



Universiteit Utrecht

Graduate School of Natural Sciences

Goal, mistake and success learning through resimulation

Curriculum learning in reinforcement learning

MASTER THESIS

Niels Scholte

Artificial Intelligence



Supervisors:

Dr. Shihan WANG
University Utrecht

Dr. Till MILTZOW
University Utrecht

January 20, 2022

Abstract

In this thesis, we improve reinforcement learning through curriculum learning. We pioneer a new approach to curriculum learning based on resimulation. We formulate 2 approaches to resimulation. Namely, Goal-Based Resimulation (GBR), where we resimulate after changing the goal, and Initial-State-Based Resimulation (ISBR), where we resimulate after changing the initial state. We construct one GBR method, namely G, where the goal is set to be the last state of the resimulated episode. G is shown to enable solving tasks that are neither solvable by using Proximal Policy Optimization (PPO) [Schulman et al., 2017], an Intrinsic Curiosity module (ICM) [Pathak et al., 2017] nor Hindsight Experience Replay (HER) [Andrychowicz et al., 2017]. We construct 2 ISBR methods, namely S+ and S-. Both methods process the advantage estimates to determine swing events, periods with high amplitude advantage estimates. S+ and S- then resimulate successes and mistakes, respectively, by setting the initial states to be the states at the start of swing events. All methods are tested on two tasks that only differed in the level of sparsity, and 3 reward ratios, controlling for the extent with which the ICM was used. The performance is measured in the solve rate and the learning speed. We find that

- G enables solving the proposed tasks.
- S+ improves the solve rate, but only on sufficiently sparse tasks and when using ICM.
- S- improves both the solve rate and the learning speed.
- G, S+ and S- can be used in unison to create a better combined algorithm.

Preface: relevance to AI

In this thesis, we work on a subfield of machine learning, called reinforcement learning. The aim is to have a robot or computer, situated in an environment, learn a task on its own, without being told what to do explicitly by programmers. For example, video game playing agents fall in this category. Reinforcement learning comes down to simulating an action taking agent in an environment over and over again, adjusting the actions it takes every time automatically through an algorithm. Eventually, this should lead to an agent that is competent at the task it was presented. The programmer of the agent has no definitive say in the strategy the agent will employ, nor will they know what the exact logic is that the agent uses to determine its actions. All that matters is that the agent gets the job done, by any means necessary, within the restrictions of the environment.

Reinforcement learning is different from supervised learning, which is also a subfield of machine learning. There, agents also have to learn some form of logic from data. Again, this logic is often unknown to the programmer. However, the big difference is that in supervised learning the objective is differentiable. This means that we can compute exactly how to slightly tweak the agent such that the agent becomes better at the task. For example, scanning medical images for a number of diseases falls in this category. In reinforcement learning, the objective is not differentiable, meaning we have to use heuristics and much noisier signals to determine how to tweak the agent. This leads to a much slower and less stable learning process. It is common that a problem is too hard, or the algorithm is not adequate for the task. Then the learning process can collapse or simply not learn at all. In such cases, it is not a matter of adjusting the hyperparameters to solve the problem, but an entirely different approach to the problem is required. This is different from supervised learning.

In this thesis, we merge another subfield of machine learning with the field of reinforcement learning, namely curriculum learning. We apply the concept of a curriculum, as seen in any kind of human education system, to reinforcement learning agents.

Contents

1	Introduction	4
1.1	Reinforcement learning	5
1.1.1	MDP	6
1.1.2	Common algorithms	6
1.2	Curriculum learning	9
2	Related work	13
2.1	Algorithms	13
2.1.1	Classification of literature	13
2.1.2	Sample-based	14
2.1.3	Multi-agent-based	16
2.1.4	Goal-based	17
2.1.5	Initial-state-based	19
2.1.6	Other methods	20
2.2	Simulation environments	22
2.2.1	Visual	22
2.2.2	Robotics	22
2.2.3	Navigation	23
2.2.4	Multiplayer	23
2.2.5	Toy	23
2.2.6	Other	24
3	Methods and implementations	25
3.1	Algorithms	25
3.1.1	GBR	25
3.1.2	ISBR	27
3.1.3	Resimulation implementation	29
3.1.4	Summary and hypotheses	32
3.2	Experimental setup	34
3.2.1	Task design	34
3.2.2	Experiment design	36
3.3	Implementation details	38
3.3.1	Model architecture	38
3.3.2	Optimizer	39

3.3.3	Normalization	39
3.3.4	Generic RL parameters	39
3.3.5	State inflation	39
4	Results	41
4.1	Quality measures	41
4.1.1	Solve rate	42
4.1.2	Learning speed	43
4.2	Data exploration	46
4.2.1	Experiment 1	46
4.2.2	Reward ratio 1:9	51
4.2.3	Experiment 2	56
4.3	Definitive results	60
4.3.1	Solve rate	60
4.3.2	Learning speed	62
5	Discussion	65
5.1	Experimental insights	65
5.1.1	Hypotheses and results	65
5.1.2	Future research	66
5.2	Experimental process	67
5.2.1	Fixed initial conditions	67
5.2.2	LSTM considerations	69
5.2.3	Insights from experiment 0	70
5.2.4	Future research	71
6	Conclusions	72
	Appendices	73
A	Supplementary figures	74

Chapter 1

Introduction

In this thesis, we focus on improving reinforcement learning through curriculum learning. More specifically, we pioneer a new approach to curriculum learning based on resimulation. The central idea is that we revisit old experiences by playing them out again, sometimes with a small change.

In short, reinforcement learning methods aim to teach an agent how to take good actions while only being told where they did well and where they did badly. This provided signal is assumed to be sparse and of poor quality. Curriculum learning aims to improve learning by providing the agent with problems that are beneficial for learning. Often this comes down to somehow first providing the agent with easy problems before moving on to harder, more interesting problems. More general, a curriculum is any ordering of information in which this information is provided to a learner, to aid learning, as opposed to sampling fully randomly from some sort of distribution from which training data is otherwise gathered.

Of course, this approach is inspired by humans. People employ curricula in almost all situations where learning takes place. Even if there is no class, workshop or supervisor to follow, the act of looking up information generates a curriculum in and of itself since the query is a non-random request for specific information.

In this thesis, we propose our own curriculum learning methods and show they improve learning in terms of the learning speed and the probability of finding a solution in a sparse task. There are two main methods that we propose, both relying on the idea of resimulation. The first method resimulates episodes (single runs of the agent, i.e. single played games) with goals that were achieved previously in the corresponding episodes. The idea here is that the agent always has some data from feasible tasks. This method is called **Goal-Based Resimulation (GBR)** because it resimulates an episode, keeping everything the same, except for the goal. It is the changing goal that gives this resimulation method its effectiveness.

The second method identifies unexpected, or **swing events** and resimulates those. The idea here is that these swing events are highly instructive (containing a clear signal on how to improve; beneficial to learning) and thus are worth revisiting. On the one hand, this aims to create a clearer reward signal around fringe cases that are otherwise too sparse to learn from (**success learning**).

On the other hand, the method stabilizes learning by resimulating cases where higher rewards were expected, and thus likely were obtained in the past (**mistake learning**). All in all, this method is called **Initial-State-Based Resimulation (ISBR)** because it changes the initial states. Again, it resimulates an episode, keeping everything the same, except for the initial state.

Once we define what exactly a swing event is and which goals will be used for GBR, we will be able to more clearly refer to exact algorithms. From then on, the exact proposed algorithms for GBR will be referred to as **G** (G for goal) and mistake and success learning will be referred to as **S-** and **S+** (S for swing) respectively. Hence, GBR and ISBR will be used to refer to the general resimulation ideas, not the exact algorithms.

We design a task and show that G improves learning by enabling the algorithm to solve the task at all. S- and S+ are then shown to improve the performance further in terms of the solve rate and the learning speed. We also show that together, the proposed methods work best. Because resimulation only affects which experiments are run, it should be able to function well in conjunction with other methods aiming to solve similar problems. By testing in conjunction with an exploration algorithm, we take a step towards more modular RL algorithms, since algorithms should be designed to work well with other methods. We find that S- improves the performance mostly independent of the exploration method, but that S+ relies on it to contribute value.

Before we go into the details of these methods, we first introduce the fields of reinforcement learning and curriculum learning, commonly used terminology and notation, and commonly used algorithms in chapter 1. This lays the groundwork for a thorough survey of curriculum learning literature in chapter 2.1, discussing the methods used to improve reinforcement learning through curriculum learning. In chapter 2.2 we revisit this pool of literature, but this time focusing on the problems that are attempted to be solved. We discuss the variety, complexity and where their implementations can be found. Having seen the curriculum learning literature, we introduce our own research in chapter 3, and we discuss how to evaluate the performance. In chapter 4 we present the results and in chapter 5 these are discussed. We conclude with the conclusions in chapter 6.

1.1 Reinforcement learning

Reinforcement learning (RL) is a field in which agents are trained to perform tasks in environments. They learn these tasks by reinforcement, by which we mean that they get given feedback on actions if they did something that was clearly good or bad. What exactly it means to be clearly good and clearly bad depends on the task. Take for example the game of chess, one could attempt to teach an agent to play chess by having it play against other existing algorithms or even itself. We could say the agent performed poorly if it lost and that it did well if it won. Intuitively, this poses a hard learning problem because the agent has to perform a long sequence of complicated actions before it gets told how well it did, and therefore it doesn't obtain a clear signal that tells it how to improve. We can help the agent by also telling it that it did well if it takes a piece and that it did poorly if it lost a piece, but this also inherently changes the task, as the agent will no longer fully focus on winning.

1.1.1 MDP

In RL literature, such tasks are often modelled as Markov Decision Processes (MDP) [Wiering and Van Otterlo, 2012]. Intuitively, this way of modelling tasks means we assume that previous states and actions do not contribute any relevant information when provided alongside the current state. An example where such a model is generally adequate is again chess. Most of the time it does not matter how players got to a certain position, the current board configuration contains all information they need to make a decision. Exceptions are edge cases like draw rules, which illustrates that a MDP is still just a model, but a useful one nonetheless.

Definition 1.1.1. (MDP) A *Markov Decision Process* M is a 4-tuple (S, A, p, r) , where S is the set of states, A is the set of actions, $p : (s, a, s') \mapsto P(s'|s, a)$ is a transition function that gives the probability of transitioning to state s' after taking action a in state s and similarly $r : S \times A \times S \rightarrow \mathbb{R}$ is a reward function that assigns the immediate reward to the tuple (s, a, s') for taking action a in state s and transitioning to state s' .

Continuing with the chess example, we see that we can model chess with a state space S that is the set of all viable board configurations, an action space A that is the set of all viable moves, a transition function that models the opponent and a reward function that indicates if there is a winner, a draw or if the game is still ongoing.

There are many extensions of the concept of an MDP. A notable one is the concept of an *episodic* MDP, with the main idea being that an agent has to navigate from starting states to goal states. As such, a starting state distribution and a set of terminal states are added to the tuple, making an episode MDP a 6-tuple.

Often, a discount factor $\gamma \in [0, 1]$ is added as a task specific property as well. The purpose of this factor is to reward past action for future rewards. This is needed when good actions do not yield immediate rewards. For example, in chess, if rewards are only obtained when a player wins, the agent will only learn game ending moves or moves that prevent an immediate loss on the next move of the opponent. In other words, it will only know what to do when there is a mate-in-one threat. This dynamic is prevented by using **accumulated rewards** (or **payoffs**) R_t instead, given by

$$R_t = r_t + \gamma R_{t+1} = \sum_{i=t}^N \gamma^{i-t} r_i,$$

where r_t is the immediate reward assigned to the transition (s_t, a_t, s_{t+1}) on time step t by the reward function r . This way, actions are also awarded for future rewards, but less for rewards that are obtained in the distant future and more for rewards obtained in the near future.

1.1.2 Common algorithms

We now move on to common RL algorithms used to solve MDPs. In this thesis we opt to use the well established general purpose RL algorithm **Proximal Policy Optimization (PPO)**, hence we will only discuss exploitation methods that aid in understanding this method. However, it is important that we also explain at least one exploration algorithm here. Most importantly because we will be using this particular method heavily throughout the thesis, but also because the trade-off between exploitation and exploration is fundamental to RL. It is also important to note that

exploration algorithms are very much related to curriculum learning, and it is important to see the differences.

There are two reasons why we make heavy use of an exploration algorithm in this thesis. First, it is because we want to ensure that our methods do more than just incentivize exploration when solving very sparse problems. In a sense it serves as a baseline, but since it is also included in most experiments, the patterns we see will not be because of exploration incentive, but rather due to other factors, since exploration incentive is already accounted for. Second, it serves as a component in a modular approach to RL. Agents often have to overcome many difficulties to solve a problem, and new algorithms are constantly being developed. However, it is not always clear how to combine these methods stably to form a better agent. Designing algorithms to work well in conjunction with other methods and verifying this behaviour makes it easier to put together multiple components. In other words, this leads to a more modular approach to RL, allowing specialized algorithms to more easily come together. Therefore, the exploration algorithm has a pivotal role in this thesis and will also be treated here.

Q-learning and SARSA

There are many algorithms for learning good policies, the maps from states to actions that agents use. The most classical ones that we discuss here are Q-learning [Watkins and Dayan, 1992] and SARSA [Wiering and Van Otterlo, 2012]. These methods aim to estimate the values (read: expected accumulated rewards) of (state, action) pairs given the agent’s policy. Since the policy depends on these valuations, the hope is that after sufficiently many iterations of playing and learning, the valuations are correct assuming perfect play. This implies that the highest possible rewards are obtained if the agent always chooses the highest valued action. Therefore, the agent will be optimal if it greedily only selects the best actions.

Both methods estimate the values similarly. Roughly, they both keep a table of (state, action) pairs (called a Q-table storing Q-values, hence the name Q-learning) and update the entries like decaying running averages, through a fixed-point iteration. They iteratively tweak the values of their tables to be more like accumulated rewards that were obtained. This table can be explicit in the case of small discrete action and state spaces, and implicit otherwise, using an artificial neural network to infer the entries of the table.

SARSA is most like this sketched approach. It uses the update rule

$$Q_{k+1}(s_t, a_t) = Q_k(s_t, a_t) + \alpha \left(r_t + \gamma Q_k(s_{t+1}, a_{t+1}) - Q_k(s_t, a_t) \right),$$

where $\alpha \in [0, 1]$ is the step size that controls how much the current value is updated each iteration and Q_k is the Q-table at update step k . Key is the term $\gamma Q_k(s_{t+1}, a_{t+1})$ that shows that the expected accumulated rewards are estimated for the actual behaviour of the agent, since a_{t+1} is the realized action that was taken at the next time step given the next state.

In contrast, Q-learning assumes that the agent only plays the action that is assigned the highest value. As such, it tries to more directly obtain the Q-table for an optimal agent. It uses similar the update rule,

$$Q_{k+1}(s_t, a_t) = Q_k(s_t, a_t) + \alpha \left(r_t + \gamma \max_a Q_k(s_{t+1}, a) - Q_k(s_t, a_t) \right).$$

The two methods are most similar in the case that the agent plays only best moves. Then they are only different if the action with the highest estimated value changes from when the sample was obtained. However, when using pure Q-learning or SARSA this means that the agent only exploits and never explores. As a consequence, the algorithm then gets stuck in local optima.

Another common pitfall arises when using a neural network to infer the entries of the Q-table. Because the samples from a single run are highly correlated, in practise it is often best to store the transition samples in a replay buffer. The algorithm then samples from this buffer instead when updating the Q-network [Mnih et al., 2013]. This will be discussed further in section 2.1.2.

A2C

A third RL algorithm is Advantage Actor Critic (A2C) [Degris et al., 2012] [Mnih et al., 2016]. A2C is a policy gradient method. It decouples value estimation from action choosing by directly predicting what actions are good. It only uses value estimation to create good updates to the policy. In essence, it directly increases the probability for actions that are good, and decreases them for actions that are bad.

To this end, we formalize the policy π as producing a probability distribution over actions, given a state. More explicitly, if one were to build such a method from scratch, a first idea could be to try to update π such that $\pi(a_t|s_t)Q_\pi(s_t, a_t)$ is high, where $Q_\pi(s_t, a_t)$ is the true expected accumulated reward of (s_t, a_t) for the policy π .

It turns out that $Q_\pi(s_t, a_t) - V_\pi(s_t)$, called the advantage of (s_t, a_t) , is a better way of measuring the quality of an action, where $V_\pi(s_t)$ is the true expected accumulated reward of s_t . However, in practise, neither Q_π nor V_π are available. This leads to the use of **advantage estimates** instead, where $Q_\pi(s_t, a_t)$ is estimated through R_t , and $V_\pi(s_t)$ is estimated through a value function v , a neural network that is trained to approximate V_π . After replacing $\pi(a_t|s_t)$ by $\log \pi(a_t|s_t)$, to intuitively only fix some issues with how the probability distribution is created, this leads to the gradient

$$\nabla_\theta \log \pi_\theta(a_t|s_t)(R_t - v(s_t)),$$

where θ is the parameterization of π_θ .

PPO

A fourth RL algorithm is Proximal Policy Optimization (PPO) [Schulman et al., 2017]. The main innovation is a simple way of not requiring replay memory. Before, if one were to perform multiple updates using the same highly correlated data, performance would either collapse due to updates that are too large, or one would have to implement complicated trust region methods that define regions where updates are still warranted. PPO greatly simplifies the latter by simply setting the gradient to be zero for samples where the new policy (the policy after some updates) differs too much from the old policy (before the updates). This approach is enabled by switching out $\log \pi_\theta(a_t|s_t)$ for $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ in the A2C loss, where θ are the parameters that are being updated, and θ_{old} are the frozen parameters of the model before the latest updating phase.

More concretely, it cleverly clips the ratio $r_t(\theta)$ in the maximization objective $r_t(\theta)(R_t - v(s_t))$ such that progress halts on samples on which too much progress has been made already. This still allows updates on samples where the policy changed in the wrong direction or where no progress has been

made at all. In other words, the policy loss is not blindly clipped if the new policy differs too much from the old policy. Because the objective is clipped to prevent overfitting, the objective can safely be optimized till convergence, even on limited highly correlated samples, making the most of the available data.

ICM

The methods we have seen above are all exploitation algorithms. They define how to make changes to the policy based on the yielded rewards. However, often rewards are very sparse or not present at all. In those cases, methods are required that learn on their own. The **Intrinsic Curiosity Module (ICM)** proposed by [Pathak et al., 2017] does this by generating its own dense rewards and then using an off-the-shelf exploitation algorithm to do the learning.

Intuitively, the method assigns high rewards if it cannot accurately predict relevant information about the next state s_{t+1} , based on the current state s_t and the taken action a_t . Crucial is that only relevant information is predicted, such that rewards are not based on the prediction error of irrelevant features. As such, the approach consists of two main components. The first is a forward model that predicts the relevant information about the next state. The second is an inverse model that has a large role in generating this relevant information. The network that extracts relevant information from states is implicitly trained through the joined effort of the forward and the inverse models.

The inverse model takes in two consecutive states s_t and s_{t+1} and predicts the action a_t in between. Since the main purpose of the inverse model is to train the relevant information extractor, this network is part of the inverse model and is run on both s_t and s_{t+1} . Similarly, the relevant information extractor is also part of the forward model, as its predictions are based on s_t and a_t . The rewards are then defined as the square of the distance between the output of the forward network and the relevant information extractor ran on s_{t+1} .

ICM is not the only one of its kind. In section 2.1 many alternative exploration algorithms will be discussed.

1.2 Curriculum learning

We now move on from the basics of RL to the main topic of this thesis, namely curriculum learning (CL). The idea is fundamental to the learning process in humans; one should learn the basics first before moving on to more difficult concepts or problems. This sparked researchers to use this concept to improve learning in machine learning. Recently, [Narvekar et al., 2020] summarized this pool of literature in a review paper. They proposed a framework for CL in terms of definitions that we loosely follow to define CL here. It's important to emphasize these concepts are solely used to help the reader think about CL and for them to see structure in the vast pool of literature. Fundamentally, CL is still nothing more than ordering experiences in some way to improve learning.

[Narvekar et al., 2020] also approach CL from the perspective of having a set of tasks T modelled as MDPs. These MDPs together then induce a set D^T of all possible transition samples, where each transition sample also includes the reward awarded by the MDP it was obtained from. Each element is then of the form (s, a, r, s') . In this tuple s is a state in which action a can be taken, s'

is a possible state to which can be transitioned and r is the immediate reward of this transition, as awarded by the relevant MDP in T . More formally, we define D^T as

$$D^T := \{(s, a, r(s, a, s'), s') | s, s' \in S, a \in A, p(s'|s, a) \neq 0, (S, A, p, r) \in T\}. \quad (1.1)$$

Similarly we can define D_M^T as the set of all transition samples obtained from a single task $M = (S, A, p, r) \in T$, namely

$$D_M^T := \{(s, a, r(s, a, s'), s') | s, s' \in S, a \in A, p(s'|s, a) \neq 0\}. \quad (1.2)$$

The idea is then that we can construct a connected directed acyclic graph where the nodes are subsets of D^T and where the directions of the edges indicate on which subsets the agent must be trained first. Therefore, we define a curriculum as follows.

Definition 1.2.1. (Curriculum) Let T be a set of MDPs. Let D^T be as in equation 1.1. A *curriculum* C for a task set T is a connected directed acyclic graph (V, E) where the vertex set V contains only subsets of D^T and E is the set of directed edges. An edge $(x, y) \in E$ indicates that one should train by sampling from x before training by sampling from y .

Note that often, curricula are build to improve the learning on a target task $M_t \in T$, and this target task is trained on last. In such a case, there is only one vertex with out-degree zero, and this vertex is $D_{M_t}^T$. With this definition, we can define a single-task curriculum for single-task curriculum learning (STCL), opposed to multiple-task curriculum learning (MTCL).

Definition 1.2.2. (Single-task curriculum) A *single-task curriculum* is a curriculum for a task set T , where $|T| = 1$.

Definition 1.2.3. (Multiple-task curriculum) A *multiple-task curriculum* is a curriculum for a task set T , where $|T| > 1$.

Using the broadest definition of a state, in which we can store whatever data we want in there, even including goals that alter the reward function, ISBR uses STCL whereas GBR generally uses MTCL. ISBR uses STCL because all initial states were previously reached, meaning that all samples obtained from ISBR can also be obtained from the original task. In general, GBR methods are MTCL methods, since the goal when resimulating can be any previously achieved goal, not necessarily one from the distribution defined by the task.

Two other frequently recurring curricula are task-level curricula, in which the agent is trained on subsequent tasks, and sequence curricula, in which the agent learns from chunks of data in a linear order, not having any invariance under the order in which an agent is trained on the nodes of the graph.

Definition 1.2.4. (Task-level curriculum) A *task-level curriculum* is a curriculum for a task set, T where $V \subseteq \{D_M^T | M \in T\}$.

Definition 1.2.5. (Sequence curriculum) A *sequence curriculum* is a curriculum where the in-degree and out-degree of each vertex is at most 1.

Notice that these definitions do not exclude one another. A curriculum, consisting of a single node for a single task, can be both a single-task curriculum, a task-level curriculum and a sequence curriculum all at the same time. Also notice that neither GBR nor ISBR use a sequence curriculum.

After all, the curricula are generated on the fly. Any resimulation task can be generated at any time depending on the performance of the agent.

From these definitions, it should be clear that the field is very broad. This is because any ordering of sampled data, explicit or implicit, is considered a curriculum. We will limit the literature review in section 2.1 to STCL and MTCL where only the reward function, goals as a part of the states, and the initial states or the terminal states are changed to form tasks. In the framework where tasks are modelled as MDPs this means that the action space A stay the same. The reward function r is allowed to be altered freely, and the state transition function p may be altered on the initial states and the terminal states. If one were not to limit the extent in which MDPs may be altered in this way, one would quickly stumble into task-level MTCL.

Task-level MTCL

According to [Narvekar et al., 2020], almost all curricula in current literature are sequence curricula. It must be noted that this very heavily relies on the exact definition of a sequence curriculum and the scope of the pool of curriculum learning literature that is considered. In practise, the lines get fairly blurry, especially when considering exploration tasks. In such cases, it is often unclear if these tasks have an objective at all. Still, there is clearly some merit to this claim if the focus is on task-level MTCL.

In this thesis, however, we will mostly avoid task-level MTCL. In part, this is because it is not immediately relevant to the methods proposed here. Although, GBR uses MTCL, it is not task-level. It just sets the goals that were achieved, regardless of some form of task order. However, it is mostly because the problems that arise in task-level MTCL are fundamentally different from problems that arise in other parts of CL. Nonetheless, task-level MTCL is an intuitive and important branch CL, too important to simply ignore. Therefore, we dedicate the rest of this section to task-level MTCL and relate it to another part of CL that is relevant to this thesis.

