

# Learning in a Small World

Paper 280

## ABSTRACT

Understanding how we are able to perform a diverse set of complex tasks has been a central question for the Artificial Intelligence community. One popular approach is to use temporal abstraction as a framework to capture the notion of subtasks. However, this transfers the problem to finding the right subtasks, which is still an open problem. Existing approaches for subtask generation require too much knowledge of the environment, and the abstractions they create can overwhelm the agent. We propose a simple algorithm inspired by small world networks to learn subtasks while solving a task that requires virtually no information of the environment. Additionally, we show that the subtasks we learn can be easily composed by the agent to solve any other task; more formally, we prove that any task can be solved using only a logarithmic combination of these subtasks and primitive actions. Experimental results show that the subtasks we generate outperform other popular subtask generation schemes on standard domains.

## Categories and Subject Descriptors

I.2.6 [Artificial Intelligence]: Learning; I.2.8 [Artificial Intelligence]: Problem Solving, Control Methods and Search

## General Terms

Algorithms, Theory, Experimentation

## Keywords

reinforcement learning, options framework, social network analysis, small world phenomenon

## 1. INTRODUCTION

Reinforcement learning (RL) is a widely studied learning framework for autonomous agents, particularly because of its extreme generality; it addresses the problem of learning optimal agent behaviour in an unknown stochastic environment. In this setting, an agent explores a state space, receiving rewards for actions it takes; the objective of the agent is to maximise its rewards accumulated over time. However,

**Appears in:** *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2012)*, Conitzer, Winikoff, Padgham, and van der Hoek (eds.), June, 4–8, 2012, Valencia, Spain.

Copyright © 2012, International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). All rights reserved.

when scaling up to larger domains, these agents require prohibitively large amounts of experience in order to learn a good policy. By allowing the agent to exploit the structure of environment or task, we can reduce the experience required.

Structure can be imposed on a learning task through either spatial or temporal abstractions. With the former, the state-space is minimised using information about the symmetries present in the domain. Spatial abstractions have been surveyed in [6]. In the latter case, high-level actions are introduced which capture sequences of primitive actions. In this light, temporal abstractions capture the notion of a “subtask”. The most common approach for temporal abstractions is the options framework proposed by Sutton, Precup and Singh [11], and we build our work on this framework also. Work by Ravindran and Barto on relativised options [10] show how temporal abstractions can be combined with spatial abstractions. Both spatial and temporal abstractions play an important role in transfer learning, where we wish to extend optimal behaviour learnt in one task to another task; a survey of such techniques can be found in [13].

While options provide a broad framework for temporal abstraction, there is still no consensus on how to choose subtasks. The prevalent view is that subtasks should represent skills, i.e. partially defined action policies that constitute a part of many reinforcement learning problems [14]. For this reason, much of the existing work centres around identifying ‘bottlenecks’, regions that the agent tends to visit frequently [8], either empirically as in [8], or, more recently, using graph theoretic methods like betweenness centrality [2] or graph partitions [9]. The intuition is that options that navigate an agent to such states helps the agent move between strongly connected components, thus leading to efficient exploration.

These option generation schemes suffer from two serious drawbacks; (i) they either require complete knowledge of the MDP, or construct a local model from trajectories, a sample-heavy approach, and (ii) options to bottlenecks can be initiated at any state, leading to a blowup in the decision space, which might cause to actually take more time to learn the task as it sorts through the unnecessary options.

If one considered these options as additional edges to the bottleneck states, in the sense that a single decision is sufficient to transit the agent from a state, to the bottleneck, the resultant state-interaction graph would now be “more” connected. To highlight the importance of the connectivity of the state-interaction graph, consider the Markov chain induced by a policy for an Markov decision process. It is well known that the convergence rate of a Markov chain (mixing

time), is directly related to its conductance [4], and thus its algebraic connectivity.

