Solving *Baba Is You* Using Reinforcement Learning

Orr Bezalely

Bachelor of Science in Computer Science and Mathematics
The University of Bath
2022/23

Solving *Baba Is You* Using Reinforcement Learning

Submitted by: Orr Bezalely

Copyright

Declaration

This dissertation is submitted to the University of Bath in accordance with the requirements of the degree of Bachelor of Science in the Department of Computer Science. No portion of the work in this dissertation has been submitted in support of an application for any other degree or qualification of this or any other university or institution of learning. Except where specifically acknowledged, it is the work of the author.

Abstract

Baba Is You is an award-winning Sokoban-like puzzle game in which the player is required to manipulate the rules of each level to create a winning condition. The human approach to completing this game is thinking in a hierarchy. This involves planning the rule manipulation in the higher levels, and investigating the moves required to achieve the goal in the lower levels. The strategy of solving a problem while planning in a hierarchy is seen in many real-life situations and is thus an important area of research. So far, limited work has been done to apply reinforcement learning to *Baba Is You* and, to the best of our knowledge, none of it involves hierarchical reinforcement learning. This dissertation applies methods from hierarchical reinforcement learning to investigate whether introducing hierarchy into a reinforcement learning agent improves learning in this domain. Our results suggest that a hierarchy is indeed effective for improving the agent's learning, both when handcrafting the subgoals, and when automatically generating the subgoals using different centrality measures.

Contents

List of Figures

List of Tables

Acknowledgements

I would like to thank my supervisor Matthew Hewitt for his endless help and guidance. I have learnt a great deal from him, both regarding the content of reinforcement learning, and also the appropriate writing style required for a dissertation.

I am also grateful to my family for their constant moral support and for the long challenging discussions we had together, often to the small hours of the morning.

Chapter 1

Introduction

1.1 Introduction to the Problem

1.1.1 *Baba Is You* Game

Baba Is You (BIY) is a puzzle game developed for the 2017 Nordic Game Jam by Arvi Teikari, who later expanded it into a full game in 2019 (Teikari, 2019). The game is divided into levels, each played in a grid-world where objects can be pushed around. There are 4 types of objects: Sprites (S) such as Baba, Wall, and Flag; Sprite Words (*SW*) corresponding to each sprite such as *Baba*, *Wall*, and *Flag*; the word *Is*; and Attribute Words (*AW*) such as *You*, *Win*, and *Stop*. The first level of the game can be seen in Figure 1.1.

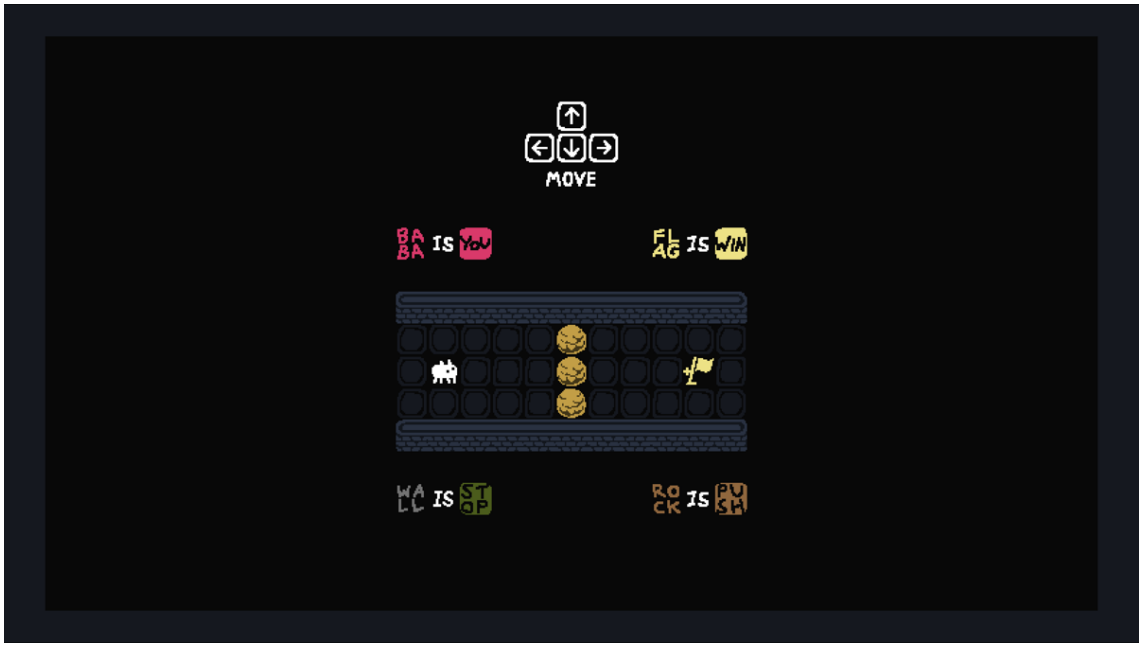

Figure 1.1: Baba Is You - Level 00 (Teikari, 2019), screenshot taken directly from the original game.

The word objects (*SW*, *Is*, *AW*) can be combined together into sentences by pushing them next to each other either left-to-right or up-to-down. Sentences which follow the structure

of *SW-Is-AW* or *SW-Is-SW* form rules, which take place in the level. For example, the rule *Baba Is You* means that the player has control of all Baba sprites, and the rule *Baba Is Flag* means that all Baba sprites are converted into Flag sprites. The player must manipulate the rules which take place in a given level in order to reach a winning state, namely one where the player controls a sprite which is in the same grid cell as a sprite which has the *Win* attribute. Figure 1.2 includes annotations of the first level for readers to familiarise themselves with the game and its features.

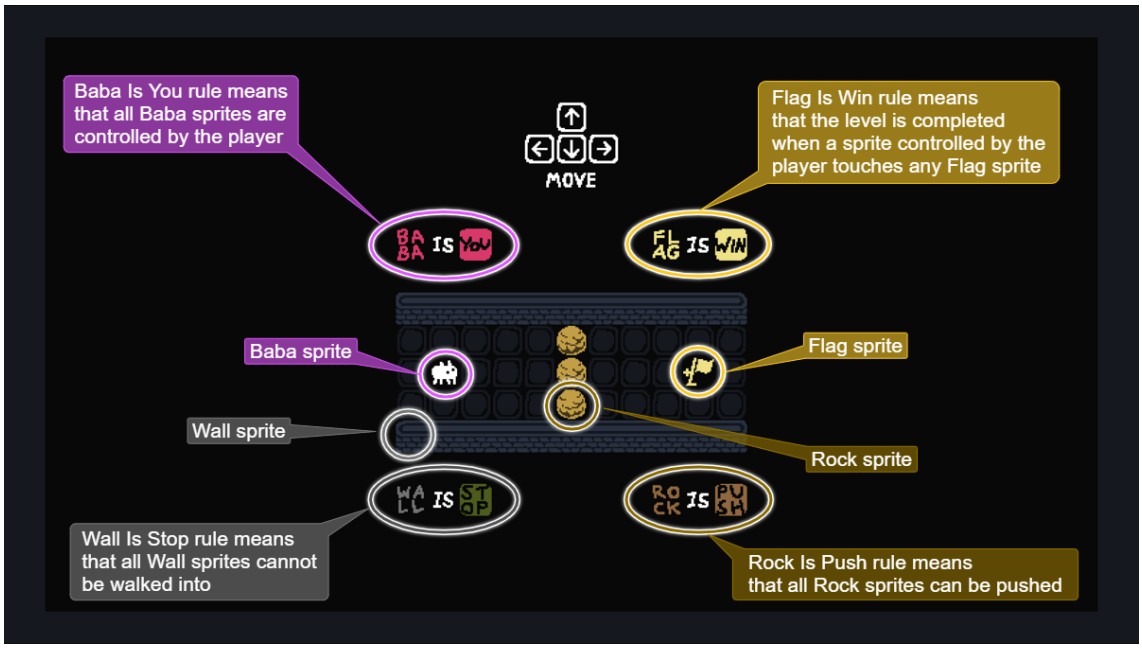

Figure 1.2: Baba Is You - Level 00 (Teikari, 2019), screenshot annotated by author to identify the features of the level.

1.1.2 Context to the Problem

BIY is an example of a Sequential Decision Problem (SDP). An SDP is a type of optimisation problem (Diederich, 2001) in which, according to Şimşek, Algorta and Kothiyal (2016), "a successful outcome depends not on a single decision but on a series of related decisions". The problem can be modelled using two components, a decision-maker and an environment. The environment responds with different rewards for each sequence of actions carried out, and the decision-maker is tasked with observing the environment and thus figuring out the optimal sequence of actions which yields the highest rewards. Each SDP can have various properties. For example, an SDP might be long-horizon, meaning that the number of decisions that the decision-maker needs to make is large. Another possible property of an SDP is that the environment will rarely give out positive rewards. This is formally known as a sparse reward environment. An SDP might also contain bottleneck states which, according to Şimşek and Barto (2004), are "states that allow transitions to a different part of the environment". By learning how to solve simpler SDPs, like BIY, we could extrapolate important strategies which could then be applied to solving more complex real-life environments.

1.2 Why the Problem is Worth Studying

Many real-life problems can be modelled as SDPs, and understanding how to solve SDPs would enable solving such problems. For example, given a text input, understanding it and generating an appropriate response is an open problem in the Natural Language Processing field. The currently-trending ChatGPT was fine-tuned using a reinforcement learning algorithm called Proximal Policy Optimisation (Schulman et al., 2017) to solve this problem, and currently it "outperforms both previous large language models and most state-of-the-art systems" (OpenAI, 2023). Thus, understanding how to solve SDPs has real-life applications, and for this reason it is a relevant area of investigation.

1.3 Structure of the Dissertation

The dissertation starts by briefly stating its aim, and the objectives required to achieve it. This is followed by a literature review, covering the background knowledge relevant to our research. Afterwards, a specific BIY level is created, and the relevant methods reported in the literature review are identified and experimented on. The experiments include hypotheses, and the results are analysed to check the validity of these hypotheses. Lastly, conclusions are drawn indicating possible paths for future work on the topic.

Chapter 2

Aims and Objectives

2.1 Aims

As stated above, many real-life problems can be modelled as SDPs. Finding the best actions to carry out in a given state is an active research area in Reinforcement Learning (RL). The aim of this dissertation is to explore the effect of introducing hierarchy into an RL agent when solving a single decision-maker, long-horizon, sparse reward SDP containing bottleneck states. This is done by comparing the performance of a non-hierarchical agent (with only primitive actions available) against a similar agent containing a hierarchy (having both primitive actions and additional temporally extended actions available) in a level from BIY displaying the above criteria.

2.2 Objectives

1. Identifying a level from the BIY environment having the properties of a single agent, long-horizon, sparse reward SDP containing bottleneck states.
2. Deciding on an appropriate pair of agents which have the same base algorithm, the difference being that one of them (the hierarchical agent) allows for the use of temporally extended actions while the other (the non-hierarchical agent) does not.
3. Handcrafting subgoals for the chosen level and deciding on the automatic methods of generating subgoals for the hierarchical agent. For each of the automatic methods, we further identify the subgoal states generated for the chosen level.
4. Implementing and comparing both agents described in point 2 with each instance of the hierarchical agent containing the subgoal states generated by one of the methods in point 3.

Chapter 3

Literature Review

3.1 Reinforcement Learning

3.1.1 Agent-Environment Interaction

Reinforcement learning is a branch of machine learning which tackles an SDP by finding actions which maximise rewards over the long term in an environment through trial and error.