Task-level MTCL is arguably the most intuitive form of CL. On a high level, it resembles human education systems the most. Students have subjects that they go through, often in parallel. These subjects can often be studied in arbitrary order, except for when advanced courses that require more basic courses. This naturally creates a graph of tasks. Similarly, in task-level MTCL, one aims to break the main tasks into multiple smaller tasks.

The biggest problems that arise in task-level MTCL come from the smaller tasks that need to be generated, ordered and the knowledge that needs to be transferred between them. In other words, the first problem is that the generation of tasks, where the state space S or the action space A are changed freely, is domain specific [Narvekar et al., 2016] [Silva and Costa, 2018]. This means that one needs to manually build a task generator using domain knowledge. In general, this is not trivial, as the target task can be very intricate. Hence, handcrafting meaningful changes to the target task that significantly scales the complexity is not a feasible approach to RL in general. It is a step away from general purpose RL algorithms.

The second problem is that ordering the tasks in task-level MTCL is non-trivial, and if done wrong can lead to negative transfer [Narvekar et al., 2017]. Intuitively, this becomes a bigger threat when tasks differ more. The third problem is the complication of knowledge transfer, which can be entirely ignored when using STCL or when tasks are very much alike.

Regardless, CL seems most relevant when an agent first has to acquire basic skills that it must employ to solve a harder task. Abstractly, this is the case when the data samples provided to the agent do not adequately describe the trajectory it has to traverse to become better at the task, but instead only describe the trajectory it has to traverse to complete the task. For example, if the agent needs a key to move through a door, it would be best for learning if the reward signal indicates that the agent needs to move to the key and then to the door, rather than pointing to the desired location beyond the door the whole time. Splitting the task in two part, where the agent first learns to obtain the key and then learns to move to the door, might be tempting, but again doesn't scale well with more difficult problems. Therefore, it is important to note that this problem can also be phrased as an exploration problem, where the agent insufficiently explores to learn on its own terms. In other words, CL problems can sometimes also be solved by exploration algorithms.

The reverse relation is also interesting. Many algorithms that try to solve exploration problems satisfy the restrictions imposed on the MDPs in the rest of this thesis, as mentioned earlier. They often aim to systematically explore the state space. The agents learn to take actions to find more and more different states they have not visited before. States closer to the starting state are explored first, and states that require more complicated action patterns to reach are explored later. This induces a curriculum! In other words, we do not need task-level MTCL to practise CL at all, although it might be the most intuitive kind of CL.

Chapter 2

Related work

Here we give a thorough survey of CL literature for RL, limited by the restrictions mentioned in section 1.2, which can be summarized as ignoring all forms of task-level MTCL. We start with an overview of CL algorithms in section 2.1 before review the same pool of literature again in section 2.2, but then from the perspective of the simulation environments used to test these algorithms.

2.1 Algorithms

2.1.1 Classification of literature

The review will be split in four parts, corresponding to a classification based on four main criteria. The first criterion identifies methods that change sampled data. Notice that a vertex in a curriculum can have infinite cordiality, since a task can induce infinitely many transition samples. For most curricula, one can obtain a different curriculum by simply sampling from the vertices and replacing the vertices by the sampled sets. Especially if these samples are obtained by a reinforcement learning agent and are highly correlated, one can imagine that the former curriculum contains much more information than the latter. Therefore, the first criterion on which papers are categorized is that the vertices of the curriculum are simply sets of sampled data, rather than that the vertices themselves can be sampled from to generate unseen transition samples. For example, the nodes of task-level curricula in MTCL intuitively contain much more information than the nodes of a STCL method that only reorder experience data. Methods that only change sampled data to create a curriculum, rather than generating new data, will be called **sample-based** methods.

The second criterion is that the curriculum is created through the use of multiple RL agents. In all aspects of life, competition has driven optimization and innovation, with the most notable example being life itself, as evolution is driven by natural selection. This can also be leveraged in RL to more efficiently explore or model. Methods that use multiple RL agents will be called **multi-agent-based** methods.

The third criterion identifies methods that construct explicit secondary goals and use these in the process of learning. Goals are simply part of the state; the agent observes them and should act according to the changed behaviour of the reward function. Goals are a large theme in CL. They can

be used by the agent to more thoroughly understand the environment. Sometimes they even allow agents to complete tasks for which they have not been explicitly training. They also allow agents to learn from repeated failures by viewing failures as successes in secondary tasks. Methods that use explicit secondary goals will be called **goal-based methods**, not to be confused with **goal-based tasks**, which are tasks in which the state contains a goal that may vary between episodes. One of our proposed methods, **Goal-Based Resimulation (GBR)**, falls in this category. It resimulates previous runs with the twist that it replaces the old goal with the goal that was previously achieved in the final state of that particular episode.

The fourth criterion is that the methods alter the initial state distribution. In a way this is the opposite of goal-based methods since now a single terminal state is attempted to be reached from many initial states, rather than attempting to reach many goals states from an often fixed initial state. Methods that explicitly alter the initial state distribution will be called **initial-state-based methods**. One of our methods, called **initial-state-based resimulation (ISBR)** is somewhat related to this approach as it also alters the initial-state distribution. However, it does not fall into this category, as it does not at all start at the solution and then works backwards. Instead, it looks for initial states from which most learning took place and resimulates from there to obtain more instructive data, aiding the learning algorithm that remains unchanged.

Many papers combine the above approaches. Recently, goal-based methods have become very common. This leads to some overlap in the categories. Therefore, we will discuss sample-based methods first in section 2.1.2 before moving on to multi-agent-based methods in section 2.1.3. After that, we review the remaining goal-based methods in section 2.1.4 and initial-state-based methods in section 2.1.5. Lastly, we finish the literature review with some smaller categories in section 2.1.6, where we treat leftover methods that are neither sample-based, multi-agent-based, goal-based nor initial-state-based.

2.1.2 Sample-based

The first sample-based method is actually a paper on supervised learning. The idea of [Bengio et al., 2009] is to verify that by training on easy examples first, and more difficult examples later, improves performance. This turns out to be the case in multiple experiments when creating multiple training sets. They also suggest sampling training samples based on a weight that promotes easy samples early on in training, but don't show any results for such an approach. In a sense, this paper implements the core idea of CL.

In [Schaul et al., 2015b] this idea is applied to reinforcement learning. Transition samples in the experience replay are no longer sampled uniformly, but instead the authors aim to sample them weighed by their significance, leading to the name Prioritized Experience Replay (PER). The temporal-difference (TD) error [Wiering and Van Otterlo, 2012] is used as a proxy for significance. The method improves the performance of DQN (Q-learning with an implicit Q-table) on many Atari games.

[Ren et al., 2018] improve PER by more actively sampling transitions that are sampled infrequently, formalized by a coverage penalty. They also condition the proxy for significance on both the TD-error and the difficulty of the current curriculum, called self-paces priority.

[Andrychowicz et al., 2017] take a different approach to sample-based methods. They do not attempt to improve learning by ordering and replaying past experiences, but rather focus on the

problem of not learning when there is a single goal state in a very large state space. Their main insight is that an agent can still learn from failed experiences by also rephrasing the failure to reach the goal state as the success to reach a state that was actually reached. This way, the transition samples from which the agent learns can contain both failures and successes. By conditioning the policy on goals as in [Schaul et al., 2015a], the agent learns the relation between states given goals, allowing it to generalize knowledge about reaching goals to find goal states in a very large state space. The method is called **Hindsight Experience Replay (HER)** as it learns from experiences knowing, in hindsight, what goals were achieved in the future of the transition samples. This is a very influential paper, and we will be returning to it often.

[Fang et al., 2019] improve HER by noticing that phrasing failures as successes can lead to learning to achieve irrelevant goals. Similar to the approach of [Ren et al., 2018], they use a similarity measure between goals as a proxy for significance and a dissimilarity measure that ensures diversity. This creates a curriculum for HER, leading to the name Curriculum-guided HER (CHER).

[Sun et al., 2019] extend HER by assuming that every state can be mapped to a goal. Every trajectory then describes many relations of how to achieve many future goals given many current states. The idea is then to gradually learn more long term relations. More concretely, denote g_t the goal induced by some state s_t in some trajectory. Then the algorithm starts by constructing a dataset that consists of pairs (s_t, a_t, g_{t+1}) that indicate that the agent needs to take action a_t in state s_t to reach goal g_{t+1} in the shortest time. This dataset can be trained on with regular supervised learning. The time step between the state in which the action is taken, and the goal is then progressively increased, leading to tuples of the form (s_t, a_t, g_{t+k}) for optimal action sequences of length k . Some checks are put in place to ensure that a_t is indeed the fastest known action to reach g_{t+k} from state s_t . This explicitly constructs a curriculum through the increase in required number of actions between the state in which the agent acts and the goal that is aimed to be reached. The policy for short action sequences should remain somewhat stable when moving to slightly longer action sequences, as it still describes optimal action sequences for the shorter paths, which should be part of the longer optimal action sequences. This process of extending a policy to a greater state space is called policy continuation, leading to the name Policy Continuation with Hindsight Inverse Dynamics (PCHID).

It is important to emphasize that PCHID uses heuristic tests to solve an important shortcoming of HER, namely that HER does not guarantee that the action sequence was taken is an adequate action sequence to reach the achieved goal in the final state. If an agent performs a random walk, it could very well end up where it stated. This does not at all mean this random walk is a good action sequence for reaching the final state, since this final state is the first state! Not moving at all would be the more intuitive thing to do. If an agent has already learned the optimal action sequence to achieve a goal, attempting to teach it suboptimal paths will cause it to start acting suboptimally again and forget the optimal path, setting back the learning process. Since it is generally much easier to achieve a goal suboptimally than optimally, this becomes a source of much undesired noise in the learning process. Rather than using heuristic tests to check if a sample is suitable for hindsight training, our approach to this problem is to avoid learning in hindsight altogether through the use of resimulation. This will be discussed further in chapter 3.

2.1.3 Multi-agent-based

Although neither of our proposed methods makes use of multiple agents to improve learning, multi-agent-based algorithms are an important part of CL research. Since they often lack a target task, being exploration algorithms, it can be difficult to classify them as MTCL or STCL. Still, since the curricula are dynamically generated they are neither task-level nor sequence curricula, making them similar to our methods. A striking exception will be discussed last, in which a multi-agent-based curriculum is used to increase the quality of the data. Similarly, we will also be constructing a curriculum that increases the quality of the data through ISBR.

The first multi-agent-based paper is [Sukhbaatar et al., 2017] where the authors aim to explore an environment in an unsupervised manner by pitting two reinforcement learning agents against each other. Both agents, Alice and Bob, learn to navigate the environment. However, each agent has a different objective through clever reward function design. Every iteration, the two agents take turns acting in the environment. Alice goes first and must, before time runs out, select a moment to terminate her turn, which resets the environment. On Bob’s turn, he is given Alice’s final state and must get to it before time runs out, being given only the time Alice had left. When their turns end, Alice and Bob will have spent time t_A and t_B time in the environment respectively, with $t_A + t_B \leq t_{max}$, the shared time. For this iteration, Alice is awarded a reward of $\max(0, t_B - t_A)$, rewarding her for Bob taking more time on his turn than her. Bob is awarded $-t_B$, rewarding him for quickly reaching Alice’s final state. This leads to Alice setting more complicated tasks if Bob can keep up, and setting easier tasks if Bob cannot. Because the agents function together but are still trained with their own objective, this algorithm is called Asymmetric Self Play.

[OpenAI et al., 2021] extend ASP by allowing Bob to also train using Alice’s trajectories via behavioural cloning. Notice that they could have gone a step further in the same line of thought if they implemented PCHID instead, training on all intermediate segments as well. They then apply ASP to robotic manipulation and evaluate its zero-shot generalization capabilities.

[Liu et al., 2019] propose a different algorithm of two interacting agents, where the overall objective of the algorithm is to promote exploration. Again, both agents, Alice and Bob, learn to navigate the environment. This time, both agents are interacting with their own version of the environment in parallel. The two runs are paired up and stored in a replay buffer. Mini batches are sampled from the replay buffer for training and are therefore of the form

$$\{(s_A^i, a_A^i, g_A^i, r_A^i, s_A^i), (s_B^i, a_B^i, g_B^i, r_B^i, s_B^i)\}_{i=1}^m$$

where m is the batch size. A small δ is fixed and for all i it is checked if there exists a j such $|s_A^i - s_B^j| < \delta$. If so, r_A^i is relabelled to be $r_A^i - 1$ and r_B^j is set to be $r_B^j + 1$. In other words, Bob is rewarded to get to states Alice also gets to, and Alice is penalized when this happens. This leads to a dynamic where Bob chases Alice and Alice tries to escape from Bob. Because the agents are competing and because experiences are being relabelled, the method is called Competitive Experience Replay (CER). They show that this approach outperforms ICM, even when both methods are used in conjunction with HER.

[Baker et al., 2019] also study agents chasing each other. This time more directly in a team based game of hide and seek, in a 3D environment with simulated physics and interactable objects. The teams show emergent strategies that become progressively more complicated as training goes on, inducing a curriculum. This demonstrates the power of combining large scale RL with competition.

Earlier, [Bansal et al., 2017] reached a similar conclusion using smaller scale experiments. The main takeaway being that simple rules can induce complex behaviour, a common phenomenon in biology.

[Vinyals et al., 2019] take large scale RL much further by training an agent, called AlphaStar, that reaches Grandmaster level in the game of StarCraft 2. They train the agent through an ecosystem of agents called Alpha League, instead of only learning from self-play, which was still the norm for AlphaGo [Silver et al., 2016]. The league contains fixed old versions of learning agents, as well as exploiters that only train against a limited number of opponents to exploit their weaknesses. This leads to a curriculum that stays sufficiently difficult and divers throughout training.

Similar to this last paper, we will also be constructing a curriculum that increases the quality of the data through ISBR. However, instead of using multiple agents, we will be processing the advantage estimates. Using this, we determine instructive segments of past runs and resimulate those. More on this in chapter 3.

2.1.4 Goal-based

We have already seen some goal-based literature above, but those were also sample-based or multi-agent-based. Here, the main focus is on designing instructive goals that aid learning. Most of these papers aim to not just learn the structure of the task through these secondary goals, but to also actively explore the space. In contrast, we propose to only use goals to, like HER, learn the structure of the problem. We then outsource the exploration to a specialized algorithm in favour of a more modular approach.

The first paper we treat here is [Baranes and Oudeyer, 2013] that introduces an algorithm called SAGG-RIAC. The method provides a learning agent with *interesting* goals in the form of states it should learn to get close to (using the euclidean distance), where goals are defined to be interesting if the agent recently got better or worse at achieving them. More specifically, goal states are interesting if the average distance to the final achieved states of the agent has been decreasing or increasing in recent runs. The key heuristic being leveraged here is that if progress has been made in a certain direction, more progress might be possible in that, or the reverse direction. This heuristic is of course not always correct, especially when using the euclidean norm to measure the state space. A learning approach to generating goals might be more effective.

This is done in [Florensa et al., 2018] where the authors generate goals through the use of a Generative Adversarial Network (GAN) [Goodfellow et al., 2014], and have an agent learn to achieve these goals. This leads to the name Goal GAN. In a way, this is a similar approach as ASP, where instead the goal generation is done through supervised learning rather than through reinforcement learning. In ASP, the tasks are naturally of an appropriate difficulty by design of the reward function. Because here the goal generation is done through supervised learning, previously generated goals need to be labelled to ensure that future generated goals are of appropriate difficulty. This is done through estimating the success rate of the agent on the goal. If after multiple attempts the success rate is too low or too high, the task is considered to be too hard or too easy. They show that the algorithm is very robust to the cut-off ratios that define “too hard” and “too easy” and that their method outperforms SAGG-RIAC. They also compare their method to ASP and show that the performance is slightly worse in an open space navigation task, but that Goal GAN outperforms ASP in a maze setting.

[Racaniere et al., 2019] also generate goals using a generative model, dubbed a setter. The main difference is that a much more complicated goal labelling scheme is used. The immediately most apparent change is the training of a judge network that predicts if the agent is able to complete the task, called feasibility. Its output is used in one of the losses of the setter. The setter is trained using 3 or 4 losses, depending on if there is a goal distribution the algorithm designer would like to sample from.

- *Validity* promotes goals that were previously achieved.
- *Feasibility* promotes goals of the right difficulty.
- *Coverage* promotes diverse goals.
- *Desirability* promotes goals from a desired goal distribution.

The algorithm is then deployed in a much more complex environment and is shown to greatly outperform Goal GAN.

[Jabri et al., 2019] also generate goals, but instead of using an explicit goal generator, they take a more probabilistic approach. With this approach, they aim to build a CL method that explores a visual environment without incoming rewards, i.e. in an unsupervised manner. This leads to the name Curricula for Unsupervised Meta-Reinforcement Learning (CARML). Their main idea is generating a reward function based on the observed state-trajectories and training the agent using that reward function. The reward function is generated by first creating a function q that assesses how probable a state is. Since states contain goals, this probability can also be computed conditioned on a goal z . Every iteration such a goal z is sampled from q , in other words a goal is obtained that makes the seen trajectories likely under q , leading to a feasible goal. The reward of a state s is then defined to be high when $q(s|z)$ is high (i.e. when the goal is achieved) and also to be high when $q(s)$ is low (i.e. when the state is improbable and thus the state diversity is high). Intuitively, this means that a curriculum is generated for which the objectives are to go where the agent has gone before, while maintaining exploration through favouring states that are improbable given the history of visited trajectories.

[Levy et al., 2019] takes a different approach to goal-based CL. Agents are still setting and achieving goals, but now in a hierarchical fashion. The introduced method, called Hierarchical Actor-Critic (HAC), has agents take a goal and the current state as input, and output another goal, a subgoal of the initial goal. The agent at the top of the hierarchy passes the state and its generated subgoal on to the next level, where there is an agent that does the exact same thing. This process iterates until the bottom level is reached. At the bottom of the hierarchy, the agent is presented with an iteratively simplified goal and the current state. This agent then has to take an action to achieve its presented goal.

The lower an agent is in the hierarchy, the easier it is to learn, as the goals are easier and there are few other agents they depend on. However, if one agent performs badly in the chain, all agents above suffer as well. Since the agents are all learning through reinforcement learning, this causes instability if all policies are learned jointly. This also means that actions taken with exploration purposes can cause instability issues higher up the hierarchy. Not learning the policies jointly, however, mitigates the gain of Hierarchical Reinforcement Learning (HRL) as the computational overhead becomes too large.

HAC overcomes this issue by extending HER to the hierarchical setting in two ways, both through

Hindsight Action Transitions (HAT) and Hindsight Goal Transitions (HGT). HATs are transition samples made by replacing the taken action (i.e. generated subgoals) by the goal induced by the next state. A HAT is then rewarded if the induced goal coincides with the goal obtained from the higher level. HGTs are transition samples made from HATs by replacing the goal obtained from the higher level with a goal that will be achieved in the future (i.e. a goal induced by a state that will be reached). The rewards are then also updated accordingly.

By extending HER to a hierarchical setting, HAC is the first HRL method that is able to jointly train three level hierarchies in tasks with continuous state and action spaces.

[Jiang et al., 2019] also propose a HRL algorithm. In contrast to the method above, this work explicitly constructs a curriculum for the top-level agent, leading to the name Hierarchical Automatic Curriculum Learning (HACL). The method is a two-level hierarchy, but is crucially different in the way the top-level agent operates. Instead of only having one policy, the top-level agent now has two policies, one for both exploration and exploitation (we dub this the exploration agent), and one for only exploitation (the exploitation agent).

The exploration agent does most of the heavy lifting and obtains intrinsic rewards by achieving tasks set by an externally kept state graph. This state graph keeps track of which states are connected by short action sequences. It does so by grouping nearby states into macro states, with the macro states being the vertices of the graph.

The information that the state graph provides to exploration agent are sequences of macro states, only in the form of their indexes. This sequence describes how to get to the final macro state of the sequence. If the efforts of the exploration agent and the bottom level agent result in reaching this final macro state of the sequence, they are awarded additional rewards. Because the difficulty of the tasks can be measured through the past performance of the agent, appropriate tasks can be generated. This gives rise to an intrinsically rewarded CL scheme tailored to HRL. We should emphasize that the top-level agent uses the macro state sequence to divide the task of reaching the final state of the sequence into subtasks, which is precisely the information the sequence contains.

The method demonstrates state-of-the-art sample efficiency and performance on the notoriously difficult game Montezuma’s Revenge.

2.1.5 Initial-state-based

The methods in this section all assume that there exists a goal state the agent needs to navigate to. They also assume that this goal state is known and that they can initialize the agent to be near that goal. They train the agents on the simple task of reaching the goal from the nearby initial state, and iterate by training the agent from initial states that are progressively farther away from the goal. This sequence of initial states is in practise a sequence of sets, in which a particular set contains initial states that are approximately equally difficult. This general approach can find applications in cases where the original initial state can be considered simple or easy to reach, but the goal state is not, for example in a key insertion task.

These methods are in a way opposite to goal-based methods. Goal-based methods change the terminal state distribution (through the use of many secondary goals) and keep the initial state the same. In contrast, initial-state-based methods change the initial state distribution (by trying to achieve one goal from many starting states) and keep the terminal state the same.

Our proposed method ISBR is different in that it does not work backwards, but rather forwards, like goal-based methods. Because of this it is possible to use it in conjunction with other conventional methods. Another difference is that it changes the initial state distribution not by swapping out one set of initial states for another, but rather by setting some simulations to start at a state from which a trajectory was deemed to be instructive in a recent run.

[Asada et al., 1996] first proposed working backwards in this manner. There, the sequence of initial states is essentially handcrafted.

[Florensa et al., 2017] generate the sequence itself by having the agent perform random walks from the current start positions to find new candidate start positions. The expected rewards for these candidate start positions are then estimated using heuristics based on trajectories saved in the replay memory. For the next iteration, they select the moderately difficult starting states.

[Wöhlke et al., 2020] generate the sequence by assuming the state space is a euclidean vector space where nearby states are easy to reach. They then compute the derivative of the value function with respect to the state space and select initial states with high gradients. As such, the method generates a Spatial Gradient Curriculum (SGC).

[Salimans and Chen, 2018] assume they have access to an entire demonstration which avoids backwards expansion altogether. The method then simply starts episodes from demonstration states and works backwards until the first state of the task is reached.

2.1.6 Other methods

The methods in this section are on the edge of still being CL. Regardless, they propose simple ideas that turn out to yield surprisingly good results and are all relevant to this thesis.

[Justesen and Risi, 2018] assume access to the reward function. Their method adjusts rewards based on how frequently they are obtained, decreasing or increasing rewards if they are obtained frequently or infrequently, respectively. This causes the agent to seek out many different rewards, aiding exploration. They also show that the method results in a more versatile policy that adapts well to critical changes in the environment. This is particularly relevant in the light of the success learning variant of ISBR, since we improve learning by reducing the sparsity of sparse rewards, and they improve learning by having sparse rewards weigh more heavily in the optimization step.

[Burda et al., 2018] introduce the idea of fixing a randomly initialized network that makes predictions on states. By then training a network to predict the output of the random network, they are able to determine if a state is novel or not by simply looking at the difference of the outputs. If a state is novel, the error is expected to be high and thus this error can be used as an exploration reward. This method fixes the noisy TV problem encountered in [Pathak et al., 2017] where the agent gets addicted to watching TV as it cannot predict what the TV will display next, showing that ICM does not entirely succeed in filtering out irrelevant information. Since a randomly initialized network is distilled into a trained one, this method is called Random Network Distillation (RND). Because we are making heavy use of the exploration algorithm ICM in this thesis, it is important to note that more work is being done on modular exploration algorithms as well, that can simply be tacked on to any conventional algorithm.

[Oh et al., 2018] introduce Self-Imitation Learning (SIL) in which an agent tries to reproduce its own past good actions. This is done by proposing a loss function that only produces a loss if

the obtained accumulated reward is bigger than the estimated accumulated reward. The approach functions similarly to PPO, where repeated optimization steps result in a zero loss, since the value function is also only updated if the reward is larger than the estimated value. They show that SIL improves other learning algorithms when used in conjunction. For example, they show that SIL improves PPO on MuJoCo tasks. Again, this idea is similar to the success learning variant of ISBR, except that it is done through loss function design, rather than through resimulation.

2.2 Simulation environments

Since simulation environments are used to test reinforcement learning algorithms, it is important to obtain an understanding of how others evaluate their work and what environments they use. Here, we review the simulation environments observed in the literature review to then be able to design our own experiments. The main take-away is that almost everyone uses a different task to benchmark their methods. For us this will mean we design our own robotics tasks as further discussed in chapter 3.