Recognising the importance of connectivity, we try to apply concepts from Kleinberg’s work on small world networks, to the context of problem solving with autonomous agents. These graphs have been shown to have exceptionally high algebraic connectivity, and thus fast Markov chain mixing times [12]. In a small-world network, each node has one non-neighbouring edge, which connected to another node with a probability inversely proportional to the distance between them. With this simple construction, Kleinberg showed that an agent can discover a short path to any destination using only local information like the coordinates of its immediate neighbours [5]. In contrast, other graph models with a small diameter only state the existence of a short path, but do not guarantee that an agent would be able to find such a path.

In our context, we construct subtasks distributed according to the small world distribution as follows; create an option that will take the agent from a state  $s$  to another state  $s'$  with a probability inversely proportional to the distance between  $s$  and  $s'$ . We prove that this set of subtasks enables the agent to easily solve any task by using only a logarithmic number of options to reach a state of maximal value (Section 3). As this scheme adds at most one additional option per state, we do not explode the decision space for the agent.

Furthermore, in Section 4, we devise an algorithm that learns small world options from optimal policies learnt for only a few tasks in the domain. Thus not only are small world options effective to use, they are also simple to learn, and do not require any global analysis of the MDP. Experiments on several standard domains show that small-world options outperform bottleneck-based methods, and that small world options require far fewer learning epochs to be effective.

The remainder of the paper is organised as follows. We present an overview of reinforcement learning, and the options framework in Section 2. We then define a small world option, and prove that given such options, an agent will require to use only a logarithmic number of them to perform a task in Section 3. From a more practical perspective, we present an algorithm to extract these options from optimal policies learnt on several tasks in the domain in Section 4. We present our experimental results in Section 5. Finally, we conclude in Section 6, where we present future directions for our work. Appendix A contains an extension of Kleinberg’s proof for the distributed search property of small-world networks which is used in Section 3.

## 2. BACKGROUND

In reinforcement learning, the standard representation of an environment and task instance is a Markov decision process (MDP). An MDP can be represented as the tuple,  $\langle S, A, P, R, \gamma \rangle$ , where  $S$  and  $A$  are finite sets of states and actions,  $P : S \times A \times S \rightarrow [0, 1]$  describes the dynamics of the world through state-action transition probabilities,  $R : S \times A \rightarrow \mathbb{R}$  describes the task at hand by ascribing rewards for state transitions, and  $\gamma \in [0, 1]$  is a discount factor that weighs the value of future rewards.

In this setting, an agent in a state  $s \in S$  chooses an action  $a \in A$ , and moves to a state  $s'$  with probability  $P(s, a, s')$ , receiving a reward  $R(s, s')$ . The objective of the agent is to find a policy  $\pi : S \times A \rightarrow [0, 1]$ , i.e. a decision procedure for selecting actions, that maximises the reward it accumulates

in the long run,  $R = \sum_i \gamma^i r_i$ .  $R$  is also called the return.

We define the value function  $V : S \rightarrow \mathbb{R}$  to be the expected return from  $s$ , and  $Q : S \times A \rightarrow \mathbb{R}$  to be the expected return from  $s$ , after taking the action  $a$ . The optimal value function must satisfy the Bellman optimality equation,

$$\begin{aligned} V(s) &= \max_a R(s, a) + \gamma \sum_{s' \in S} P(s, a, s') V(s') \\ Q(s, a) &= R(s, a) + \gamma \sum_{s' \in S} P(s, a, s') \max_{a'} Q(s', a'). \end{aligned}$$

Given an optimal  $Q$ , an agent can construct an optimal policy,  $\pi(s, a^*) = 1$  when  $a^* = \operatorname{argmax}_a Q(s, a)$ , and 0 otherwise. In principle, if the agent knew the MDP, it could construct the optimal value function, and from it an optimal policy. However, in the usual setting, the agent is only aware of the state-action space,  $S$  and  $A$ , and must learn  $Q$  through exploration. The Q-learning algorithm learns  $Q$  with a simple update for every step the agent takes,

$$Q(s, a) = Q(s, a) + \alpha \left[ r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right],$$

where  $\alpha \in [0, 1]$  is a parameter that controls the learning rate. It has been shown that the Q-learning algorithm converges to the optimal value function in the limit with fairly permissive assumptions.