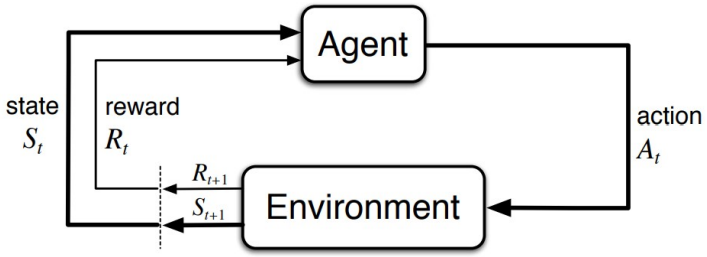

Figure 3.1: Agent-environment interaction, taken from page 48 in Sutton and Barto (2018)

There are two main components in RL: an agent and an environment. The agent is the entity that makes decisions regarding which actions to take, and the environment is the entity with which the agent interacts and receives feedback from. The process occurs in discrete time steps $t = 0, 1, ...$

In each time step t, the agent receives a reward r_t and its current state s_t in the environment. The agent then supplies an action a_t to be taken from its policy π — a mapping from states to a probability distribution of actions to be taken.

This process is repeated to produce a sequence $\tau = (s_0, a_0, r_1, s_1, a_1, r_2, ...)$ called a trajectory. The agent is tasked with improving its policy via repeated interaction with the environment, the goal being to find an optimal policy π^* — a policy which maximises a particular objective function. The most commonly used objective function is the total discounted return (Sutton and Barto, 2018):

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{1}$$

where $\gamma \in [0, 1]$ is a constant called the discount factor.

If the next state-reward pair of an environment depends only on the current state and action, which is mathematically represented in Sutton and Barto (2018) as

$$P(s_{t+1}, r_{t+1} \mid s_0, a_0, s_1, a_1, \dots, s_t, a_t) = P(s_{t+1}, r_{t+1} | s_t, a_t) \tag{2}$$

for all t, then the SDP is said to have the Markov property and can thus be described as a Markov Decision Process (MDP) (Sutton and Barto, 2018).

3.1.2 MDP Definition

An MDP (Ross, 1983) is a mathematical model that can be described as a tuple of 4 elements $< S, A, T, R >$ where:

1. S is the state-space, containing every possible state in the environment.
2. A is the action space, containing every possible action in the environment with $A(s) \subseteq A$ being the possible actions that can be taken in a given state $s \in S$.
3. T is the transition model, containing the probability distribution $p(s'|s, a)$ for $s, s' \in S$, $a \in A(s)$.
4. R is the reward model, containing the reward r given for transitioning from $s \in S$ to $s' \in S$ using action $a \in A(s)$.

3.1.3 Value Functions

Given a state $s \in S$, to decide which action would maximise the objective function G_t, the action with the highest expected return needs to be determined. This involves predicting which actions will be chosen in future time-steps, and hence requires having a policy π. Thus, the value of action a in state s following policy π is given by Sutton and Barto (2018) as:

$$Q_\pi(s, a) = \mathbb{E}_\pi [G_t \mid s_t = s, a_t = a] \tag{3}$$

A similar definition for the value of state s following policy π (Sutton and Barto, 2018) is:

$$V_\pi(s) = \mathbb{E}_\pi [G_t \mid s_t = s] \tag{4}$$

Tabular methods approximate the optimal policy by iteratively updating either the Q-values $Q(s, a)$ for every state-action pair or the state-values $V(s)$ for every state. Hence, the function that returns the Q-values or the state-values is a lookup of a dictionary comprising (s, a) keys and $Q(s, a)$ values, or s keys and $V(s)$ values respectively.

Many tabular algorithms exist, such as Value Iteration (Puterman and Shin, 1978), Monte Carlo First Visit (Singh, Sutton and Kaelbling, 1995), and SARSA (Rummery and Niranjan, 1994). A particularly popular tabular algorithm is Q-learning (Watkins, 1989). In Q-learning, actions are chosen uniformly at random with $\varepsilon \in [0, 1]$ probability and are chosen greedily with $1 - \varepsilon$ probability. If chosen greedily, then (Watkins, 1989):

$$a_t = \pi(s_t) = \text{argmax}_{a \in A(s_t)} Q(s_t, a) \tag{5}$$

Furthermore, $Q(s_t, a_t)$ is calculated in an iterative process using the update rule (Watkins, 1989):

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha(r_{t+1} + \gamma \max_{a \in A(s_{t+1})} Q(s_{t+1}, a) - Q(s_t, a_t)) \quad (6)$$

where $\alpha \in [0, 1]$ is a constant called the learning rate, and $\gamma \in [0, 1]$ is the discount factor.

3.2 Hierarchical Reinforcement Learning

Hierarchical Reinforcement Learning (HRL) is an active research sub-field in RL which "enables autonomous decomposition of challenging long-horizon decision-making tasks into simpler subtasks" (Pateria et al., 2021). The agent first temporally abstracts the problem, where they "attempt to find a time scale that is adequate for describing the actions of an AI system" (Precup, 2000). The assigned task is then divided into simpler subtasks using the detected time scale, where each subtask is viewed as its own problem. This process continues recursively until each subtask at the lowest level is a short-horizon problem, which can be solved using primitive actions. Once the simpler subtasks are solved, the more difficult ones are then solved using the solutions to the simpler subtasks, which last over multiple time-steps and are called temporally extended actions.

Theoretically, HRL should have great success in solving long-horizon problems with sparse rewards and bottleneck states. This is due to multiple reasons. Firstly, HRL allows the rewarding of temporally extended actions instead of only primitive actions, and so manages to combat the credit assignment problem — the issue where the decision-maker might struggle to identify which sub-sequence of actions from a long sequence has led to a positive reward being received. It does so by assigning rewards to a sequence of actions rather than to each action individually. In addition, HRL abstracts detail on higher levels, and thus reduces the number of decisions that need to be made in these levels. This means that the rewards received further along in the sequence are propagated back to earlier states more rapidly. Thirdly, by abstracting away details which are problem-specific, an HRL agent could transfer generalisable skills learnt only once to multiple environments.

Empirically, there are long-horizon sparse reward problems that have been solved using HRL agents, which non-hierarchical agents struggled to solve. For example, the h-DQN algorithm (Kulkarni et al., 2016) managed to show learning in the Atari game Montezuma's Revenge, which can be found on OpenAI Gym (Brockman et al., 2016), whereas its non-hierarchical counterpart DQN (Mnih et al., 2015) showed no learning progress after many time-steps.

Since temporally extended actions last for multiple time-steps, we cannot use an MDP and must transfer to a discrete-time Semi Markov Decision Process (SMDP) (Younes and Simmons, 2004).

3.2.1 SMDP Definition

A discrete-time SMDP (Bradtke and Duff, 1994) is a mathematical model that can be described as a tuple of 5 elements $< S, A, T, R, \tau >$ (Barto and Mahadevan, 2003) where:

1. S, A, and R are the same as in the definition of an MDP (see Section 3.1.2).
2. τ is the run time (in smallest time scale) of state $s \in S$ when a temporally extended action $a \in A$ is executed.

3. T is the transition model containing the probability distribution $p(s'|(\tau|s), a)$ for $s, s' \in S$ $a \in A$, τ as above.

3.2.2 HRL Frameworks

Many frameworks can be used to generate temporally extended actions. Some examples are MAXQ (Dietterich, 1999), HEXQ (Hengst, 2002), Feudal RL (Dayan and Hinton, 1992), and Hierarchies of Abstract Machines (HAMs) (Parr and Russell, 1998), but by far the most common framework is Options (Sutton, Precup and Singh, 1999).

The Options framework allows temporally extended actions called options to be added to the action set A available to the agent (Barto and Mahadevan, 2003).

Options are represented as a tuple of 3 elements $< I, \pi, \beta >$ where:

1. I is the set of states from which a temporally extended action can begin.
2. π is a policy according to which the temporally extended action behaves by.
3. β is the termination condition of the temporally extended action, which is generally determined stochastically.

Once the options are created, they are then added to the action set A and can be initiated precisely in the states contained in I and terminate when determined by β.

3.2.3 SMDP Q-Learning and Intra-Option Q-Learning Agent with Fixed Options

Sutton, Precup and Singh (1999) describe an agent, henceforth to be called the options agent, that is given fixed options defined using primitive actions, and that uses Q-learning to find an optimal policy across the fixed options. It does so using a two-level hierarchical agent. The lower level is an option which, given a state, returns a primitive action from its fixed policy. The higher level is the agent which, given a state, returns an option from the set of available options — those which contain the given state in their initiation set. The chosen option then determines the next action(s) to take until the option is terminated. The higher level chooses an option by a similar method to the Q-learning agent: approximating the Q-values of every state-option pair as $Q(s, o)$, choosing randomly with $\varepsilon \in [0, 1]$ probability and choosing greedily with $1 - \varepsilon$ probability.

The options agent performs two different types of iterative updates, namely SMDP Q-learning and intra-option Q-learning.

SMDP Q-Learning

SMDP Q-learning is used when an option that lasts more than one time-step is terminated and its Q-value needs to be updated. To do this, the state in which the option was initiated originally is maintained, along with the rewards received every time-step until termination. Then, the following formula is used as the update rule (Sutton, Precup and Singh, 1999):

$$Q(s_t, o) = Q(s_t, o) + \alpha \left[\left(\sum_{k=0}^{\ell-1} \gamma^k r_{t+1+k} \right) + \gamma^\ell \left(\max_{o' \in \text{Available Options}} Q(s_{t+1+\ell}, o') \right) - Q(s_t, o) \right] \tag{7}$$

Intra-Option Q-Learning

Intra-option Q-learning is used after every time-step to update multiple options. The update rule of normal Q-learning relies on the action that was executed. Thus, if multiple options choose to execute the same action, it is irrelevant which option executes the action itself. Therefore, each consistent option can be looped over in order to update its Q-value. This update rule (Sutton, Precup and Singh, 1999) is:

$$Q(s_t, o) = Q(s_t, o) + \alpha \left(r_{t+1} + \gamma U(s_{t+1}, o) - Q(s_t, o) \right) \tag{8}$$

where

$$U(s, o) = (1 - \beta(s)) Q(s, o) + \beta(s) \max_{o' \in \text{Available Options}} Q(s, o') \tag{9}$$

where $\beta(s)$ is the probability that option o terminates in state s.

3.3 Graph-Based Methods for Discovering Subgoals

In order to produce fixed options, relevant subgoals which the options will represent must first be identified. One method is for a human expert in the field to analyse the environment and identify the important subgoals. However, this process can also be automated. A method in HRL which aims to automate the process of identifying important subgoals is the graph-based approach for subgoal discovery (Mendonça, Ziviani and Barreto, 2019), falling under the independent subtask discovery in Figure 3.2.

