Small-World Options in Reinforcement Learning - In the Automatic Guided Vehicle Domain

submitted by

VIKRAM RAO S

in partial fulfilment of the requirements
for the award of the degree of

BACHELOR OF TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY MADRAS.

May 2013

THESIS CERTIFICATE

This is to certify that the thesis titled **Small-World Options in Reinforcement Learning - In the Automatic Guided Vehicle Domain** submitted by **Vikram Rao S**, to the Indian Institute of Technology, Madras, for the award of the degrees of **Bachelor of Technology**, is a bona fide record of the research work done by him under our supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Dr. Balaraman Ravindran
Research Guide
Associate Professor
Dept. of Computer Science and Engineering
IIT Madras, 600 036

Place: Chennai

Date:

ACKNOWLEDGEMENTS

Firstly, I would like to thank my excellent advisor, Dr. Ravindran Balaraman for his guidance, motivating words and never-ending enthusiasm and support. He was much more than a guide or teacher to me. I convey my gratitude to him for making my undergraduate days and my project interesting and enjoyable. I am thankful to Dr. C Siva Ram Murthy for his continuous support and encouragement in his role as my faculty advisor.

My interest in intelligent systems led me to take courses by Dr. Hema Murthy, Dr. Deepak Khemani and Dr. Sutanu Chakraborty, in addition to my guide Dr. Ravindran Balaraman - to all of whom I am indebted for introducing me to and deepening my interest in learning techniques and their applications.

I thank the Aditya Birla and the OP Jindal Groups for generously funding my education at IIT Madras through their scholarships.

I am ever grateful to my friends Sujeet Gholap, Devesh Y, Aravind Anil, Arijit Bannerjee, Madhavi Y, many others and my project partner M Vijay Karthik for making my journey interesting and enjoyable. My seniors Kirtika Ruchandani, Arun Chaganty, Pranesh Srinivasan and Vasuki N have played the role of both friends and mentors in all matters, and I must thank them deeply for that.

I would like to specifically acknowledge the role of Dr. R Kalyanakrishnan in my life at IIT Madras. Both during his Professorship and his retired life, he has served as a mentor, philosopher, guide, friend and a fatherly figure. I will never forget my visit to his home and all the inspirational anecdotes he so vividly recounts.

I also thank my parents, Geetha and Sudarshan Rao, whose support is immeasurable. Last but not least, I must thank the Computer Science Department and the RISE Lab as a whole. The remarkable teaching and research talent of our Faculty is admirable and I shall strive to emulate them.

ABSTRACT

One characteristic of intelligent behaviour is the ability to learn from past experience. Learning from one's successes and failures is thus an important trait of intelligent agents. The Reinforcement Learning framework provides a crisp, yet general formalization of agents learning from past behaviour in an environment. A key challenge in the area is that of generalizing experience with some tasks in an environment to other tasks.

A novel solution to this challenge was proposed by Chaganty *et al.* (2012), where the Options Framework for temporal abstractions was combined with ideas from the theory of Small-World Networks. The solution uses minimal knowledge of the environment and generates options, whose number doesn't complicate the agent's decision space by a significant amount.

One of the aims of this project is to empirically study whether the results proposed in Chaganty *et al.* (2012) hold in larger and more complex domains than were previously tested on. The domain of choice is the Automatic Guided Vehicle Domain.

Another aim of this project is to make the proposed method practical by minimizing further, the knowledge of the environment required for generating options. Finally, we observe that the approach proposed implicitly makes certain assumption which render it inapplicable to some domains. We aim to address this limitation by suitably modifying the scheme.

TABLE OF CONTENTS

LIST OF FIGURES

List of Algorithms

ABBREVIATIONS

RL	Reinforcement Learning
MDP	Markov Decision Process
AGV	Automatic Guided Vehicle
SAS	State-Action-State
SA	State-Action

CHAPTER 1

Introduction

Reinforcement Learning (RL) addresses the problem of learning optimal agent behaviour in an unknown stochastic environment. An agent explores the environment, performing actions that lead the environment to transition between states. Some actions in certain states also fetch the agent rewards. The objective of Reinforcement Learning is for the agent to learn policies which maximize the total reward accumulated by the agent over time. The RL framework's generality allows a wide variety of domains to be expressed.

1.1 Exploiting Structure

In the original setting, an agent learns the behaviour to maximize rewards for each new problem from scratch. One form of abstracting the structure of a problem, *temporal abstraction*, involves the notion of a "subtask" ie., introducing higher level actions which are sequences of primitive actions. Such high level actions are called options, according to the framework proposed by Sutton *et al.* (1999); this is the framework we use. Temporal abstractions play a significant role in transfer learning, where the optimal behaviour learnt in one task is extended to other tasks. A survey of many other transfer learning techniques can be found in Taylor and Stone (2009).

Also, options speed up an agent's learning process by reducing the number of decisions an agent has to take on average to reach the goal ie., by making each decision taken perform more work. An example motivating the need for options is given in the next section.

1.2 Motivating Options: Corridor with Rooms

Consider a corridor with rooms on one side. If the objective of the agent is to enter one of the rooms, given that it starts from any of the other rooms uniformly at random, it is clear that the agent must learn how to enter the corridor from the room where it starts.

Thus, the macro-actions or options described in Figure 1.1 help the agent in learning the task faster. The options shown are optimal ways of entering the corridor from each of the rooms. In other words, instead of the agent deciding at each step whether to move up, down, left or right, it decides to execute the option in one decision. The option's policy then navigates the agent onto the corridor efficiently, without the agent having to learn how to behave at each step along the option.

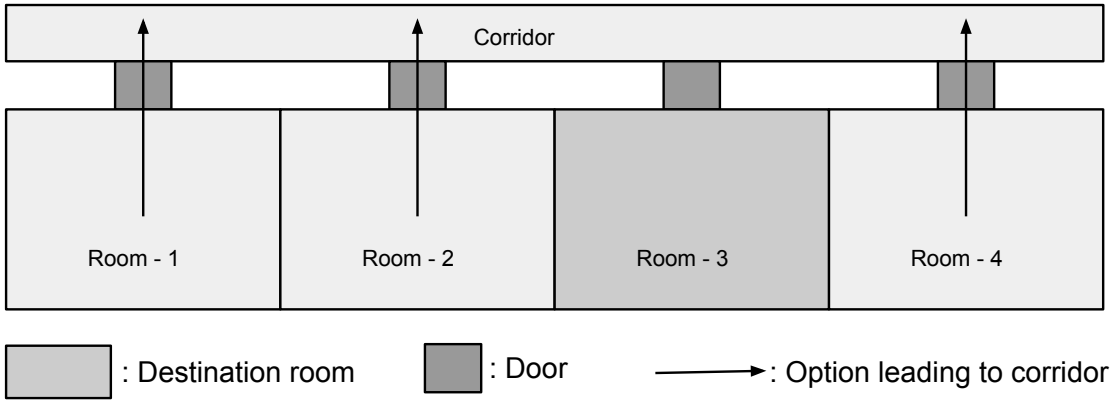


Figure 1.1: Options in a corridor with rooms

1.3 Learning Temporal Abstractions

As mentioned earlier, an "option" is a special action that the agent can choose to take, upon which it follows a pre-specified behavioural policy till the option's termination condition has been satisfied. The options framework also

describes how learning algorithms should be modified in order to include options.

Unfortunately, the framework does not describe how the options themselves should be constructed, and neither is there a consensus in the community on the same. The prevalent view is that subtasks should represent skills (Thrun and Schwartz, 1995). For this reason, most of the existing work centres around identifying "bottlenecks", regions connecting well-connected regions of the state-space (McGovern and Barto, 2001), either empirically as in (McGovern and Barto, 2001), or, using graph theoretic methods like betweenness centrality (Şimşek and Barto, 2008) or graph partitions (Menache *et al.*, 2002). The intuition is that options that navigate an agent to such states helps the agent move between well connected components, thus leading to efficient exploration of the state-space.

These option generation schemes suffer from two major drawbacks; (i) they either require complete knowledge of the MDP (ii) there are, in general, several options to bottlenecks that can be initiated by the agent. This leads to a blowup in the decision space, often causing the agent to take more time to learn the task as it learns to ignore unnecessary options.

1.4 Small-World Networks

Kleinberg (2000) shows that certain kinds of graphs possess the property of efficient navigability using only local information. Specifically, a graph with $|S|$ nodes can be navigated in $\mathrm{O}\big((\log |S|)^2\big)$ steps with each node using information only about its neighbours. Such graphs are referred to as small-world graphs.

As a specific case, it was proved that if a regular grid was augmented with one additional long-range edge per node s, whose destination t was selected

from the graph with a probability $\propto dist(s,t)^{-r}$, certain exponents r lead to the augmented graph being a small-world graph.

The navigation becomes efficient because each long-range edge leads exponentially closer to the destination. The logarithmic number of steps required arises for the same reason.

For a 2-dimensional grid graph, it was shown that $r = 2$ is the desired exponent. $r < 2$ leads to long long-range links which fail to lead exponentially closer to the destination at nodes that are close to the destination, whereas for $r > 2$, the long-range links are very short, thus failing at nodes far away from the destination. Similarly, for a cycle (a 1-dimensional grid), $r = 1$ is the desired exponent.

Chaganty *et al.* (2012) describe a method of learning subtasks which requires minimal knowledge of the environment and also doesn't cause an explosion of the decision space. The method uses ideas from the theory of small-world networks and addresses the limitations of options which are described in Section 1.3. Our work deals with and builds on top of the said work.

1.5 Motivation

Chaganty *et al.* (2012) describe a method to generate options using minimal knowledge of the MDP and show that small-world options perform well, and outperform options generated by other means in the Rooms domain. We were motivated to empirically verify the reported results about the efficiency of small-world options on larger and more complex domains.