The options framework provide a temporal abstraction for subtasks. An option  $\langle \mathcal{I}, \pi, \beta \rangle$  is described by an initiation set  $\mathcal{I} \subset S$ , a policy  $\pi$ , and a terminating condition  $\beta$ . An agent can exercise an option in any state  $s \in \mathcal{I}$ , following which, it will follow the policy  $\pi$  described by the option, until the terminating condition  $\beta(s)$  is satisfied. The terminating condition  $\beta$  can be stochastic.

Several learning algorithms have been proposed for agents using options [11, 1]. One simple such method that we will use is MacroQ, a generalisation of the Q-learning algorithm described above. The MacroQ algorithm updates the value function only after completion of the option. If the option  $o$  was initiated in the state  $s$ , and continues for  $k$  steps before terminating in  $s'$ , the corresponding  $Q$  function update will be,

$$Q(s, o) = Q(s, o) + \alpha \left[ r + \gamma^k \max_{o' \in AUO} Q(s', o') - Q(s, o) \right].$$

Different tasks in the same domain can be described by different  $R$ . Let  $R$  be sampled from the family  $\mathcal{R}$ . Our objective then is to find a set of options  $O$  that reduces the expected learning time over  $\mathcal{R}$ .

*Example 1.* To make the discussion more tangible, let us look at an example, the Taxi domain, shown in Figure 1. The agent is a taxi navigating in this road-map. It must pick up a passenger at one of the 4 pads, R, G, B or Y. Subsequently, it must carry the passenger to a destination, which is also one of the above four pads. The states of the taxi would then be a tuple containing the location of the passenger (in one of the four pads, or within the taxi), the destination of the passenger, and location of the taxi in the map. The actions the taxi can perform are moving up, down, left or right in the map, as well as pick up or drop a passenger at a pad. Typical options for such a domain would be an option that can be started anywhere, and has a policy that takes the taxi to the one of the pads in the shortest possible manner. Such an option is generic, and

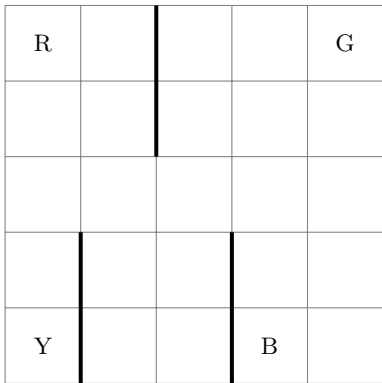


Figure 1: The Taxi Domain

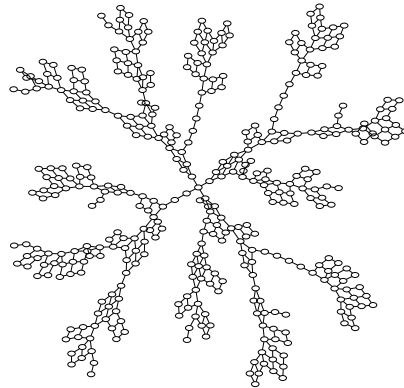


Figure 2: The State Space Graph for Taxi

does not depend on where the passenger or destination are. The RL agent must then learn to choose the right option when picking up the passenger.

### 3. SMALL WORLD OPTIONS

In Kleinberg’s small-world network model, each node  $u$  is given one ‘long-range’ edge to a node  $v$ , which was chosen with a probability  $P_r(u, v) \propto \|u - v\|^{-r}$ , where  $\|u - v\|$  denotes the distance between  $u$  and  $v$  in the graph. Similarly for each state  $s$ , we add a single ‘path option’ to another state  $s'$ , where  $s'$  is chosen with probability  $P_r(s, s') \propto \|s - s'\|^{-r}$ . A path option  $o_p(s, s')$  is an option with  $\mathcal{I} = \{s\}$ ,  $\beta = \{s'\}$ , and an optimal policy to reach  $s'$  for  $\pi$ . Intuitively, it is an option that takes the agent from  $s$  to  $s'$ . In practice, we may generate path options only for a subset of  $|S|$ . Note that while  $O(|S|)$  options are created, only one additional option is available in any state, and thus the decision-space for the agent is not significantly larger.