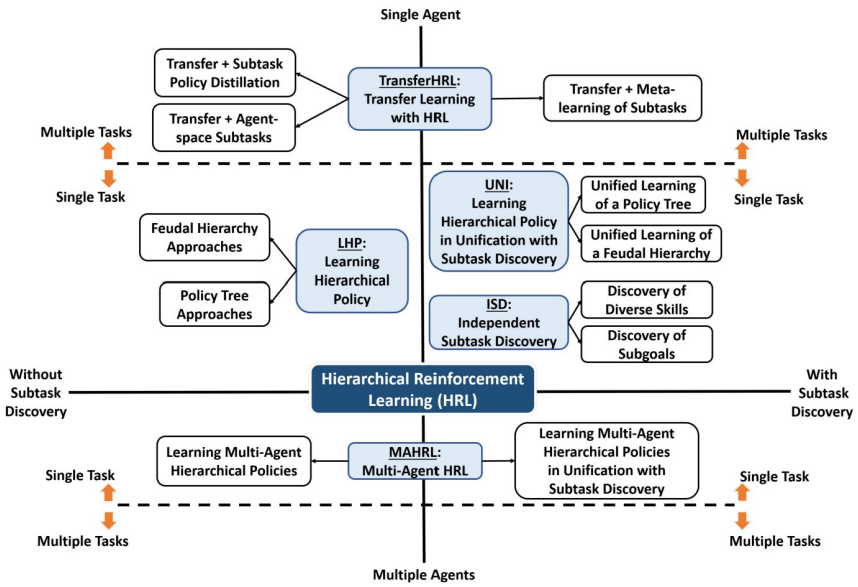

Figure 3.2: Different sub-fields in hierarchical reinforcement learning and their approaches, taken from page 8 in Pateria et al. (2021).

The class of graph-based approaches uses a graph representation of the MDP, called the State Transition Graph (STG), to find properties about the MDP that can be used to build temporally extended actions. The STG (Mendonça, Ziviani and Barreto, 2019) represents an MDP with nodes representing the states and edges representing the transitions and rewards.

Three common approaches in the literature divide into centrality, clustering and spectral analysis. Our focus is on the centrality approach group.

3.3.1 Centrality

Centrality measures are maps from nodes to values representing the importance of a given node. The higher the value, the more important the node is using a given measure. Given a directed graph $G = (V, E)$, some examples include:

1. Degree centrality, first introduced conceptually by Shaw (1954) and more formally defined by Nieminen (1974) and Freeman (1978) – Measures the number of nodes that are connected to a given node. Mathematically, this is the sum of the number of edges going into the node plus the number of edges going out of the node (Freeman, 1978):

$$d(v) = d_G(v) + d_{G'}(v) \tag{10}$$

   where $G' = (V, E')$ is the original graph with the edges reversed.

2. Closeness centrality, first introduced conceptually by Bavelas (1948) and more formally defined by Sabidussi (1966) and Freeman (1978) – Measures the inverse mean distance from a given node to all other nodes. Mathematically (Freeman, 1978), this is:

$$c(v) = \frac{|V| - 1}{\sum_{w \in V} \text{dist}(v, w)} \tag{11}$$

   where $\text{dist}(v, w)$ is the length of the shortest path between v and w.

3. Betweenness centrality, first introduced conceptually by Shaw (1954) and more formally defined by Freeman (1978) – For a given node, it measures the sum of the fraction of the shortest paths between any two nodes that go through the given node. Mathematically (Freeman, 1978), this is:

$$b(v) = \sum_{s \neq v \neq u \in V} \frac{\sigma_{su}(v)}{\sigma_{su}} \tag{12}$$

   with $\sigma_{su}(v)$ being the number of shortest paths between s and u passing through v and σ_{su} being the total number of shortest paths between s and u.

   A variation of this measure, more appropriate to RL, was introduced by Şimşek and Barto (2008), where weights are added to each part of the sum to account for the reward function. Mathematically (Şimşek and Barto, 2008), this is:

$$b(v) = \sum_{s \neq v \neq u \in V} \frac{\sigma_{su}(v)}{\sigma_{su}} w_{su} \tag{13}$$

   with w_{su} being the weight assigned to paths from s to u.

Many examples of other centrality measures can be found in Das, Samanta and Pal (2018).

Chapter 4

Methodology

4.1 Choice of Environment

4.1.1 Baba Is Y'all and Keke AI Competition

Baba Is Y'all (BIYa) (Charity, Khalifa and Togelius, 2020) is a collaborative mixed-initiative system based on the game BIY. It allows users to build custom levels with the rules described for BIY, found in Section 1.1.1. This system was utilised to create an AI competition, called Keke AI Competition (KAIC) (Charity and Togelius, 2022), where coders attempt to develop agents that solve various BIY levels devised by the hosts of the competition. In KAIC, states are represented as a combination of the following:

1. ASCII representation of the map.
2. List of active rules.
3. List of non-interactive objects.
4. List of word sprites.
5. Lists of sprite objects that can be assigned by attribute according to point 2.

This representation of states is useful for creating our own agents later on.

4.1.2 Suitability of the KAIC Environment

We are looking for a level having the properties of a single decision-maker, long-horizon, sparse reward SDP containing bottleneck states. Using the KAIC environment, one can create levels which have the required features. In particular:

1. BIY is a single-player game which indicates that any level in the KAIC environment is a single decision-maker environment. Despite actions having the ability to control multiple sprites, they all have access to the same information and thus do not fall under the definition of multiple agents given by Panait and Luke (2005).
2. BIY levels vary in solution lengths. If KAIC is used to create a level which has a lengthy minimum solution, then the level will have the long-horizon property.

3. For a given level in BIY, the player is rewarded only for completing the level, and is punished only for reaching a terminal non-winning state — one where the player does not control any objects. We suggest a reward system which evolves naturally from BIY involving a large positive reward of 200 when a winning state is reached, and a large negative reward of -200 when a terminal non-winning state is reached. Additionally, as we wish the level to be solved in as few moves as possible, a small negative reward of -1 is allocated for each move made. The reward system suggested above ensures that we have a sparse reward system since the decision-maker is only rewarded with positive rewards when reaching a goal state (which is a small percentage of the states on the average level).
4. BIY levels, and hence KAIC levels, contain natural bottleneck states when the decision-maker creates/destroys sentences. For example, until there is no sentence involving the *Win* word, there is no way for the decision-maker to complete the level by reaching a winning state, no matter where on the board they are located. Once the sentence is formed, it unlocks access to states in which the decision-maker can win.

Thus, we use KAIC to create a level which has the above required properties. To the best of our knowledge, this is the second piece of literature that applies RL to solve any BIY environment, the first being Charity and Togelius (2022) which is more focused on the competition structure than on the agents themselves.

4.1.3 Level Creation

We take inspiration from the two-room gridworld environment in Şimşek and Barto (2004) to create our level for the following experiments. The two-room gridworld environment is a map with the layout of two rooms connected together by one passageway (this can be thought of as two rooms connected by a door). That way, if any entity wishes to move from one room to another, it will be forced to pass through the passageway. In this way, we guarantee that there is at least one bottleneck state as the passageway is the only method of moving between the rooms. To add to the complexity of the two-room gridworld environment, and to introduce the rule-manipulation concept from the BIY environment, we forcethe agent to push a word from one room to the other in order to create a win condition.

Figure 4.1: Augmented two-room environment, created by author using Baba Is Y'all (Charity, Khalifa and Togelius, 2020).

As seen in Figure 4.1, this level has multiple bottleneck states, including when the agent tries to push the *Win* word through the passageway from the left room to the right, and when the agent attempts to create a win condition by composing the sentence *Flag Is Win*. Furthermore, the level's shortest possible solution requires the agent to manoeuvre the *Win* word across the passageway and into the grid-cell containing the *Stop* word, and then to walk all the way back to the Flag in the first room, which takes 39 time-steps, making it a relatively long-horizon problem, for the sake of our experiments.

Thus, this level is suitable as a single decision-maker, long-horizon, sparse-reward SDP containing bottleneck states.

4.2 Formalising the Problem as an MDP

As the next state of BIY, and hence KAIC, is produced only by the current state and action taken, the environment has the Markov property and hence can be described as an MDP.

Thus, for the given level in KAIC, the MDP $< S, A, T, R >$ (see Section 3.1.2) comprises the following:

1. The state-space S consisting of all possible states in the KAIC representation of the chosen level.
2. The action-space $A = \{\text{north, south, east, west}\}$ with $A(s) = A$ for all $s \in S$.
3. The transition model T of the deterministic environment containing

$$p(s'|s,a) = \begin{cases} 1 & \text{if KAIC nextMove function returns } s' \text{ from state } s \text{ taking action } a \\ 0 & \text{otherwise} \end{cases}$$

4. The reward model R with $r = R(s,a,s') = \begin{cases} 200 & \text{if } s' \text{ is a winning terminal state} \\ -200 & \text{if } s' \text{ is a losing terminal state} \\ -1 & \text{otherwise} \end{cases}$

4.3 Choice of Agents

In this dissertation, we carry out small-scale experiments to show a proof-of-concept, prompting larger experiments for future work. We therefore limit our scope to tabular algorithms.

4.3.1 Choice of Non-Hierarchical Agent

The Q-learning algorithm is popular amongst the other tabular algorithms for multiple reasons. Firstly, Q-learning bootstraps (compared to Monte Carlo methods which do not), meaning that estimates are updated based on estimates from previous time-steps in the same episode. This allows Q-learning to learn quicker than methods which do not bootstrap. Q-learning is also off-policy (compared to SARSA which is on-policy). In particular, the Q-learning agent learns about a different policy from the one it is following, which allows for exploration of the environment without accounting for exploration in the learnt policy. This allows Q-learning to guarantee convergence to an optimal policy when the number of times that each state is visited tends to infinity (Watkins and Dayan, 1992). Additionally, perfect knowledge of the environment's dynamics is not always available; Q-learning does not require perfect knowledge (compared to value iteration which does).

For the above reasons, we are choosing Q-learning as the non-hierarchical agent for our experiments.

4.3.2 Choice of Hierarchical Agent

Having chosen the base algorithm, we now need to find an agent that when being run without temporally extended actions results in the Q-learning agent. The Options framework allows for the introduction of hierarchy to agents in a way that when the hierarchy is not used, the resulting agent is the same as the base agent with no hierarchy. For the specific case of Q-learning, using the options agent described in Section 3.2.3, the Options framework represents each primitive action as an option $< I, \pi, \beta >$ where I contains every state (in other words the option can be initiated in every state), the policy π returns the primitive action chosen for any given state, and the termination probability β is 1 for every state. Thus, there is a one-to-one correspondence between primitive options and primitive actions. The further added options allow the options agent to pick an action from a wider range of options. However, once the non-primitive options are removed, what remains is a Q-learning agent in disguise.

We are therefore choosing the options agent to be the hierarchical counterpart of our non-hierarchical Q-learning agent.

4.4 Choice of Fixed Options for Hierarchical Agent

Having chosen our hierarchical agent to be the options agent, we now need to choose the additional fixed options that the agent would receive. This process is divided into two. The first part involves identifying important subgoals, and the second involves creating fixed options for each subgoal identified.

4.4.1 Choice of Subgoals

We define a subgoal as a desirable set of states to be in. One method to identify subgoals is to handcraft them. This involves a human expert with deep understanding of the environment to identify important subgoals. As described in Section 3.3, another method of identifying important subgoals is to represent the MDP of the problem as an STG and to use graph-based approaches to identify important sets of nodes which are associated with important sets of states. We focus on the centrality approach because it is a common method in mathematical literature used on graphs to identify central nodes (Das, Samanta and Pal, 2018).

The specific centrality measures we are choosing to use are the degree, closeness, and betweenness centralities. We chose these centralities for two main reasons. Firstly, they are the most basic centrality measures in mathematical literature. This is because they are the three centralities discussed in Freeman (1978), one of the most foundational papers about different centrality measures. A second reason we chose these is that many centrality graph-based approaches for creating subgoals in RL are based on these specific centrality measures, as suggested by Mendonça, Ziviani and Barreto (2019). We are also including one adapted variation from RL, namely the variation to the betweenness centrality of Şimşek and Barto (2008), which can be found in Section 3.3.1, as another method of subgoal generation. The centrality measures chosen require us to create a global STG of the MDP and run each centrality measure to get a value for each node, representing its importance using the given centrality measure.