Also, they do not address the question of which small-world exponent causes the best performance. We aim to also address that question.

In addition, the method described uses the full MDP when picking the des-

tination of an option. Thus, while generating options in such a manner was sufficient to empirically test the usefulness of small-world options, the method isn't practical. To be a practically useful method, options need to be generated without needing to examine the full MDP. This motivated us to make the scheme proposed in Chaganty *et al.* (2012) practical.

1.6 Our Contribution

We test our proposed option-generation schemes on a simple version and a full-fledged, complex version of the Automatic Guided Vehicle domain described in Ghavamzadeh and Mahadevan (2003). The simple version is presented in Karthik (2013). The domain is large (has a large number of states) as well as complex in its dynamics (reactions) and the semantics of the task to be performed. While the largest domains previously tested on had around $10k$ states, we tested on domains having $30k$ states and having much more complex structure.

We also tried to analyze the variation of the performance of small-world options with the small-world exponent.

In addition, we propose different option-generation schemes in which the agent keeps track of a model of the environment, which is then used as an approximation to the MDP. Our proposed schemes do not require complete knowledge of the Markov Decision Process to generate small-world options.

Limitations of the above schemes, as well as the scheme proposed by Chaganty *et al.* (2012) for certain domains are described and a new option-generation scheme using graph-partitions and small-world based options is proposed to overcome the said limitations.

1.7 Organisation of Thesis

We present an overview of concepts in reinforcement learning, Q-learning, the options framework and small-world options in Chapter 2 and present an overview of various temporal abstraction frameworks in Chapter 3. We present a domain used in testing our methods in Chapter 4. The proposed option-generation scheme is described and some experimental results are presented in Chapter 5. A limitation of the option-generation schemes proposed in Chaganty *et al.* (2012) and those presented in Chapter 5 is described in Section 5.2. A new option-generation scheme which overcomes the said limitations is proposed in Chapter 6, and some results are presented. We finally conclude our work and present possible future directions in Chapter 7.

For the curious reader, the efficiency of using small-world options is proved in Appendix A.

CHAPTER 2

Background

2.1 Reinforcement Learning

2.1.1 Markov Decision Process

The environment in Reinforcement Learning is represented as a Markov Decision Process (MDP). A Markov Decision Process is denoted by the 5 tuple $\langle S, A, P, R, \gamma \rangle$, where

S is the set of states of the environment

A is the set of actions available to the agent

P represents the state transition probabilities ($S \times A \times S \rightarrow [0, 1]$). It defines the probabilities of reaching each state in the MDP when an action is taken from a particular state.

R is the reward function ($S \times A \rightarrow [-k, k]$). It defines the reward gained by the agent for performing an action in a state.

γ : Discount factor. It determines the values of future rewards.

When the agent takes an action $a \in A$ from the state $s \in S$, it moves to the state $t \in S$ with probability $P(s, a, t)$

2.1.2 RL Agents

An agent chooses an action to take at each state. A policy ($\boldsymbol{\pi} : S \times A \to [0, 1]$) defines the decision the agent takes at every state. The objective of the agent is to find a policy which maximises its cumulative reward.

The extent to which an agent values its future rewards is given by γ. The reward accumulated by the agent in the long run is given by

$$R = \sum_i \gamma^i r_i \tag{2.1}$$

where i represents the ith time step, and r_i is the reward that the agent gets in the i^{th} time step. R is called the return.

The agent makes a decision based on how much it values a state ($V : S \to \mathbb{R}$), or how much it values an action to be taken in a state ($Q : S \times A \to \mathbb{R}$). The value function $V(s)$ is the expected return from the state s when following the policy π.

$$\begin{aligned} V^\pi(s) &= \mathbb{E}_\pi\Big[\sum_{t=0}^{\infty} \gamma^t R_t | S_0 = s\Big] \\ &= \max_a R(s, a) + \gamma \sum_{t \in S} P(s, a, t) V^\pi(t) \end{aligned} \tag{2.2}$$

The value function $Q(s, a)$ is the expected return from the state s after taking the action a.

$$\begin{aligned} Q^\pi(s, a) &= \mathbb{E}_\pi\Big[\sum_{t=0}^{\infty} \gamma^t R_t | S_0 = s, A_0 = a\Big] \\ &= R(s, a) + \gamma \sum_{t \in S} P(s, a, t) Q^\pi(t, a') \end{aligned} \tag{2.3}$$

The optimal value functions (V^* and Q^*) satisfy

$$V^*(s) = \max_a R(s,a) + \gamma \sum_{t \in S} P(s,a,t) V^*(t) \tag{2.4}$$

$$Q^*(s,a) = R(s,a) + \gamma \sum_{t \in S} P(s,a,t) \max_{a'} Q^*(t,a') \tag{2.5}$$

A greedy policy is defined as

$$\pi_{greedy}(s) = \arg\max_a Q(s,a) \tag{2.6}$$

If the Q is the optimal value function, then the greedy policy will be an optimal policy. The Q values can be learnt through exploration of the environment.

Q-Learning Agent

The Q-learning algorithm is one of the ways for an agent to update its Q values when exploring. If the agent takes an action a from state s and the environment transitions to state t, then the update done is

$$Q(s,a) = Q(s,a) + \alpha \Big[r + \gamma \max_{a'} Q(t,a') - Q(s,a) \Big] \tag{2.7}$$

where $\alpha \in [0,1]$ controls the learning rate of the agent. The Q-learning algorithm eventually converges to optimal Q values subject to some assumptions.

2.2 The Options Framework

As mentioned in Section 1.3, options provide a temporal abstraction. An option is represented by a 3 tuple $\langle \mathcal{I}, \pi, \beta \rangle$ where

$\mathcal{I}$ is the set of states where an option can be initiated

π is the policy that the agent follows at each state during the option

β is the terminating condition. It can be stochastic.

An agent can start an option o in any state $s \in \mathcal{I}_o$. After initiating an option, the agent will follow the policy π_o prescribed by the option o, until the terminating condition β_o is satisfied.

2.2.1 Learning in an Options Framework : MacroQ

The Q-learning algorithm presented in Section 2.1.2 can be modified to incorporate options, along with primitive actions. When the agent executes an option o lasting k steps, ending at t, and gets a total reward r during the option, it can update its Q values as

$$Q(s,o) = Q(s,o) + \alpha\Big[r + \gamma^k \max_{o' \in A \cup \mathcal{O}} Q(t,o') - Q(s,o)\Big] \tag{2.8}$$

where $\mathcal{O}$ is the set of options.

Options are like stored experience which can be used to reduce the expected learning time of the agent.

2.3 Small World Ideas in Reinforcement Learning

2.3.1 Small World Phenomenon

Small world graphs are graphs where efficient navigation is possible by using only local information (information about neighbors of a node). Small world graphs are a common occurrence, for example, in online social networks.

One way of creating small world graphs from existing graphs is by modifying them by adding an edge to each node connecting it to another node selected with a probability inversely proportional to the distance between them (probability $\propto dist(source, destination)^{-r}$[1]). It is proved in Kleinberg (2000) that a greedy local search algorithm can find a short path ($\mathrm{O}\big((\log |S|)^2\big)$ steps) to any destination in a small world graph by using only local information like the coordinates of its immediate neighbors. Though other graph models with a small diameter state an existence of a short path, they do not guarantee an greedy local search algorithm will find such a path.

2.3.2 Small World Options

Small world options have been proposed by Chaganty *et al.* (2012). A small world option initiating from a state s is defined by

$\mathcal{I}$ = $\{s\}$

β = $\{(d, 1.0)\}$. The option ends in state d with a probability of 1.0. The state d is any one of the states of the MDP chosen with probability $\propto dist(s, d)^{-r}$[2]

π is a policy to take the agent from s to d using the shortest path (minimum number of actions)

[1] Kleinberg's small-world probability distribution
[2] Kleinberg's small-world probability distribution

These options require virtually no information about the environment. It is proved in Chaganty *et al.* (2012) that when small world options are added to an MDP, an ϵ-greedy agent[3] can reach a state of maximum value by taking only $O((\log(|S|))^2)$ decisions.

The number of options added is bounded by $O(|S|)$. Therefore, only a constant number of options are added to each state, and the action space does not grow exponentially by adding options.

[3] An agent which takes the greedy action $1 - \epsilon$ fraction of times, and a random action ϵ of times

CHAPTER 3

Related Work

In this chapter, we present an overview of the various schemes for temporal abstraction which have been proposed and used in Reinforcement Learning. We discuss a few schemes for generating Options and one scheme for Hierarchical Reinforcement Learning. We finally review the emperical results of using small world options in reinforcement learning in simple domains presented in Chaganty *et al.* (2012).

3.1 Bottleneck based options

Bottleneck states are states which connect well connected portions of a graph. Bottleneck based options are a very popular form of options. These options lead to bottleneck states.

When an agent is learning, it is likely for it to remain within strongly connected regions of the state space. Options leading to bottleneck states will tend to connect separate strongly connected regions. Bottlenecks often correspond to subtasks. For example, in the Rooms domain, the bottleneck states correspond to the doors and corridors. An example case of where such options are useful is given in Section 1.2

3.1.1 An algorithm by McGovern and Barto (2001)

One way to identify bottleneck states would be to use the frequency with which an agent visits each state. This is not a good measure because bottleneck

states are visited often only in *successful* trajectories [1]. Hence a better measure to identify bottlenecks would be to maintain counts of first visits to each state in successful trajectories.

The probability of a bottleneck state being in a trajectory given that the trajectory is successful, is high. The probability of a bottleneck state being in a trajectory given that the trajectory is unsuccessful, is low. Bottleneck states are identified using these measures.