On an  $r$ -dimensional lattice,  $\mathcal{K}_r$ , the distance from any node  $u$  to a target node  $t$  is bounded by  $\|u - t\|$ , a quantity which is locally computable. When given long-range edges distributed according to  $P_r$ , Kleinberg showed that the greedy distributed algorithm  $\mathcal{GA}$  that chooses a neighbour  $v$  closest to  $t$  will reach  $t$  with an expected time  $O(\log(|V|)^2)$ . This follows as a trivial corollary of the following theorem,

**THEOREM 1.** *Let  $f : V \rightarrow \mathbb{R}$  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  $0 \leq c_2 \leq \frac{c_1}{2}$ . Let  $M_f$  be the global maxima of  $f$ . Let  $\mathcal{GA}_\epsilon$  be an  $\epsilon$ -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.  $N(u) = \operatorname{argmin}_v \|f(v) - f(M_f)\|$ .*

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

**PROOF.** The key insight of the proof is that with edges distributed according to  $P_r$ , there will always be edge within the neighbourhood of a node to an exponentially smaller neighbourhood of the target. Thus, the agent will only require to hop through  $\log |V|$  ‘neighbourhoods’. By bounding the time spent in each neighbourhood to  $\log |V|$ , we arrive at the result. We refer the reader to [Appendix A](#) for the complete proof.  $\square$

It is easy to construct a graph  $\mathcal{G}_M$  out of the state-space described by an MDP. The states  $S$  become the nodes of the graph, and actions  $A$  become the edges, with the transition probabilities as weights. The edges can also be attributed with the rewards described by  $R$ . Options can be viewed to be paths along the graph. As an example, the Taxi domain defined earlier translates to the graph shown in [Figure 3](#).

Consider an MDP  $M_{\mathcal{K}_r}$  with states connected in a  $r$ -dimensional lattice, and noisy navigational actions between states. We claim that using path options distributed according to  $P_r$ , an  $\epsilon$ -greedy agent can reach a state of maximal value using  $O(\log(|S|)^2)$  options. Clearly, the value function  $V$  is a local property of the state. Thus, if we can relate  $V$  to the distance from the target state, we can apply [Theorem 1](#).

*Definition 1.* A robust path option  $o(u, v)$ , where  $u, v \in S$  is an option that takes the agent from  $u$  to  $v$  ‘robustly’, in the sense that in each epoch, the agent moves closer to  $v$  with a probability  $1 - \epsilon > \frac{1}{2}$ .<sup>1</sup> Note that this  $\epsilon$  includes any environmental effects as well.

The following lemma shows that  $V$  satisfies the properties of an embedded function required for [Theorem 1](#).

**LEMMA 1.** *Let  $o(u, v)$  be the preferred option in state  $u$ , and let  $\|u - v\|_V = |\log V(v) - \log V(u)|$ . Then,*

$$k_1 \|u - v\| - c_1 \leq \|u - v\|_V \leq k_2 \|u - v\|,$$

where  $k_1 = \log \frac{1}{\gamma}$ ,  $k_2 = \log \frac{1}{(1-\epsilon)\gamma}$ , and  $c_1 = \log \frac{1}{1-\gamma}$ .

**PROOF.** For notational convenience, let  $\bar{\epsilon} = 1 - \epsilon$ . From the Bellman optimality condition, we get the value of  $o(u, v)$  to be,

$$Q(u, o(u, v)) = E_l \left[ \gamma^l V(v) + \sum_{i=1}^l \gamma^{i-1} r_i \right],$$

where  $l$  is the length of the option, and  $r_i$  is the reward obtained in the  $i$ -th step of following the option.