2.2.1 Visual

We start off with visual environments, i.e. environments where states contain images. Tasks in visual environments are often computationally expensive, since agents both have to learn how to observe the environment and how to take good actions.

Atari games are most frequently used. [Schaul et al., 2015b], [Oh et al., 2018], [Burda et al., 2018] and [Ren et al., 2018] test their results of an assortment of Atari games, for example [Schaul et al., 2015b] used 49 Atari games to test their method. Since many Atari games are now considered to be too easy, papers sometimes focus on solving the hardest Atari game, namely Montezuma’s Revenge. This is done in [Jiang et al., 2019], [Salimans and Chen, 2018] and [Burda et al., 2018]. Atari games are implemented in OpenAI Gym [Brockman et al., 2016].

Other papers focus on the first-person shooter game Doom instead, namely [Pathak et al., 2017], [Justesen and Risi, 2018] and [Jabri et al., 2019]. ViZDoom is implemented by [Wydmuch et al., 2018].

[Racaniere et al., 2019] makes their own environment altogether, in which an agent has to navigate a room with simulated 3D physics and has to perform tasks that involve moving specific objects based on their colour.

2.2.2 Robotics

Environment in which robotics task have to be performed are also common. [Asada et al., 1996], [Baranes and Oudeyer, 2013], [Andrychowicz et al., 2017], [Fang et al., 2019], [Sun et al., 2019], [Jabri et al., 2019] and [OpenAI et al., 2021] all perform tasks that are in some way related to robotics.

The most commonly solved problems are those from the four Fetch environments, implemented in OpenAI Gym and proposed in [Plappert et al., 2018]. In these environments, a robotic arm with a gripper at the end has to perform tasks.

- FetchReach: The gripper has to be moved to a given point in space.
- FetchPush: An object needs to be pushed to a location within reach.
- FetchSlide: An object needs to be pushed to a location out of reach.
- FetchPickAndPlace: An object needs to be picked up and moved to a target location.

Like all 3D environments in OpenAI Gym, they rely on MuJoCo [Todorov et al., 2012] for the physics simulations.

[OpenAI et al., 2021] increases the difficulty of these tasks by also including visual inputs, from which an agent also has to perform the correct action on the correct object.

The four Shadow Dexterous Hand OpenAI Gym environments are much harder robotics tasks. In the first three environments, either a block, a ball or a pen has to be manipulated to the correct target positions by a single humanoid hand. In the fourth environment, the hand is trained to reach a given target position itself. [Fang et al., 2019] use these environments.

The older papers [Asada et al., 1996] and [Baranes and Oudeyer, 2013] use different environments, but we won't go into them, as they appear to be outdated.

2.2.3 Navigation

Navigation tasks are tasks where an agent has to learn to navigate the environment, often with a goal. OpenAI Gym has an assortment of movement tasks in which an agent has to learn to navigate using a 3D physical body, e.g. a ball with 4 legs (again relying on MuJoCo). RLLab [Duan et al., 2016] has more complicated navigation tasks, such as mazes and worm-like creatures that have to gather food (SwimmerGather). These tasks in RLLab also rely on MuJoCo.

[Florensa et al., 2018], [Florensa et al., 2017], [Liu et al., 2019] and [Wöhlke et al., 2020] use the mazes and [Sukhbaatar et al., 2017] use SwimmerGather. [Levy et al., 2019] do similar goal-based MuJoCo navigation experiments with walls, and [Oh et al., 2018] test on an assortment of navigation tasks with different bodies using OpenAI Gym.

2.2.4 Multiplayer

Competitive and frequently zero-sum multiplayer games often induce the most complexity, as agents can adapt to one another and create vastly complicated strategies. Renown examples are Go [Silver et al., 2016] and StarCraft [Vinyals et al., 2019]. However, where Go is still easy to implement and run, StarCraft is the opposite. In [Baker et al., 2019] agents are tasked with playing hide and seek in a 3D environment with interactable objects. In [Bansal et al., 2017] multiple competitive MuJoCo environments are created.

In all the above papers it requires many iterations and many strategies of the competing agents to see the benefit of using a multiplayer environment as the behaviour of interest is inherently complicated and refined.

2.2.5 Toy

Toy problems are used to illustrate a concept and to quickly iterate when in development. Arguably the most iconic type of toy problem is a grid world in which agents occupy a square on a grid and can move to adjacent squares. Grid worlds are still used in [Schaul et al., 2015a], [Oh et al., 2018] and [Sun et al., 2019]. Another frequently studied toy problem is MountainCar and is implemented by both RLLab and OpenAI Gym. It is a 1-dimensional task in which an agent has to reach a flag on a hill. To reach the flag, the agent has to build up momentum by moving left and right, to swing out of the basin it started in. Its main use is studying exploration algorithms, since a sequence of coordinated movements is required before obtaining a reward. MountainCar is used in [Sukhbaatar et al., 2017].

2.2.6 Other

[Florensa et al., 2017] and [Wöhlke et al., 2020] use custom MuJoCo tasks, with the most prominent environment being a key insertion task, ideal for testing initial-state-based curricula. [Sukhbaatar et al., 2017] use many environments, but only one remains undiscussed, namely training marines in StarCraft: Brood War using TorchCraft [Synnaeve et al., 2016]. This is a task with sparse rewards that are only obtained after a rather specific action sequence, also great for testing exploration algorithms.

Chapter 3

Methods and implementations

3.1 Algorithms

3.1.1 GBR

As we have seen before, reinforcement learning is a notoriously time-consuming process that often fails to find satisfying solutions. There are many attributes that a problem can have that will make it more difficult. Two of those attributes are if the problem is goal-based, i.e. if the agent needs to achieve a given goal that changes every run, and if the problem has sparse rewards, e.g. if rewards are only obtained in a “success” state. Combining these two attributes in a single problem can lead to convergence issues when deploying a standard RL algorithm, such as PPO or Q-learning. This happens because rewards are hardly ever obtained due to their sparsity. Even if they are obtained, there is no simple action sequence that reliably increases the probability of obtaining a reward. This is due to the goal-based nature of the problem. After all, the required action sequence changes with the goal.

As an example, consider continuous control tasks where the reward also contains a component that promotes taking small or few actions. In such a case, the agents simply learn to take no actions at all, as this part of the reward is the only reliable signal. Even adding an exploration algorithm, such as ICM, does not change this. The method will only guide exploration in directions that are neutral with respect to the environment rewards. The reason for this is as follows:

We want to use an RL algorithm to find a global optimum for the environment rewards. This means that we want the behaviour of the agent to become optimal and thus predictable, which makes the ICM rewards directly oppose the environment rewards in its global optimum. Given that we are looking for optimal behaviour, we do not know what this behaviour is, and we do not know the associated environment rewards either. This means that if we want the ICM rewards to always be dominated by the environmental rewards in the optimal state, we must normalize the rewards dynamically before combining them. This yields the combined algorithm that is designed to still converge to optima of the environment rewards, and can therefore only actively explore in directions mostly neutral with respect to the environment rewards. Hence, it is unable to break out of a no-actions optimum because every means of exploration directly opposes the only present

reward signal, which is the no-actions objective. In short, exploration alone is not enough. The agent also needs to learn the underlying structure of the task in some other way such that when simulating, the agent makes genuine attempts at solving the entire objective.

It is important to emphasize that this is also how ICM is implemented in this thesis. We first normalize the environment and ICM rewards before merging them through a weighted average. This weighting is generally done using a 1:9 ratio for the ICM and environment rewards, respectively. In other words, this means multiplying the ICM and environment rewards by 0.1 and 0.9 before adding them. Likewise, a 2:8 ratio means multiplying the ICM and environment rewards by 0.2 and 0.8 before adding them. More details on this can be found in section 3.3.3.

Much research has been done on the topic of sparse goal-based learning, most notably HER and its many follow-up works, as discussed earlier in section 2.1.4. Taking a closer look at HER, we note that if one combines learning in hindsight with a standard policy gradient algorithm such as PPO, this comes with a fundamental drawback. Namely, one first needs to select the hindsight goal, and then hope that the initially obtained action sequence is not too dissimilar from an action sequence that could have been obtained if the agent had been presented with the hindsight goal in the first place. In other words, if some actions have probabilities that are too low in hindsight, which can easily happen; especially in problems where precise manoeuvring is required, gradients for the policy loss can get out of control and value estimations become inaccurate. After all, the action sequence that resulted in this final state does not even have to resemble the action sequence that would have resulted from tasking the agent to take actions towards this last state. Therefore, a more stable approach is desired. Not just because learning in hindsight can result in convoluted paths for the reached goals, but mostly because it is not easily combined with a well established exploitation algorithm. A more modular solution would make it much easier to create powerful RL algorithms.

Similarly, other papers also focus on using feasible goals to improve learning. In a scenario where learning no longer takes place in hindsight, but rather by running experiments with these goals, one can simply dedicate some computational power allocated to collecting data from environment interactions to collecting data with generated goals. We have seen multi-agent examples of this in section 2.1.3, as well as a more sophisticated method that explicitly tried to optimize multiple criteria to increase the quality of the goal in section 2.1.4.

Curiously, it appears no one has combined these two simple concepts before, where one simply chooses the goal induced by the last state of an episode and reruns the experiment with this goal that HER would have used to learn in hindsight. This approach should avoid the above-mentioned stability concerns of learning in hindsight, and should still teach the agent the underlying structure of the problem. As mentioned before, this approach where we resimulate using a reached goal will be called **goal-based resimulation (GBR)** and the algorithm where we specially select the goal induced by the last state as the goal to resimulate with will be called **G**.

Again, in contrast to many works that simulate using generated goal, what this method does not do is explore. This is worth noting because **G** learns by reaching states it reached before, which has the looming threat of a positive feedback loop. Therefore, it is important to prevent the agent from getting stuck in performing some arbitrary action sequence that is not constructive to solving the task. This can be done by combining the method with a sufficiently strong exploration algorithm. In other words, a modular approach is not only a desired, it can turn out to be a necessity.

Because of this setup, we will be decoupling teaching the agent the fundamental structure of the task from exploration. This gap in exploration potential will be filled with the exploration algorithm ICM, as discussed in section 1.1.2.

3.1.2 ISBR

We are now fully in the domain of curriculum learning as we are designing problems that aid learning. However, this need not be limited to generating goals. Some papers are dedicated to improving learning by changing initial conditions instead. However, these methods focus on starting the agent near the goal and then searching for nearby states that can function as new initial states. These initial-state-based methods, discussed in section 2.1.5, assume much more of the environment than the goal-based methods that only require a map from states to goals. This is because they are required to find new candidate initial states as well as the first initial-state near the goal.

Now note the similarity in these different lines of research. On the one hand, people try to simulate with instructive* goals and on the other hand, people try to simulate with instructive initial conditions. As seen earlier in section 2.1.4, [Racaniere et al., 2019] comprehensively defined instructive to be a combination of validity (the tasks are solvable), feasibility (the tasks are of a reasonable difficulty), coverage (the tasks are diverse) and desirability (the tasks are desired by the developer). However, most papers only implicitly consider instructive to be “on the edge of what the agent can achieve” which makes curricula naturally valid and feasible. If then some form of exploration is also present, these curricula automatically come with some level of coverage as well.

Hence, a natural and modular combination of these two approaches is an initial-state-based method that does not search backwards, but that does change the initial states such that the resulting runs end up “on the edge of what the agent can achieve”. This is the basis for our proposed approach that we call **Initial-State-Based Resimulation (ISBR)**.

Just like GBR, we are resimulating previous episodes. Instead of keeping everything constant except for the goal, now everything is kept constant except for the initial state. However, because the initial state was obtained from the episode we are resimulating, ISBR simply repeatedly resets the environment to some point during that particular episode.

Aside from resimulating with goals versus resimulating with initial states, there is another major difference between the methods. GBR can resimulate with any goal induced by the last state of an episode because this automatically means that the goal is “on the edge of what the agent can achieve”. After all, we are proposing a task that is both sufficiently achievable, because it has been achieved from that particular initial state before, and sufficiently difficult, because it took the agent an entire episode to reach that state. In contrast, what makes an initial state result in a task that is “on the edge of what the agent can achieve”, is entirely depending on how that particular episode played out. In other words, to decide if an initial state is a good candidate for resimulation we must first decide if the episode is a good candidate for resimulation.

To make this more concrete, imagine a game where an agent has to navigate to a goal state from a starting state, in which the state happens to be equivalent to the position of the agent. The agent is rewarded for being at the target location after a given number of actions. If the given goal is to

*Something being instructive refers to it containing or resulting in a clear signal on how to improve, and therefore being beneficial to learning.

reach the initial state and the agent does not move at all, this results in a perfect score. However, this can hardly be considered instructive. After all, not moving does not lead to interaction with the environment, which in turn leads to the agent not learning the consequences of its actions. Similarly, if the goal is unattainable given the agent’s skill, the episode cannot be considered instructive either. The target location is simply out of reach and will never be achieved, regardless of the number of attempts. Because of this, the agent will not be rewarded differently for different actions, meaning it will not learn from the experiences. Naturally, in such uninformative episodes, no state can be considered an informative initial state for ISBR.

If we consider less extreme cases, the agent might still learn from the resimulated episodes. However, this would not be better than sampling episode configurations randomly. After all, there is no benefit to resimulating uninformative data. In fact, it would probably be worse, because resimulation also reduces the sample diversity. All in all, it is clear that to find informative initial states, we will have to put in some work.

Swing events

To find informative initial states, we now take a step back and revisit policy gradient methods, such as A2C and PPO, as discussed in section 1.1.2. These methods learn by increasing or decreasing the probability of actions based on whether the measure of the quality of that action (i.e. the advantage estimates for the purpose of this thesis) was positive or negative. It also changes the probability of the action more drastically, if the amplitude of the quality measure is greater. One could say a great amplitude reflects an unexpected event, since it means there was a great difference between the accumulated reward that the agent expected, and the one that was actually obtained. In the case of a goal-based game with sparse rewards, this reflects unexpectedly reaching or not reaching the goal.

Since the obtained rewards are discounted to also reward past actions for future payoffs, one can imagine there sometimes being a state before which the agent predicts the rewards to go one way, but then after that point it predicted it to go another way. Such states we call **swing states**. It is important to note that the changing predictions of future payoffs is directly reflected in the advantage estimates. After all, discounting rewards with a discount factor means that nearby actions in time are rewarded similarly. If there is a sudden change in the expected returns, this causes a difference in advantage estimates between nearby actions.

Swing states are actually quite common. Aside from the fact that this can be observed by looking at the trajectories of the expected payoffs compared to the realized payoffs of many episodes, see figure 3.1 for an example, we can reason on it as well. Notice that as the agent gets closer to the final state, the variance on what the future trajectory of the agent will be, decreases. This means that the expected accumulated rewards get more accurate. If swing states were not common, it would imply the variance remains constant throughout the episode, as the agent never clearly deviates from its original expected payoffs. This cannot be, as it implies that it is always the last action of an episode that determines the rewards.

Note that a swing state occurs when it becomes apparent to the agent that the past value estimations were inaccurate[†]. This means that the critical actions that led to the unexpected outcome happened

[†]Or at least we can say that. The agent is still just a model, potentially without memory.

at or before the swing state. In other words, swing states indicate that the (recent) past was instructive.

All in all, because swing states have instructive pasts, are common, and are directly observable in the advantage estimates, we can use them to find instructive initial-states for ISBR, extracting more utility from them. More formally, if we define **swing states** to be the states where the value estimation matches the true discounted reward after having differed greatly for many time steps, we can define the state with the most **swing** to be the state for which the area between the value estimation curve and the discounted reward curve is greatest before this state. Since we are now measuring over time as well, we will call the event running up to a swing state a **swing event**. In other words, the duration of this event is the number of steps for which the difference between the expected and realized payoffs does not change sign, running up to the swing state. The swing of a swing state (or swing event) measures the absolute area between the expected and realized payoffs during the swing event.

The first states of swing events with lots of swing are instructive initial states. This is because in a previous episode, they lead to a sequence of states with large advantage estimates, all with the same sign. Since the advantage estimates are the signal that is directly used to adjust the action probabilities, large advantage estimates directly translate to large changes of the action probabilities. Because the advantage estimates also have the same sign, the quality of the entire action sequence is clear, which in turn means there is a clear and substantial signal indicating how to improve. This makes the first states of swing events instructive initial states for ISBR.

We can now formulate a rudimentary algorithm that identifies the swing events with the most swing, and then resimulates those. However, this completely ignores the different nature of swing events if the area between the curves have different signs. On the one hand, if the expected payoffs are greater than the realized payoffs, the event marks an unexpected failure. For example, this could imply diminished performance compared to previous runs, possibly caused by other updates to the model or some source of noise. On the other hand, if the expected payoffs are lower than the realized payoffs, the event marks an unexpected success. For example, this could imply increased performance compared to previous runs, possibly caused by luck and some source of noise.

In this light, we define the mistake learning algorithm **S**− that only resimulates swing event where the payoffs are worse than expected, and we defined the success learning algorithm **S**+ that only resimulates swing events where the payoffs are better than expected. We define **S** to be the algorithm that does both in a 1:1 ratio, hence it explicitly does not only look at which swing events have the most swing.

S− is expected to increase the stability of the algorithm, as it puts emphasis on maintaining the expected (presumably previously observed) performance. **S**+ is expected to increase the exploration capacity of the algorithm, as it should decrease the sparsity of the problem by making rare rewards more accessible.

3.1.3 Resimulation implementation

When running RL experiments, tasks are instantiated as environments. An agent can repeatedly take actions in such an environment. After every action, the environment updates, after which the agent can take another action. At some point, the environment terminates, after which it resets to a state sampled from the initial state distribution.

In this thesis, we are choosing for the classical phased approach of deep reinforcement learning; data is first collected from the environments, before using this data to update the model. After each iteration, this process repeats, again gathering data from the environments with the updated model to update it again afterwards. Such an iteration will be called an **epoch**, not to be confused with an **optimization epoch**, which refers to looping over the data once when training the agent.

To reduce computational overhead, the data gathering happens in parallel, with at most 18[‡] parallel environments gathering data at the same time. If one terminates, it resets on its own and begins its own new episode. After time runs out, all parallel environments terminate and only completed episodes are collected to form the dataset with which the agent is again updated.

Resimulation is implemented by dedicating some portion of the available parallel environments to collecting resimulation data. More explicitly, when using resimulation, we always dedicate 2/3 of the available resources to gathering resimulation data. This means that of the 18 parallel environments, 12 are dedicated to resimulation and the remaining 6 are running the primary unaltered task. The 12 resimulation environments will only have slightly different termination and initialization rules from the 6 primary environments. This depends on the specific resimulation methods for which these 12 environments are being used. Often, these rules also differ within the 12 resimulation environments. Namely, this happens when multiple resimulation algorithms are being used.

To make this more concrete, we will list the environment splits for the algorithms we will be comparing: G, SG+, SG− and SG. If a particular experiment tests the performance of G, then 12 environments are performing G and 6 are performing the primary task. SG+ and SG− test the performance of G combined with S+ and S− respectively. This is done by having half of the resimulation environments run GBR (namely, G), and half run ISBR, with the ISBR algorithm being S+ or S−. This means that 6 environments are running the primary task, 6 that are running G and 6 are running S+ or S− respectively. SG combines G, S+ and S− by having half of the resimulation environments run GBR (namely, G) and half run ISBR, but now half of the ISBR are running S+ and half are running S−. This means that 6 environments are running the primary task, 6 are running G, 3 are running S+ and 3 are running S−.

In practice, episodes are 50 steps long and the total step limit for gathering data is 250, every parallel environment performs 5 episodes. Because the resimulation environments behave differently from the primary environments, this results in the resimulation environments repeating the same experiment 5 times, starting at the same initial state, aiming to reach the same goal. In contrast, the primary environments reset randomly every episode. For ISBR, this has a clear purpose. It enables the agent to collect more data on a sparse signal, creating a more instructive signal. For GBR, this serves less of a purpose. After all, any reached goal should suffice. However, to keep the implementations consistent, and to better compare between GBR and ISBR without having to worry about sample diversity concerns, this is implemented in the same way. For S, this means that only a limited number of swing events can be resimulated. Therefore, we always choose the swing events with the most swing.

Knowing this we can formulate pseudocode for algorithms S, S−, S+ and G. However, since algorithms S, S− and S+ are very similar, we will only spell out pseudocode for S and G in algorithm 2 and 1 respectively. An importance difference between the two methods is that G only draws goals only from the primary task, whereas all variants of S draw initial states from all other tasks.

[‡]Preferably chosen as high as possible, but here restricted to 18, the maximum number supported by my laptop.

This means that when running SG (S and G combined) G is drawing goals from the 6 primary environments, whereas S is drawing initial states from both the 6 G environments and the 6 primary environments. This is to prevent a feedback loop where the same task, originally generated by the primary task, keeps cycling in the resimulation environments. ISBR is chosen to sample from all environments because its main function is supporting other methods by desparsifying and stabilizing learning. For this, it needs access to all other environments. There is no trade off here either, because for GBR it does not matter where the reached goals came from. Any reached goal will suffice.

Algorithm 1: G

- Data:** The simulation states of the first and last states of episodes obtained from the primary task.
- Result:** Setting the GBR environments with start states that led to reaching the given goals in the past.
- 1 Assign a primary environments to every GBR environments, s.t. the number of times a single primary environment is assigned differs by at most 1;
 - 2 **for** *Every GBR environment* **do**
 - 3 Obtain a (first state, last state) pair from the assigned primary environment and set the initial state of the GBR environment to be the first state, and the goal to be the goal induced by the last state.
-

Algorithm 2: S

- Data:** Simulation states obtained from all other tasks and all accompanying advantage estimates
- Result:** Setting the ISBR environments to resimulate swing events.
- 1 **for** *every episode* **do**
 - 2 Compute the signed area under the advantage estimate curve for all segments where the advantage estimate does not change its sign.
 - 3 Save the first state of the segment with the greatest positive area.
 - 4 Save the first state of the segment with the greatest negative area.
 - 5 Select the first states of the saved first states with the greatest positive area for resimulation of successes on one half of the ISBR environments[§];
 - 6 Select the first states of the saved first states with the greatest negative area for resimulation of failures on the other half of the ISBR environments;
-

In figure 3.1 we present an example of how S processes the advantage estimates to identify the beginning of swing events. This figure contains data obtained from 6 G environments and 6 primary environments, since S draws its initial states from both kinds of environments.

[§]In our case, the 3 or 6 states with the most positive swing, for S and S+ respectively.



Figure 3.1: This figure summarizes how S selects its next starting states, illustrated using the algorithm SG. The red line plots the advantage estimates (y-axis) as a function of their serialized time stamp (x-axis) of a single epoch by serializing all episodes in the epoch that are not obtained through ISBR. The episodes are separated by the dotted lines at the bottom. The advantage estimate trajectories of adjacent episodes (groups of 5) in the first half (x-axis, until 1500) look similar since they are obtained from the same GBR processor, hence they share their starting state and their goal. The dotted lines at the top indicate the start states for ISBR in the next epoch. Notice that these states are the start of swing events, as the signed area under the curve from these states onward is either very positive or very negative. Notice that there are exactly 3 starting positions representing positive advantage (S+) and exactly 3 starting positions representing negative advantage (S-). Notice that the swing states are also visible, as those are the points that the advantage estimate tends back to zero. Advantage estimates are clipped to $[-3, 3]$ to avoid outliers.

3.1.4 Summary and hypotheses

In summary, we propose 3 distinct new algorithms, namely G, S+, S-. They are implemented by allocating most computational resources to resimulation. This means that per epoch, almost the same number of transitions are observed in all methods. The algorithms can also be combined to form more complicated algorithms, namely SG+, SG-, S and SG, and should be combinable with an exploration algorithm such as ICM, in favour of a modular approach to RL. The hope is that the methods interact in a complementary way, adding their strengths to form better combined algorithms.

G

We expect that G will enable the agent to learn the structure of the data stably, by always providing the agent with some feasible goals. The expected drawback of this method is that it can create a positive feedback loop, which causes the agent to learn a fixed action pattern[¶]. In other words, if G fails to find solutions in the primary environments due to it learning a fixed action pattern (on a sparse problem!), it is expected that there is room for improvement by somehow breaking this positive feedback loop.

An exploration algorithm such as ICM is expected to be able to break and prevent these positive feedback loops by stimulating diversifying actions in the primary environments. This should be

[¶]E.g., the agent learns to reach the same state over and over again, regardless of the goals.

beneficial because, by diversifying the action pattern in the primary environments, the goals in the GBR environments also diversify. Using ICM for this purpose should be possible because, if G fails to find solutions in the primary environments due to the agent performing a fixed action pattern, no other clear reward signal should be present. S+ is also expected to prevent such positive feedback loops since S+ would resimulate unexpected successes, i.e. successes not due to the converging fixed action pattern.