Once we identify bottlenecks, options are learnt which lead to these bottleneck states. These options can be started from any state prior to the bottleneck state in trajectories which have these bottleneck states.

It is empirically shown in McGovern and Barto (2001) that agents using Bottleneck options learn faster than agents with no options.

3.1.2 An algorithm by Precup and Stolle (2002)

An algorithm for learning bottleneck based options based on frequency with which states are visited is described in Precup and Stolle (2002). While it is clear how a policy can be learnt given that an option's start and end sets are specified, finding the start and end sets is a question that needs to be addressed. The question is addressed as follows.

Random start and destination state pairs $< S, D >$ are picked. For each pair, an agent is first run for N_{train} episodes to learn a policy to navigate from S to D, by giving a reward for reaching D. Later, the agent is run for N_{test} episodes and the number of times each state is visited $n(s)$ is recorded.

The states visited most frequently correspond to bottlenecks ie., the states connecting well-connected regions of the state-space. Thus, each of the most

[1] A trajectory of length n starting at time step t is defined as the sequence of n states visited by the agent after time t. A trajectory is *successful* if the agent reaches the goal.

frequently visited states D_{max} is chosen as the destination state and an option is constructed leading to it.

The start set of the option is constructed as follows. Given the state D_{max}, find the counts $n(s, D_{max})$ of the number of times state s occurs on paths to D_{max}. The average $avg(D_{max})$ of all such counts $n(s, D_{max})$ is found. The start set is constructed as follows:

$$\mathcal{I} = \{\ s | n(s, D_{max}) > avg(D_{max})\ \}$$

A policy is then learnt by running an agent which gets a reward for reaching D_{max}. This procedure is repeated for other frequently visited states and multiple options are thus generated.

It is empirically shown that the algorithm performs well for a Rooms domain with multiple rooms, which has bottleneck states as well as Rooms domains with a single room, where there are no bottleneck states.

3.2 Hierarchical Reinforcement Learning

Consider a task such as travelling on a business trip. While this task can easily be decomposed into various subtasks, such as picking the mode of transport, the best route between the two cities, the place of stay at the destination etc. which are important decisions, other factors such as the side of the bed one gets up from, the door of the house used for exiting it etc. are unlikely to be important with respect to the final outcome.

Hierarchical reinforcement learning (HRL) provides a framework for Reinforcement Learning agents to model such pragmatics of a domain. Many approaches to Heirarchical Reinforcement Learning have been proposed, including Options. In hierarchical reinforcement learning the overall problem

may be decomposed into smaller subtasks and represented as a task hierarchy. Subtasks at the top of the task hierarchy typically model the more global aspects of the problem at a coarser level. Subtasks near the bottom of the task hierarchy typically model the more local aspects of the problem in finer detail. Hierarchical reinforcement learners generally search for an optimal solution to the overall problem by considering the subtasks at *all* levels in the task hierarchy. While it can be inefficient to search a full subtask hierarchy, particularly when there are several levels with a high branching factor, in practice, for many problems, it may be possible to find good solutions by ignoring details at the lower levels when deciding the subtask policies at higher levels.

Hengst (2002) present approximations which significantly reduce construction, learning and execution time as well as storage requirements of a task hierarchy, while not leading to a significant performance drop compared to the original Hierarchical Reinforcement Learning framework which requires searching the entire task hierarchy.

3.3 PolicyBlocks

PolicyBlocks is an algorithm presented in Pickett and Barto (2002) for creating useful options in reinforcement learning.

Given a sample set M of k Markov Decision Processes having the same state space, but each with a different reward function $R(m)$, and a set of optimal policies $L = \{L_1, L_2, \ldots L_k\}$, a set of options are created using 3 steps.

1. Generate a set of candidate options
2. Score the generated options
3. Remove redundant options

Policies (π) used in these options are *partial* policies. These policies need not be completely defined. For a particular state, $\pi(s)$ can either be a valid action, or $null$. A policy is said to be *full* policy if every state has a valid action.

For the options created, the set of start states $\mathcal{I}$ are the states where the policy of that option defines a valid action ($\pi(s) \neq null$). The option terminates with a probability of 0.001 for the states in $\mathcal{I}$ and with a probability of 1 for states not in $\mathcal{I}$. Once an agent chooses an option, that option's policy is followed till termination.

Options are generated by merging policies (logical AND). The resultant policy has a $null$ action for a state s if the actions of the merging policies differ. If the actions of the merging policies for a state s are identical, then the resulting policy has the same action for the state s.

The initial list of candidate options are obtained from L. The next iteration of candidate options are generated by merging random subsets of L. Each candidate option is scored according to the number of non $null$ actions and the number of times it occurs in L. The option with the highest score is selected and added to the set of options $\mathcal{O}$, and options similar to the selected option are removed from $\mathcal{O}$.

It is empirically shown in Pickett and Barto (2002) that agents using Policy-Blocks options learn faster than agents with no options.

3.4 Small World Options on Rooms Domain

Small-World ideas and Small-World Options were introduced in Section 2.3.

The Rooms domain is a 2-D grid where come cells are walls. The agent cannot move through walls. The objective of the agent is to reach a particular cell in the grid. An example map is given in figure 3.1 and example options in

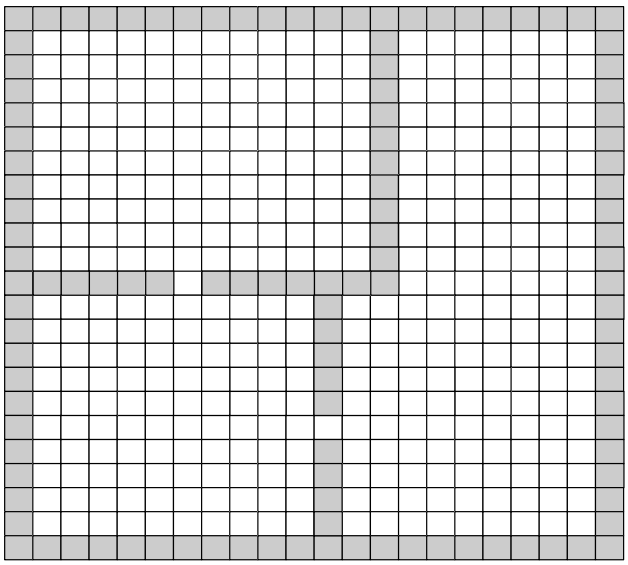


Figure 3.1: Rooms Domain: Map

the Rooms domain are shown in figure Figure 3.2.

Small world options have been evaluated on Rooms domain and the results are presented in Chaganty *et al.* (2012). Agents using small-world options outperformed agents using other option-generation algorithms such as betweenness based options in the Rooms domain.

In the Rooms domain, small-world options generated using a smaller exponent ($r \in [0, 1]$) had better performance than larger exponents.

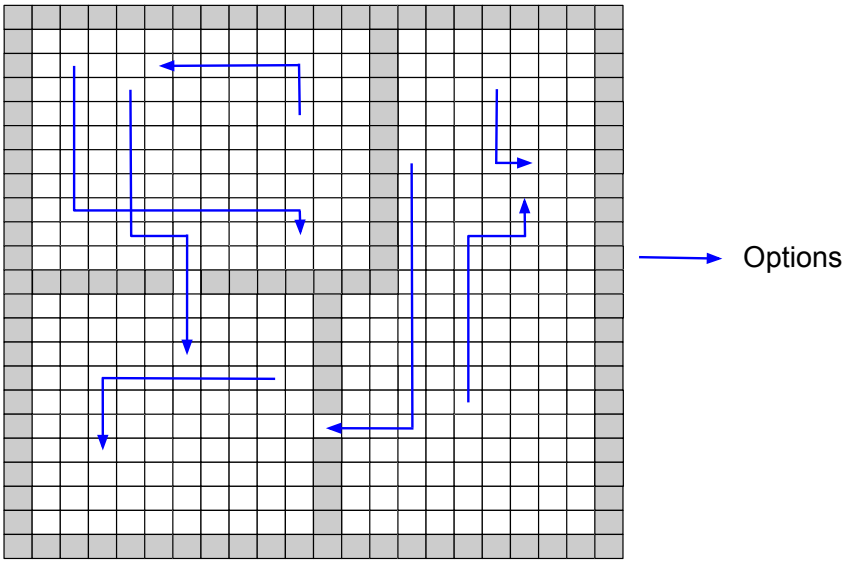


Figure 3.2: Rooms Domain: Options

CHAPTER 4

Domains Used

The complex domain which we use for empirically verifying the results of Chaganty *et al.* (2012) is the Automatic Guided Vehicle domain described briefly in Ghavamzadeh and Mahadevan (2003). The domain was picked in particular for the complexity of the task to be performed in order to earn a reward.

A simpler version of the same domain is experimented with in Karthik (2013).

The domain is now described in detail, following which the challenges faced in dealing with *large* and *complex* domains are explained.