Following that, there are multiple ways of deciding how to create subgoals. Three intuitive ones are as follows:

1. Choosing a fixed number of nodes (e.g. 10), which have the highest values out of all nodes, to be the subgoals.
2. Choosing precisely those nodes which have a higher value than some threshold value (e.g. for degree centrality if there are more than 6 neighbours) to be the subgoals.
3. Choosing precisely those nodes which have local maxima values to be the subgoals. These are the nodes which have the highest values in their neighbourhood (by neighbourhood we mean direct neighbours, which are one node away — but this can be further expanded).

We choose to apply the third method since it covers most clusters of nodes in the STG, whereas the first and second methods do not necessarily do so.

4.4.2 Creating Fixed Options from Subgoals

Having subgoals, we now need to determine how to create an option that achieves those subgoal. This involves defining 3 elements for each subgoal: the initiation set, the policy, and

the termination condition. The combination of the three definitions allows the creation of an option $< I, \pi, \beta >$ which is well-defined, and can be fed into our options agent.

Initiation Set

Given a set of states representing a subgoal, we could define the initiation set as any subset of the set of states that are firstly not part of the subgoal set, and secondly that can reach at least one of the subgoal states after some number of moves. This is because any state which cannot reach all subgoal states is not useful to achieve the required subgoal, and any state which is in the subgoal set already does achieve it.

To reduce the initiation set further, one could consider only states which are closer to the subgoal set, measured as some number of moves away. However, we use the largest possible initiation set as we wish for the options to be utilised more frequently.

Policy

To find the optimal policy, one could start by defining an internal reward system which gives a large positive reward for reaching any of the subgoal states, and a small negative reward for every move which does not achieve that. Using the primitive actions available at state s as the possible actions, and the same transition model, one could attain a well-defined MDP, and thus could use most RL algorithms to find the optimal policy for the MDP.

For our purposes, we do not need to create an RL agent to solve this subtask, but instead we use the STG to define the policy in a way that actions are taken greedily. This means that the action chosen by the policy will follow the shortest path to reach the closest state in the subgoal set.

Termination Condition

A natural way to define the termination condition deterministically is to terminate with probability of 1 for any state not in the initiation set, and with probability 0 for any state that is in the termination set. Using this termination condition and the above policy, which is optimal, we ensure that the agent will terminate in a subgoal state.

4.5 Formalising the Problem as an SMDP

Having access to the fixed options, we can now formulate the problem as an SMDP $< S, A, T, R, \tau >$ (see Section 3.2.1) which comprises the following:

1. The state-space S is the same as in the MDP in Section 4.2.
2. The action-space A has every primitive option corresponding to a primitive action in the MDP (north, south, east, west) and the further fixed options found, with $A(s)$ containing all options whose initiation set contains state $s \in S$.
3. The run time τ of a given option is the number of time-steps since the option was initiated.

4. The transition model $p(s' \mid (\tau \mid s), o)$ for $s, s' \in S$, $o \in A$, τ as above, evaluates to the same transition model used in the MDP due to option o deciding on a primitive action to take using $(\tau \mid s)$.
5. The reward model is the same as in the MDP in Section 4.2.

Chapter 5

Experiment 1

In this experiment, we wish to test whether introducing hierarchy to an RL agent could have a positive effect in speeding up the learning process when solving a single decision-maker, long-horizon, sparse-reward SDP containing bottleneck states. To test this, we will use the level created in Section 4.1.3 as our environment, and then follow the steps below:

1. Create handcrafted subgoals from the chosen level.
2. Create fixed options from the handcrafted subgoals and supply them to the handcrafted options agent.
3. Run both the Q-learning agent and the handcrafted options agent on the chosen level.
4. Analyse the results produced by the experimentation.

Thus, the independent variable is whether or not the agent is introduced to hierarchy; the dependent variable is the value of rewards received for each time-step; and the control variables are the following:

1. The level created on which to run the experiment (found in Section 4.1.3), which determines the SDP.
2. The base algorithm which the agents use, which has already been described in Section 4.3, namely Q-learning.

Intuitively, introducing the agent to useful options should decrease the number of episodes taken for the agent to learn since this increases the chances that the agent will pick an option that brings it closer to its goal. Conversely, introducing the agent to options which are not useful should increase the number of episodes taken for the agent to learn since this increases the chances that the agent will pick an option that brings it further away from its goal.

5.1 Hypothesis

H0: In the chosen level, the handcrafted options agent will have similar or worse performance — measured by the average number of episodes (averaged over 100 trials) to achieve a score of over 100 — compared to the normal Q-learning agent.

H1: In the chosen level, the handcrafted options agent will have better performance — measured by the average number of episodes (averaged over 100 trials) to achieve a score of

over 100 — compared to the normal Q-learning agent.

We will average over 100 trials to avoid any one-off outlier trial from affecting the entire analysis. This increases the precision of the results and ensures our experiment and its results are more repeatable.

5.2 Choice of Handcrafted Subgoals

The approach taken in which we problem-solve this level as humans is as follows. There is only one goal to begin with, namely reaching a state where the player controls a sprite which is in the same grid cell as a sprite which has the *Win* attribute. Looking at this level, we first note that we are currently controlling the Baba sprite since the rule *Baba Is You* is active. We will now analyse which objects are good candidates to have the *Win* attribute. There are 3 possible sprite word candidates: *Baba*, *Wall*, and *Flag*. However, both *Baba* and *Wall* are not accessible to push around as they are surrounded by wall sprites, and the rule *Wall Is Stop* is active. Hence, these are not viable candidates.

Thus, the remaining possibility, *Flag*, is the object which will need to have the *Win* attribute. As there are wall sprites below, above, and to the left of the *Flag* word, the rule must be created left-to-right, with the *Flag* word already being in its current location. Thus, we must manoeuvre the *Win* word to where the *Stop* word is currently located in order to disable the rule *Flag Is Stop* and instead enable the rule *Flag Is Win*. From there, we will need to reach the Flag sprite. Thus, the subgoal set that we handcrafted contains the states where the *Flag Is Win* rule is active.

We use the same method of creating the initiation set, policy, and termination condition as described in Section 4.4.2, namely, the initiation set is the set of states which does not contain the *Flag Is Win* rule, but which can reach a state containing the rule; the policy returns the action required to follow the shortest path to reach the closest state in the subgoal set; and the termination condition has probability 1 for any state in the initiation set, and 0 otherwise.

5.3 Experiment Details

Both agents have the following hyperparameter settings: $\varepsilon = 0.1$, $\alpha = 0.2$, and $\gamma = 0.9$ (see Section 3.1.3). The choice of ε was a balance between the need to be high enough for random actions to be taken in order to allow for exploration at the beginning, but low enough so that states which the agents deduce to be more useful are exploited more frequently near the end. The choice of α is also a balance, and needs to be high enough so that learning is not too slow (as we only have limited computational power), but low enough so that the agent will be able to approximate the value of a state-action pair more precisely. The choice of γ matters more in environments with a more dense reward system, but does not have much effect in our level of choice. The choice of γ does not matter in our level since the environment returns the same reward of -1 for any move which results in a non-terminal state, and the moves which result in the terminal state receive a fixed reward of 200. So although changing the value of γ will change the value of a state-action pair, it will always maintain the same ranking compared to other state-action pairs from the same state.

In addition, both agents are given a maximum of 200 steps per episode to solve the level, after which the episode is terminated, since if the agent gets itself into an unwinnable state, the

episode would never terminate. This method was put in place with consideration to scaling up the complexity of the levels since determining whether a given state is winnable in BIY on a general level is undecidable (Geller, 2022). Furthermore, both agents are averaged over 100 trials, with each trial being trained for 15,000 episodes due to limited computational power.

5.4 Experiment Results and Discussion

The two agents were run on the chosen level for 100 trials, with each trial having 15,000 episodes, along with three baseline agents. The three baseline agents are a random agent with only primitive actions, a random agent with both primitive actions and an additional handcrafted option, and an optimal agent created using the STG. The two random agents were also run for 100 trials, 15,000 episodes per trial. This produced the graph displayed in Figure 5.1 (raw results can be found in Appendix B). The orange line corresponds to the Q-learning agent with primitive actions only, and the blue line corresponds to the handcrafted options agent.

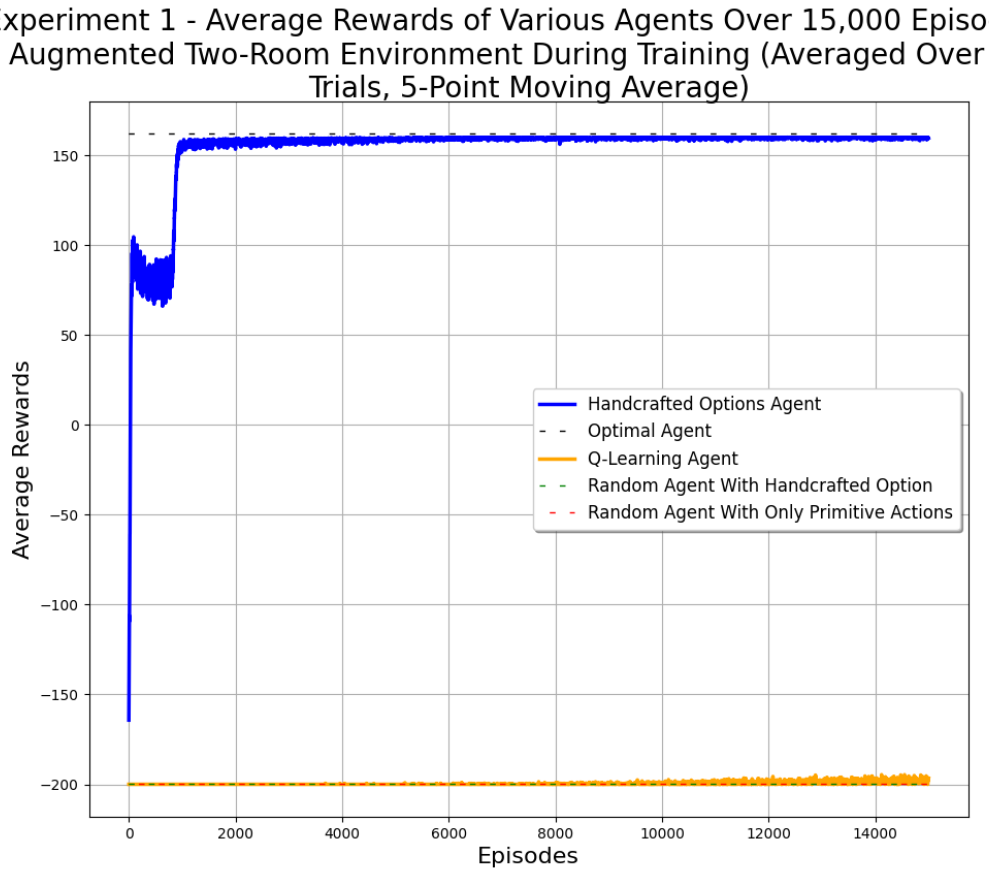

Figure 5.1: Experiment 1 training graph (smoothed), created by author.

The Q-Learning agent did not manage to show a significant improvement in rewards received throughout the 15,000 episodes. This might be because the Q-learning agent frequently reached unwinnable states (i.e. states from which no sequence of moves could lead to a win —

see Figure 5.2), and so any other action that it chooses after reaching those states is irrelevant as it has already made a mistake in the beginning sequence of moves.