S+

We expect that S+ will enable the agent to learn from a very sparse signal, through resimulating unexpected successes. The expected drawback of this is that when barely reachable goals become available, all S+ environments will be resimulating very similar goals, which can have a destabilizing effect due to a reduced sample diversity. This seems a particularly likely scenario when S+ obtains the initial states from G, because then multiple episodes are run with the same goals and initial states. S- is expected to be able to inhibit this destabilizing effect by preventing catastrophic forgetting.

S+ is also expected to have an amplifying interaction with ICM. This is because S+ effectively resimulates unseen successes, which will cause ICM to produce high rewards if the associates states are also unseen. A priori, the result of such an interaction is unclear.

S-

We expect that S- will prevent the agent from forgetting learned skills by resimulating unexpected failures, which implies that previously, the agent performed better. Again, this method is expected to supplement S+ very well, since it should prevent S+ from causing catastrophic forgetting. It is also expected to perform well if the task requires precise manoeuvring (e.g. a task with a continuous action space with tight tolerances), since it maintains the easily lost skills of the agent.

The expected downside of S- is that it inhibits exploration by not allowing suboptimal actions to be learned. In other words, if S- fails to find a solution, it is expected that there is room for improvement by adding S+ or ICM for their expected explorative properties.

More concretely, in light of these expectations, we will be designing tasks that are sparse as well as precise and goal-based. For these tasks, we hypothesize that:

1. G must be used to stably solve these tasks, since they are otherwise unsolvable by naively applying PPO and ICM, with or without HER.

Here, we define **solving** a task as stably reaching at least some goals and maintaining this performance for a reasonable number of iterations. In other words, in terms of reaching environment goals, we hypothesize that on the to be proposed tasks, PPO and ICM with or without HER will effectively perform no better than random, whereas G will perform better. This hypothesis will turn out to be true. The reason for this was already discussed in section 3.1.1.

For the purpose of studying the performance of ISBR algorithms, we revise this definition of solving a task. Essentially, only requiring the agents to do better than random now sets the bar too low. As such, in this context, **solving** a task (in a **solving** run) will be defined as reaching a constant sufficiently high reward threshold at least once, while also maintaining this level of performance.

Given that G is required to solve the proposed tasks introduced in section 3.2, we formulate the following hypotheses on the performances of S+ and S-. These are the main subject of study for this thesis.

2. S+ improves the solve rate.
3. S+ improves the learning speed.
4. S+ has some amplifying interaction with ICM.
5. S- improves the learning speed.
6. S outperforms both S- and S+.
7. Fixed action patterns are more likely to be observed for SG- with a low ICM:environment reward ratio, and less likely to be observed for SG+ with a high reward ratio.

Here, the solve rate refers to the probability of solving a task. We also hypothesize that ICM improves the solve rate and that harder tasks will cause a lower performance. However, these results are of no interest to us here. They will only be studied to correct for their effects in the analysis of ISBR methods, and for their possible interaction effects with ISBR.

3.2 Experimental setup

3.2.1 Task design

To test the hypotheses formulated in section 3.1.4, we design our own robotics task with the following properties:

1. It must be a goal-based continuous control task with sparse rewards.
2. It must contain a globally suboptimal local optimum, and it must be sufficiently difficult that the methods can demonstrate a performance increase.
3. It must be sufficiently simple such that episodes can be short, since we only have very limited computational resources available.

Since “reacher-v2”, a robotics task implemented in OpenAI Gym, is already an episodic goal-based continuous-control task with an episode length of only 50, it already satisfies requirement 3, and we can augment it further to satisfy all requirements. An illustration of the task can be seen in figure 3.2.

To satisfy requirement 1, the reward function is changed to make it sparse. More concretely, given some distance δ that defines the acceptable distance from the goal to the tip of the reacher, the obtained reward function for reaching goals becomes

$$r(x_{\text{goal}}, x_{\text{reacher}}) = \begin{cases} 1, & \text{if } \|x_{\text{goal}} - x_{\text{reacher}}\| < \delta \\ 0, & \text{otherwise,} \end{cases}$$

where x_{goal} and x_{reacher} are the coordinates of the goal and the tip of the reacher respectively. Note, this is only the reward function for reaching the goal. The additional reward function that

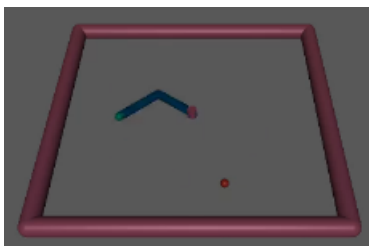


Figure 3.2: An example state of our task, a modification of “reacher-v2” implemented by OpenAI Gym, visually identical to the original task. The agent controls the force exerted on the two joints of the arm. The modified game is sparse, as the agent obtains a reward of one if the tip of the arm is sufficiently close to the goal, displayed as a red dot, and zero otherwise.

promotes taking small actions is still left unchanged and is given by $r'(a) = -\sum_i a_i^2$, where a is a vector containing the continuous actions taken by the agent.

Lastly, we put friction on the second joint (the one not in the center) as well as a small force that pushes the arm into a straight position that increases linearly with how far the tip of the arm is from the edge. We also ensure that all goals are in the interior of the circle that the arm can reach, but without extending all the way to the edge. In other words, there is a force on the arm that pushes it away from all possible goals. Complementing this is that the initial state is a still, stretched out position with only minimal fluctuations in the initial position and velocity. With the already present reward that promotes small actions in this continuous control task, the agent now has the globally suboptimal local optimum of not moving at all, never finding any goals, but also not being penalized for taking actions. Note that we can make the game arbitrarily difficult by first tuning δ , the size of the area around that goal that is considered a success state, and second by tuning how tight the circle with goals is around the center, satisfying requirement 2.

After these changes, figure 3.2 is deceiving. A better representation would be replacing the flat surface with a cone-like shape that has a continuously increasing slope towards a peak in the center. All goals are then situated somewhere on the steep slope of the concave cone. The higher the goal, the steeper the slope.

The design of this task was inspired by the toy problem MountainCar, a problem we discussed earlier in section 2.2.5. This was done because for MountainCar, we know the optimal parametrization for combining the ICM rewards with the environment rewards, namely a weighted average with a roughly 1:9 ratio, as discussed earlier in section 3.1.1. However, although we know the ratio for MountainCar, the optimal ratio for this task is very much unclear, which is problematic for designing experiments.

It is unclear for several reasons. In MountainCar, the agent has to manoeuvre up a 1-dimensional slope by swinging to build up momentum. Here, the agent has to manoeuvre to a very precise point on a 2-dimensional slope, and then has to fight gravity in different amounts, depending on how high it is on the slope. Because there is friction on the second joint, momentum cannot be used to climb the slope and the agent has to pay a price to climb the cone in terms of a reward penalty for moving. To find compensation for climbing the cone, it must be able to precisely manoeuvre to the goal and stay there for long enough. In contrast, in MountainCar, the agent can find the top of the

hill almost through luck. There is a chance that it will find the solution by just swinging back and forth for some time.

In other words, the problem is sparser than MountainCar, indicating a greater need for exploration. However, it also requires much more precise manoeuvring, indicating a greater need for exploitation. This makes it the role of ICM in solving this task unclear and something to be investigated.

3.2.2 Experiment design

In all experiments that are covered in this section, we use the designed task of section 3.2.1 to confirm or reject the hypotheses formulated in 3.1.4. In total, we perform three experiments, numbered 0, 1 and 2, that become progressively more difficult.

The easiest experiment, experiment 0, has the sole purpose of confirming hypothesis 1. There we show that PPO and ICM, with or without HER, cannot solve the easiest task. Because G can solve the harder tasks in experiment 1 and 2, hypothesis 1 turns out to be true. In experiment 1 and 2 we investigate the performance of S+ and S-. In experiment 1 we make the problem require more precision and in experiment 2 we make the problem sparser. In short, here we show that S- improves performance across the board, in every metric. S+ is shown to interact both with ICM and the sparsity of the task. As a consequence, the use of S+ is limited to sparse problems and must be accompanied by an exploration algorithm such as ICM to cause improved performance.

Before we can go into detail on the specifics of the experiments, we must cover some hyperparameters to understand how the difficulty of the game is scaled. The parameters of experiment 1 will be referred to as the standard parameters. We will go over these now, and refer to them later to more clearly show the difficulty relations between the tasks.

Using the standard parameters, the two arms of the reacher both have a length of 0.1, making it possible to reach all goals within a disk of radius 0.2 from the origin. We set the disk in which the goals appear to have a radius of 0.15. This means only the inner 56% of the space contains goals generated by the environment. These goals are said to be reached if the tip of the reacher’s arm reaches the disk round the goal with radius of 0.005, meaning that only 0.06% of the total space is considered a success state.

Experiment 0

In experiment 0 we demonstrate the performance of PPO combined with ICM and PPO combined with ICM and HER, both with a reward ratio of 1:9. We show that on the both algorithms do not find any solutions on the given tasks.

To show this, the task was made easier by increasing the radius of the disk around goals from 0.005 to 0.01, making the task require less precision. This doubles the area that is considered a success state from 0.06% to 0.25% of the total space. Because these results are unsurprising, they are shown in the discussion in figure 5.1b as the lines indicated by “OG” and “OG HER”.

In both cases, the algorithms systematically collapse, indicated by the lines dropping below the x-axis in the figure. In section 5.2.1 more experiments on the same easier task are discussed. Sadly, the rest of these experiments cannot be used, even though the results obtained there are similar to the results of experiment 1. The reason for this comes down to the peculiarities of the experiment setup used there.

Experiment 1

In experiment 1 we compare G, SG+, SG− and SG, all for the ICM to environment reward ratios 1:9 and 2:8. This yields the following experimental setup where we compare 8 algorithms, all fundamentally relying on PPO as an exploitation algorithm:

Algorithm acronym	reward ratio (ICM : environment)
G	1:9
SG+	1:9
SG−	1:9
SG	1:9
G	2:8
SG+	2:8
SG−	2:8
SG	2:8

In isolation, the results from this experiment are deceiving. However, there were two things that did become clear. The first was that the solve rate was very high. If we wanted to push the algorithms to their limits, the sparsity would have to be increased. The second was that increasing the reward ratio from 1:9 to 2:8 brought no benefits, and an interaction between S+ and ICM was observed. These two observations combined led to the design of experiment 2.

Experiment 2

In experiment 2 we again compare G, SG+, SG− and SG, and we make the task sparser. The difficulty of the task is increased by decreasing the radius of the disk in which the environment generates goals. However, it is important to stress that the task designed in section 3.2.1 was constructed to easily scale up in terms of difficulty. At first glance, it might not seem too impactful to reduce the radius of this disk. However, because the force on the second joint grows linearly with the distance to the edge of the accessible space, and because the tip of the reacher moves in a circle around the second joint, a small change in this angle significantly changes the minimum effort required to reach environment goals. In fact, we can compute this exactly through a simple line integral. After all, the parametrization of the curve is simply $r(\theta) = [\cos(\theta) \quad \sin(\theta)]$ for $0 < \theta < \pi$ and the scalar field of interest is $f(x, y) = 1 - x$. If we then want to know the effort required to reach a certain angle ϕ from a stretched out position, this gives the line integral,

$$\int_0^\phi f(r(t)) \left\| \frac{\partial r}{\partial t} \right\| dt = \int_0^\phi 1 - \cos(t) dt = \phi - \sin(\phi).$$

In other words, the minimum effort required for reaching environment goals using the standard parameterization is $\cos^{-1}(0.5) - \sin(\cos^{-1}(0.5)) = 0.181172$, where this number does not mean very much in an absolute sense. However, by solving for ϕ in $\phi - \sin(\phi) = 0.181172 \cdot 1.5$, we find the angle for which the problem becomes 50% more difficult to be, $\phi = 1.20593$. This translates to a radius of 0.135682. Therefore, we will be using this radius for experiment 2. For context, by decreasing the radius to be near 0, the effort required to reach environment rewards can increase to over a factor of 17 compared to the standard task.

Because the radius is reduced from 0.15 to 0.135682, the percentage of the total area in which goals can appear goes down from 56% to 46%. and the angle required to reach the environment goals goes up from 60° to 69°.

In experiment 1 we learned that increasing the ICM : environment reward ratio leads to a decreased learning speed, and that this is particularly devastating for S+. Because of this, we now compare not using ICM (i.e., a 0:1 ratio) to the default ratio (1:9) This yields the following experimental setup, where we compare 8 algorithms, all fundamentally relying on PPO as an exploitation algorithm:

Algorithm acronym	reward ratio (ICM : environment)
G	0:1 (no ICM)
SG+	0:1 (no ICM)
SG-	0:1 (no ICM)
SG	0:1 (no ICM)
G	1:9
SG+	1:9
SG-	1:9
SG	1:9

3.3 Implementation details

Before moving on to the results, we go over the most important hyperparameters, what their values are and how they were chosen. This section is aimed at people looking to reproduce the experiments of this thesis, and those that are just generally interested in hyperparameters.

3.3.1 Model architecture

The used models are all multi-layer perceptrons (MLPs) of size 32 with ELU activation functions, implemented using PyTorch [Paszke et al., 2019].

The ICM model has the expected structure, consisting of a forward model, an inverse model, a model that processes the taken action and a model that processes the given states. All components are chosen to be the same size for simplicity, namely consisting of 3 layers. The model from the original paper is not copied since, as discussed in section 2.2, that model processes visual inputs, which is not relevant here.

The agent’s model (the model that produces the policy output and the value function estimates) can be summarized as just a large stack of fully connected layers, where the exact hyperparameters are irrelevant. For the sake of completeness and for those interested in reproducing or extending this work, we will now give a more detailed description. However, the exact model architecture is expected to have no influence on the results.

After being presented with a state, the agent’s model processes the given state for some number of layers before splitting into two ways, one for the value function and one for the policy output. From the fork, the value function is processed with a few more layers before being mapped to the value function output. In contrast, the policy output does not receive any additional processing and is mapped to the output directly.

In an early model exploration stage, it turned out that providing the value function with a bit more processing sometimes increases performance. The amount of layers was chosen to again be 3 layers per segment. However, since the implementation supports an LSTM (and comparisons with an LSTM model, see section 5.2.2 for more details on this support), the processing from state

to action is considered to be 2 segments. Namely, one before and one after a hypothetical LSTM insertion. This leads to a total of 6 layers from state to action output. For the sake of code maintainability, this then became the structure of 5 layers before the fork, followed by 1 layer for the action output and 3 layers for the value function output.

3.3.2 Optimizer

The models are optimized using the Adam optimizer [Kingma and Ba, 2014]. The advised parameter 0.9 is used for β_1 , but β_2 is changed from 0.999 to, 0.99999 ($\approx 0.999^{\frac{1}{54}}$) since we are doing 54 ($= 3 \cdot 18$) optimization steps every epoch on roughly the entire dataset. Here, it is important to note that the number of times a particular transition occurs in the loss of one optimization loop is 20 (> 18) times, as can be seen in table 3.1.

3.3.3 Normalization

The rewards before discounting are normalized over time to ensure the ICM rewards and the environment rewards can be easily merged using a 1:9 weighting ratio (see section 3.1.1 for a more detailed explanation). The rewards are normalized using a cumulatively updating mean and variance using the StandardScaler from the sklearn.preprocessing python package [Pedregosa et al., 2011], of which we decay the number of samples seen by a factor of 0.9 every epoch to create exponential smoothing. Similarly, we normalize the states with a 0.999 decay factor.

The exponential smoothing is necessary to obtain proper smoothing of the obtained data given the policy. Since the policy is constantly changing during the optimization procedure, the statistics of the data are also constantly changing and cumulative statistics are therefore bad measurements. The decay factors were tuned to work well for PPO with ICM on the MountainCar task of OpenAI Gym.

3.3.4 Generic RL parameters

The discount factor was chosen to be 0.99 to allow long-term dependencies. The policy loss and the value loss were not scaled and simply added to produce the final loss. The clipping parameter of PPO was chosen to be 0.1, lower than the generally used [0.2, 0.3] range, since we are effectively doing 36 optimization steps using the entire dataset, meaning that stability becomes a concern.

3.3.5 State inflation

Because of the chosen discount factor, that ensures long-term dependencies can be found, in combination with the short episode length, all accumulated rewards are highly time dependent. For the used tasks, a proper estimate of the value function is hardly possible without information on how many steps the agent has left, because this greatly affects how many steps the agent will eventually spend in the region where the rewards are obtained. Therefore, the number of steps left until termination was included in the state of the agent.

For example, because the episodes are of length 50, after 5 steps on a primary environment 45 steps are left. Therefore, the number 45 is concatenated to the state. On ISBR environments, the number of steps required to reach the initial state are saved. For example, if 7 steps were already

taken to reach the initial state, after 3 steps, only 40 steps are left until termination. Therefore, the number 40 is concatenated to the state.

Table 3.1: Hyperparameters.

Description	Hyperparameter
# epochs	5000
# parallel environments	18
Episode length	50
# transitions per parallel env. per epoch	250
# episodes per epoch	$90(= \frac{250}{50} \cdot 18)$
# optimization loops per epoch	3
# batches per optimization loop	18
Data duplication factor for LSTM compatibility	20
Effective # times any transition occurs in the loss	$60(= 3 \cdot 20)$
Clipping parameters ϵ of PPO	0.1
Policy loss : value loss ratio in the final loss	1:1
Discount factor	0.99
Exp. decay of moving avg. and var. for reward standardization	0.9
Exp. decay of moving avg. and var. for state standardization	0.999
Optimizer	Adam
Adam parameter β_1	0.9,
Adam parameter β_2	0.99999
Learning rate	0.0002
# nodes per layer	32
Non-linearity of layers	ELU
# layers from state to action output	6
# layers from state to value function	8
# shared layers between policy and value function	5
# layers in ICM inverse model	3
# layers in ICM forward model	3
# layers in ICM state encoder model	3
# layers in ICM action encoder model	3
Total # layers in ICM	12

Chapter 4

Results

Here we discuss the results of the experiments introduced in section 3.2.2. The data obtained from these experiments is complicated. It consists of 4 changing variables, namely the task, the reward ratio, the use of S+ and the use of S-. This gives a total of $2 \cdot 3 \cdot 2 \cdot 2 = 24$ different parameter settings, of which we test 16, 8 per experiment. For each parameter setting, 10 runs are performed. This yields a dataset of 160 learning trajectories resulting from the 160 runs. At 7 hours per run, it took 2 months to collect all data for experiments 1 and 2, and at 14 hours per run, it took another 2 months to collect all data discussed in section 5.2.1. For each run, we save the sum of the obtained rewards per episode in the primary environments, together with the number of transitions that were learned from at that point in the run. After all, we are ultimately interested in the performance on the original task, not in the performance on the environments of which we alter the goal or the initial state.

Because the data is rather involved, we will slowly go through it in section 4.2, explaining the different statistical methods and figures we use along the way. We want to emphasize that because we are going to analyse subsets of the data, these results are not definitive. It is mostly meant as a setup for a complete analysis where all data is processed at once, but also as an attempt to find patterns that can be used to further improve the ISBR methods. Because of this, we will be using an α of 0.1 when testing for statistic significance in this section. After all, the sole purpose of this section is exploring and reasoning on the data, not making definitive claims.

Definitive results will be presented in section 4.3 using an α of 0.05. There we analyse the data as a whole, fully reducing the task to an explanatory variable. At that point, these results can be readily understood in the light of the data exploration that is done in section 4.2. However, before we can get to any of that, we first discuss the quality measures we use, in section 4.1.

4.1 Quality measures

As mentioned before, in this thesis, we split the overall performance into two components, learning speed and solve rate. For the learning speed, we only consider solving runs. As such, the two required quality measures measure distinctly different properties.

We are not interested in asymptotic performance, the performance of the agent after running for a very long time. This has two reasons. First, it appears to be the same for all algorithms. Second, runs take much longer to complete if we look at asymptotic behaviour, preventing us from gathering enough data for a statistical approach to investigating the other properties of the algorithms.

4.1.1 Solve rate

For this approach, it is also important that the used measures satisfy the assumptions of the used models and tests. For the solve rate, this is simple to achieve. Because of the binomial nature of the solve rate, we use a logistic regression with likelihood ratio tests. These do not make any substantial assumptions that are not automatically satisfied through our experimental setup. As such, when redefining the solve rate, we only have to make sure that it properly distinguishes runs that do not reliably obtain sufficiently high rewards from runs that do. However, what exactly is sufficiently high appears to be subjective. To find natural candidates for making this distinction, we consider the average rewards per episodes of a run. In other words, we look at the sum of rewards individual episodes achieve on average, in a particular run.

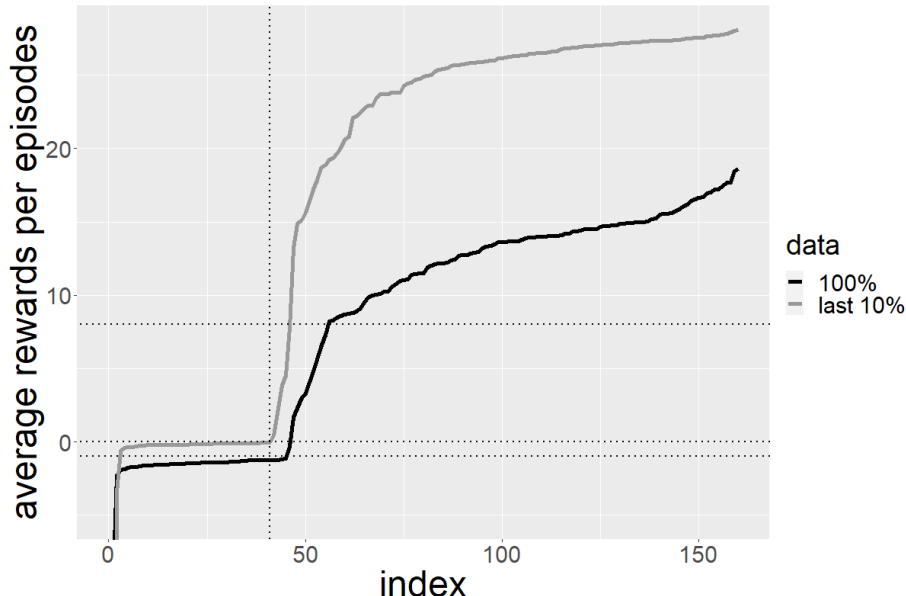


Figure 4.1: The sorted average rewards per episode in experiments 1 and 2 for both all the data and the last 10% (measured in observed transitions).

In figure 4.1 we plot this for all runs, sorted to be in ascending order. We first consider the line corresponding to using 100% of the data. It is clear that there are two main modes. The first is where the run never find any rewards, indicated by the runs with an average reward below -1. The second is where the runs properly converge, indicated by runs with a reward above 8. In-between these two modes, there are many runs that presumably either started learning late, learned slowly, or experienced some other anomaly. These runs could contain valuable information for the analysis

of the learning speed and, at least for now, should not be lumped with the bottom mode. Still, we have to ensure that the runs reaching a cut-off point of -1 actually managed to obtain environment rewards, and did not simply learn to stay still extraordinarily quickly. We also have to ensure that the negative rewards obtained while exploring, do not hide the fact that the agent did indeed reach environment goals, even if it took very long to reach them. We must also note that preferably, we want to use a cut-off point at 0, which has the nice interpretation of being the point where the obtained rewards from reaching goals outweighs the cost of moving.

For all these reasons, we compute the same metric, but now only for the rewards obtained in the last 10% of transitions in every run, as also displayed in figure 4.1. We do indeed see that the bottom node now stops at 0. Because we are only looking at a very late stage in learning, we may assume that if the agent does not reach an average reward greater than 0, then the agent is simply not reaching any goals. Because of this, this metric satisfies all our requirements and, for now, we redefine a **solving** run as a run that reached an average reward greater than 0 on the primary environments, in the last 10% of the observed transitions.

4.1.2 Learning speed

For learning speed, things are more complicated. Still, because of our experimental setup, when using a linear model we only have to worry about the residuals following a normal distribution and the variance of the residuals not being dependent on the explanatory variables (homoscedasticity). The most obvious candidate statistic is again taking the sum of all obtained rewards. If it is high, the agent performed well, if it is low, the agent performed poorly. However, even when only considering solving runs, this immediately runs into issues with both assumptions. Sticking with the statistics assigned to each run as displayed in figure 4.1, we obtain p-values ranging from 10^{-6} to 10^{-16} for the Shapiro–Wilk test of normality and the Breusch–Pagan test of homoscedasticity*. In other words, we have to look for inspiration elsewhere to manually design a statistic that reduces learning curves to a single real number.