If  $o(u, v)$  is the preferred option in state  $u$ , then  $V(u) =$

<sup>1</sup>This condition is equivalent to saying that the option takes the agent from  $u$  to  $v$  in finite time, and hence is not particularly strong.

$Q(u, o(u, v))$ . Using the property that  $0 \leq r_i \leq 1$ ,

$$\begin{aligned} \text{E}_l[\gamma^l V(v)] &\leq V(u) \leq \text{E}_l[\gamma^l V(v) + \sum_{i=1}^l \gamma^{i-1}] \\ \text{E}_l[\gamma^l] V(v) &\leq V(u) \leq \text{E}_l[\gamma^l] V(v) + \frac{1}{1-\gamma}. \end{aligned} \quad (1)$$

$\text{E}_l$  is an expectation over the length of the option. Using the property that  $o(u, v)$  is robust, we move closer to  $v$  with probability  $\bar{\epsilon}$ ; this is exactly the setting of the well-studied gambler’s ruin problem, where the gambler begins with a budget of  $\|u - v\|$ , and wins with a probability of  $\epsilon$ .

LEMMA 2. Consider the gambler’s ruin problem where the gambler begins with a budget of  $m$ , and has a winning probability of  $q < \frac{1}{2}$  (the dealer has a winning probability of  $p = 1 - q$ ). Let  $L$  be a random variable for the length of the game. Then,

$$g_m(x) = \sum_{l=0}^{\infty} P(L=l) x^l = \frac{1}{\lambda_1^m(x) + \lambda_2^m(x)},$$

where  $\lambda_1(x) = \frac{1 + \sqrt{1 - 4pqx^2}}{2px}$ , and  $\lambda_2(x) = \frac{1 - \sqrt{1 - 4pqx^2}}{2px}$ . Further, when  $x \leq 1$ ,

$$(px)^m \leq g_m(x) \leq x^m.$$

PROOF. For the first part, refer [3]. The second part follows as a corollary,

$$\begin{aligned} \frac{1}{(\lambda_1(x) + \lambda_2(x))^m} &\leq g_m(x) \leq \sum_{l=m}^{\infty} P(L=l) x^l \\ \frac{1}{(\frac{2}{2px})^m} &\leq g_m(x) \leq \sum_{l=m}^{\infty} P(L=l) x^m \\ (px)^m &\leq g_m(x) \leq x^m. \end{aligned}$$

□

In our situation,  $p = 1 - \epsilon = \bar{\epsilon} > \frac{1}{2}$ ,  $q = \epsilon$ , and  $m = \|u - v\|$ .

$$(\bar{\epsilon}\gamma)^{\|u-v\|} \leq \text{E}_l[\gamma^l] \leq (\gamma)^{\|u-v\|}.$$

Returning to Equation (1),

$$\begin{aligned} \text{E}_l[\gamma^l] V(v) &\leq V(u) \leq \text{E}_l[\gamma^l] V(v) + \frac{1}{1-\gamma} \\ (\bar{\epsilon}\gamma)^{\|u-v\|} V(v) &\leq V(u) \leq \gamma^{\|u-v\|} V(v) + \frac{1}{1-\gamma} \end{aligned}$$

$$\|u - v\| \log \frac{1}{\gamma} - \log \frac{1}{1-\gamma} \leq \|u - v\|_V \leq \|u - v\| \log \left( \frac{1}{\bar{\epsilon}\gamma} \right).$$

## 4. OPTIONS FROM EXPERIENCE

In Section 3, we remarked that we need to generate  $O(|S|)$  options. Given the scale of this number, we require an algorithm to efficiently generate options within a budget of training epochs in order for small world options to be practical. Drawing insight from the proof of Theorem 1, the objective of small world options is to bring the agent into an exponentially smaller neighbourhood of the maximal value state. This suggests that cheaply generated options may still be acceptable.