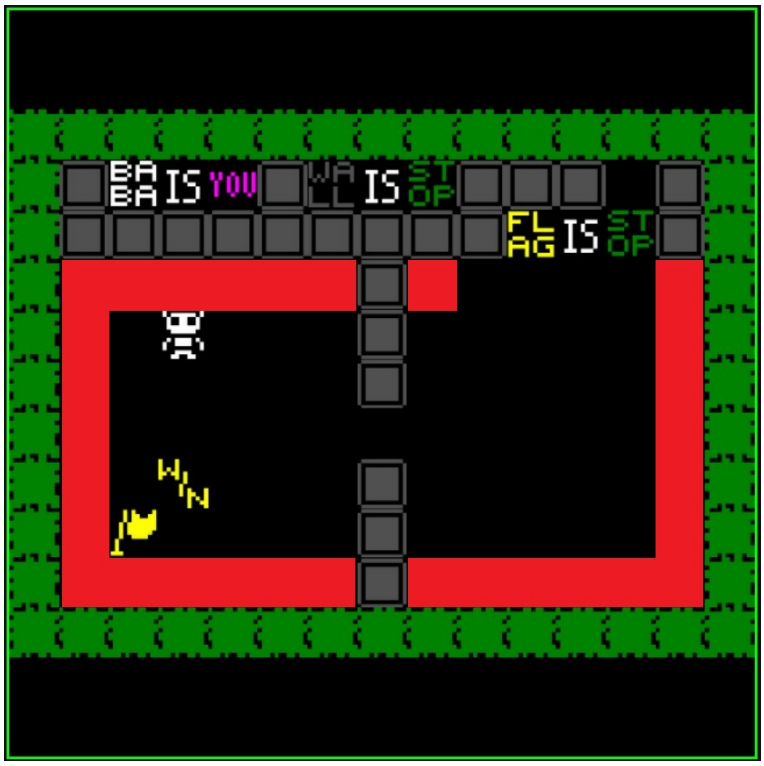

Figure 5.2: Augmented two-room environment, created by author using Baba Is Y'all (Charity, Khalifa and Togelius, 2020), unwinnable states are precisely those having the *Win* word in any of the locations coloured in red (labelled by author).

Thus, the Q-Learning agent mostly learnt what it should not do in the earlier episodes since it kept getting a small negative reward of -1 for these moves, but did not receive many positive rewards. Once it managed to eliminate much of what it should not be doing, it started randomly creating the *Flag Is Win* rule, after which it randomly reached the winning state more frequently, and so it began receiving large positive rewards.

However, as rewards are propagated back slowly in the Q-learning algorithm, the information about whether an action led to a successful trajectory was not available immediately in the episodes following a success, and so the agent did not know better about which action to take in an early state in those episodes. The results shown in Figure 5.1 strengthen this theory since until episode 4000 the Q-learning agent shows almost no successful episodes. However, starting from around episode 4000, it does start showing minimal signs of success at an increasing rate. The successful episodes, however, do not happen often enough, so when averaging them over 100 trials, no significant learning is displayed. If we were able to run each trial for more episodes, we would expect to see a classic learning curve developing, since the Q-learning algorithm converges to an optimal policy when given enough time (Watkins and Dayan, 1992). However, due to computational limitations, we were not able to do this. Instead, to prove this point, we repeated the experiment on a scaled-down level with a similar layout to show that the Q-learning agent is viable and eventually learns. The experiment results can be found in Appendix A.

On the other hand, the handcrafted options agent managed to show significant increase in rewards received throughout the 15,000 episodes. This suggests that the handcrafted options agent has significantly learnt throughout its training. One contributing factor is that it reached the unwinnable states (see Figure 5.2) a lot less frequently. This is due to the handcrafted option always being available in the earlier states which are still winnable, and so the handcrafted options agent is statistically more likely to pick that option, which guarantees it not reaching an unwinnable state thereafter for that episode. However, if that alone were the deciding factor for its success, then the random agent with the handcrafted option would also have managed to get positive rewards, but it does not. This suggests that the options agent learns. The reason for its precise reward-gaining pattern might be explained as follows:

In nearly all episodes, the handcrafted options agent ended up picking the handcrafted option at some point during the episode, whether by choosing the option uniformly at random with 0.2 probability, or during the later episodes by picking it greedily since it has a large positive Q-value. Either way, the option took the agent to the state attained after creating the *Flag Is Win* rule. The agent then acted like a Q-learning agent from that point in the episode onward since the handcrafted option was no longer available to it. However, unlike in the full problem of the original Q-learning agent, the solution from that point requires fewer decisions. Therefore, with the remaining time-steps for that episode, the agent explores different states reachable from that point, including the winning state. This happened at the end of nearly every episode of the first 1,000 episodes, hence justifying the receiving of positive rewards. However, during these episodes, the rewards did not yet propagate all the way back to the states visited in the earlier time-steps, and so those states received only a small negative reward.

Thus, the handcrafted options agent did not know that what it did was good, and so it tried exploring new states before choosing the handcrafted option, thus resulting in episodes which took longer. Hence there is a decrease of rewards received by the handcrafted options agent for these episodes. After around 1,000 episodes, the rewards were already propagated to the state just after the rule creation, and so there was a path of positive Q-values from there until the end. In other words, the agent knew how to reach the end from that point. It then took only one more SMDP Q-learning update rule for the agent to propagate the rewards backwards from that state all the way to the state in which the handcrafted option was initiated, which is likely to have been a few steps away from the original state. After that, it took only a few more episodes to propagate the rewards back to the original state, thus solving the problem. From this point onward, the options agent repeated what it had learnt, which is represented by the rewards it received, shown by the plateau in Figure 5.1 from around episode 1,000.

After training each agent, we tested their performance by reducing ε to 0, hence making the agents pick actions greedily. Table 5.1 shows the score of the two agents after being trained, averaged over 100 trials. All 100 trials of the handcrafted options agent managed to find the optimal path by the end of training, while no trials of the Q-learning agent managed to do this. The random agents' score as well as the optimal agent's score were the same as in training due to their policy not being changed during training.

Table 5.1: Experiment 1 results table after training, with agents acting greedily (averaged over 100 trials).

Agent	Rewards Received After Training
Q-Learning Agent	−200.00
Handcrafted Options Agent	162.00
Random Agent With Only Primitive Actions	−200.00
Random Agent With Handcrafted Options	−200.00
Optimal Agent	162.00

5.5 Experiment Conclusion

Having carried out the experiment, it is clear that the addition of the hierarchy introduced to the base algorithm in this case was what caused the performance to improve. However, the subgoal in this experiment was handcrafted. We wish to explore methods which do not require handcrafting the subgoals. We will continue to explore this in Chapter 6.

Chapter 6

Experiment 2

Having carried out experiment 1, we determined that introducing hierarchy to an agent can be beneficial, assuming the options created are useful. We would therefore now like to explore the usefulness of different subgoals generated automatically through the three base centrality measures discussed in Section 3.3.1, namely degree, closeness, and betweenness centralities; and the RL variation of betweenness suggested by Şimşek and Barto (2008). We will do so by using the level created in Section 4.1.3 as our environment, and then follow the steps below:

1. Compute the STG of the level.
2. Compute the values of each node in the STG using each of the centrality measures above.
3. Create subgoals for each centrality measure by using the values of each node calculated in point 2.
4. Generate fixed options using the different subgoals generated in point 3 for each centrality measure.
5. Supply the options agent with each set of fixed options generated in point 4 for each centrality measure.
6. Analyse the results produced by the experimentation.

Thus, our independent variable is the centrality measure used; the dependent variable is the value of rewards received for each time-step; and the control variables are the following:

1. The level created on which to run the experiment (found in Section 4.1.3), which determines the SDP.
2. The method of subgoal generation from each centrality measure.
3. The method of fixed option creation from each subgoal generated.
4. The agent type employed, which uses each set of fixed options created as input.

The method of subgoal generation from each centrality measure, fixed option creation, and agent type that we will be using has already been described in Section 4.

Intuitively, the choice of centrality measure should change the value of rewards received during every episode since the different centrality measures take into account different factors of the

STG. For example, the degree centrality is only concerned about immediate neighbours and so is not likely to find bottleneck states, which usually have fewer neighbours by nature. The closeness centrality, on the other hand, is more likely to find bottleneck states due to these being on the boundary between two clusters of states, and so are more likely to lie in the middle of the STG. Similarly, naive and improved betweenness centrality measures are also more likely to identify bottleneck states due to the measures considering paths between nodes.

6.1 Hypothesis

H0: In the chosen level, different subgoals generated by the degree, closeness, naive betweenness, and improved betweenness centralities will have similar performances — measured by the average number of episodes (averaged over 100 trials) to achieve a score of over 100.

H1: In the chosen level, different subgoal states generated by the degree, closeness, naive betweenness, and improved betweenness centralities will have different performances — measured by the average number of episodes (averaged over 100 trials) to achieve a score of over 100.

6.2 Experiment Details

The details for this experiment follow those of the first experiment, which can be found in Section 5.3.

6.3 Experiment Results and Discussion

The four agents created using the different options generated from the various centrality measures were run on the chosen level for 100 trials, with each trial having 15,000 episodes, along with the agents created in Experiment 1 (found in Section 5.4), which now act as baseline agents. This produced the graph displayed in Figure 6.1 (raw results can be found in Appendix B).

The green, red, blue, and purple lines correspond to the degree, closeness, naive betweenness and improved betweenness options agents respectively.

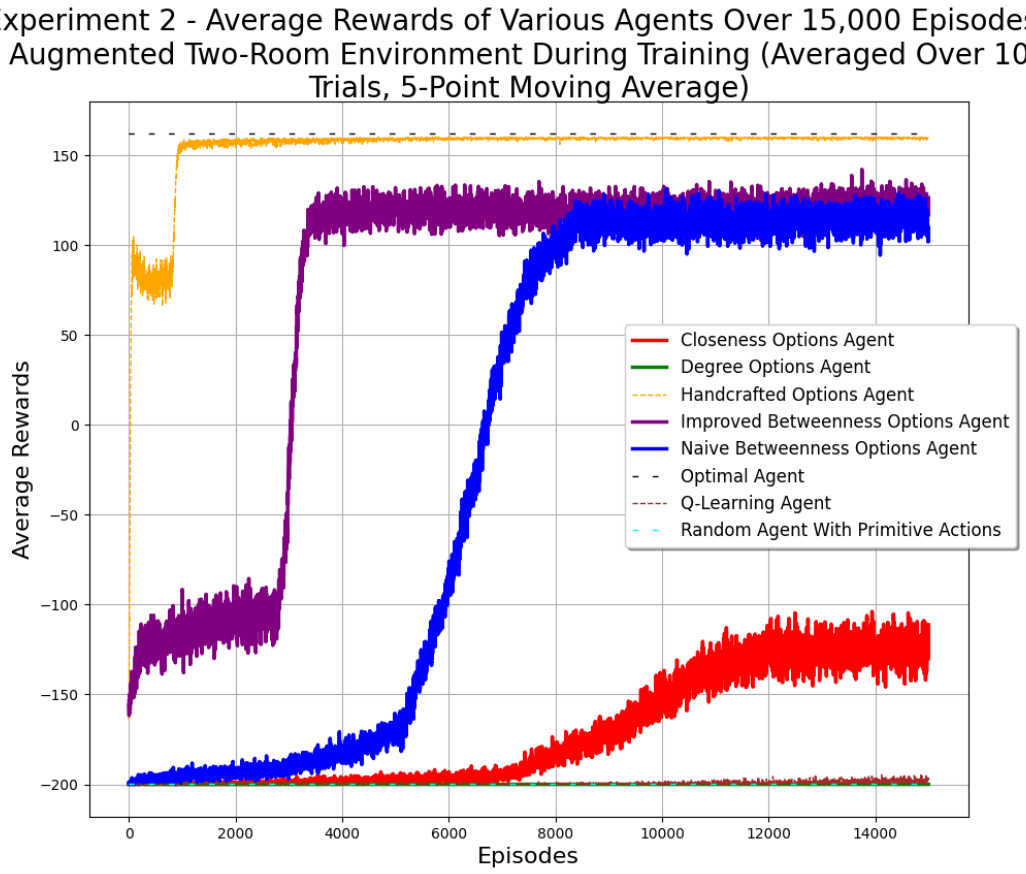

Figure 6.1: Experiment 2 training graph (smoothed), created by author.