Time-to-threshold

A commonly used statistic for speed is time-to-threshold [Narvekar et al., 2020]. This measure does exactly what you would expect, it measures how long it took the agent to reach a reward threshold. However, this is not a suitable measure for our purpose for a few reasons. First, if the agent on a particular run never reaches the threshold, we do not obtain a value. This limits the threshold to relatively low values. Second, our data consists of rewards obtained from episodes, accompanied by the number of processed transitions. Because of this, a single easy episode could entirely devalue this metric. As such, some form of smoothing is required.

The root of the second problem appears to be that the data does not reflect the performance level of the agent well, as it should, but rather relies too heavily on the goals of individual episodes. Because of this, the data is preprocessed to better reflect the performance of the agent on a particular epoch. This is done by taking the mean of the rewards obtained in the episodes of every epoch. Therefore, we say that the new dataset consists of **mean-episodic-rewards-per-epoch**, accompanied by the unchanged number of processed transitions that is the same for all episodes in a particular epoch. However, this does not solve the second problem with time-to-threshold. On a particular epoch,

*Using the appropriate model, see section: 4.3.2.

the agent can still get lucky and get a lot of easy episodes on the primary environment. Because of this, time-to-threshold really does require some form of severe smoothing.

This would mean that we also have to design and use a smoothing method. However, using a smoothing method to reduce a learning curve to a real number seems to be unnecessarily complicated and to lose too much information in the process of obtaining the statistic. This brings up a third problem with time-to-threshold, and that is that all information in the data after reaching the threshold is disregarded entirely. This is strange, because this is the region where the performance of the agent matters most, hence the focus of many papers on asymptotic performance. This is especially problematic since the threshold is limited to relatively low values, as discussed above.

Threshold fraction

Because of these issues with this existing measure, we design our own, but still based on the same concept. We still define a reward threshold, but instead of measuring the time it takes to reach this threshold, we count the number of times a mean-episodic-reward-per-epoch was obtained that is greater than the threshold. In other words, instead of focussing on how many epochs it takes the agent to get to the threshold, we focus on how many epochs are above the threshold. The **threshold count** (\mathbf{TC}_x for a threshold x) is a great measure for three main reasons. First, it means that outliers (in terms of epochs) are no longer an issue. If the agents are presented with easy epochs, these are just single counts. Second, it means that if the agent performs very well, it will reach above the threshold even when presented with many difficult episodes, reaching a higher count than poorly performing agents. Third, it means that if an agent reaches the threshold faster, then more epochs are played where the agent reaches a reward above the threshold, leading to a higher count when learning is faster. Therefore, using this counting method for a sufficiently high threshold is a great way of measuring the performance of solving runs.

To account for ISBR methods having a very slightly increased number of epochs per transition and a very slightly lower number of processed transitions (a difference of at most 0.7%), we cut the dataset after a fixed number of transitions obtained by every run and instead measure the fraction of the epochs in which the threshold is exceeded. We call this fraction the **threshold fraction** (\mathbf{TF}_x for a threshold x).

We now have a good method of condensing a learning curve to a single real number, but we still have to set the specific threshold that will be used. From figure 4.2 we observe that the reward-threshold-interval from 23 to 24 (x-axis) is the last interval between integers, in which no runs reach their highest reward. After this plateau, the fraction steadily declines. We also observe that 21 is the last integer after a very long plateau. Preferably, the threshold should be either 21 or 24. After all, the threshold should be as high as possible, otherwise reaching the threshold does not properly reflect the quality of the agent. Still, it should not be too high, otherwise too many runs obtain a count of 0. Therefore, we consider the integers in the interval $[20, 25]$, as we do not want to over optimize this threshold either.

For each threshold in the interval, we test the normality through the Shapiro-Wilk test, and the homoscedasticity through the Breusch–Pagan test. It turns out that 24 is the value for which the p-values are best, namely 0.859 and 0.63532, respectively*. Since neither of the two is near significant, the assumptions required for the tests are satisfied, and we can freely use \mathbf{TF}_{24} for measuring the

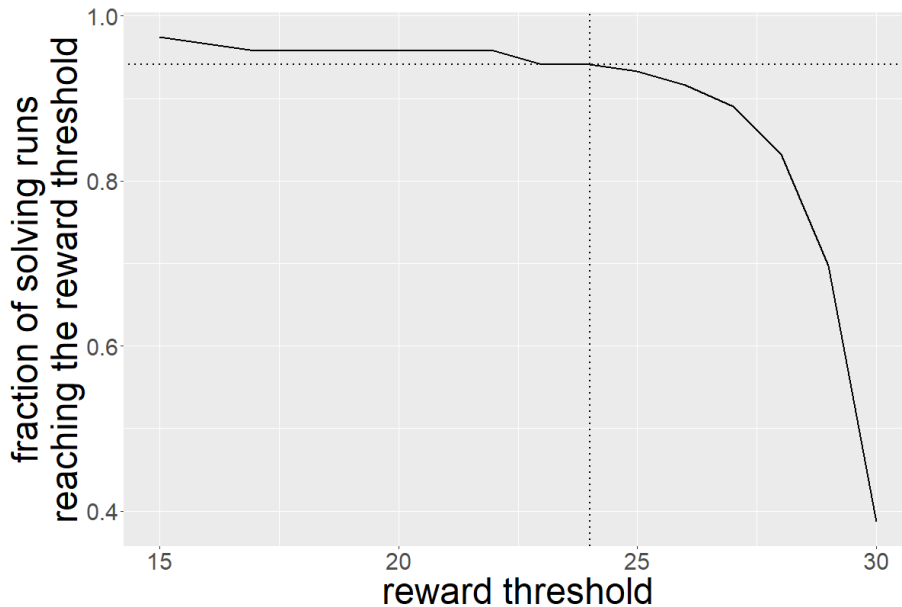


Figure 4.2: For the integer reward thresholds between 20 and 30, we show the fraction of all solving runs that pass that reward in at least 1 epoch. It presents the values in $[20, 25]$ as candidate thresholds for the quality measure of speed.

quality of solving runs.

Still, these measures for the solve rate and speed are not yet fully satisfactory. One should note the resemblance of figures 4.1 and 4.2, they essentially portray the same shape, but with the axes swapped. Because we are looking at learning speed, figure 4.2 only contains data points of solving runs, in other words, where the 10% line in figure 4.1 is greater than 0. Because there are still some runs on a steep slope up between the modes, from figure 4.1 it is to be expected that there would be some number of outliers that simply do not reach a reasonably high counting threshold. From figure 4.2 we see that this percentage of outliers lies at 6%. On the one hand, it is good to appreciate that here, reasoning on the shapes of these curves, and identifying outliers just from sight, gives the same results as running normality and homoscedasticity tests. However, on the other hand, we are setting the counts of 6% of the solving runs to zero simply because the observed performance is extraordinarily poor. This begs the question if these runs should be considered to be solving runs to begin with. As mentioned in section 3.1.4, we answer this question with no. Only requiring the agent to achieve a reward greater than 0 sets the bar too low.

Summary

In conclusion, we now have our two performance measures. At first, what constitutes a solving run seemed relatively straight forward to define, we consider the sum (or equivalently the mean) of all obtained rewards (from the primary environments) in the last 10% of the observed transitions, and we say a run is solving if this sum is greater than 0. This occurs when the rewards obtained from

reaching the environment goals outweighs the cost of moving, in the last part of the run. However, after also considering how to measure the quality of solving runs, the additional requirement was imposed that the run should also contain at least one epoch where it reaches an average episodic reward of at least 24. Therefore, a solving run should be interpreted as a run that solves the problem almost entirely within the given time frame. If a run is not solving, then it either never reached the environment rewards, or it is an outlier in the sense that the performance is extraordinarily poor. Exactly 70% of the runs in experiments 1 and 2 are solving.

Similarly, finding a valid measure for the learning speed required some effort. Still, no compromises had to be made. First, because computing the fraction of epochs above a sufficiently high reward threshold is a great measure of the performance of a run. Second, because 24 is sufficiently high since the fraction of runs reaching the threshold rapidly drops above 24. And third, because the assumptions of linear models are satisfied for this particular threshold, especially after requiring solving runs to reach a mean-episodic-reward of at least 24 in at least one epoch. As such, TF_{24} will be used to measure the learning speed.

4.2 Data exploration

Here, we analyse subsets of data to gain insight in the behaviour of the ISBR methods. The sole purpose is to be able to readily understand the definitive results in section 4.3.

4.2.1 Experiment 1

Experiment 1, as the name implies, is the first experiment where we investigate the performance of ISBR methods. Although the task already requires a great amount of precision to complete (in favour of S-), as it turns out, the problem is not yet sparse enough. As such, the benefits of using S+ will only become apparent in experiment 2.

Solve rate

S+	S-	Algorithm name	Reward ratio	
			1:9	2:8
✘	✔	SG-	1	1
✔	✔	SG	1	0.9
✘	✘	G	0.9	0.7
✔	✘	SG+	0.6	0.5

Table 4.1: The fraction of solving runs for the different parameters settings of experiment 1. The statistics for each cell are obtained from 10 runs. This table is visualized in figure 4.3.

We first investigate the solve rate. In table 4.1 we can see the fraction of solving runs per algorithm. This should give the impression that S- improves the solve rate, but that S+ reduces it. However, using only this table, it is still difficult to tell what the effects are of the individual methods. This is mainly because of the varying reward ratio and because the methods are used in conjunctions in SG.

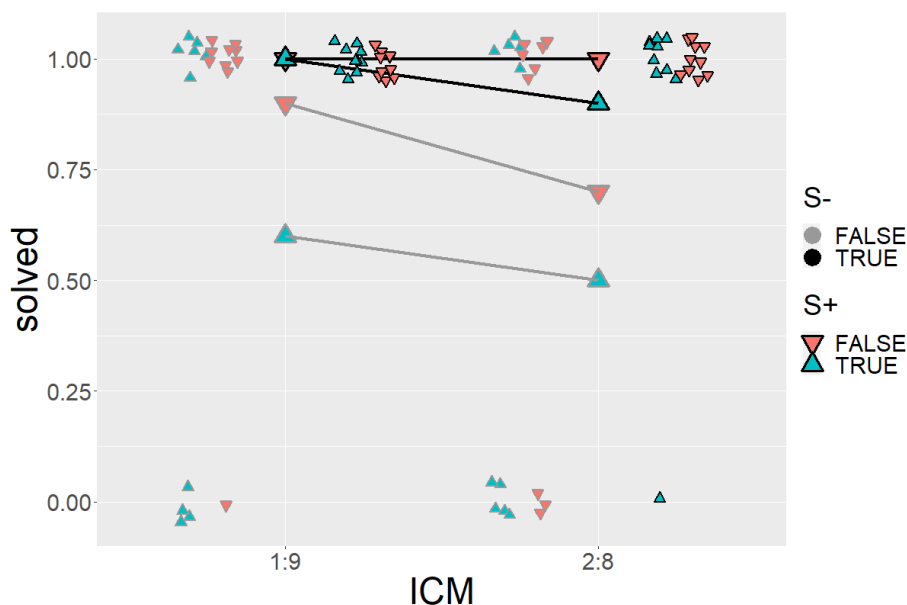


Figure 4.3: A visual representation of the contents of table 4.1. This figure makes it easier to see the effect of the individual variables, by enabling looking at the individual colour changes. Because of this, it is possible to compare all runs simultaneously. The fraction of solving runs (y-axis) are plotted for each algorithm using large markers. Each algorithm is identified by the use of S+ and the use of S-, through the colour of the marker and the edge colour, respectively. For example, SG uses both S- and S+, so it has a blue marker with a black edge. Small markers indicate individual runs. These are situated at 1 if they are solving, and 0 otherwise. Solve rates are plotted for both reward ratios (1:9 and 2:8) that control for the extent with which ICM is used (x-axis).

The same data is illustrated in figure 4.3, which enables seeing the effect of individual variables by comparing all algorithms simultaneously. There we see that this suspicion appears to be correct. After all, the black lines (using S-) are situated above the corresponding grey lines (not using S-), and the blue markers (using S+) are situated below the corresponding red markers (not using S+). We also see that moving from a reward ratio of 1:9 to 2:8 leads to the markers going down, suggesting a negative effect of ICM on the solve rate. We do not see any lines crossing, and the order of the markers is maintained in every subgroup (e.g. face colour when all markers are grouped by border colour), hence no evidence for interaction between the explanatory variables is observed.

Putting these observations to the test, we use a logistic regression and apply likelihood ratio tests. The results are shown in table 4.2. It turns out that indeed, S- and S+ have a significant positive and negative effect on the solve rate respectively. However, the negative effect of increasing the reward ratio from 1:9 to 2:8 is not statistically significant.

[†]In case the reader is unfamiliar with logistic regressions, this can be thought of as having a flat solve rate of around 0.88, rather than 0.5 which would have been the case if the effect of the constant was 0, ignoring the margin of error.

Variable	Effect	Std. Error	Df	LR χ^2	p-value
(Constant)	1.9754 [†]	0.7170	1		
S-	3.1202	1.0939	1	15.3303	$9.026 \cdot 10^{-5}$ ***
S+	-1.3519	0.7133	1	3.9345	0.04731 *
ICM(2:8)	-0.9141	0.6956	1	1.8023	0.17943

Table 4.2: The result from a logistic regression on the data displayed in table 4.1, applying likelihood ratio tests. The results show a significant positive effect of S- on the solve rate, and a significant negative effect of S+. The stars following the p-values indicate their significance levels as: $0 \leq *** < 0.001 \leq ** < 0.01 \leq * < 0.05 \leq \cdot < 0.1 \leq ' < 1$.. Naturally, no likelihood test was performed on the constant, since it should not be removed from a (generalized) linear model.

Learning speed

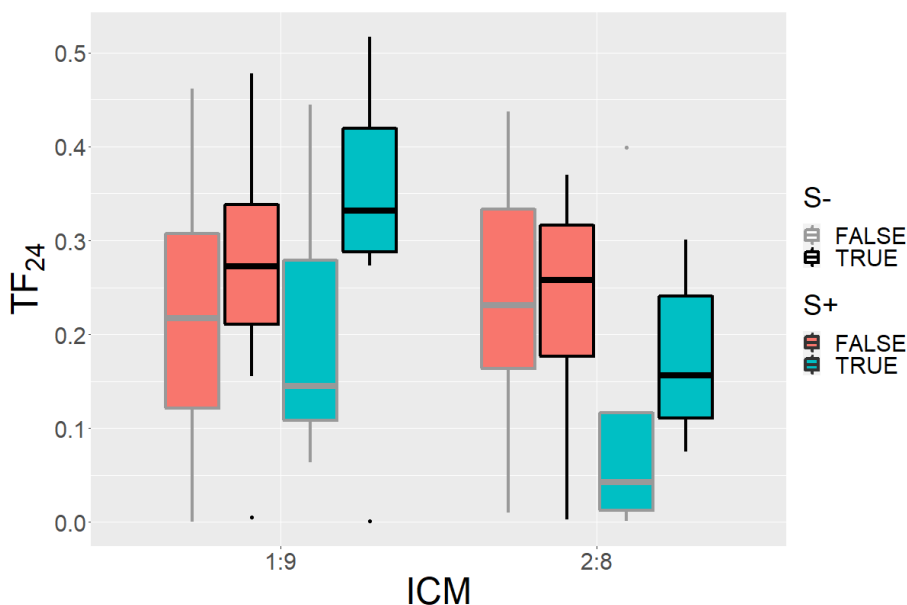
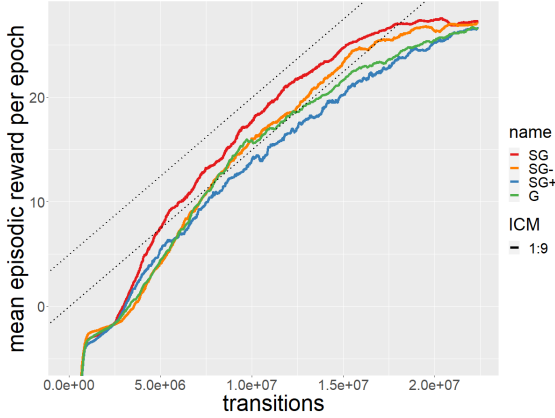


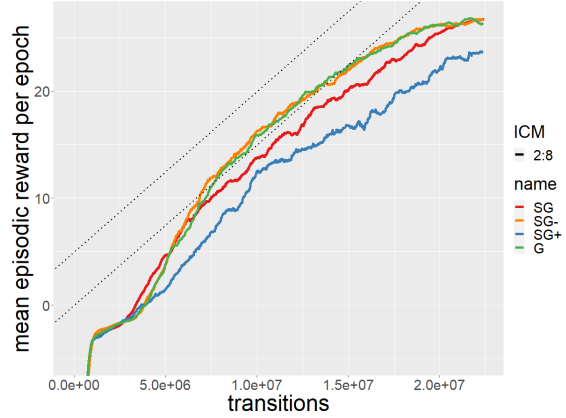
Figure 4.4: A box plot displaying the learning speed measured through TF_{24} [‡] for the various parameter settings in experiment 1. Again, like figure 4.3, the border colour indicates the use of S-, and the face colour indicates the use of S+. S- appears to increase performance, since moving from any box with a grey border to the corresponding box with a black border leads to an increase of the median (center line of the box). S+ shows an interaction effect with an increased reward ratio (x-axis), since enabling S+ specifically when the reward ratio is 2:8, leads to a decrease in performance.

For the learning speed, we present results using TF_{24} in figure 4.4 and smoothed learning curves

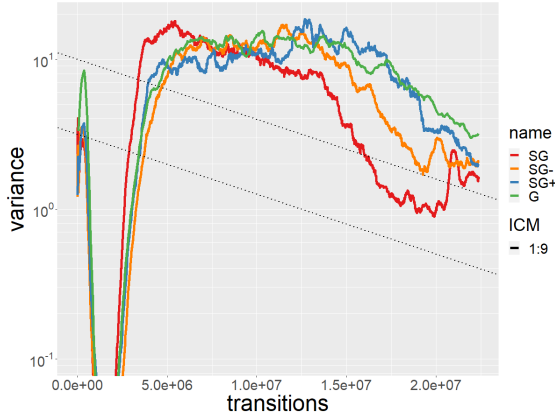
[‡]Recall that TF_{24} is the fraction of epochs with a mean-episodic-reward of at least 24 within a run, as discussed in section 4.1.2.



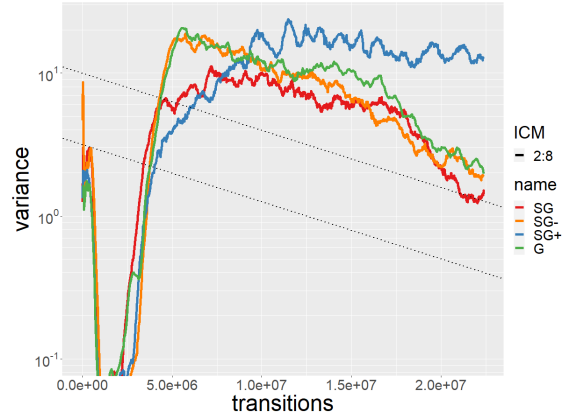
(a) Rewards obtained with a 1:9 reward ratio.



(b) Rewards obtained with a 2:8 reward ratio.



(c) A variance measure of the rewards obtained with a 1:9 reward ratio.



(d) A variance measure of the rewards obtained with a 2:8 increased reward ratio.

Figure 4.5: The smoothed learning and variance curves of the different parameter settings of experiment 1, combining all solving runs into single curves. In figures a and b, the mean-episodic-rewards-per-epoch from the primary environments (y-axis) are plotted against the number of observed transitions (x-axis). The lines are obtained through smoothing by taking the median of the surrounding 2% (half on each side) of the data from all solving runs combined. In figures c and d, a variance measure of the rewards is plotted. This variance measure is the variance, where we replace expected value computations with medians. In other words, we take the squares of the differences with the running medians presented in a and b and then smooth by again taking the median of the surrounding 2% of the data. Because medians are less sensitive to outliers, the produced curves are also less sensitive to outliers. Therefore, each curve gives a better indication of a typical trajectory, compared to a curve produced by smoothing with averages. Figure A.1 shows all lines combined into a single figure.

in figure 4.5. From figure 4.4, we notice a similar pattern for S- as observed for the solve rate.

Again, using S− leads to an increase in performance in every comparison. However, when looking at the effect of S+ we see things are no longer as simple. We observe a decrease in performance when enabling S+ in every case, except for when S+ is used in conjunction with S− for a reward ratio of 1:9, then it increases performance. This is striking, because it causes it to be part of the best performing algorithm of the experiment!

We also notice that the performance drops much more when using S+ if the reward ratio is 2:8. This suggests that S+ interacts with both S− and ICM. Still, we cannot statistically prove an interaction between S+ and S− here. This is because the supporting evidence is only present in 10 out of 66 samples, and because of the dominating negative interaction between ICM and S+ that is also present. However, what we can do is look at the effect of S− separately. The results of fitting a linear model and applying an ANOVA test are presented in table 4.3. The table shows that the both the positive effect of S−, and the negative effect of the interaction between ICM and S+, are statistically significant[¶].

Variable	Effect	Std. Error	Sum sq	Df	F-value	p-value
(Constant)	0.21079	0.03535	0.63345	1	35.5603	$1.347 \cdot 10^{-7}$ ***
S−	0.05817	0.03356	0.05353	1	3.0047	0.0881 .
S+	0.03519	0.04541	0.01070	1	0.6004	0.441
ICM(2:8)	-0.01191	0.04461	0.00127	1	0.0713	0.790
S+:ICM(2:8)	-0.11778	0.06613	0.05650	1	3.1719	0.0799 .
Residuals			1.08663	61		

Table 4.3: The results of a linear regression, on the data of experiment 1, predicting TF_{24} (displayed in figure 4.4). An ANOVA test with type III sum of squares[§] is applied. The results show a significant[¶] positive effect of S− on the learning speed, and a significant negative effect of the interaction between S+ and the increased reward ratio.

The smoothed learning curves of figures 4.5a and 4.5b shows identical results. However, the main contribution of figure 4.5 is not presenting a different perspective on the same pattern. Rather, it is that we can compare the relative position of the learning curves with the relative positions of the variance curves in figures 4.5c and 4.5d. These variance curves essentially show how similar the obtained rewards are of all runs combined (y-axis) for a given number of observed transitions (x-axis). If the variance curves are low, the obtained rewards are very similar, even across runs. The main thing to notice here is the similarity between the relative positions of the trajectories in the smoothed rewards and the smoothed variances. Or rather, the dissimilarity for SG in the case that the reward ratio is 2:8. Both figures 4.5b and 4.2 show that there, SG performs worse than G and S−, but figure 4.5d shows that the variance across runs is the same. Presumably, this is not noise, but rather the effect of S− as its sole function is maintaining the performance of the agent, reducing the variance of the obtained rewards. Sadly, Because this phenomenon is inherently a measure of all runs combined, we cannot verify it statistically. The best we can do is conclude

[§]Type III sum of squares means we always account for the variance in all other groups first, before testing the variable of interests. This is necessary because of the unequal sample sizes, in this case due to the varying number of solving runs.

[¶]Recall that statistical significance is determined with an α of 0.1 in section 4.2, as discussed in the beginning of chapter 4.

that for a reward ratio of 2:8, apparently, the SG runs were extraordinarily similar, even though we expected them to be more dissimilar.

Summary and discussion

When it comes to the solve rate, we have seen that S⁻ has a positive effect and that S⁺ has a negative effect. Neither of these results were expected. For S⁻ we formulated no hypothesis regarding the effect on the solve rate. In contrast, for S⁺ we formulated hypothesis 2, expecting the opposite result. As mentioned earlier in section 3.2.2, the main critique here is that a solve rate is too high to be clearly improved upon, hence a sparser task is required. When it comes to ICM, we saw, at a p-value of 0.18, that ICM might play a role in decreasing the solve rate. However, that p-value is mostly irrelevant. It effectively^{||} tests if the parameter controlling for the increased reward ratio is statistically different from 0. Given a negative effect of ICM, this is not interesting to us. We only include ICM in these experiments to see the interaction with an approach also commonly deployed on such tasks, and to ensure that the benefits of ISBR are exclusive to ISBR. If no benefit is observed from increasing the reward ratio, then there is no reason to keep using ICM at a higher reward ratio.