The algorithm (Algorithm 1) we propose takes a given MDP  $M$ , and trains an agent to learn  $T$  different tasks (i.e. different  $R$ ) on it, evenly dividing the epoch budget amongst

them. With each learned task, we certainly will have a good policy for path options from any state to the state of maximal value,  $M_v$ . However, we observe that will also have a good policy for path options from  $u$  to  $v$  is the path is ‘along the gradient’ of  $Q$ , i.e. when  $V(u) < V(v) < V(M_v)$ . Observing that  $V(s) \approx \text{argmax}_v Q(s, \pi(s))$ , we detail the algorithm to construct options from the  $Q$ -value function in Algorithm 2. We use this algorithm to construct many options from a single task solution.

---

### Algorithm 1 Small World Options from Experience

---

**Require:**  $M, \mathcal{R}, r, n$ , epochs,  $T$

- 1:  $O \leftarrow \emptyset$
  - 2: **for**  $i = 0 \rightarrow T$  **do**
  - 3:  $R \sim \mathcal{R}$
  - 4:  $Q \leftarrow$  Solve  $M$  with  $R$  using  $\frac{\text{epochs}}{T}$  epochs
  - 5:  $O' \leftarrow$  QOptions( $Q, r, \frac{n}{T}$ )
  - 6:  $O \leftarrow O \cup O'$
  - 7: **end for**
  - 8: **return** A random subset of  $n$  options from  $O$
- 

---

### Algorithm 2 QOptions: Options from a $Q$ -Value Function

---

**Require:**  $Q, r, n$

- 1:  $O \leftarrow \emptyset$
  - 2:  $\pi \leftarrow$  greedy policy from  $Q$
  - 3: **for all**  $s$  in  $S$  **do**
  - 4: Choose an  $s'$  according to  $P_r$
  - 5: **if**  $Q(s', \pi(s')) > Q(s, \pi(s))$  **then**
  - 6:  $O \leftarrow O \cup \{s, \pi, \{s'\} \cup \{t \mid Q(s', \pi(s')) < Q(t, \pi(t))\}\}$
  - 7: **end if**
  - 8: **end for**  $s$  in  $S$
  - 9: **return** A random subset of  $n$  options from  $O$
- 

We note here except for sampling  $s'$  from  $P_r$ , we do not require any knowledge of the MDP, nor do we need to construct a local model of the same.  $s'$  can approximately be sampled using  $\frac{E[l]}{\log(|S|)}$  in place of  $P_r$ .

## 5. EXPERIMENTAL RESULTS

We trained MacroQ learning agents on several standard domains, and measured the cumulative return obtained using the following option generation schemes:

- **None:** No options were used.
- **Random:** Options were generated by randomly connecting two nodes in the domain (this is equivalent to  $P_0$ ).
- **Betweenness:** As a representative of bottleneck-based schemes, options were generated to take any node to a local maxima of betweenness centrality, as described in [2].
- **Small World:** Options were generated randomly connecting two nodes of the domain using an inverse square law, as described in Section 3.

Each experiment, unless mentioned otherwise, was run for 10 randomly generated tasks in the domain; each task ran for 40,000 epochs, and was averaged over an ensemble of 20 agents.

|        | Arbt. Navi                | Rooms                   | Taxi                |
|--------|---------------------------|-------------------------|---------------------|
| None   | -31.82                    | -1.27                   | -16.90              |
| Random | -31.23                    | -10.76                  | -18.83              |
| Betw.  | -18.28                    | -8.94                   | <b>80.48</b>        |
| Sm-W   | <b>-14.24</b> [ $r = 4$ ] | <b>8.54</b> [ $r = 2$ ] | 0.66 [ $r = 0.75$ ] |

Table 1: Cumulative Return

## 5.1 Optimal Options

The agents were run on the following three domains using the algorithm sketched in Section 3:

- **Arbitrary Navigation:** The agent must reach an arbitrary goal state in an obstacle-free  $x \times y$  grid-world.
- **Rooms:** The agent must navigate a floor plan with 4 rooms to reach an arbitrary goal state.
- **Taxi:** This is the domain described in Example 1.