4.1 Complex AGV Domain

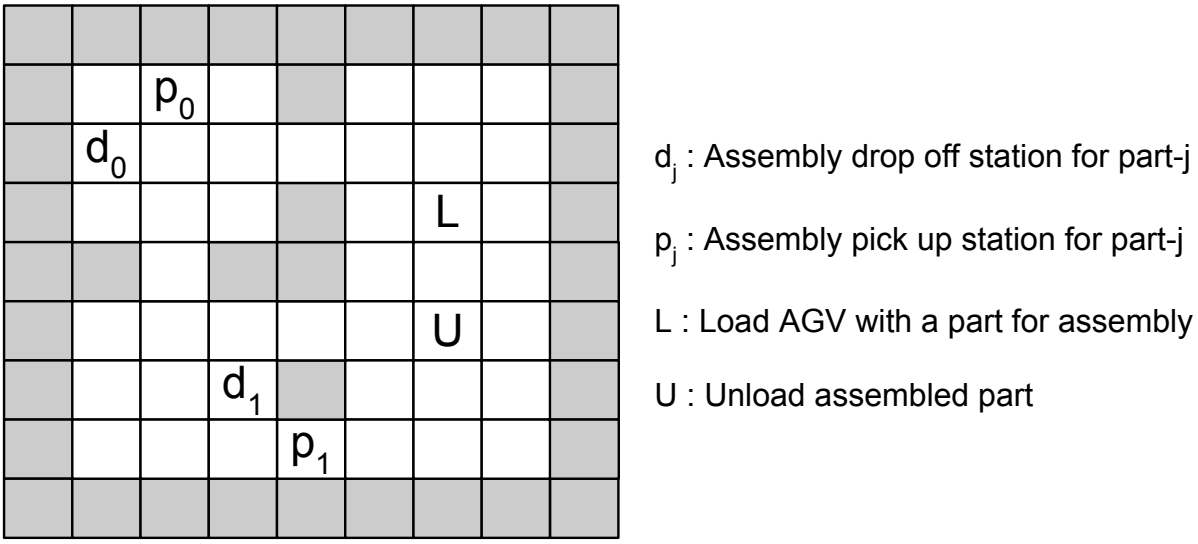


Figure 4.1: Complex AGV: Map

The domain consists of a map, containing a loading station, an unloading station and several pick-up and drop-off points - one for each type of part available in the domain. The gray cells are walls where the agent is not allowed to go. Each component of the domain is described in detail below.

L The loading station generates one of the n_{mach} types of parts picked uniformly at random. The time-delay t_{gen} between generation of successive parts is also a random variable uniformly distributed in $[1, gen_{time}]$.

d_k The drop-off point for the k^{th} machine-part. Once a part has been dropped, its assembly begins. The amount of time needed for assembly is a a random variable uniformly distributed in $[1, a_{time}]$. After the part is assembled, it is available for pick-up at the corresponding pick-up point p_k.

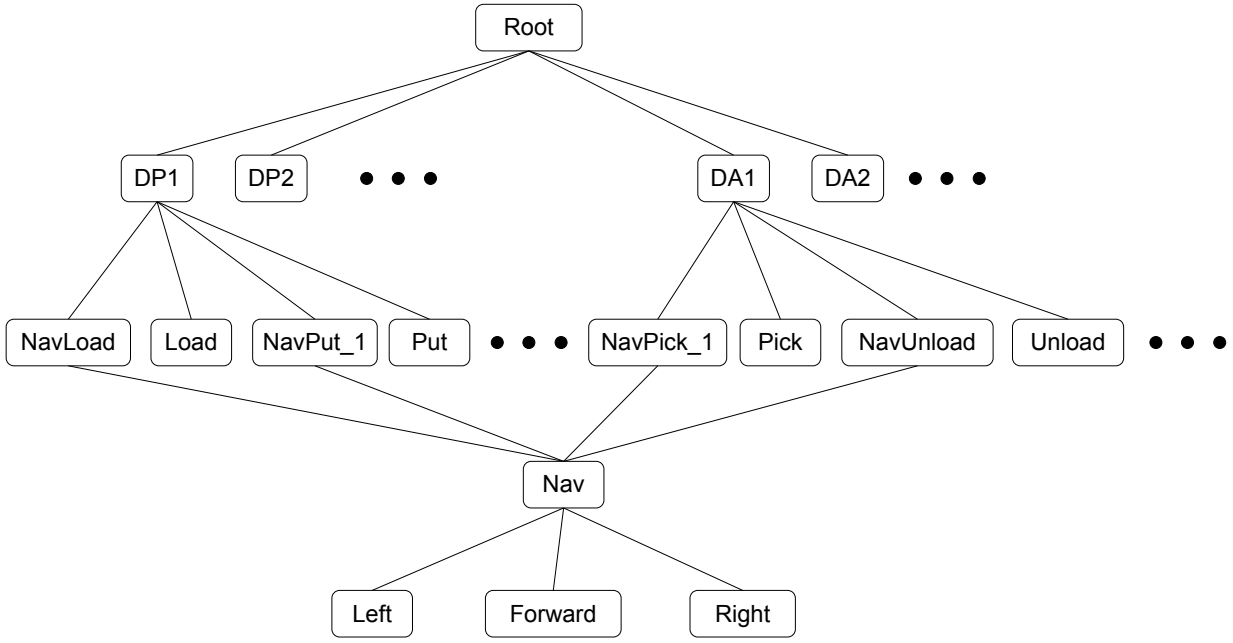
p_k The pick-up location for the k^{th} part, which can be picked up after its assembly.

U The unload-station. The agent must drop an assembled part here to earn a reward, after which the episode ends.

The environment gives a reward for dropping an assembled part at the unloading station. In addition, every other action yields a step-reward of -1.0, in order to encourage discovery of short paths.

Since the entire state of the environment, including the time intervals since the last part-generation, since a part was dropped for assembly at each assembly station etc. needs to be stored in the state, the domain consists of a large number of states. If the map is of size y_{max} x x_{max}, the number of states in this domain is $y_{max}.x_{max}.2.(1 + gen_{time}).(2.n_{mach} + 1).(1 + a_{time})^{n_{mach}}.2^{n_{mach}}.2$. The map which was used in our experiments consisted of $30k$ states.

The task to be performed by the agent is depicted schematically in Figure 4.2. The domain's semantics are complex, with the task to be performed



DP_i : Drop part at assembly drop-off station i
DA_i : Pick up assembled part at assembly pick up station i
NavLoad : Navigate to the loading station
NavPut_i : Navigate to assembly drop-off station i
NavPick_i : Navigate to Pick up station i
NavUnload : Navigate to unload station

Figure 4.2: Complex AGV: Subtasks

consisting of picking up a part, dropping it at the correct assembly drop-off point and then picking it up and dropping it at the unloading station.

Also, since the response of the environment is very probabilistic, with the part generated, t_{gen} and assembly time being random variables, the domain's dynamics is also complex.

4.2 Challenges with Large and Complex Domains

Large and complex domain pose many conceptual and implementation challenges. Some challenges that were faced during this work are described below.

1. Since the dynamics and semantics of the domain are both complex, a large budget of epochs is needed for the agent to learn good policies and generate good options.

2. Much of the book-keeping by the environment and the agent requires space $\propto |S|$ and some require space $\propto |S|^2$, where S is the state-set. Thus, start sets, end sets, rewards and transition-probabilites cannot be stored directly, but need to be computed when required. A similar problem arises when storing Q-values.:

3. Each option takes space $\propto |S|$. Since at least one option is generated from each state, $|S|^2$ amount of space is required.

4. In large domains, Reinforcement Learning agents face the problem of learning policies well only for states that are on or near the path from the start-state to the goal. Having a large-epoch budget does not solve this issue.

   Hence, the agent was teleported to a random location on the map at the end of each episode. This leads to the exploration of a larger portion of the state-space. Consequently, the epoch-budget needed also increases, as good policies now need to be learned for more states.

5. Since the semantics of the task to be performed is complex, the agent might never learn it. The issue is that the final reward for unloading an assembled part propagated very slowly to each state along the load-unload path. Thus, while the agent improved its policy across load-unload cycles, the learning during a load-unlaad cycle was limited.

To encourage the agent to learn faster, one popular method is to provide intermediate rewards for performing subtasks leading to the goal. Thus, rewarads were added for dropping a part at the assembly drop-off point and for picking it up from the pick-up point.

While this encourages the agent to move in the right direction, there is the danger that the agent might get stuck in a loop, continuously performing a subtask and earning the intermediate reward, while making no progress towards solving the original task.

We experimented with this approach and decided not to use it, as (i) a large epoch-budget was being used anyway to deal with teleportation to random start-states, and (ii) it is considered good practice not to use intermediate rewards.

CHAPTER 5

Option Generation Schemes

In this chapter, we explore the results of an option generation scheme which we proposed and implemented. The proposed scheme overcomes Chaganty *et al.* (2012)'s limitation of needing complete knowledge of the environment's MDP while generating options. This is done by having the agent keep track of a model of the environment, which serves as an approximation to the actual MDP.

While a brief description of the final option-generation scheme proposed by us follows, a complete description is provided in Karthik (2013). In addition, Karthik (2013) also describes slightly different schemes in which the agent's model of the environment is different. The other schemes each suffer from drawbacks, which led us to propose the scheme described below.

5.0.1 Option-generation Scheme: SAS Transitions + Modified Greedy Policy

As defined in Section 2.2, an option is defined by its start set $\mathcal{I}$, policy π and terminating condition β. The terminating condition can also be a single state or a set of states.

To generate options, an agent is first run for a certain number of eposides. A greedy policy is extracted from the agent and is used to generate options. The algorithm is described below in Algorithm 1, and the exact method of selecting

destinations and setting the option's policy are described later on.

Algorithm 1: Agent-generated Options

Data: Environment E, Number of goals g, Episode budget k

Result: Options O

```
O = {};
for i from 1 to k do
    Reset the environment E and its goals;
    Run an agent for k/g episodes;
    π_g = Agent's greedy policy, from Equation (2.6);
    for state s in E.states do
        Choose destination state d;
        O = O ∪ { option( {s}, π_g, {d} ) };
return O;
```

The destination state of an option is selected using Kleinberg's small-world probability distribution [1] from among the states in the agent's greey path from the start of the option to the goal. This path is constructed from the information on $\langle state, action, state \rangle$ transition counts ($count_{SAS}$), which is kept track of by the agent. The algorithm is described in Algorithm 2.