When it comes to the learning speed, we have seen that S⁻ has a positive effect, but that an interaction between S⁺ and the 2:8 reward ratio has a negative effect. These results were mostly expected in hypotheses 4 and 5. A priori, we did not know if the interaction between S⁺ and ICM was positive or negative. Again, no positive effect of increasing the reward ratio is observed. Because clear benefits of using S⁺ or a higher reward ratio are not present, in experiment 2, the sparsity of the task is increased and that the reward ratio is decreased.

Still, there were two remarkable observations. Most importantly, the best performing algorithm (for both the solve rate and the learning speed) turned out to be SG, used in conjunction with a reward ratio of 1:9. This suggests a positive effect of the interaction between S⁺ and S⁻. At the very least, it supports hypothesis 6. Second, we notice that the variance of the rewards across the different solving runs is surprisingly low when using SG for the increased reward ratio. This suggests an unusually reliable learning trajectory.

4.2.2 Reward ratio 1:9

Having concluded that a higher ICM ratio brings no benefits, we ignore that portion of the data for now and focus on a reward ratio of 1:9. This means we compare the results of both experiment on the overlapping reward ratio. As a consequence, half the data discussed here is identical to that of the previous section. It also means half of all figures and tables entries are identical to those of the previous section. This enables us to more effectively study the effect of the new task, along with any potential interaction effects. It also means you will not have to awkwardly compare graphs in different sections as we present all relevant data where it is discussed, even if this means showing the same figure twice. The results of the entirety of experiment 2 will be discussed in section 4.2.3.

S+	S-	Algorithm name	Task	
			Exp. 1	Exp. 2
✓	✓	SG	1	0.9
✓	✗	SG+	0.6	0.8
✗	✓	SG-	1	0.6
✗	✗	G	0.9	0.2

Table 4.4: The fraction of solving runs for the different parameters settings given that the reward ratio is 1:9. The statistics of each cell are obtained from 10 runs. This table is visualized in figure 4.6.

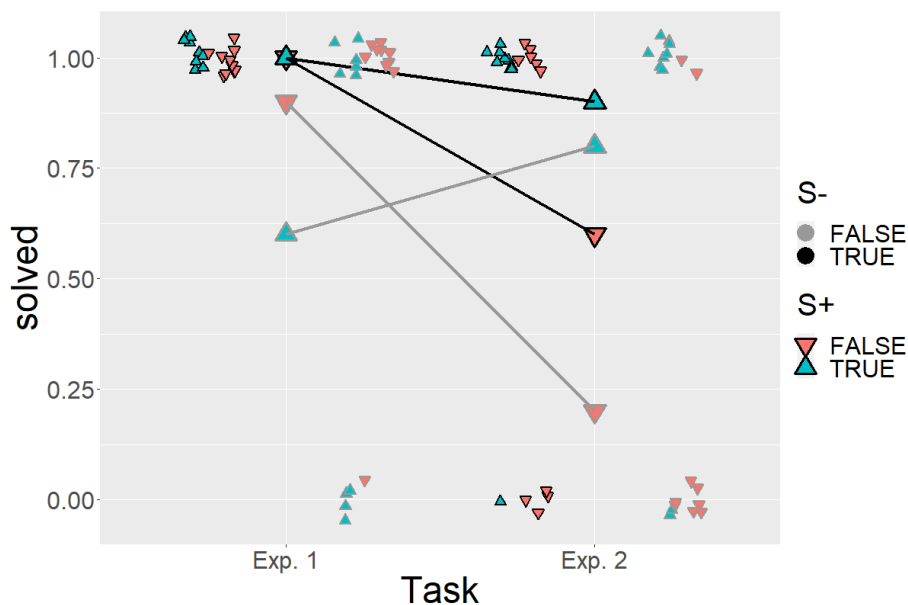


Figure 4.6: A visual representation of the contents of table 4.4. See figure 4.3 for more details on what is depicted.

Solve rate

Again, we first investigate the solve rate. In table 4.4 we can see the fraction of solving runs per algorithm. From this table, it should be clear that increasing the sparsity of the task by 50% makes all the difference. For experiment 2, both S+ and S- appear to have a positive effect on the solve rate. Especially since the results for experiment 1 and 2 are not in agreement, using only this table it is difficult to tell what the underlying patterns are.

Again, the data from the table is visualized, this time in figure 4.6, which has the same structure as figure 4.3. We still see that the black lines (using S-) are situated above the corresponding grey

[†]This is because t-tests and ANOVA tests are equivalent for categorical variables of two levels.

lines (not using S−), indicating an unchanged positive effect of S− on the solve rate. However, this time, we see that the blue markers (using S+) are mostly situated above or at the corresponding red markers (not using S+), also indicating a positive effect of S+. Because there is an exception to this rule when not using S−, we obtain crossed lines, indicating an interaction effect between the task and S+ instead.

Furthermore, we see that when moving from the task of experiment 1, to the task of experiment 2, the markers tend to go down, suggesting a negative effect of increasing the sparsity on the solve rate. This last point is of course entirely understandable and precisely the point of making the task sparser in experiment 2.

Variable	Effect	Std. Error	Df	LR χ^2	p-value	
(Constant)	-1.5187	0.6840	1			
S−	2.0097	0.7392	1	9.3088	0.002281	**
S+	2.5859	0.8911	1	10.7220	0.001059	**
Task(Exp. 1)	3.8653	1.2127	1	17.7503	$2.519 \cdot 10^{-5}$	***
S+:Task(Exp. 1)	-4.2531	1.5067	1	10.0448	0.001528	**

Table 4.5: The result from a logistic regression on the data displayed in table 4.4, applying likelihood ratio tests. The results show significant positive effects on the solve rate of both S+, given the sparse task of experiment 2, and S−. The interaction between S+ and decreasing the sparsity of the task is also shown to have a negative effect on the solve rate. Unsurprisingly, an increase in the sparsity of the task is also shown to negatively effect the solve rate**

Interpreting (generalized) linear models

Putting these observations to the test, we again use a logistic regression and apply likelihood ratio tests. The results are shown in table 4.5. However, this time there is a significant base effect of S+ while the interaction term with the task is also significant.

Therefore, it is worth mentioning that the order of the levels of the task factor matters for the outcome of the base effect of S+. This is because the “base” effect of S+ is not simply the flat effect of S+ anymore because of the interaction with the task that is included in the model. it is now the effect given the first level of the task factor. The interaction effect is still defined with respect to this base effect as if it is an effect that is present for all levels of the task, but it should no longer be interpreted as such.

In the table, we see that the task of experiment 1 obtains its own parameter. This means that the first level of the task factor is the task of experiment 2. As such, the row indicated by S+ should be interpreted as the effect of **S+ given the sparser task**. Just to clarify, the effect of S− should not be interpreted as such, since there is no interaction term of S− with the task included in the model.

Having said that, it turns out that, indeed, both S+, given the sparse task of experiment 2, and S− have a positive effect on the solve rate. We also see that the interaction between S+ and decreasing

**Equivalently, the decrease in sparsity of experiment 1 has a significant positive effect, as more directly displayed in the table.

the sparsity of the task has a significant negative effect on the solve rate. Since in applications, the task is generally given and an adequate algorithm has to be chosen, this means that if the problem is sparse enough and a reward ratio of 1:9 is used, S+ will most likely have a positive effect on the solve rate.

Learning speed

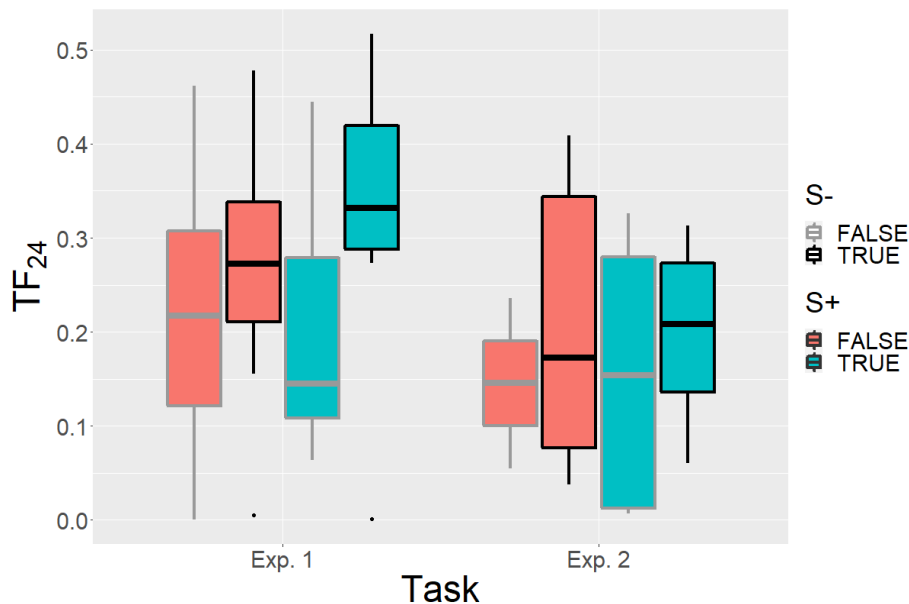
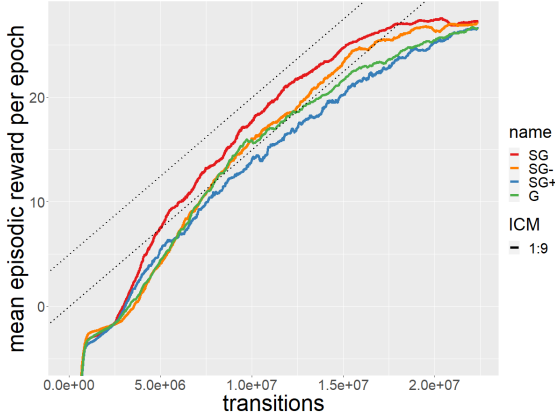


Figure 4.7: A box plot displaying the learning speed measured through TF_{24}^\ddagger for the various parameter settings using a reward ratio of 1:9. See figure 4.4 for more details on what is depicted.

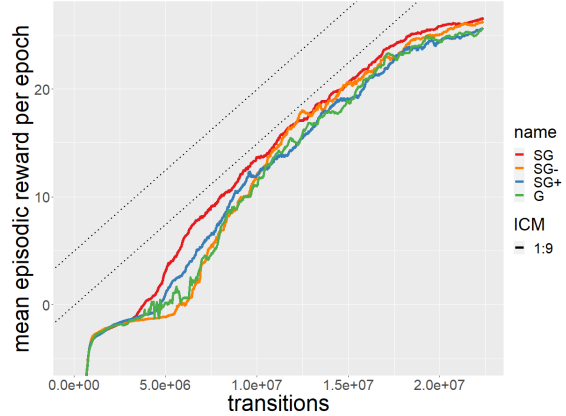
Variable	Effect	Std. Error	Sum sq	Df	F-value	p-value	
(Constant)	0.12183	0.04207	0.15017	1	8.3870	0.005381	**
S-	0.07604	0.03506	0.08421	1	4.7030	0.03437	*
S+	0.02222	0.03561	0.00697	1	0.3891	0.53529	
Task(Exp. 1)	0.08468	0.03595	0.09936	1	5.5493	0.02201	*
Residuals			1.00269	56			

Table 4.6: The results of a linear regression, on the data with a reward ratio of 1:9 (displayed in figure 4.7), predicting TF_{24} . An ANOVA test with type III sum of squares is applied. The results show a significant positive effect of S- on the learning speed, and a significant positive effect of the reduced sparsity in the task in experiment 1.

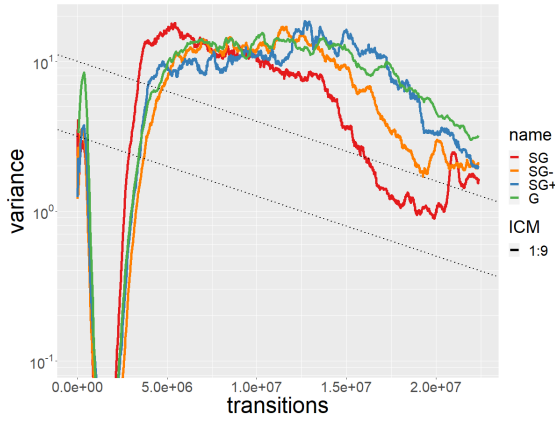
For the learning speed, we present results using TF_{24} in figure 4.7 and the smoothed learning curves in figure 4.8. We first discuss figure 4.7. Again we see that S- has a positive effect on the learning speed since it always causes the performance to go up. We also see that the sparser task of



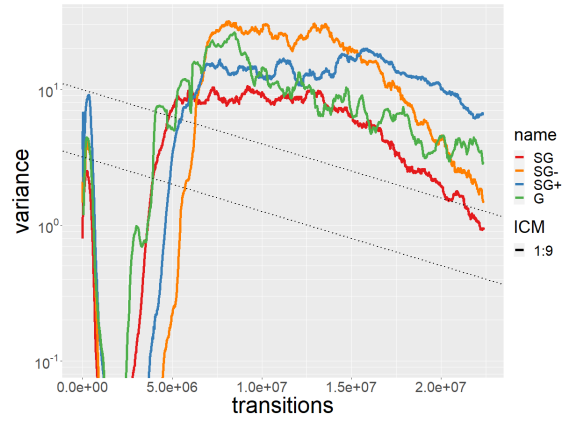
(a) Rewards obtained from experiment 1.



(b) Rewards obtained from experiment 2.



(c) A variance measure of the rewards obtained from experiment 1.



(d) A variance measure of the rewards obtained from experiment 2.

Figure 4.8: The smoothed learning and variance curves of the different parameter settings for a reward ratio of 1:9, combining all solving runs into single curves. See figure 4.5 for more details on what is depicted.

experiment 2 leads to a lower learning speed in every case. For S+, no substantial effect seems to be present. These observations match the results of fitting a linear model and applying an ANOVA test, as shown in table 4.6.

Figure 4.7 also shows these patterns. The performance in subfigures b and d is generally worse than in a and c, and within each subfigure, SG and SG- perform best. This last point is also the main contribution of this figure. We have already seen that SG has the highest solve rate on both tasks, and now we also see that SG has the highest learning speed on both tasks. Taking a closer look at figures 4.7 and 4.8, we remark that this holds from the perspective of TF_{24} , the smoothed learning curves, and the smoothed variance curves.

Summary and discussion

When it comes to S⁻, we have seen that it again has a positive effect on both the solve rate and the learning speed. We have also seen that S⁺ has a positive effect on the solve rate if the task is sparse enough. In other words, by considering a sparser task, we found support for hypothesis 2. We also made the observation that SG again had the highest solve rate and the highest learning speed. At the very least, this suggests that the benefits of S⁺ and S⁻ are additive and can be used together to form a better algorithm, which supports hypothesis 6.

4.2.3 Experiment 2

We now focus on the results of experiment 2. Since the task is constant, this means we go back to studying the effect of the ISBR methods and their interactions with ICM.

Solve rate

S ⁺	S ⁻	Algorithm name	ICM	
			None	1:9
✓	✓	SG	0.6	0.9
✓	✗	SG+	0.3	0.8
✗	✓	SG-	0.7	0.6
✗	✗	G	0.5	0.2

Table 4.7: The fraction of solving runs for the different parameters settings on the task of experiment 2. The statistics for each cell are obtained from 10 runs. This table is visualized in figure 4.9.

Again, we first investigate the solve rate. In table 4.7 we can see the fraction of solving runs per algorithm. What should jump out is that the order of the best performing algorithms differs greatly for the different reward ratios. This makes the table hard to read. Therefore, we immediately focus on the visualization in figure 4.9. We still see that the black lines (using S⁻) are situated above the corresponding grey lines (not using S⁻), indicating an unchanged positive effect of S⁻ on the solve rate. For S⁺ we again see crossed lines, indicating an interaction effect between S⁺ and the use of ICM.

Putting these observations to the test, we again use a logistic regression and apply likelihood ratio tests. The results are shown in table 4.8. It turns out that, indeed, both S⁺, given a reward ratio of 1:9, and S⁻ have a significant positive effect on the solve rate. We also see a significant negative effect of the interaction between S⁺ and not using ICM at all. In other words, S⁻ always improves the solve rate, but S⁺ only improves the solve rate if ICM is used.

It is worth emphasizing that this is the third time we find an interaction effect containing S⁺, and the second time S⁺ is found to be interacting with ICM. In experiment 1 the interaction with ICM lead to a reduced learning speed, presumably caused by an excessively high reward ratio. In contrast, now we find an increased solve rate when using a reasonable reward ratio, compared to not using ICM at all.

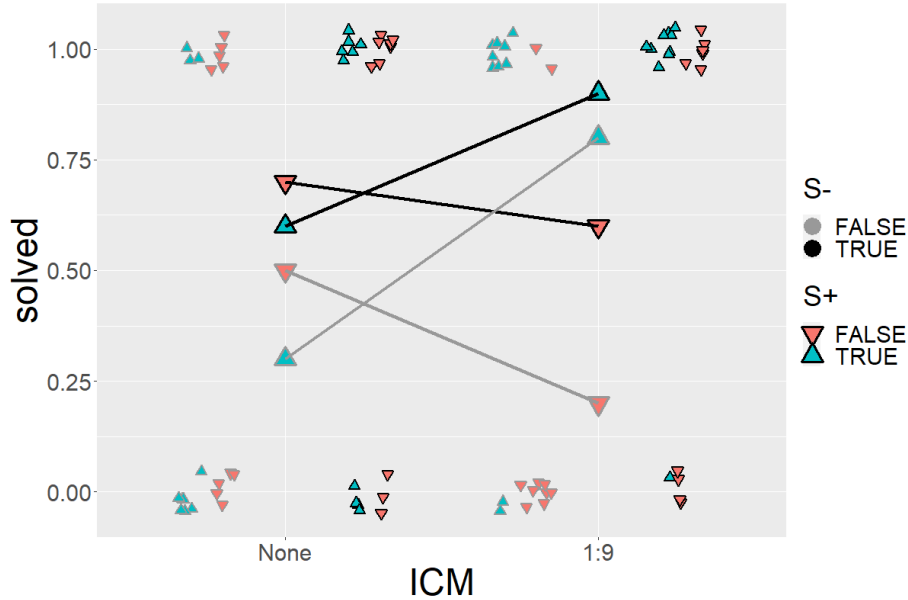


Figure 4.9: A visual representation of the contents of table 4.7. See figure 4.3 for more details on what is depicted.

Variable	Effect	Std. Error	Df	LR χ^2	p-value
(Constant)	-1.0515	0.5576	1		
S-	1.2161	0.5131	1	5.9612	0.014624 *
S+	2.3054	0.8094	1	9.7590	0.001785 **
ICM(None)	0.8869	0.6772	1	1.7588	0.184774
S+:ICM(None)	-2.9686	1.0581	1	8.7294	0.003131 **

Table 4.8: The result from a logistic regression on the data displayed in table 4.7, applying likelihood ratio tests. The results show significant positive effects on the solve rate of both S+, given that a reward ratio of 1:9 is used, and S-. It also shows that the interaction between S+ and not using ICM has a negative effect on the solve rate.

Learning speed

For the learning speed, we present results using TF_{24} in figure 4.10 and the smoothed learning curves in figure 4.11. We first discuss figure 4.10. The most important thing to realize here is that the number of solving runs is greatly reduced compared to experiment 1. As such, many of the boxes displayed here do not contain many data points. This is mainly an issue for S+ since, as we have seen, it greatly relies on ICM to obtain solving runs. Still, we can see that S- has a positive effect on the learning speed since it always causes the performance to go up. We also see that using ICM has a negative effect on the learning speed. This is not too surprising since its main function is exploration. After the environment goals are found, it essentially only disturbs the

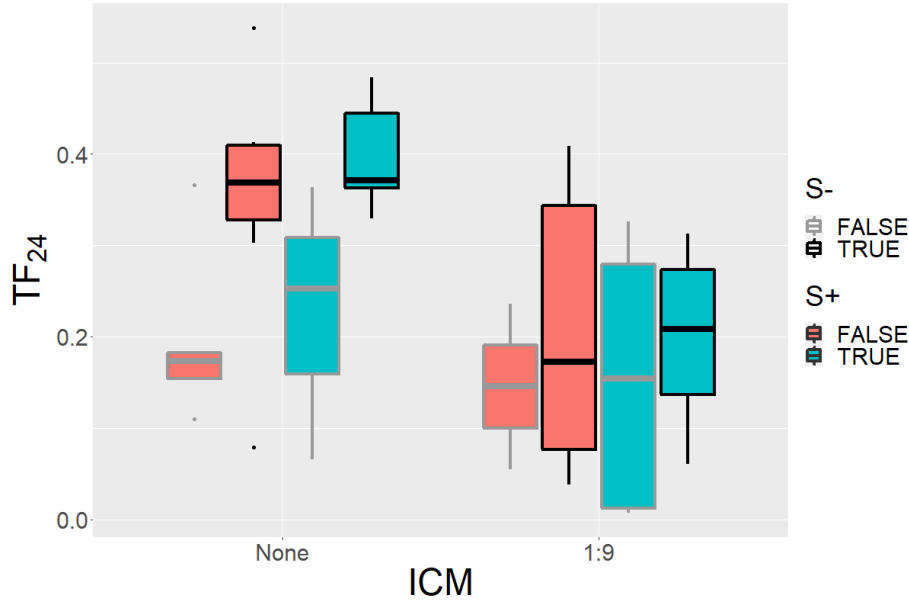


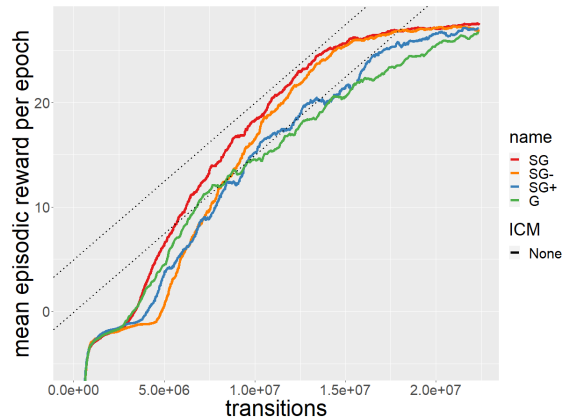
Figure 4.10: A box plot displaying the learning speed measured through TF_{24}^{\ddagger} for the various parameter settings of experiment 2. See figure 4.4 for more details on what is depicted.

Variable	Effect	Std. Error	Sum sq	Df	F-value	p-value	
(Constant)	0.10175	0.04201	0.08457	1	5.8652	0.0198400	*
S-	0.10377	0.03638	0.11734	1	8.1379	0.0066996	**
S+	0.02728	0.03701	0.00783	1	0.5433	0.4651831	
ICM(None)	0.13248	0.03674	0.18751	1	13.0049	0.0008186	***
Residuals			0.60558	42			

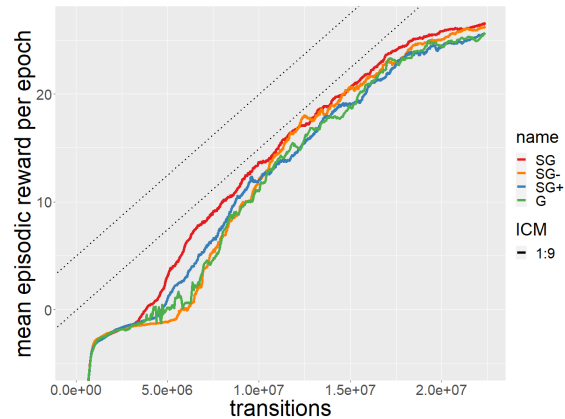
Table 4.9: The results of a linear regression, on the data of experiment 2 (displayed in figure 4.10), predicting TF_{24} . An ANOVA test with type III sum of squares is applied. The results show a significant positive effect on the learning speed of both S- and not using ICM.

learning process, adding no further value. These observations match the results of fitting a linear model and applying an ANOVA test, as shown in table 4.9.

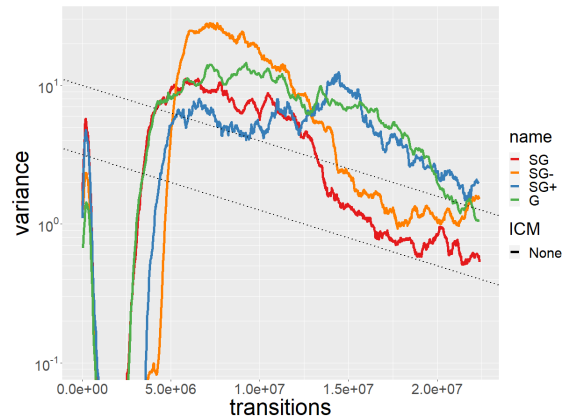
Figure 4.10 also shows these patterns. Again, the performance in subfigures b and d is generally worse than in a and c, and within each subfigure, SG and SG- perform best. Just like in figure 4.7, this last point is also the main contribution of this figure. Taking a closer look at figures 4.10 and 4.11, we again remark that SG is the fastest learning algorithm for both tasks. This holds from the perspective of TF_{24} , the smoothed learning curves, and the smoothed variance curves.