Optimal policies were given to the options generated according to the schemes described above.

The results of these experiments are summarised in Table 1. Small world options perform significantly better than the other schemes in navigation-oriented tasks like Rooms or Arbitrary Navigation. In the Taxi domain, options generated by the betweenness scheme outperform the small world options. This is expected because the goal states in this domain lie at betweenness maxima.

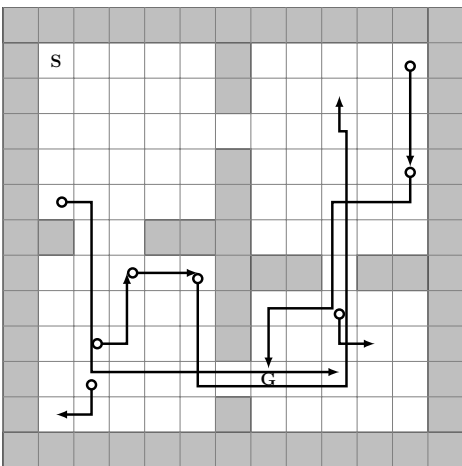


Figure 3: Rooms: Options learnt

Some of the small world options preferred in Rooms domain are shown in Figure 5.1. The graph shows several examples of options that compose together to arrive near the goal state. We have also plotted the learning behaviour in Figure 5.1.

## 5.2 Sensitivity of $r$

We do not yet have a clear understanding of how the exponent  $r$  should be chosen. Figure 5.1 plots  $r$  versus the cumulative return on the Rooms domain. The performance of the agent without options after 20,000 epochs is also plotted for reference. There is a range of  $r$  ( $\approx 0.75$  to 1.5) with good performance, after which the performance steadily drops. This behaviour is easily explained; as the exponent goes up,