Also, the greedy policy as determined by the agent's Q-values, as shown in Equation (2.6) is augmented by by the SAS counts. The greedy action in state t is now defined as the action which is most taken from that state, given by Equation (5.1).

$$\pi(t) = argmax_a\Big(\Big\{\sum_u count_{SAS}(t,a,u)\Big\}\Big) \tag{5.1}$$

We marginalize $count_{SAS}(t,a,u)$ over u and find the action a which max-

[1] Given a source s, the probability that the destination is d is $\propto dist(s,d)^{-r}$

imises $count_{SA}(t, a)$.

Algorithm 2: Greedy path to the goal

Data: State s

Result: Get a greedy path from state s to the goal

```
state n = s;
path=[n];
while not at goal and not looping do
    Best action a_b = π(n);
    Next state n = argmax_v({c_{n,v,b} ∀v ∈ nbr(n)});
    path.append(n);
return path;
```

In summary, the following is the way of picking the destination d and the policy π of the small-world option described in Section 2.3.2.

Candidate Destination States: On the path from s to goal g. The path to the goal is constructed using a count of $\langle state, action, state \rangle$ transitions seen by the agent, as described in Algorithm 2.

Policy: Greedy Policy of the Agent modified by $\langle state, action, state \rangle$ transitions, as desecribed in Equation (5.1).

This greedy policy will follow the path to the goal constructed as both of them use counts of $\langle state, action, state \rangle$ transitions. This option generation scheme was found to work without any errors or loops, as is described in detail in Karthik (2013).

Some experimental results using this scheme are examined in the next section.

5.1 Experimental Results: Complex AGV Domain

Small-world options were generated on the Complex AGV domain and the resulting options were used by an agent to accumulate rewards. The parameters of the experiment were as follows:

The agent was run for 2e6 episodes with the following parameters and the options thus generated were used by different agents to evaluate them.

- γ (reward decay rate): 0.995
- α (learning rate): 0.1
- ϵ (of ϵ-greedy policy): 0.1
- Per-step reward : -1.0
- Reward for Success : 125 * partNumber
- Rate of decay of ϵ
  - 0, when learning options ie., no decay.
  - 0.01, when running the agent When evaluating performance, as the agent learns, it becomes more greedy to reduce the variance in its reward. This is the reason for the decaying ϵ.

The options were generated using different exponents r from 0 to 1 in steps of 0.1 and from 1 to 5 in steps of 0.5.

For evaluating the performance, 20 agents were run for 4000 episodes each and performance was measured. The metric used for measuring performance is the cumulative discounted reward of an episode, which represents the throughput of the AGV.

As can be seen in Figure 5.1, the performance of the options is poor for large exponents, as the generated options are too short for being of use. Also, there

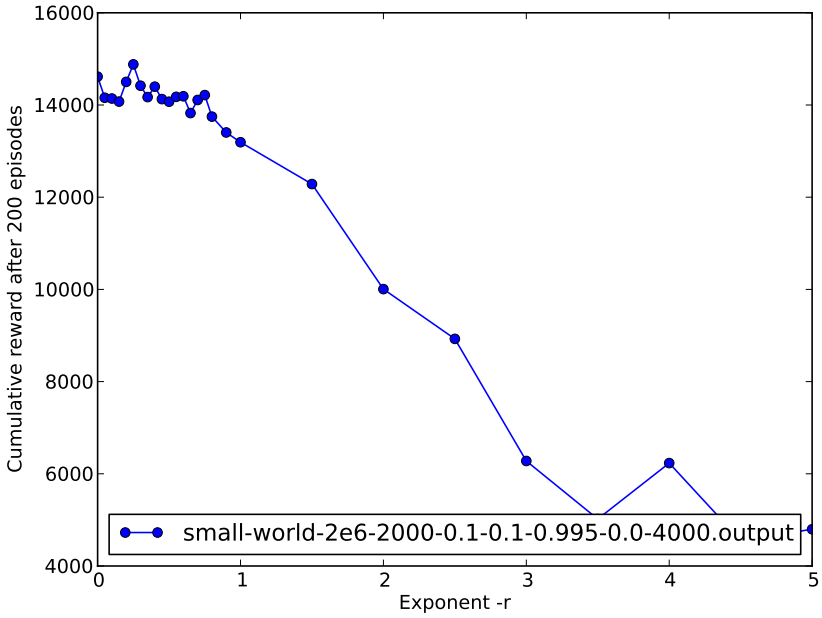


Figure 5.1: Dependence of performance of 2000 Small-World Options on Exponent

is peak performance for an exponent in $(0, 1)$, which corresponds to the correct small-world exponent for the MDP. These validate the fact that the option-generation scheme proposed in Chaganty *et al.* (2012), in its suitably modified form for not requiring the complete MDP, indeed gives performance benefits for complex domains like the Complex AGV.

However, there are two anomalous behaviours seen.

1. Unlike expected, an exponent of 0 does not perform significantly poorer than other exponents in $(0, 1)$, as seen in Figure 5.1. This behaviour is expected because a zero exponent disregrards distances while picking the destination of an option.

2. Using more options leads to strictly better performance, as seen in Figure 5.2. This suggests that every option is contributing to the agent's navigation towards the goal.

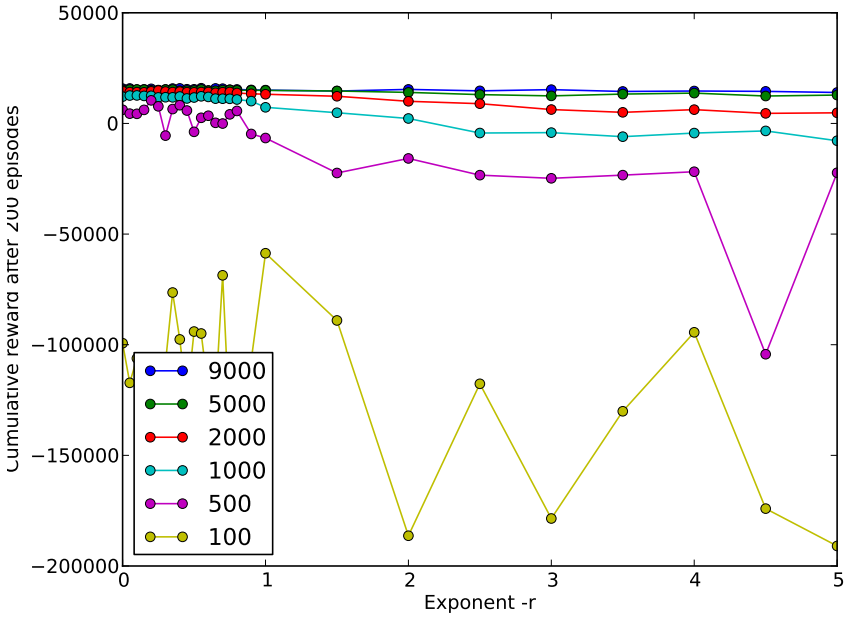


Figure 5.2: Dependence of performance of Small-World Options on Exponent and Number of Options

The explanation suggested for the anomalous behaviour is further validated by Figure 5.3. There is expected to be dip in cumulative reward as the agent learns a good policy. While this dip is observed for a small number of options, there is no dip for a large number of options. This strengthens the suspicion that all options being goal directed may be the cause for the anomalous behaviour.

5.2 Limitations of Current Option-generation Schemes

The option-generation schemes proposed in Chaganty *et al.* (2012) and their practical counterparts as proposed in Chapter 5 suffer from the limitation that they make certain assumptions about the domain. They assume that the domain is such that goal states can be changed without affecting the semantics of the domain. While this may be true for simple domains like Rooms and

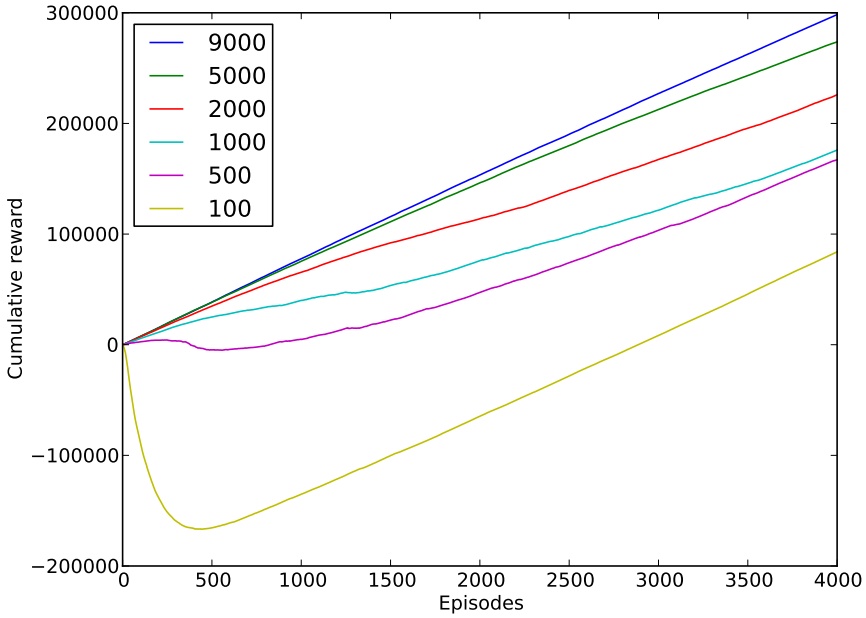


Figure 5.3: Dependence of Cumulative Reward of Small-World Options on Number of Options

Taxi, the assumption fails for domains with complex semantics, for example, the Complex AGV domain described in Chapter 4.

In the Complex AGV domain, there is a single goal state and it does not make sense to set an arbitrary state as the goal state. Thus, all the generated options are directed towards the domain's goal state. Consequently, using more options leads to a better performance, as all options are goal-directed and using more options increases the probability of reaching closer to the goal. This is the anomalous behaviour observed in Section 5.1.