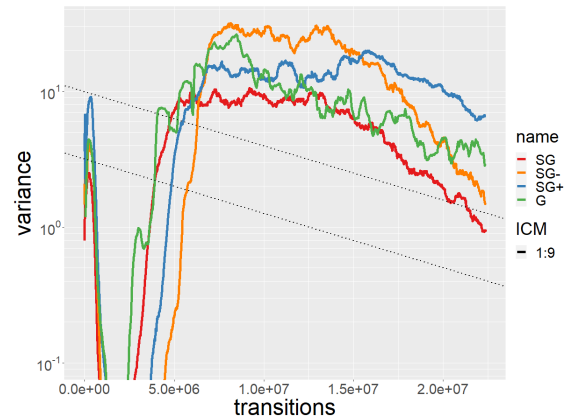
(a) Rewards obtained without ICM.



(b) Rewards obtained with a 1:9 reward ratio.



(c) A variance measure of the rewards obtained without ICM.



(d) A variance measure of the rewards obtained with a 1:9 reward ratio

Figure 4.11: The smoothed learning and variance curves of the different parameter settings for experiment 2, combining all solving runs into single curves. Figure A.2 shows all lines combined into a single figure. See figure 4.5 for more details on what is depicted.

Summary and discussion

When it comes to S₋, we have seen that it again had a positive effect on both the solve rate and the learning speed. We have also seen that S₊ has a positive effect on the solve rate if ICM is used. A potential explanation for this improved performance is that ICM is more able to drive exploration and find the environment rewards through the use of S₊, since S₊ causes the agent to visit uncommon states more often. However, given that the effect of this interaction can be both positive and negative depending on the statistic measured (solve rate versus learning speed), the reward ratio and the sparsity of the task, there is most likely room for improvement.

Since the advantage estimates are analysed to determine the initial states for ISBR methods, it

seems that the ICM rewards being part of the advantages estimates is a natural candidate for the cause of this interaction. However, if this is the case, one would also expect a negative effect of the interaction between S− and ICM, which we have not found yet. However, with the increased sample size that is used in the complete analysis, we still find evidence in this direction, presenting us with a path for future research.

When it comes to the performance of SG, we again observe that SG had the highest learning speed, and was very close to having the highest solve rate. This supports hypothesis 6.

4.3 Definitive results

We have now looked at all the data, focussing on one aspect at the time. This approach is great for gaining insight into the data, but for definitive conclusions these analyses need to be unified. Because of this, we go over the data one last time, verifying all found patterns again on the complete dataset.

4.3.1 Solve rate

S+	S−	Algorithm name	Task			
			Exp. 2	Exp. 1		
			ICM			
			None	1:9	1:9	2:8
✓	✓	SG	0.6	0.9	1	0.9
✓	✗	SG+	0.3	0.8	0.6	0.5
✗	✓	SG−	0.7	0.6	1	1
✗	✗	G	0.5	0.2	0.9	0.7

Table 4.10: The fraction of solving runs for the different parameters settings. The statistics for each cell are obtained from 10 runs. This table is visualized in figure 4.12.

Again, we present the data in table 4.10 and visualize it in figure 4.12. It is important to note that now, for the first time, we have a variable with 3 levels. Namely, the reward ratio can be either 0:1 (None), 1:9 or 2:8. From figure 4.12a, we can clearly see that the effect of the reward ratio is not linear as it increases. After all, there is an interaction with S+ that only occurs on the task of experiment 2, when comparing no ICM to a reward ratio of 1:9. This means that treating the different levels of the reward ratio as categories, rather than as a continuous variable, is still the correct procedure.

For S+ there are two interaction effects. The interaction effect with ICM is clearly visible in the crossed lines for task 2. The interaction effect with the task is also visible, since the effect of S+ changes depending on the task. This is particularly apparent for a reward ratio of 1:9. For S− we simply see an improved solve rate everywhere. Knowing this, we move on to table 4.11 where we present the results of the logistic regression and the likelihood ratio tests.

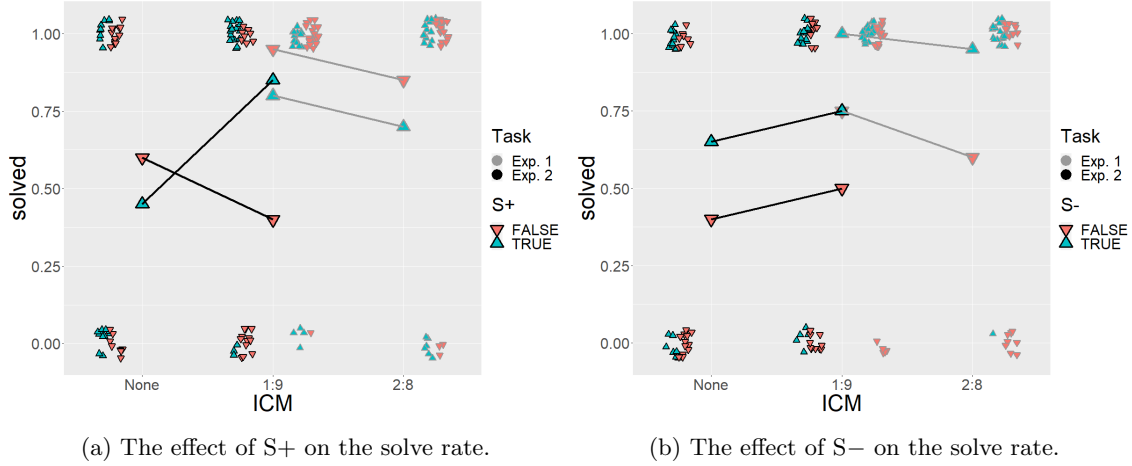


Figure 4.12: A visual representation of the contents of table 4.10. See figure 4.3 for more details on what is depicted. Notice that the colour of the lines now indicates the task, and that the marker colour indicates the use of the ISBR method.

Variable	Effect	Std. Error	Df	LR χ^2	p-value	
(Constant)	1.3465	0.5607	1			
S-	1.7252	0.4364	1	18.0804	$2.118 \cdot 10^{-5}$	***
S+	2.4705	0.8344	1	10.4769	0.001209	**
Task(Exp. 1)	3.7365	1.1660	1	17.5098	$2.858 \cdot 10^{-5}$	***
ICM(None, 2:8)			2	3.1311	0.208971	
S+:Task(Exp. 1)	-4.1158	1.4599	1	9.8103	0.001735	**
S+:ICM(None, 2:8)			2	9.5861	0.008287	**
ICM(None)	0.9678	0.7076	1	1.9191	0.165956	
ICM(2:8)	-1.2660	1.2252	1	1.2137	0.270595	
S+:ICM(None)	-3.1945	1.0995	1	9.3699	0.002206	**
S+:ICM(2:8)	0.6591	1.4557	1	0.2126	0.644717	

Table 4.11: The result from a logistic regression on the data displayed in table 4.10, applying likelihood ratio tests. The results show significant positive effects on the solve rate of S+, given the task of experiment 2 and that a reward ratio of 1:9 is used, S-, and the task of experiment 1. It also shows that the interactions of S+ with both not using ICM, and the task of experiment 1, have a negative effect on the solve rate.

This table present two very important results. First, we see that S- has a clear positive effect on the solve rate with a p-value of $2.118 \cdot 10^{-5}$. Second, we see that, given the task of experiment 2 and a reward ratio of 1:9, S+ also has a clear positive effect on the solve rate with a p-value of 0.001209. This demonstrates that both S- and S+ can improve the solve rate.

Furthermore, the table also shows that S+ relies on the use of an exploration algorithm, and on

the sparsity of the task to realize this performance increase. This illustrates two things. First, it shows that S+ in combination with ICM does not monotonically improve the solve rate. Because on the easier task the effect of this interaction decreases the performance, which it should not, we speculate that there is mostly likely some implementation artefact that is causing decreased performance on easier tasks. Luckily, there are numerous ways to improve the implementation, as will be discussed in the chapter 5 Second, it shows that the use of an exploration algorithm such as ICM is essential to the success of S+. As mentioned earlier, we speculate that the relation is the other way around and that S+ is helping ICM by repeatedly visiting infrequently seen states, rather than ICM helping S+. Regardless, the latter should not be excluded. It very well might be the case that ICM is causing the agent to visit unseen states, which S+ can then capitalize on once G sets far-away goals.

4.3.2 Learning speed

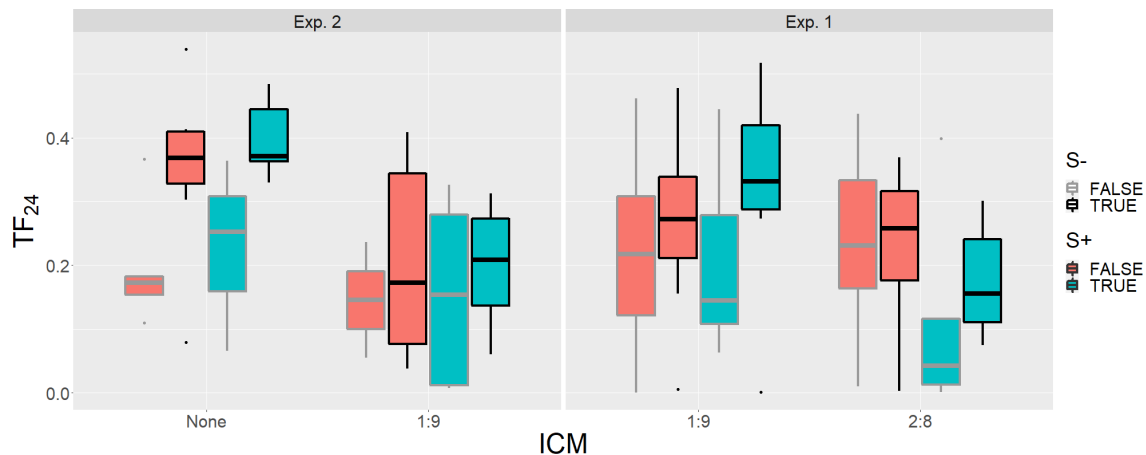


Figure 4.13: A box plot displaying the learning speed measured through TF_{24}^{\ddagger} for all parameter settings. See figure 4.4 for more details on what is depicted.

For the learning speed, we present results using TF_{24} in figure 4.13. Previously, we found significant results for the effects of S-, ICM, the task, and the interaction between S+ and ICM. However, by looking at the entirety of the data, we can see another pattern, namely an interaction between S- and ICM. This interaction of ICM with the ISBR methods is mostly visible as a general trend down when increasing the reward ratio, except for when neither S- nor S+ is used.

Because now the interaction extends to all reward ratios, the contribution of ICM is modelled as a continuous variable, where the reward ratios 0:1, 1:9 and 2:8 are assigned the values -1, 0 and 1, which are used as their numerical values in the regression. This assignment is in line with all other tests, where a reward ratio of 1:9 was also the first level of the ICM factor. The results are presented in table 4.12. From this, it becomes apparent that the interactions of both S+ and S- with ICM have small effect sizes, but that these effects are estimated to be almost the same. Likewise, the p-values are also very similar, and both very close to being significant. This explains why this pattern was not found earlier, we simply required more data since the pattern is rather

Variable	Effect	Std. Error	Sum sq	Df	F-value	p-value	
(Constant)	0.14997	0.03023	0.39474	1	24.6036	$2.724 \cdot 10^{-6}$	***
S-	0.08279	0.02464	0.18114	1	11.29025	0.001087	**
S+	-0.00117	0.02442	0.00004	1	0.0023	0.961893	
ICM	-0.02073	0.03573	0.00540	1	0.3365	0.563090	
Task(Exp. 1)	0.07501	0.03333	0.08124	1	5.0635	0.026520	*
S-:ICM	-0.06795	0.03651	0.05557	1	3.4634	0.065537	.
S+:ICM	-0.06360	0.03577	0.05071	1	3.1604	0.078339	.
Residuals			1.68462	105			

Table 4.12: The results of a linear regression on all the data (displayed in figure 4.13), predicting TF_{24} . An ANOVA test with type III sum of squares is applied. The results show significant positive effects on the learning speed of S-, given a reward ratio of 1:9, S-, given a reward ratio of 0:1 (None), and the task of experiment 1. It also shows that the interactions of both S+ and S- with increased reward ratios have a negative effect on the learning speed.

subtle.

Taking a closer look, we see that the effect of the interactions are negative. This means that without ICM, we obtain an estimated effect of $0.08279 - (-0.06795) = 0.15074$ for S-, which is even better than the effect we observe in the table. There the presented value is the effect of S- given an ICM ratio of 1:9. Knowing this, it should come as no surprise that even if we assume that the (near significant) interaction effect is present, we obtain a significant^{††} positive effect of S- for the two relevant reward ratios (0:1 and 1:9). It also means that although the effect of S- is not significant for the reward ratio of 2:8, the effect is still estimated to be positive. More specifically, the effect is estimated to be $0.08279 - 0.06795 = 0.01484$.

It should be clear that if the interaction term between S- and ICM was not included, that then the base effect of S- is still significant. Namely, it would have an estimated effect of 0.0767233 with a p-value of 0.002437 **. In other words, we can definitively conclude that S- also has a clear positive effect on the learning speed.

This only leaves the interaction between ICM and the ISBR methods to be more closely examined. Because the interaction effects of using ICM with S+ and S- are very similar, we suspect that ICM interacts with the ISBR scheme rather than with the individual methods. Therefore, we test the interaction using the effect of using S+ or S-. The results are shown in table 4.13. There we see that there is indeed a significant interaction effect between using an ISBR methods and ICM.

^{††}With a p-value of 0.00165 ** in the case that ICM is not used.

Variable	Effect	Std. Error	Sum sq	Df	F-value	p-value	
(Constant)	0.152178	0.030205	0.39474	1	25.3837	$1.942 \cdot 10^{-6}$	***
S-	0.081325	0.024510	0.18114	1	11.0096	0.001243	**
S+	-0.003219	0.024268	0.00004	1	0.0176	0.894732	
ICM	-0.011731	0.040137	0.00540	1	0.0854	0.770640	
Task(Exp. 1)	0.074526	0.033353	0.08124	1	4.9929	0.027549	*
(S+ or S-):ICM	-0.102326	0.041923	0.05557	1	5.9575	0.016311	*
Residuals			1.70303	106			

Table 4.13: The results of a linear regression on all the data (displayed in figure 4.13), predicting TF_{24} . An ANOVA test with type III sum of squares is applied. The results show a significant negative effect on the learning speed of the interactions of ISBR with increased reward ratios.

Chapter 5

Discussion

The discussion consists of two parts. First, we wrap up the results of experiment 1 and 2 in section 5.1. Here, we suggest future research based on the experimental finding. Second, we go over the experimental process in section 5.2. There we discuss how the ISBR methods were designed as well as some obstacles that had to be overcome in the process of designing an algorithm and testing it. There, we suggest future research based on ideas that came up while designing ISBR methods.

5.1 Experimental insights

5.1.1 Hypotheses and results

Summarizing the results, we found conclusive evidence that shows that

1. G enables solving the proposed tasks.
2. S+ improves the solve rate, but only on sufficiently sparse tasks and when using ICM.
3. S- improves both the solve rate and the learning speed.
4. Combining ICM with ISBR methods reduces the learning speed.

These results confirm hypotheses 1, 2, 4 and 5. This leaves us with hypotheses 3, 6 and 7.

Hypothesis 6, S outperforms both S+ and S-, is the easiest to treat. Not only did we not find any evidence supporting any form of negative interaction between S+ and S-, S was also the overall best performing algorithm in both solve rate and learning speed, as discussed in sections 4.2.2 and 4.2.3. This certainly supports hypothesis 6. In addition, it is worth mentioning that S was observed to be extraordinarily stable, even when using an excessively high reward ratio.

Hypothesis 3, S+ improves the learning speed, requires more effort to analyse. This is because the dominating pattern for S+ is the interaction with ICM. Measured directly, no statistically significant evidence was found that supports that S+ improves the learning speed. However, through the interaction with ICM, we can still find support for hypothesis 3.

Namely, we have seen that given a reward ratio of 1:9, S+ is estimated to have no effect on the learning speed. We have also seen that that given a reward ratio of 0:1, S+ is estimated to have a positive effect on the learning speed. This implies that if the negative interaction between ISBR and ICM is the consequence of some implementation artefact, that S+ can have a positive effect if this artefact is found and resolved. This is because ICM should not interact with ISBR in general. In other words, although a direct benefit is not observed, the data does suggest that it is possible to use S+ to improve the learning speed. On top of that, because we found that S outperforms S- in terms of learning speed, it is implied that S+ also contributed to the improved learning speed.

Lastly, hypothesis 7, about fixed action patterns, turns out to not be testable. This is because many of the recordings did not save properly due to memory errors. This makes it impossible to properly investigate this phenomenon. The data is simply too sparse.

5.1.2 Future research

ISBR and ICM

From the results, we see several ways to improve the proposed algorithms. A prominent direction is resolving the negative interaction between ICM and ISBR. There are 4 main candidates for the cause of this interaction. The first 3 are the presence of ICM rewards in the advantage estimates that

1. are used to determine the initial state for resimulation.
2. indicate the quality of actions taken in the ISBR environments.
3. indicate the quality of actions taken in the primary environments.

Potential cause 1 seems like a natural candidate, but since selecting initial states is a discrete process, it is most likely that the ICM rewards are simply drowned out by the much larger environment rewards. Potential cause 2 is a much more likely candidate. After all, the goals on ISBR environments are feasible by construction. Therefore, ICM should not have much to add there. In light of potential cause 2, 3 might seem unlikely. However, there could also be a more indirect effect where the ICM models get very familiar with the resimulated episodes, which then causes the agent to avoid these states in the primary environments due to the ICM rewards. In other words, if potential cause 3 is found to be relevant, one should consider somehow limiting the extent by which the ICM models learn the structure of the data from the ISBR environments.

On top of that, the current hypothesis is that S+ obtains an improved solve rate through the use of ICM on the S+ environments. Since we would like to obtain methods that yield strictly better performance in all cases, it is important to investigate all possible causes of the slowed learning speed. This way, one could find an ISBR implementation such that S+ improves both the solve rate and the learning speed.

The fourth candidate cause has nothing to do with the presence of the ICM rewards in the advantages, but rather with the setup of the ISBR environments. Namely, it is that

4. GBR environments run multiple episodes with the same goals.

This is an issue because in turn, it causes multiple ISBR environments to obtaining the same goal, which greatly decreases the sample diversity. In other words, if acting on potential cause 3 turns out to increase the learning speed, 4 could be the deeper underlying problem for exactly the same reasons we just discussed for 3. Similarly, one could also lower the repetitions of ISBR algorithm from 5 to 3, or even 1, just to increase the sample diversity. The potential drawback of this is that if goals are very difficult to achieve, a single resimulation episode might not be enough to reach any goals at all.

S+ and task sparsity

Another prominent direction is the effect of the sparsity of the task on the benefits of the interaction between ICM and S+. This effect should simply not be present. A possible cause is that S+ and ICM together form some form of semi fixed action pattern that explores very far away regions, ignoring the goals in the GBR environment. This behaviour was observed in a recording of a final epoch. If the angle to reach the environment goals then becomes slightly larger, the solve rate then remains roughly the same if the “extent” of this semi fixed action pattern can still reach the inner area with goals. After all, if goals from the GBR environment are ignored, why would the movement penalty not also be ignored. The solve rate is then determined mostly by the direction of exploration. Acting on potential cause 4 could possibly resolve this by limiting the domination of the loss by similar samples obtained from the S+ environments.

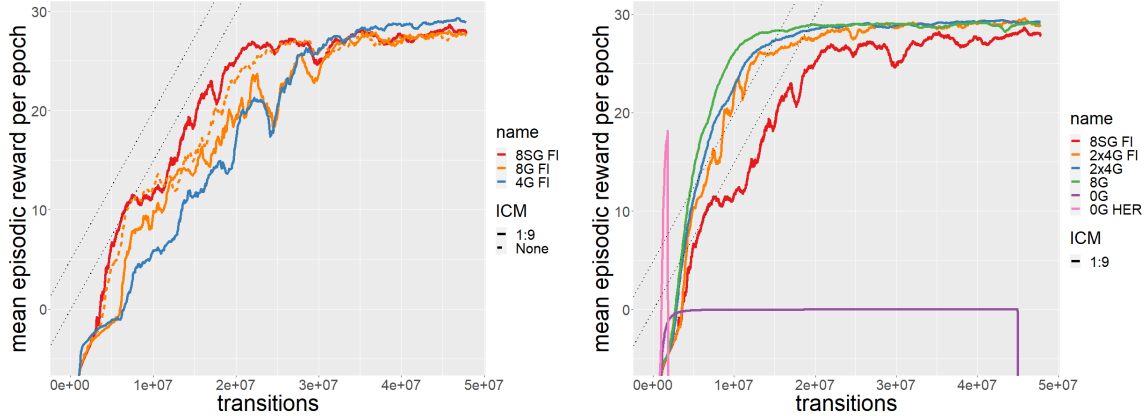
5.2 Experimental process

This section consist of disjoint summaries of important events and thought processes that do not directly contribute to studying GBR or ISBR, as presented in the rest of this thesis. Namely, in section 5.2.1 we discuss an experimental setup that did not function properly, but still taught us the importance of data diversity. In section 5.2.2 we discuss an early ISBR algorithm and how, at the time, this led to using an agent with memory. In section 5.2.3 we discuss the fascinating behaviour of the agent without GBR on experiment 0, and we discuss the possible causes of this behaviour. Finally, in section 5.2.4, we suggest future research based on general ideas and concepts that came up during the entire process.

5.2.1 Fixed initial conditions

Around halfway through the thesis, it was imposed that from that point forward, all experiments had to be run with predetermined initial states and goals. The idea behind this was that it would reduce the variance of the results, making it clearer to see the difference between different methods. This did not work. The reason for this is most likely that in all experiments, initial conditions hardly vary. It is the actions of the agent that entirely determines the course of an episode, and is influenced next to none by the initial conditions. Especially at the beginning of learning when actions are mostly random, the initial conditions play no role.

What made matters worse was that in practise, fixing the initial conditions meant fixing the primary environments using the machinery built for resimulation. This had the side effect that the primary environments no longer reset randomly every time an episode terminated. As a consequence, the optimization process was done with highly correlated data. This lead to catastrophically misleading results, as any form of increased sample diversity lead to better performance.



(a) Rewards obtained from the naive experiment. (b) Rewards obtained from further investigating sample diversity.

Figure 5.1: Results on the task of experiment 0 on the learning speed. In the legends, FI stands for fixed initial conditions, where the initial conditions are reset to predetermined values when episodes end. For a more detailed description, see figure 4.5. The curves are only based on solving runs, except if no runs were solving, then all runs are used. See table 5.1 for more information on the solve rate.

Algorithm name	# Solving runs	# Runs	Solve rate
8G	7	7	1
2x4G	7	7	1
2x4G FI	8	8	1
8SG FI	10	10	1
8G FI	9	10	0.9
8G FI (no ICM)	8	10	0.8
4G FI	2	10	0.2
0G	0	10	0
0G HER	0	10	0

Table 5.1: The fraction of solving runs for the different algorithms ran on the task of experiment 0.

To make this more concrete, in figure 5.1 the smoothed learning curves are shown on the easier variant of the sparse reacher task of experiment 0, and in table 5.1 the solve rates are shown. The number in front of the algorithm name indicates the number of environments dedicated to resimulation. Note that at the time, 12 parallel environment were used rather than 18. For example, 8G refers to dedicating 8 out of 12 environments to resimulation, which is the same ratio as using 12 out of 18 in the rest of the thesis.

We first focus on the algorithms 8SG FI, 8G FI and 4G FI, the algorithms ran with fixed initial conditions. In table 5.1 we see that from these algorithms, only 8SG FI and 8G FI reliably converged.

This means a majority of the environments need to be dedicated to resimulation. When comparing 8SG FI to 8G FI in figure 5.1a, we see that SG does indeed learn faster than G.

Because of this, one might suspect that we have confirmed that SG outperforms G in terms of speed. However, the performance gap is most likely due to the fact that there are only 4 primary environments with fixed initial conditions. This means that there are only 4 goals. As a consequence, G is also running with only 4 goals on 8 environments. If one adds algorithm S, the number of goals has the potential to go up to 12 because S also draws from the G environments. This increases sample diversity and therefore increases performance.