The degree centrality measure generated 8 subgoal states: 4 of these were unwinnable states, and the other 4 were states that had the *Flag Is Stop* rule disabled and the Baba sprite and *Win* word located near the Flag sprite. The earlier subgoal states created options which, if picked, led the agent to unwinnable states. Furthermore, the latter subgoal states created options which were not helpful either, as the agent spent 40 moves to execute the options, yet was not any closer to the winning state than an agent that had not executed the options (we label options like these as ineffective). This supports our initial intuition, and the results match this.

The degree options agent ended up performing worse than the Q-learning agent, showing no signs of improvement over the 15,000 episodes that it was trained for, as it did not manage to reach the winning state in any episode of any trial. This is because the options generated distracted the agent from the main goal and wasted moves or worse, brought the agent to unwinnable states.

On the other hand, the rest of the agents, namely the closeness and the two betweenness options agents, did show signs of learning throughout the 15,000 episodes, and even managed to gain more rewards than the Q-learning agent did.

The closeness centrality measure generated 26 subgoal states: 18 of these were unwinnable states, but the other 8 were useful. The first 4 of the 8 were subgoal states that had the *Win* word in the same vertical level as the passageway between the two rooms and the other 4

subgoal states had the *Win* word in the top half of the room on the right. Thus, overall, some of the subgoal states generated by the closeness measure ended up being useful, so when the closeness options agent was trained, it was able to differentiate between the useful subgoals and the non-useful ones. This allowed the closeness options agent to learn, as can be seen in Figure 6.1 where the agent starts displaying an increase in rewards gained, showing the beginning of a learning curve until episode 11,000.

However, from episode 11,000 onward, the rewards received by the agent begin to plateau. We believe that this plateauing does not suggest that the agent stopped learning, but is due to the combination of two factors which negate each other. Firstly, for a similar reason as explained in Experiment 1 for the handcrafted options agent, the closeness options agent explored further states in the early time-steps of the episodes, as the rewards had not yet been propagated to the starting state. This sequence of inefficient moves led to the decrease of rewards gained in those episodes. On the other hand, the closeness options agent in the later time-steps of the episodes improved its policy from the point after the additional options were executed, and so the number of moves to reach the winning state decreased, which led to the increase of rewards gained in those episodes.

The combination of these two elements makes it look as if the agent's learning is plateauing, but we believe that if we could have run the experiments for longer, the agent's line would have recovered from the plateau and would transition into a learning curve. However, as training was cut at 15,000 episodes due to computational limitations, the agent was not able to attain the full learning curve, so similarly to in Experiment 1, more episodes per agent are needed to see the full curve.

The naive betweenness centrality measure generated 400 subgoal states: 138 of these were unwinnable states, and a further 95 were states where the *Win* word was in the left room (49 in the top half and 46 in the bottom half); 61 states had the *Win* word in the same vertical level as the passageway between the two rooms, and a further 105 states were states where the *Win* word was in the right room (58 in the top half and 47 in the bottom half). In addition, 1 subgoal state (subgoal state number 319) had the *Flag Is Win* rule active and had the Baba sprite located in the passageway, being only 7 moves away from the goal state. With just under half of the subgoals generated being useful and advancing the agent in the right direction, this agent managed to reach the goal in most episodes, and thus managed to propagate the rewards backwards to the beginning state. This is supported by the classic learning curve displayed on the graph generated from this agent.

However, the agent did not manage to finish its learning within the 15,000 episodes. Theoretically, if the agent would have reached an optimal policy, the expected rewards during training (when $\varepsilon = 0.1$) would have been

$$0.9 \times 162 + 0.1 \left[\frac{138}{404} \times (-200) + \frac{95}{404} \times 100 + \frac{167 + 4}{404} \times 162 \right] \approx 148 \qquad (14)$$

assuming that any option that leads to a subgoal state which is ineffective results in an approximate 100 average score as it wastes around 60 moves and then takes a further 40 moves to reach the winning state, and any effective winnable subgoal state gets a score of 162. This is more than what the agent attained in the final training episodes, which is approximately 115.

It is worth noting that the theoretical value is less than what the handcrafted options agent managed to achieve because if at any point of the episode the naive betweenness agent picked an action that took it to an unwinnable state (which happened with probability 0.34 when an action is picked uniformly at random), then the agent ended up getting -200 for the entire episode. This did not happen with such a high probability in the handcrafted options agent. In addition, 95 of the options created were ineffective, having a similar effect to the subgoals discussed with regards to the degree options agent.

Lastly, the improved betweenness options agent had additional information about the reward system of the environment, and thus generated only 51 subgoal states (compared to the 400 generated by the naive betweenness measure); 33 of the 51 were unwinnable states, and a further 17 were states where the *Win* word was in the top half of the room on the right. In addition, there was 1 subgoal state (subgoal state number 47) that was the winning state. Due to many of the options generated being useful, and one of them already leading the agent directly to the winning state, the improved betweenness options agent started gaining some positive rewards from the very first episodes, averaging out with a score of -152 in the first episode.

Then, over the next 3,000 episodes, the agent learnt not to pick the options which led it to unwinnable states, and so the rewards increased slowly. After about 3,000 episodes there was a steep learning curve where the agent learnt to pick the useful options, and the rewards were propagated back to the beginning state. The gradient of the curve in those episodes is steeper than the gradient of the curve of the naive betweenness options agent due to the rewards taking fewer steps to propagate back to the beginning state. This is due to the useful options of the improved betweenness options agent being more beneficial, on average, than the useful options of the naive betweenness options agent. After the steep learning curve, the agent began to plateau, getting close to having learnt an optimal policy. In this instance, the agent also did not manage to finish its learning within the 15,000 episodes. Theoretically, if the agent would have reached an optimal policy, the expected rewards during training (when $\varepsilon = 0.1$) would have been

$$0.9 \times 162 + 0.1 \left[\frac{33}{55} \times (-200) + \frac{18+4}{55} \times 162 \right] \approx 137 \tag{15}$$

assuming any effective winnable subgoal state results in a score of 162. This is more than what the agent attained in the final training episodes, which is approximately 125.

After training each agent, we tested their performance by reducing ε to 0, hence making the agents pick actions greedily. Table 6.1 shows the score of the agents we tested after being trained, averaged over 100 trials. Both betweenness options agents managed to find the winning state in all 100 trials, while the closeness agent found it in only 32 trials (with the other 68 trials failing to reach the winning state). The degree agent failed to reach the winning state in all 100 trials.

Table 6.1: Experiment 2 results table after training, with agents acting greedily (averaged over 100 trials).

Agent	Rewards Received After Training
Degree Options Agent	−200.00
Closeness Options Agent	−84.16
Naive Betweenness Options Agent	162.00
Improved Betweenness Options Agent	162.00
Q-Learning Agent	−200.00
Handcrafted Options Agent	162.00
Random Agent With Only Primitive Actions	−200.00
Optimal Agent	162.00

6.4 Experiment Conclusion

Having carried out this experiment, it is clear that the generation of subgoals from different centrality measures affects the performance of the options agents that utilise the subgoals as options. The betweenness centrality measure ended up performing the best out of the 3 basic ones, with its variation outperforming it.

One problem that occurred with all agents, and in particular the betweenness agents, is that many of the initiation sets of different options were overlapping. Consequently, the agents had too many options to choose from at any given state, and so they took time to explore all the options. A related problem is that there were pairs of options whose initiation sets contained the subgoal state generated by the other option. This led to a situation where option 1 was picked from the subgoal state of option 2, which then took the agent to the subgoal state of option 1. Immediately afterwards, option 2 was picked, which brought the agent back to the subgoal state of option 2. This resulted in an ineffective sequence of moves (i.e. the same position was reached, but with fewer moves left until the episode was reset), which also hindered the performance of the agents.

One way to overcome both these issues is to limit the initiation set of each option further than we did, to include only states within a certain number of moves from the subgoal state (e.g. 5 moves away). We will leave this for future work.

Chapter 7

Future Work and Conclusion

7.1 Future Work

There are several ways to develop on this dissertation for future work. Firstly, remaining in small-scale experiments, more centrality measures and variations of centrality measures need to be tested. We tested only 4 of them (3 basic and 1 RL variation), but many more should be done. Furthermore, our experiments should be run for more than 15,000 episodes per trial to allow time for the agents to optimise their policy, and should be averaged over a larger number of trials to improve the precision of the experiments.

Another important point is to test more agent types. We compared a Q-learning agent and an options agent, but there are more base algorithms that have a hierarchical equivalent which should be explored. For example, in order to scale up to more complicated environments, the base algorithm will need to be changed from a tabular method to a function approximated method, due to the number of states being vast and memory space being an issue. Further aspects need to be changed in order to scale up the environment, including the method of subgoal generation and option creation, for which we used an STG, but this would not be possible for scaled-up environments due to memory constraints.

7.2 Conclusion

We have achieved all the objectives of this dissertation, namely, we identified a suitable level of BIY that displays the criteria of a single agent, long-horizon, sparse reward SDP containing bottleneck states; we found an appropriate pair of agents which have the same base algorithm, one of which has hierarchy while the other does not; we decided on methods of subgoal generation for the hierarchical agent, and implemented those methods for the chosen level; and we implemented and compared the agents created by the methods above. Through fulfilling these objectives, we explored the effect of introducing hierarchy into an RL agent when solving a single decision-maker, long-horizon, sparse reward SDP containing bottleneck states, which was the overarching aim of this dissertation.

Furthermore, the environment we identified, namely the game BIY, is a relatively novel environment in connection with RL. To the best of our knowledge, only one other piece of literature used this domain in relation to RL, and it did not attempt to use HRL. Through this

dissertation, we have demonstrated that BIY can serve as an effective environment for future RL research and, in particular, for the development of HRL agents.

Number of words: 9585

Bibliography

Barto, A. and Mahadevan, S., 2003. Recent advances in hierarchical reinforcement learning. *Discrete event dynamic systems: Theory and applications* [Online], 13. Available from: https://doi.org/10.1023/A:1025696116075.

Bavelas, A., 1948. A mathematical model for group structures. *Human organization* [Online], 7(3). Available from: https://www.jstor.org/stable/44135428.

Bradtke, S.J. and Duff, M.O., 1994. Reinforcement learning methods for continuous-time markov decision problems [Online]. *Proceedings of the 7th international conference on neural information processing systems*. Cambridge, MA, USA: MIT Press, p.393–400. Available from: https://dl.acm.org/doi/10.5555/2998687.2998736.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J. and Zaremba, W., 2016. Openai gym. *Corr* [Online], abs/1606.01540. Available from: http://arxiv.org/abs/1606.01540.