This also explains the good performance of the exponent 0. Since all options are directed towards the goal, using any option whose destination was picked independently of distance is still better than using such options when all options aren't goal directed. In the latter case, some options in the exponent 0 case navigate the agent away from hte goal, thus leading to a dip in performance, whereas such behaviour doesn't occur in the current scenario.

Thus, there is a need to suitably modify the option-generation scheme in order to have options directed in different directions. Our proposed scheme is described in Chapter 6.

CHAPTER 6

Partition-based Option-generation

As described in Chapter 5.2, the assumption made my the studied option-generation methods is the relocatability of the goal state. While this may not make sense on the entire domain, it does make sense if states are separated into semantically related partitions that repsent subtasks to be performed by the agent. This reasoning can clearly be seen in the graphical representation of a Rooms domain shown in Figure 6.1, where rooms (squares), doors (bottlenecks) and corridors (chains) can clearly be identified.

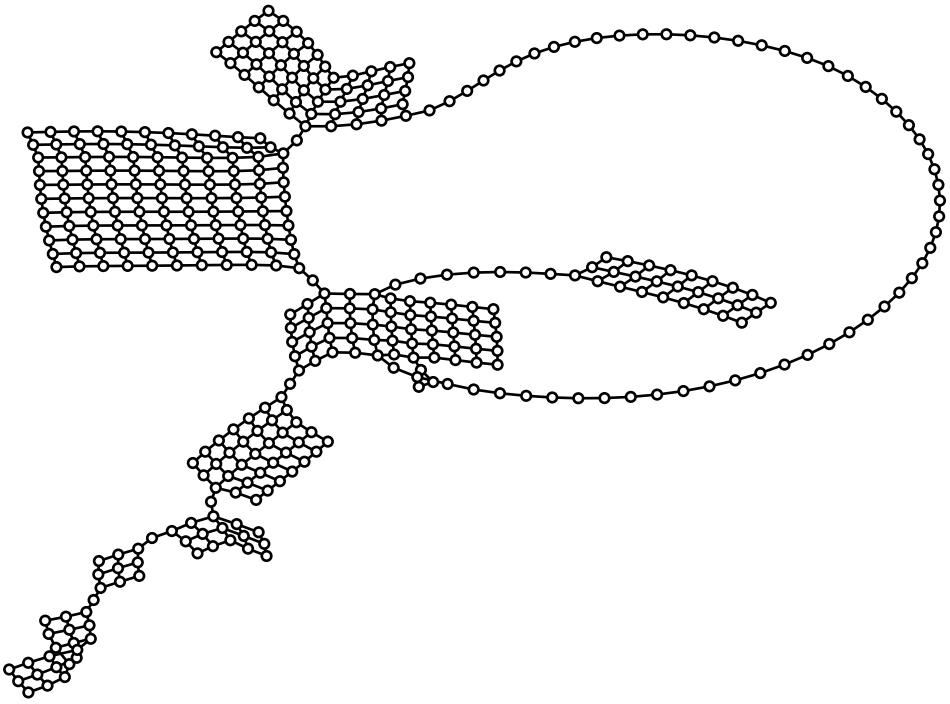


Figure 6.1: Rooms Domain: Graphical Representation

The idea is to use a graph partitioning algorithm to partition the state-space into multiple partitions. Each partition is expected to represent a subtask to be performed by the agent and be a collection of semantically related states. A partition is analogous to a room in the Rooms domain, and it makes sense to have arbitrary goal-states within a partition.

The partitioning algorithm used is Spectral Clustering, as proposed in Kumar *et al.* (2013). The results of using the algorithm on the rooms domain shown in Figure 6.2 are shown in Figure 6.3.

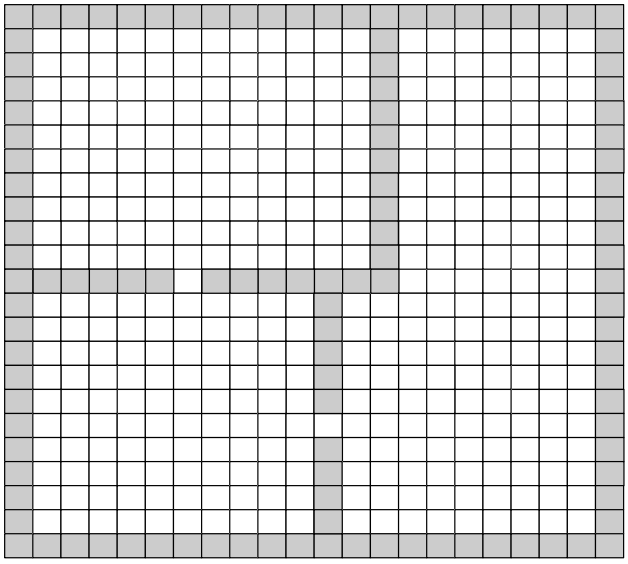


Figure 6.2: Example Rooms Domain: Before Partitioning

6.1 Option-generation Scheme

Since goal states can be set arbitratily in the partitions, our approach learns small-world options to multiple goal states within each partition. The agent gets a step-reward of -1.0 and a high reward for reaching the goal state. The agent is prevented from leaving the partition. The small-world options thus learnt are used for navigating within a partition. The algorithm is described in

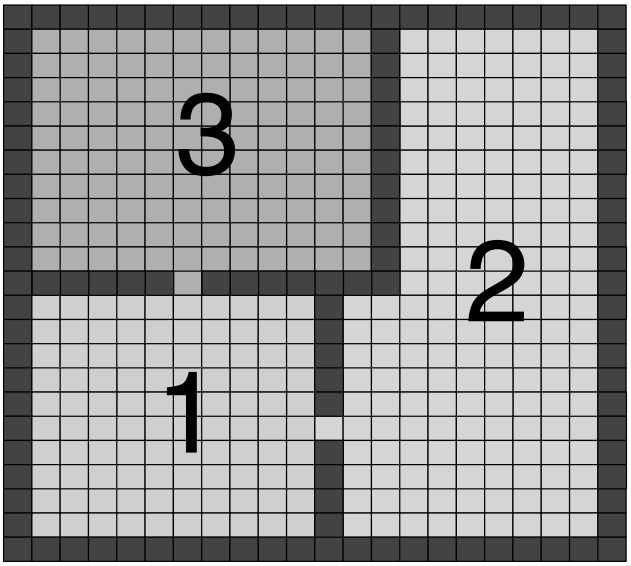


Figure 6.3: Example Rooms Domain: Partitions

Algorithm 3.

Algorithm 3: Small-world options in Partitions

Data: MDP P, number of goals k

Result: Small world options for MDP P

```
Partition P using Spectral Clustering;
for each partition do
    for i in 1..k do
        Set an arbitrary goal state g;
        Learn small-world options to g;
return options learnt;
```

While this leads to efficient navigation within partitions, the problem of navigation between partitions still exists. This is solved by adding a certain number of betweenness-based options to the option set.

Betweenness centrality is a measure of a node's centrality in a network. It is equal to the number of shortest paths from all vertices to all others that pass through that node. The betweenness-based options thus help an agent to navigate to "bottleneck" states which connect well-connected components of

the graph, which correspond to the boundaries between the partitions found by spectral clustering.

The algorithm for finding betweenness-based options is described in Algorithm 4.

Algorithm 4: Betweenness-based Options

Data: MDP P

Result: Betweenness-based options for MDP P

b_{max} = Betweenness Maxima in P, above a threshhold;

for *state* s *in* b_{max} **do**

    Learn options to s;

return options learnt;

The full algorithm that uses both small-world options and betweenness-based options is described in Algorithm 5.

Algorithm 5: Partition-based Options

Data: MDP P, number N, fraction f

Result: Options for MDP P

options = { $(1-f).N$ options generated by Algorithm 3 };

options = options $\cup$ { $f.N$ options generated by Algorithm 4 };

return options;

An illustrative example of the options generated by Algorithm 5 is shown in Figure 6.4.

6.2 Experimental Results: Rooms Domain

The Rooms domain is described in Chapter 3. Partition-based options were generated and used for running an agent on a large Rooms domain. The performance of 15% betweenness-based options and 85% small-world options, using a total of 125 options, is shown in Figure 6.6. As expected, the zero expo-

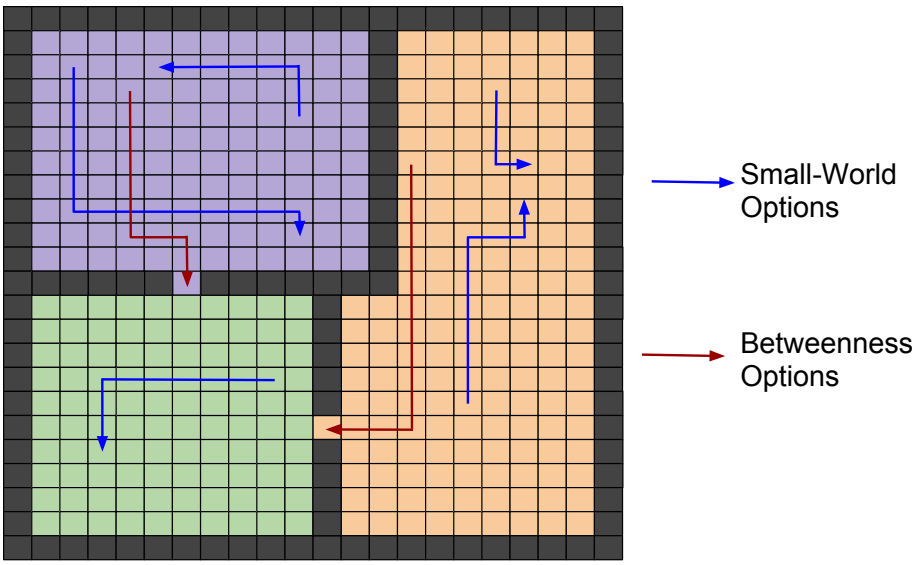


Figure 6.4: Example Rooms Domain: Illustrative Examples of Options

nent performs poorly and there is peak performance for an exponent in $(0, 1)$.