If one then compares to applying G in two segments (2x4G FI), with similar dataflow to SG, performance increases drastically, as expected. This is shown in figure 5.1b. It reveals that indeed the results in figure 5.1a cannot be trusted. This result led to getting rid of the fixed initial conditions, reverting to the simple randomly resetting environments when episodes finish. Sadly, at this point only 2 months were left until the thesis had to be entirely wrapped up, since figure 5.1 collectively took 2 months to produce.

5.2.2 LSTM considerations

Originally, the aim was not to look at the signed area under the advantage estimate curve for ISBR, but rather at segments with monotonic increases and decreases. The intuition here is that when the agent is closing in on a reward, the agent will have a very good value estimate. This is because the future action sequence until this reward is short. In other words, the agent can easily estimate the rewards that are about to be gained and the amount this reward needs to be discounted. Therefore, if the agent has a large error in the value estimate, this error will decrease when the agent approaches the reward (or absence thereof). Therefore, if we observe a large increase or decrease in the advantage estimate, that tends towards zero, we can speculate that near the beginning of this trend the cause of the trend can be found. This is different from the current method where focus is on the point where the advantage estimates actually crosses zero, but it is similar in that the start of the swing event is situated before the start of the trend towards zero.

However, this approach requires a minimal natural fluctuation in the outputs of the value function, caused by back-to-back matrix multiplications containing constantly changing unconverged parameters. In an attempt to decrease these fluctuations, the goal was to use an LSTM to have the model effectively predict if the value of a state increased or decreased after the previous state, leading to a smoother advantage curve. This also has the pleasant by product of the agent being able to keep track of time, reducing the noise in the value outputs even more. After all, the discounted rewards of short episodic games with high discount factors are highly time dependent. It would even allow for testing on exciting tasks where memory is a prerequisite.

Although the implementation was very complicated, it was possible and everything presented in this thesis also works perfectly with this model that has memory. However, since we are doing episodic reinforcement learning, to be able to perform clean experiments, the hidden state needs to be reset between episodes to really start the agent off again from scratch. If we do not, and just include an indicator in the state indicating that a new episode has started, the value function needs a few transitions to produce to a reasonable estimate again after a new game starts. Ironically, this behaviour is the exact reason why we opted for the LSTM approach in the first place. Sadly, this causes unreasonable increases in the value loss, destabilizing the parameters of the entire network,

including those of the policy network. Simple fixes such as just not including these first few states in the loss do not work because then the problem just gets pushed to further on in the action sequence.

Therefore, the only solution is to keep track of the hidden state when passing batches of data through the network and resetting them accordingly when an episode ends and another begins. However, this means we cannot run a batch using optimized CUDA code, but instead rely on going over the length of the small memory segments manually. This means that instead of running one chunk of code, we now have an additional for-loop with the length of the memory horizon we allow. If we choose a memory horizon of 20 transitions, the computation cost increases more than 3-fold, making this entire endeavour infeasible considering the experiments are run on a laptop and the experiments already take very large amounts of time.

One could argue that taking a small horizon, maybe of 1 or 2, would be enough if the goal was only to get a time-dependence in the value function. One could even suggest to just add the value estimate of the previous state to the next state, as well as the time step and ditch the LSTM altogether and get the exact same gain with an MLP. However, at the time it did not seem like the right thing to temper with the state provided to the agent as it would make comparing to literature harder. It would also prohibit memory-dependent tasks. However, in hindsight, this is exactly what should have been done from the beginning. Memory dependent tasks were already off the table due to increased computation costs, the LSTM would not fix the natural fluctuations in the value function to a high enough degree, the number of transitions until termination ended up getting added to the state as it was required to stabilize the value function, and comparing to literature is not done anyway as one has to write and run their own baselines.

Since the LSTM did not turn out to be a solution to the original problem, it was eventually decided on to change the ISBR method to the one presented in this thesis. However, support for an LSTM was always maintained throughout development such that if the opportunity arose in which it turned out to be useful, it could easily be compared against the results obtained using the MLP. This is why the MLP optimization loop still contains a data-inflation step, assuming a memory horizon of 20 steps in the LSTM implementation. This support is possible because PPO is made for running many optimization steps on the same data without causing problems. Had this data inflation step not been there, the number of optimization epochs would simply have been much higher, and the number of batches per optimization epoch much lower. Therefore, the results are indifferent to having LSTM support, although it may look strange at a first glance.

5.2.3 Insights from experiment 0

In experiment 0, we showed that PPO and ICM, both with and without HER, collapses. Where HER collapsed almost immediately, without HER the algorithm stably learned to do nothing for a while before collapsing. Remarkably, when looking at the resulting behaviour of the agent, both cases look the same. Both are spinning very fast with the tip of the reacher collapsed on the origin. However, the reason for this behaviour is most likely different.

With HER, we see two probable causes, likely amplifying each other. The first we have discussed already in section 3.1.1. As the agent learns, hindsight action sequences become progressively less likely, causing large policy updates, resulting in a collapse of the algorithm. As a second cause, consider the effect of HER early on. The agent will quickly learn to take large actions to get to

the goals it achieved in hindsight. However, if the agent takes large uncontrolled actions, the agent naturally swings the tip of the reacher to the origin. This then becomes a hindsight goal where the agent is told that to reach the origin, large actions must be taken. Now the agent is learning to take large actions, leading to it frequently getting stuck in the center, causing it to get stuck in the positive feedback loop created by HER.

Without HER, the agent learns to do nothing. This means that the agent learns to no longer condition on the state and simply output a normal distribution with 0 mean and a very low variance. If the policy then, through some noise, no longer outputs exactly 0 as the mean of this normal distribution, the agent starts to move very consistently due to the low variance. Because the agent expects no movement at all, the advantages become a slightly negative. After rescaling, they become larger, and sufficiently non-zero to induce a policy update. The variance of a normal distribution is governed by an exponential, so once the variance is very small, it will remain very small. However, the mean will be changed. This creates positive feedback where the agent starts to move in one direction (because it is no longer conditioned on the state), resulting in the observed spinning behaviour.

5.2.4 Future research

Here, we discuss potential future research based on ideas that came up during the entire process.

Variable time

An alternative ISBR algorithm that was on the table is a slight variation of the method in this thesis. It focusses on tasks where the agent has to reach a success state, leading to termination of the environment. An example of such a task is MountainCar.

The method tries to build a more diverse curriculum by also making the task harder. More explicitly, it does so by sometimes reducing the available time to be the exact time required in the previous run to reach the terminal success state. The aim should be to teach the agent more clearly which actions lead to better performance, as well as train the agent to produce better value estimates in edge cases, which should in turn lead to better value estimates and thus more instructive advantage estimates.

Swing state segments

The ISBR methods presented in this thesis rely on identifying swing events, and starting the resimulation at the start of such an event. However, it does not utilize that the event also ends at some point. Instead, it runs until natural termination. One could try to utilize the length of the swing event to do resimulation in a more targeted manner by only simulating for a limited number of steps from the beginning of the swing event. Given that the value estimates should be accurate again after the swing event, one could then run the value function on the last state after the last action during resimulation, discount it and treat it as a reward for the final action. This way, one can simulate swing events in a more targeted manner, possibly freeing up computation for the resimulation of multiple swing events per parallel environment. However, the implementation of this is technically involved since it strays away from the standard flow of data in RL algorithms. Regardless, it could be interesting future research for very long tasks.

Chapter 6

Conclusions

In this thesis, we proposed 2 types of resimulation, GBR and ISBR, and 3 resimulation algorithms, G, S+ and S-. These methods were tested on two tasks, that only differed in the level of sparsity, and 3 reward ratios, controlling for the extent with which the ICM was used. The performance was measured in the solve rate and the learning speed. From these experiments, we found that

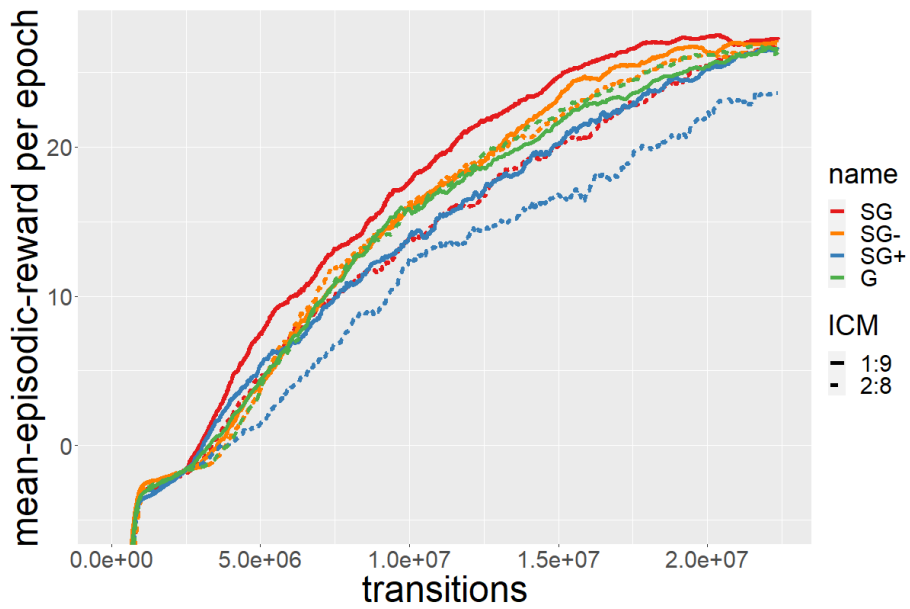
- G enables solving the proposed tasks.
- S+ improves the solve rate, but only on sufficiently sparse tasks and when using ICM.
- S- improves both the solve rate and the learning speed.
- G, S+ and S- can be used in unison to create a better combined algorithm.
- Combining ICM with ISBR methods reduces the learning speed.

Lastly, we argued that, given the unexpected interaction between ICM and ISBR, it is likely that the performance of S+ and S- can be further improved by further investigating this phenomenon.

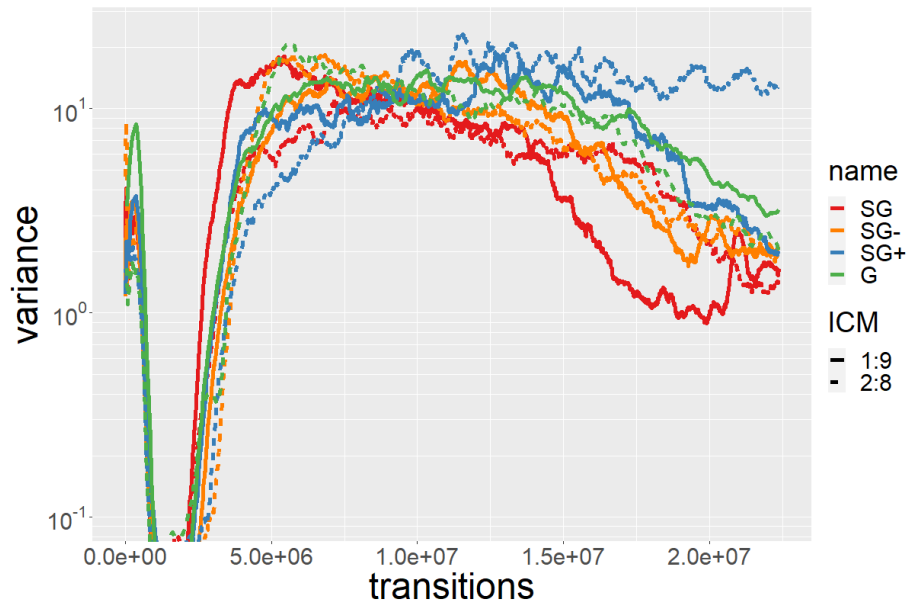
Appendices

Appendix A

Supplementary figures

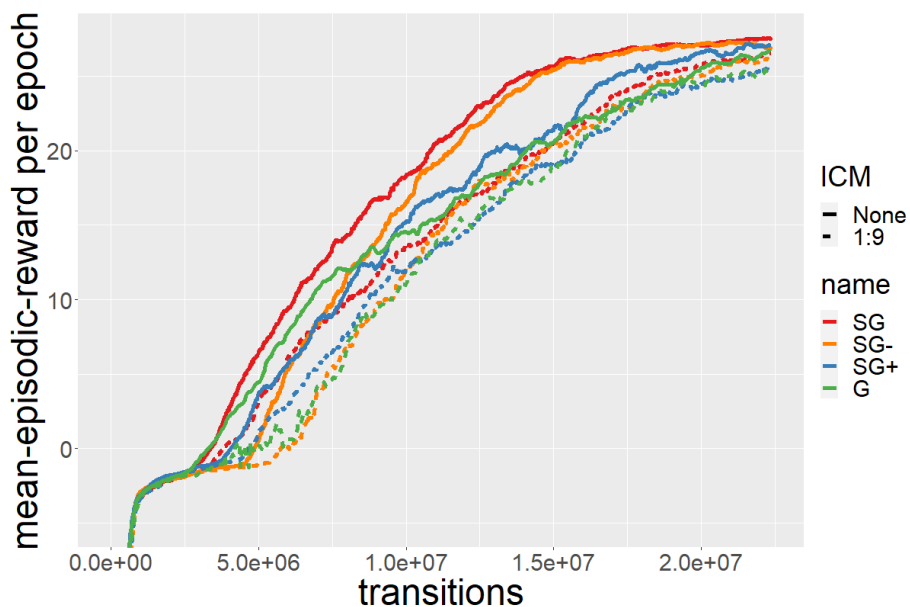


(a) Rewards obtained from experiment 1.

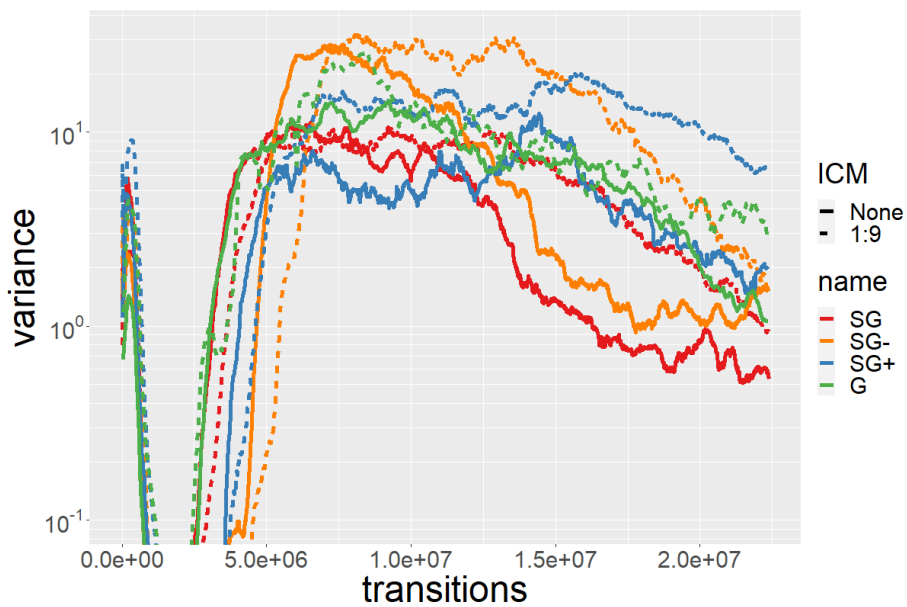


(b) A variance measure of the rewards obtained from experiment 1.

Figure A.1: The smoothed learning (a) and variance (b) curves of experiment 1. It is a supplement to figure 4.5; here we combine all lines into single figures.



(a) The obtained rewards.



(b) A variance measure of the obtained rewards.

Figure A.2: The smoothed learning (a) and variance (b) curves of experiment 2. It is a supplement to figure 4.11; here we combine all lines into single figures.

Bibliography

- [Andrychowicz et al., 2017] Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Abbeel, P., and Zaremba, W. (2017). Hindsight experience replay. *arXiv preprint arXiv:1707.01495*.
- [Asada et al., 1996] Asada, M., Noda, S., Tawaratsumida, S., and Hosoda, K. (1996). Purposive behavior acquisition for a real robot by vision-based reinforcement learning. *Machine learning*, 23(2):279–303.
- [Baker et al., 2019] Baker, B., Kanitscheider, I., Markov, T., Wu, Y., Powell, G., McGrew, B., and Mordatch, I. (2019). Emergent tool use from multi-agent autotutorials. *arXiv preprint arXiv:1909.07528*.
- [Bansal et al., 2017] Bansal, T., Pachocki, J., Sidor, S., Sutskever, I., and Mordatch, I. (2017). Emergent complexity via multi-agent competition. *arXiv preprint arXiv:1710.03748*.
- [Baranes and Oudeyer, 2013] Baranes, A. and Oudeyer, P.-Y. (2013). Active learning of inverse models with intrinsically motivated goal exploration in robots. *Robotics and Autonomous Systems*, 61(1):49–73.
- [Bengio et al., 2009] Bengio, Y., Louradour, J., Collobert, R., and Weston, J. (2009). Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48.
- [Brockman et al., 2016] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). Openai gym. *arXiv preprint arXiv:1606.01540*.
- [Burda et al., 2018] Burda, Y., Edwards, H., Storkey, A., and Klimov, O. (2018). Exploration by random network distillation. *arXiv preprint arXiv:1810.12894*.
- [Degris et al., 2012] Degris, T., Pilarski, P. M., and Sutton, R. S. (2012). Model-free reinforcement learning with continuous action in practice. In *2012 American Control Conference (ACC)*, pages 2177–2182. IEEE.
- [Duan et al., 2016] Duan, Y., Chen, X., Houthoofd, R., Schulman, J., and Abbeel, P. (2016). Benchmarking deep reinforcement learning for continuous control. In *International conference on machine learning*, pages 1329–1338. PMLR.
- [Fang et al., 2019] Fang, M., Zhou, T., Du, Y., Han, L., and Zhang, Z. (2019). Curriculum-guided hindsight experience replay. In Wallach, H., Larochelle, H., Beygelzimer, A., d’Alché-Buc, F.,

- Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.
- [Florensa et al., 2018] Florensa, C., Held, D., Geng, X., and Abbeel, P. (2018). Automatic goal generation for reinforcement learning agents. In *International conference on machine learning*, pages 1515–1528. PMLR.
- [Florensa et al., 2017] Florensa, C., Held, D., Wulfmeier, M., Zhang, M., and Abbeel, P. (2017). Reverse curriculum generation for reinforcement learning. In *Conference on robot learning*, pages 482–495. PMLR.
- [Goodfellow et al., 2014] Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial networks. *arXiv preprint arXiv:1406.2661*.
- [Jabri et al., 2019] Jabri, A., Hsu, K., Eysenbach, B., Gupta, A., Levine, S., and Finn, C. (2019). Unsupervised curricula for visual meta-reinforcement learning. *arXiv preprint arXiv:1912.04226*.
- [Jiang et al., 2019] Jiang, N., Jin, S., and Zhang, C. (2019). Hierarchical automatic curriculum learning: Converting a sparse reward navigation task into dense reward. *Neurocomputing*, 360:265–278.
- [Justesen and Risi, 2018] Justesen, N. and Risi, S. (2018). Automated curriculum learning by rewarding temporally rare events. In *2018 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 1–8. IEEE.
- [Kingma and Ba, 2014] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.
- [Levy et al., 2019] Levy, A., Konidaris, G., Platt, R., and Saenko, K. (2019). Learning multi-level hierarchies with hindsight. In *International Conference on Learning Representations (ICLR)*.
- [Liu et al., 2019] Liu, H., Trott, A., Socher, R., and Xiong, C. (2019). Competitive experience replay. *arXiv preprint arXiv:1902.00528*.
- [Mnih et al., 2016] Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR.
- [Mnih et al., 2013] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.
- [Narvekar et al., 2020] Narvekar, S., Peng, B., Leonetti, M., Sinapov, J., Taylor, M. E., and Stone, P. (2020). Curriculum learning for reinforcement learning domains: A framework and survey. *Journal of Machine Learning Research*, 21(181):1–50.
- [Narvekar et al., 2016] Narvekar, S., Sinapov, J., Leonetti, M., and Stone, P. (2016). Source task creation for curriculum learning. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, pages 566–574.
- [Narvekar et al., 2017] Narvekar, S., Sinapov, J., and Stone, P. (2017). Autonomous task sequencing for customized curriculum design in reinforcement learning. In *IJCAI*, pages 2536–2542.

- [Oh et al., 2018] Oh, J., Guo, Y., Singh, S., and Lee, H. (2018). Self-imitation learning. In *International Conference on Machine Learning*, pages 3878–3887. PMLR.
- [OpenAI et al., 2021] OpenAI, O., Plappert, M., Sampedro, R., Xu, T., Akkaya, I., Kosaraju, V., Welinder, P., D’Sa, R., Petron, A., Pinto, H. P. d. O., et al. (2021). Asymmetric self-play for automatic goal discovery in robotic manipulation. *arXiv preprint arXiv:2101.04882*.
- [Paszke et al., 2019] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In Wallach, H., Larochelle, H., Beygelzimer, A., d’Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.
- [Pathak et al., 2017] Pathak, D., Agrawal, P., Efros, A. A., and Darrell, T. (2017). Curiosity-driven exploration by self-supervised prediction. In *International Conference on Machine Learning*, pages 2778–2787. PMLR.
- [Pedregosa et al., 2011] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.
- [Plappert et al., 2018] Plappert, M., Andrychowicz, M., Ray, A., McGrew, B., Baker, B., Powell, G., Schneider, J., Tobin, J., Chociej, M., Welinder, P., et al. (2018). Multi-goal reinforcement learning: Challenging robotics environments and request for research. *arXiv preprint arXiv:1802.09464*.
- [Racaniere et al., 2019] Racaniere, S., Lampinen, A. K., Santoro, A., Reichert, D. P., Firoiu, V., and Lillicrap, T. P. (2019). Automated curricula through setter-solver interactions. *arXiv preprint arXiv:1909.12892*.
- [Ren et al., 2018] Ren, Z., Dong, D., Li, H., and Chen, C. (2018). Self-paced prioritized curriculum learning with coverage penalty in deep reinforcement learning. *IEEE transactions on neural networks and learning systems*, 29(6):2216–2226.
- [Salimans and Chen, 2018] Salimans, T. and Chen, R. (2018). Learning montezuma’s revenge from a single demonstration. *arXiv preprint arXiv:1812.03381*.
- [Schaul et al., 2015a] Schaul, T., Horgan, D., Gregor, K., and Silver, D. (2015a). Universal value function approximators. In *International conference on machine learning*, pages 1312–1320. PMLR.
- [Schaul et al., 2015b] Schaul, T., Quan, J., Antonoglou, I., and Silver, D. (2015b). Prioritized experience replay. *arXiv preprint arXiv:1511.05952*.
- [Schulman et al., 2017] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.
- [Silva and Costa, 2018] Silva, F. L. D. and Costa, A. H. R. (2018). Object-oriented curriculum generation for reinforcement learning. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pages 1026–1034.

- [Silver et al., 2016] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489.
- [Sukhbaatar et al., 2017] Sukhbaatar, S., Lin, Z., Kostrikov, I., Synnaeve, G., Szlam, A., and Fergus, R. (2017). Intrinsic motivation and automatic curricula via asymmetric self-play. *arXiv preprint arXiv:1703.05407*.
- [Sun et al., 2019] Sun, H., Dai, B., Li, Z., Liu, X., Xu, R., Lin, D., and Zhou, B. (2019). Policy continuation and policy evolution with hindsight inverse dynamics. In *Optimization Foundations for Reinforcement Learning Workshop at NeurIPS*.
- [Synnaeve et al., 2016] Synnaeve, G., Nardelli, N., Auvolat, A., Chintala, S., Lacroix, T., Lin, Z., Richoux, F., and Usunier, N. (2016). Torchcraft: a library for machine learning research on real-time strategy games. *arXiv preprint arXiv:1611.00625*.
- [Todorov et al., 2012] Todorov, E., Erez, T., and Tassa, Y. (2012). Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE.
- [Vinyals et al., 2019] Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., et al. (2019). Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354.
- [Watkins and Dayan, 1992] Watkins, C. J. and Dayan, P. (1992). Q-learning. *Machine learning*, 8(3-4):279–292.
- [Wiering and Van Otterlo, 2012] Wiering, M. and Van Otterlo, M. (2012). Reinforcement learning. *Adaptation, learning, and optimization*, 12(3).
- [Wöhlke et al., 2020] Wöhlke, J., Schmitt, F., and van Hoof, H. (2020). A performance-based start state curriculum framework for reinforcement learning. In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, pages 1503–1511.
- [Wydmuch et al., 2018] Wydmuch, M., Kempka, M., and Jaśkowski, W. (2018). Vizdoom competitions: Playing doom from pixels. *IEEE Transactions on Games*.