Charity, M., Khalifa, A. and Togelius, J., 2020. Baba is y'all: Collaborative mixed-initiative level design [Online]. *2020 ieee conference on games (cog)*. pp.542–549. Available from: https://doi.org/10.1109/CoG47356.2020.9231807.

Charity, M. and Togelius, J., 2022. Keke ai competition: Solving puzzle levels in a dynamically changing mechanic space [Online]. *2022 ieee conference on games (cog)*. IEEE Press, p.570–575. Available from: https://doi.org/10.1109/CoG51982.2022.9893650.

Das, K., Samanta, S. and Pal, M., 2018. Study on centrality measures in social networks: a survey. *Social network analysis and mining* [Online], 8(1). Available from: https://doi.org/10.1007/s13278-018-0493-2.

Dayan, P. and Hinton, G.E., 1992. Feudal reinforcement learning [Online]. In: S. Hanson, J. Cowan and C. Giles, eds. *Advances in neural information processing systems*. Morgan-Kaufmann, p.271–278. Available from: https://proceedings.neurips.cc/paper_files/paper/1992/file/d14220ee66aeec73c49038385428ec4c-Paper.pdf.

Diederich, A., 2001. Sequential decision making [Online]. In: N.J. Smelser and P.B. Baltes, eds. *International encyclopedia of the social behavioral sciences*. Oxford: Pergamon, pp.13917–13922. Available from: https://doi.org/https://doi.org/10.1016/B0-08-043076-7/00636-7.

Dietterich, T.G., 1999. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Corr* [Online], cs.LG/9905014. Available from: https://arxiv.org/abs/cs/9905014.

Freeman, L.C., 1978. Centrality in social networks conceptual clarification. *Social networks* [Online], 1(3). Available from: https://doi.org/10.1016/0378-8733(78)90021-7.

Geller, J., 2022. Baba is you is undecidable. *Arxiv* [Online], abs/2205.00127. Available from: https://arxiv.org/pdf/2205.00127.pdf.

Hengst, B., 2002. Discovering hierarchy in reinforcement learning with hexq [Online]. *Proceedings of the nineteenth international conference on machine learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., p.243–250. Available from: https://dl.acm.org/doi/10.5555/645531.656017.

Kulkarni, T.D., Narasimhan, K., Saeedi, A. and Tenenbaum, J.B., 2016. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. *Corr* [Online], abs/1604.06057. Available from: http://arxiv.org/abs/1604.06057.

Mendonça, M.R.F., Ziviani, A. and Barreto, A.M.S., 2019. Graph-based skill acquisition for reinforcement learning. *Acm computing surveys* [Online], 52(1). Available from: https://doi.org/10.1145/3291045.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M.A., Fidjeland, A.K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S. and Hassabis, D., 2015. Human-level control through deep reinforcement learning. *Nature* [Online], 518(7540). Available from: https://www.nature.com/articles/nature14236.

Nieminen, J., 1974. On centrality in a graph. *Scandinavian journal of psychology* [Online], 15(4). Available from: https://www.semanticscholar.org/paper/On-the-centrality-in-a-graph.-Nieminen/cc873600a1efe9af5dff98e99721554f32909a7d.

OpenAI, 2023. *Gpt-4 technical report* [Online]. 2303.08774, Available from: https://doi.org/10.48550/arXiv.2303.08774.

Panait, L. and Luke, S., 2005. Cooperative multi-agent learning: The state of the art. *Autonomous agents and multi-agent systems* [Online], 11(3). Available from: https://doi.org/10.1007/s10458-005-2631-2.

Parr, R. and Russell, S., 1998. Reinforcement learning with hierarchies of machines [Online]. *Proceedings of the 1997 conference on advances in neural information processing systems 10*. Cambridge, MA, USA: MIT Press, p.1043–1049. Available from: https://dl.acm.org/doi/10.5555/302528.302894.

Pateria, S., Subagdja, B., Tan, A.h. and Quek, C., 2021. Hierarchical reinforcement learning: A comprehensive survey. *Acm computing surveys* [Online], 54(5). Available from: https://doi.org/10.1145/3453160.

Precup, D., 2000. *Temporal abstraction in reinforcement learning* [Online]. Thesis (PhD). University of Massachusetts Amherst. Available from: https://www.proquest.com/docview/304606445.

Puterman, M.L. and Shin, M.C., 1978. Modified policy iteration algorithms for discounted markov decision problems. *Management science* [Online], 24(11). Available from: http://www.jstor.org/stable/2630487.

Ross, S.M., 1983. *Introduction to stochastic dynamic programming* [Online]. San Diego ; London: Academic Press. Available from: https://doi.org/10.1016/C2013-0-11415-8.

Rummery, G. and Niranjan, M., 1994. *On-line q-learning using connectionist systems*. Available from: http://mi.eng.cam.ac.uk/reports/svr-ftp/auto-pdf/rummery_tr166.pdf.

Sabidussi, G., 1966. The centrality index of a graph. *Psychometrika* [Online], 31(4). Available from: https://doi.org/10.1007/BF02289527.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A. and Klimov, O., 2017. Proximal policy optimization algorithms. *Corr* [Online], abs/1707.06347. Available from: http://arxiv.org/abs/1707.06347.

Shaw, M.E., 1954. Group structure and the behavior of individuals in small groups. *The journal of psychology* [Online], 38(1). Available from: https://doi.org/10.1080/00223980.1954.9712925.

Şimşek, Ö., Algorta, S. and Kothiyal, A., 2016. Why most decisions are easy in tetris—and perhaps in other sequential decision problems, as well [Online]. In: M.F. Balcan and K.Q. Weinberger, eds. *Proceedings of the 33rd international conference on machine learning*. New York, NY, USA: PMLR, pp.1757–1765. Available from: https://proceedings.mlr.press/v48/simsek16.html.

Şimşek, Ö. and Barto, A.G., 2004. Using relative novelty to identify useful temporal abstractions in reinforcement learning [Online]. *Proceedings of the twenty-first international conference on machine learning*. New York, NY, USA: Association for Computing Machinery, pp.751–758. Available from: https://doi.org/10.1145/1015330.1015353.

Şimşek, Ö. and Barto, A.G., 2008. Skill characterization based on betweenness [Online]. In: D. Koller, D. Schuurmans, Y. Bengio and L. Bottou, eds. *Advances in neural information processing systems*. Curran Associates, Inc. Available from: https://proceedings.neurips.cc/paper_files/paper/2008/file/934815ad542a4a7c5e8a2dfa04fea9f5-Paper.pdf.

Singh, S., Sutton, R. and Kaelbling, P., 1995. Reinforcement learning with replacing eligibility traces. *Machine learning* [Online], 22(1-3). Available from: https://doi.org/10.1023/A:1018012322525.

Sutton, R.S. and Barto, A.G., 2018. *Reinforcement learning: An introduction*. 2nd ed. Cambridge, Massachusetts: Bradford Books.

Sutton, R.S., Precup, D. and Singh, S., 1999. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence* [Online], 112(1). Available from: https://doi.org/10.1016/S0004-3702(99)00052-1.

Teikari, A., 2019. *Baba is you* (v.477) [video game]. Available from: https://hempuli.com/baba/.

Watkins, C., 1989. *Learning from delayed rewards* [Online]. Thesis (PhD). King's College, Cambridge. Available from: http://mi.eng.cam.ac.uk/reports/svr-ftp/auto-pdf/rummery_tr166.pdf.

Watkins, C. and Dayan, P., 1992. Technical note: Q-learning. *Machine learning* [Online], 8(3). Available from: `https://doi.org/10.1007/BF00992698`.

Younes, H.L.S. and Simmons, R.G., 2004. Solving generalized semi-markov decision processes using continuous phase-type distributions [Online]. *Proceedings of the 19th national conference on artifical intelligence*. San Jose, California: AAAI Press, p.742–747. Available from: `https://dl.acm.org/doi/10.5555/1597148.1597267`.

Appendix A

Experiment 1 Scaled-Down Environment

This appendix contains the experiment results of the scaled-down augmented two-room environment. In particular, the level can be found in Figure A.1, and the training graphs and results table can be found in Figure A.2 and Figure A.3, and Table A.1 respectively. Notably, the Q-learning agent manages to display a full learning curve as predicted in Section 5.4.

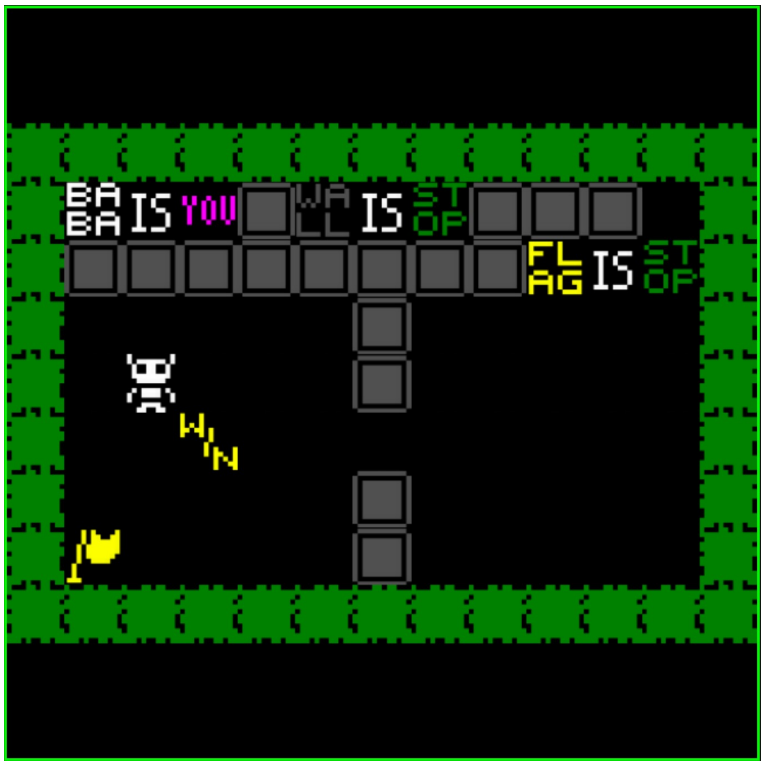

Figure A.1: Scaled-down augmented two-room environment, created by author using Baba Is Y'all (Charity, Khalifa and Togelius, 2020).