The need for betweenness-based options for navigating between the various partitions of the state-space can be seen from Figure 6.7. The dotted curves show the case when 15% betweenness-based options were used. It can be seen that using the betweenness-based options along with partition-based options results in significantly higher cumulative rewards than using partition-based options alone.

The graphical representation of the MDP of the domain used is shown in Figure 6.5. Features like rooms and connecting doors are clearly observable in the graph.

More tests of the scheme on other domains are described in Karthik (2013). Those tests also show that the anomalous behaviour of better performance with more options doesn't occur with partition-based options. Thus, proposed approach addresses both the limitations of the earlier approaches.

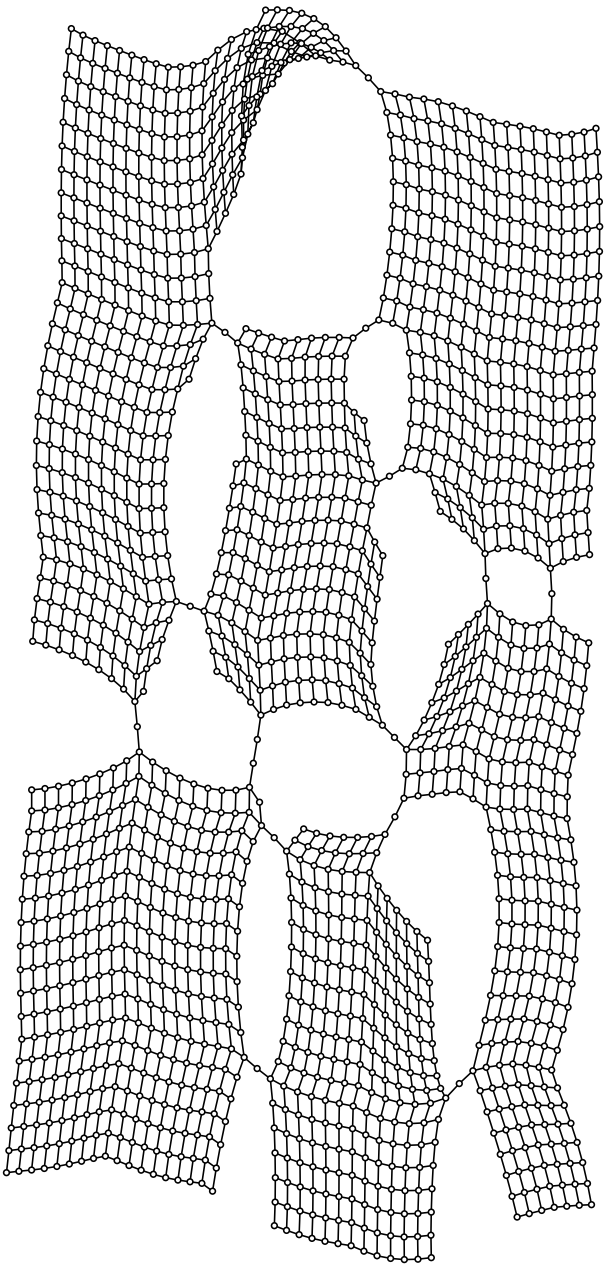


Figure 6.5: Graphical Representation of a large Rooms Domain's MDP

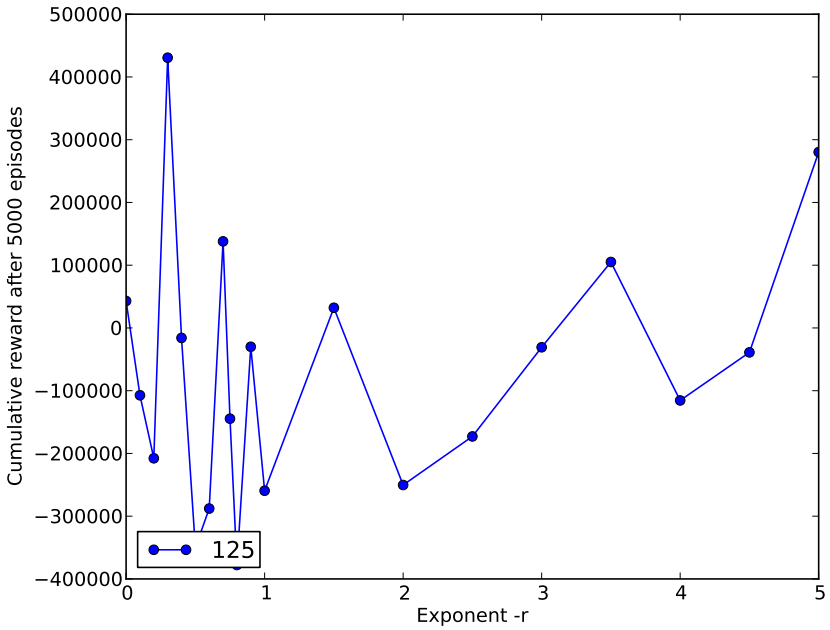


Figure 6.6: Dependence of performance of Partition-based options on Exponent in the Rooms Domain

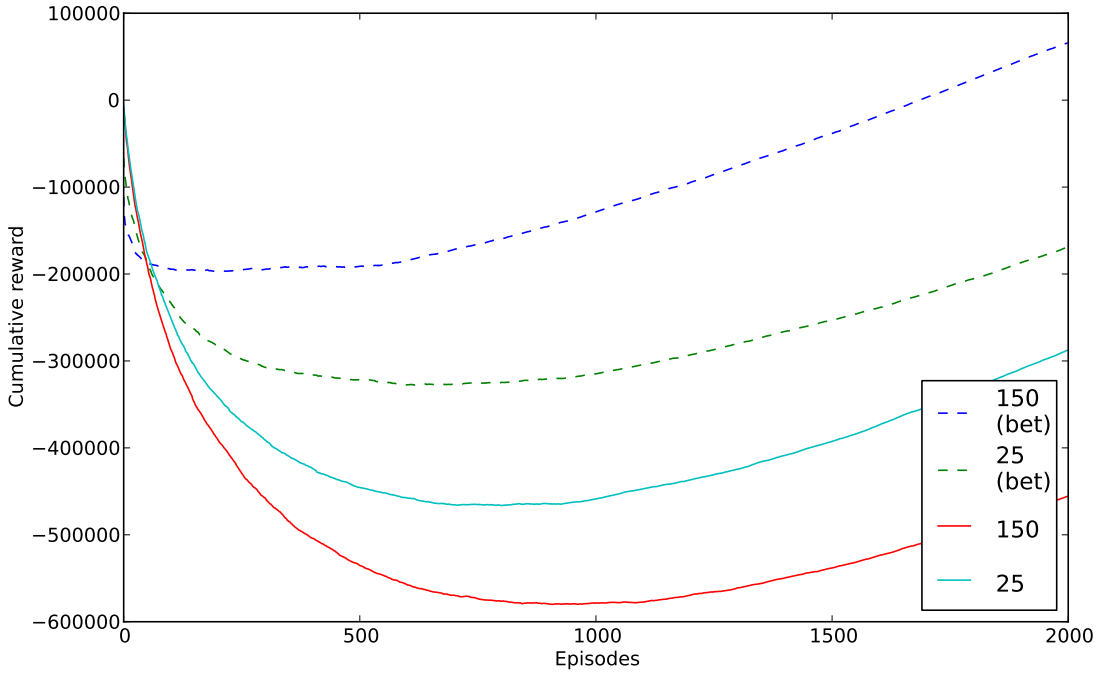


Figure 6.7: Comparison of cumulative reward for partition-based options with and without 15% betweenness-based options

CHAPTER 7

Conclusions and Future Directions

7.1 Complex Domains

We empirically verified the efficiency of using small-world options in complex domains, such as the simple and more complicated versions of the Automatic Guided Vehicle domains. It was observed that:

1. An exponent of 0 performed poorer than some non-zero exponents.
2. Peak performance occurred for an exponent in $(0, 1)$.
3. Larger exponents i.e., shorter options performed poorly.

7.2 Practical Option-generation Schemes

Option-generation schemes like those in Chaganty *et al.* (2012), but which do not require complete knowledge of the MDP were proposed and were shown to perform well. The exact behaviour as described in Section 7.1 was seen when using these schemes.

However, it was observed that an exponent of 0 performed much better than in the optimal case. The reasons for the anomalous behaviour were examined and isolated and a new option-generation scheme was proposed which overcame the said limitation.

7.3 Partition-based Option-generation

An option-generation scheme which uses small-world options for navigating within partitions and betweenness-based options for navigating between partitions was proposed and examined. The specific limitations of previous methods were addressed.

7.4 Future Work

A mathematical analysis of the proposed option-generation schemes hasn't been performed. It is as yet not proved that using small-world options results in faster learning when compared to random-options (exponent 0), though our results suggest that to be the case empirically. A mixing-time analysis could yield formal insights into the speed of learning.