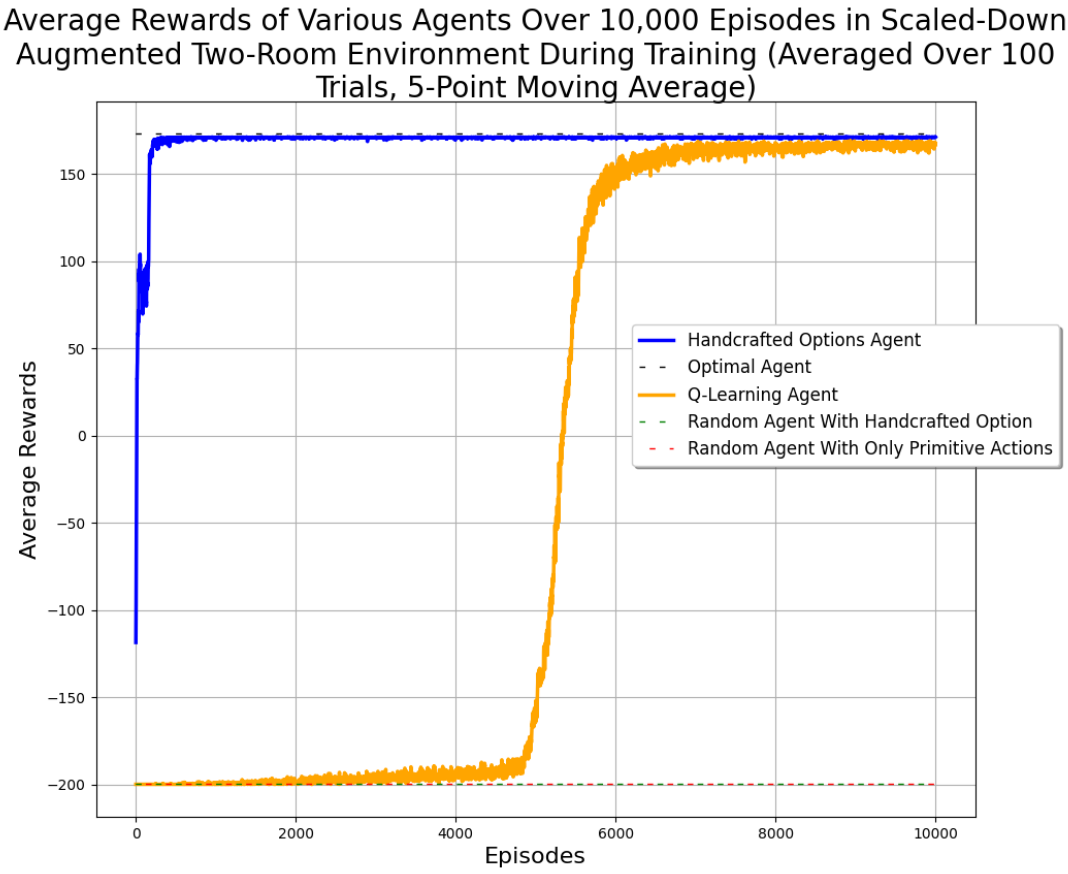

Figure A.2: Scaled-down augmented two-room environment training graph (smoothed), created by author.

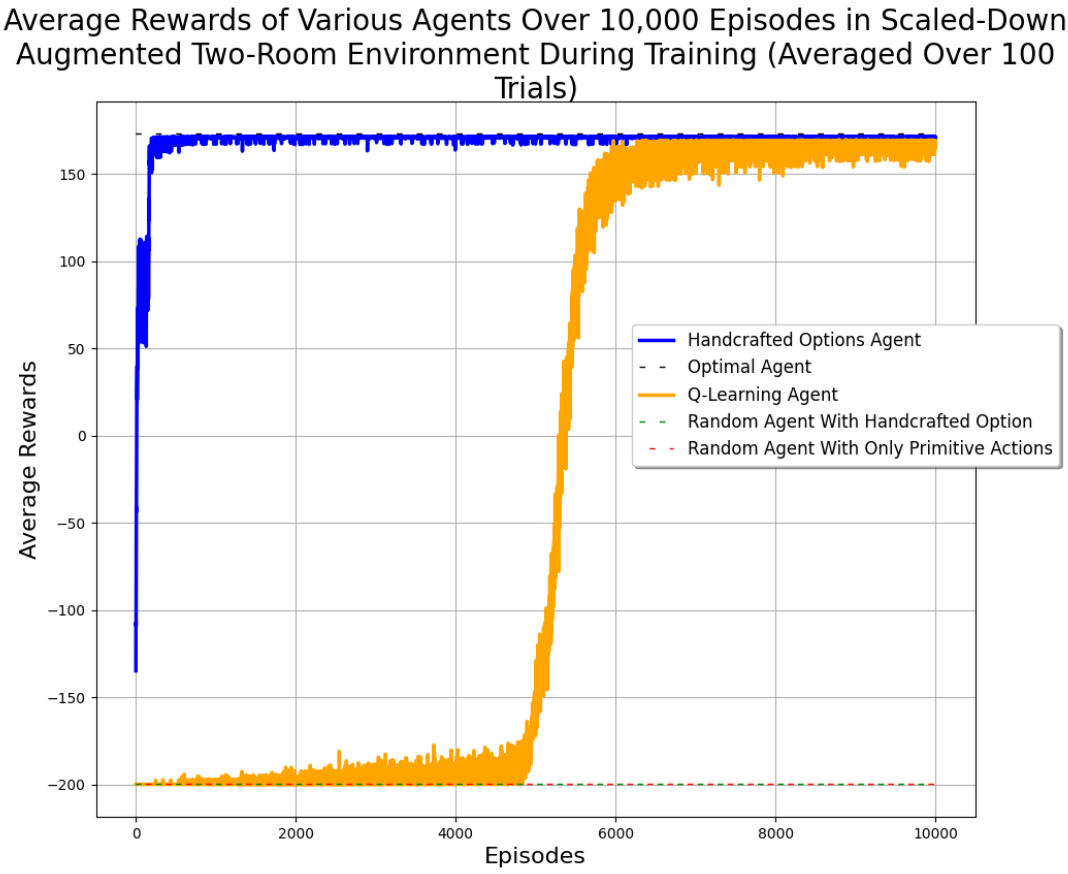

Figure A.3: Scaled-down augmented two-room environment training graph (raw), created by author.

Table A.1: Scaled-down augmented two-room environment results table after training, with agents acting greedily (averaged over 100 trials).

Agent	Rewards received after training
Q-Learning Agent	173.00
Handcrafted Options Agent	173.00
Random Agent With Only Primitive Actions	−200.00
Random Agent With Handcrafted Options	−200.00
Optimal Agent	173.00

Appendix B

Experiments 1 and 2 Raw Graphs

This appendix contains the raw experiment results of the augmented two-room environment. In particular, the raw results of experiment 1 training can be found in Figure B.1, and the raw results of experiment 2 training can be found in Figure B.2.

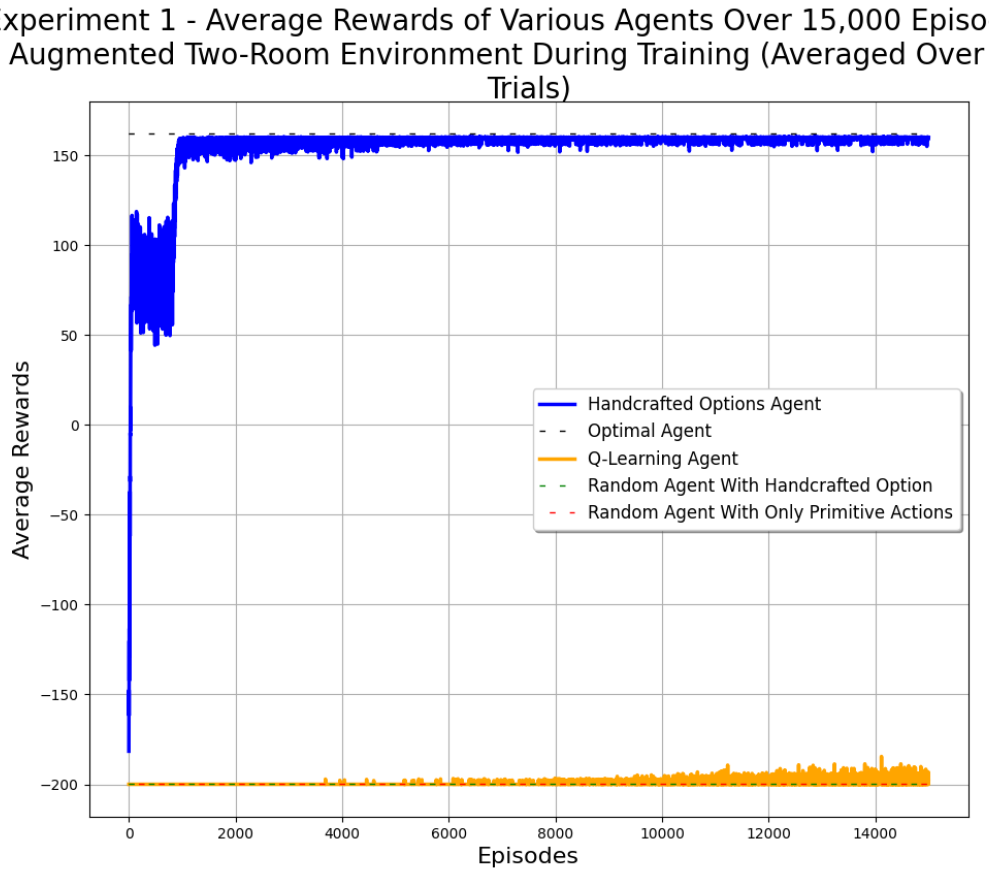

Figure B.1: Experiment 1 training graph (raw), created by author.

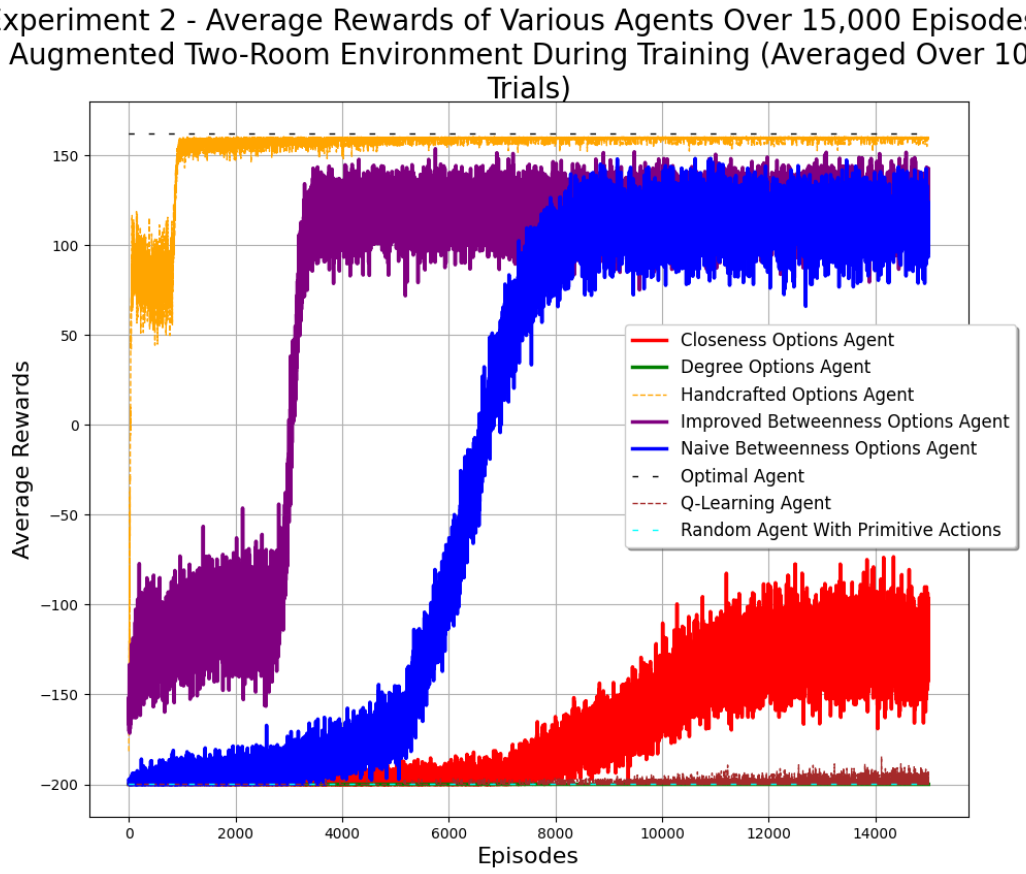

Figure B.2: Experiment 2 training graph (raw), created by author.