APPENDIX A

SMALL WORLDS

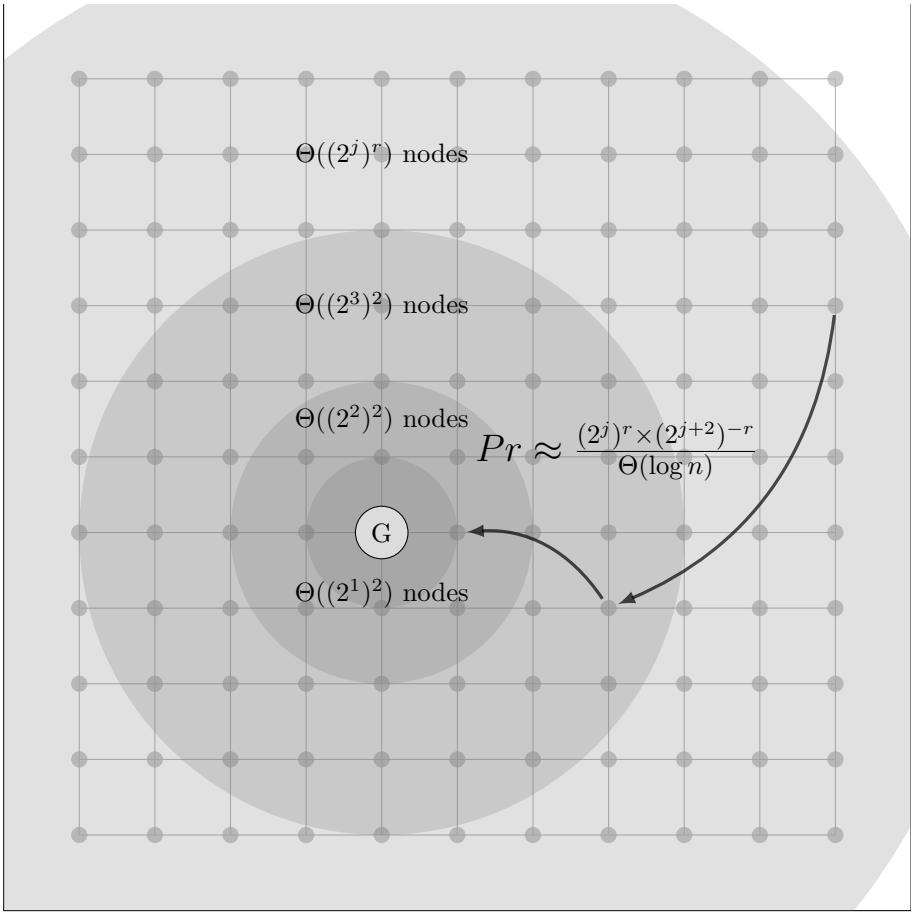


Figure A.1: Exponential Neighbourhoods

In this section we will tackle the proof of the main theorem in section 2.3.

Theorem 1. *Let $f : V \to \mathbb{R}$ be a function embedded on the graph $\mathcal{G}(V, E)$, such that, $\kappa_1||u - v|| - c_1 \leq ||f(u) - f(v)|| \leq \kappa_2||u - v|| - c_2$, where $0 \leq \kappa_1 \leq \kappa_2$, and $0 \leq c_1 \leq c_2$. Let M_f be the global maxima of f . Let $\mathcal{GA}_\epsilon$ be an ϵ-greedy algorithm with respect to f , i.e. an algorithm which chooses with probability $1 - \epsilon$ to transit to the neighbouring state closest to M_f , i.e. $N(u) = argmin_v||f(v) - f(M_f)||$.*

If $\mathcal{G}(V, E)$ is r-dimensional lattice, and contains a long distance edge distributed according to P_r : p(u, v) $\propto ||u - v||^{-r}$, then $\mathcal{GA}_\epsilon$ takes $\mathrm{O}\big((\log |V|)^2\big)$ steps to reach M_f.

Proof. This result is a simple extension of Kleinbergs result in Kleinberg (2000), and follows the proof presented there, albeit with the somewhat cleaner notation and formalism of Martel and Nguyen (2004). We begin by defining the necessary formalism to present the proof.

Definition 2. *Let us define $B_l(u)$ to be the set of nodes contained within a "ball" of radius l centered at u, i.e. $B_l(u) = \{v \mid ||u - v|| < l\}$, and $b_l(u)$ to be the set of nodes on its surface, i.e. $b_l(u) = \{v \mid ||u - v|| = l\}$.*

Given a function $f : V \to \mathbb{R}$ embedded on $\mathcal{G}(V, E)$, we analogously define $B_l^f(u) = \{v \mid |f(u) - f(v)| < l\}$. For notational convenience, we take B_l^f to be $B_l^f(M_f)$.

The inverse normalised coefficient for $p(u, v)$ is,

$$\begin{aligned} c_u &= \sum_{v \neq u} ||u - v||^{-r} \\ &= \sum_{j=1}^{r(n-1)} b_j(j) j^{-r} \end{aligned} \tag{A.1}$$

It can easily be shown that the $b_l(u) = \Theta(l^{k-1})$. Thus, c_u reduces to a harmonic sum, and is hence equal to $\Theta(\log n)$. Thus, $p(u, v) = ||u-v||^{-r}\Theta(\log n)^{-1}$.

We are now ready to prove that $\mathcal{GA}$ takes $\mathrm{O}\big((\log |V|)^2\big)$ decisions. The essence of the proof is summarised in figure A.1. Let a node u be in phase j when $u \in B_{2^{j+1}}^f \; B_{2^j}^f$. The probability that phase j will end this step is equal to the probability that $N(u) \in B_{2^j}^f$.

The size of $B_{2^j}^f$ is at least $|B_{\frac{2^j+c_2}{\kappa_2}}| = \Theta(\frac{2^j+c_2}{\kappa_2})$. The distance between u and a node in $B_{2^j}^f$ is at most $\frac{2^{j+1}+c_1}{\kappa_1} + \frac{2^j+c_2}{\kappa_2} < 2\frac{2^{j+1}+c_2}{\kappa_2}$. The probability of a link

between these two nodes is at least $(\frac{2^{j+2}+2c_1}{\kappa_1})^{-r}\Theta(\log n)^{-1}$. Thus,

$$\begin{aligned} P\Big(u, B^f_{2^j}\Big) &\geq \frac{1-\epsilon}{\Theta(\log n)}\Big(\frac{2^j+c_2}{\kappa_2}\Big)^r \times \Big(\frac{2^{j+2}+2c_1}{\kappa_1}\Big)^{-r} \\ &\geq \frac{1-\epsilon}{\Theta(\log n)}\Big(\frac{\kappa_1}{4\kappa_2}\Big)^r \times \Big(\frac{1+\frac{c_2}{2^j}}{1+\frac{c_1}{2\times 2^j}}\Big)^r \\ &\geq \frac{1-\epsilon}{\Theta(\log n)}\Big(\frac{\kappa_1}{4\kappa_2}\Big)^r \times \Big(\frac{1+c_2}{1+\frac{c_1}{2}}\Big)^r \end{aligned} \tag{A.2}$$

Let number of decisions required to leave phase j be X_j . Then,

$$\begin{aligned} E[X_j] &\leq \sum_{i=0}^{\infty}\Big(1-P\Big(u, B^f_{s^j}\Big)\Big)^i \\ &\leq \frac{1}{P(u, B^f_{2^j})} \\ &\leq \Theta(\log n)\frac{1}{(1-\epsilon)}\Big(\frac{4\kappa_2}{\kappa_1}\Big)^r\Big(\frac{1+\frac{c_1}{2}}{1+c_2}\Big)^r \\ &\leq \Theta(\log n). \end{aligned} \tag{A.3}$$

Thus, it takes at most $\mathrm{O}\big(\log n\big)$ decisions to leave phase j. By construction, there are at most $\log n$ phases, and thus at most $\mathrm{O}\big((\log n)^2\big)$ decisions.

□

REFERENCES

1. **Chaganty, A.**, **P. Gaur**, and **B. Ravindran** (2012). Learning in a small world. *AAMAS*.

2. **Şimşek, O.** and **A. G. Barto** (2008). Skill characterization based on betweenness, 1–8.

3. **Ghavamzadeh, M.** and **S. Mahadevan** (2003). Hierarchical average reward reinforcement learning.

4. **Hengst, B.** (2002). Model Approximation for HEXQ Hierarchical Reinforcement Learning, 144–155.

5. **Karthik, M. V.** (2013). *Small-World Options in Reinforcement Learning*. B.tech. thesis.

6. **Kleinberg, J.** (2000). The Small-World Phenomenon : An Algorithmic Perspective. *ACM Theory of Computing*, **32**, 163–170.

7. **Kumar, P.**, **L. Niveditha**, and **B. Ravindran** (2013). Spectral clustering as mapping to a simplex.

8. **Martel** and **Nguyen** (2004). Analyzing kleinberg's (and other) small-world models. *PODC*, 179–188.

9. **McGovern, A.** and **A. G. Barto** (2001). Automatic Discovery of Subgoals in Reinforcement Learning using Diverse Density, 1–8.

10. **Menache, I.**, **S. Mannor**, and **N. Shimkin** (2002). Q-Cut - Dynamic Discovery of Sub-Goals in Reinforcement Learning, 295–306.

11. **Pickett, M.** and **A. G. Barto** (2002). Policyblocks: An algorithm for creating useful macro-actions in reinforcement learning, 506–513.

12. **Precup, D.** and **M. Stolle** (2002). Learning Options in Reinforcement Learning, 212–223.

13. **Sutton, R. S.**, **D. Precup**, and **S. Singh** (1999). Between MDPs and Semi-MDPs : Learning , Planning , and Representing Knowledge at Multiple Temporal Scales at Multiple Temporal Scales. *Artificial Intelligence*, **112**, 181–211.

14. **Taylor, M. E.** and **P. Stone** (2009). Transfer Learning for Reinforcement Learning Domains: A Survey. *Journal of Machine Learning Research*, **10**, 1633–1685.

15. **Thrun, S.** and **A. Schwartz** (1995). Finding Structure in Reinforcement Learning, 385–392.