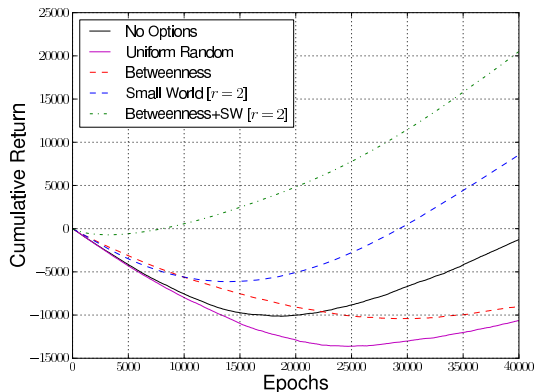


Figure 4: Rooms: Cumulative Return with 200 options

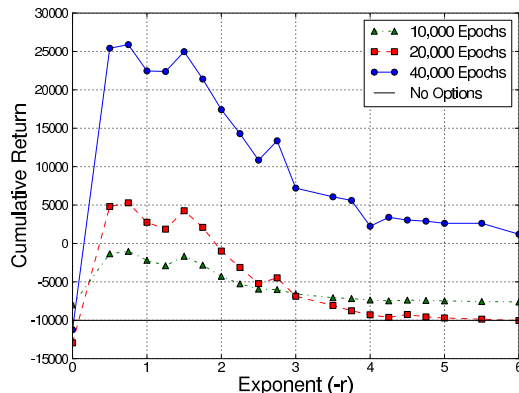


Figure 5: Rooms:  $r$  vs Cumulative Return

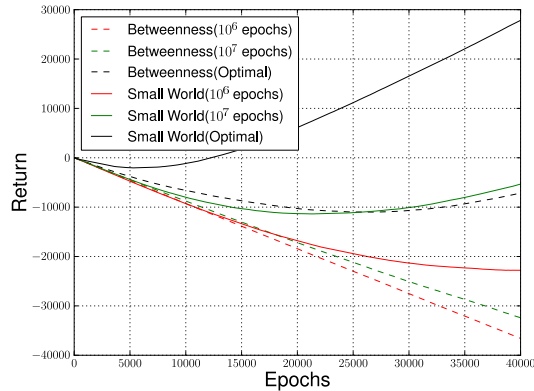
the small world options generated are very short, and do not help the agent get nearer to the maximal value state. The optimal range of  $r$  is slightly counter-intuitive because the Rooms domain is a two dimensional lattice with some edges removed. As a consequence of the reduced connectivity, and perhaps due to stochastic factors, longer range options are preferred.

## 5.3 Options Learnt on a Budget

In Section 4, we describe an algorithm to construct small world options efficiently when given a limited number of learning epochs. We compared the performance of these options with betweenness options learnt similarly, and have plotted our results in Figure 5.2. Despite using many more options, the small world options thus created significantly outperform betweenness options learnt with the same budget, and are even comparable to the optimal betweenness options.

## 6. CONCLUSIONS AND FUTURE WORK

We have devised a new scheme to generate options based on small world network model. The options generated satisfy an intuitive criteria, that the subtasks learnt should be easily composed to solve any other task. The options greatly



**Figure 6: Rooms: Options Learnt on a Budget**

improve the connectivity properties of the domain, without leading to a state space blow up. Finally, they are interesting from a theoretical perspective, as they require only a logarithmic number of decisions required in a learning task.

Experiments run on standard domains show significantly faster learning rates using small world options. At the same time, we have shown that learning small world options can be cheaper than learning bottleneck options, using a natural algorithm that extracts options from a handful of tasks it has solved. Another advantage of the scheme is that it does not require a model of the MDP.

As future work, we would like to characterise what the exponent  $r$  should be in a general domain. Given the ease with which options can be discovered, it would be interesting to experiment with a dynamic scheme that adds options on the fly, while solving tasks.

## 7. REFERENCES

- [1] A. G. Barto and S. Mahadevan. Recent Advances in Hierarchical Reinforcement Learning Markov and Semi-Markov Decision Processes. pages 1–28, 2003.
- [2] O. Şimşek and A. G. Barto. Skill characterization based on betweenness. In *NIPS*, pages 1–8, 2008.
- [3] W. Feller. *An Introduction to Probability Theory and Its Applications*, volume 1. Wiley, 1968.
- [4] M. Jerrum and A. Sinclair. Conductance and the rapid mixing property for markov chains: the approximation of permanent resolved. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, STOC ’88, pages 235–244, New York, NY, USA, 1988. ACM.
- [5] J. Kleinberg. The Small-World Phenomenon : An Algorithmic Perspective. *ACM Theory of Computing*, 32:163–170, 2000.
- [6] L. Li, T. J. Walsh, and M. L. Littman. Towards a Unified Theory of State Abstraction for MDPs. In *Proceedings of the Ninth International Symposium on Artificial Intelligence and Mathematics*, pages 531–539, 2006.
- [7] C. Martel and V. Nguyen. Analyzing Kleinberg’s (and other) Small-world Models. In *PODC*, volume 2, 2004.
- [8] A. McGovern and A. G. Barto. Automatic Discovery of Subgoals in Reinforcement Learning using Diverse Density. In *ICML*, pages 1–8, 2001.
- [9] I. Menache, S. Mannor, and N. Shimkin. Q-Cut - Dynamic Discovery of Sub-Goals in Reinforcement Learning. In *ECML*, 2002.
- [10] B. Ravindran and A. G. Barto. Relativized Options : Choosing the Right Transformation. In *International Conference on Machine Learning*, 2003.
- [11] R. S. Sutton, D. Precup, and S. Singh. Between MDPs and Semi-MDPs : Learning , Planning , and Representing Knowledge at Multiple Temporal Scales at Multiple Temporal Scales. *Artificial Intelligence*, 112:181–211, 1999.
- [12] A. Tahbaz-Salehi and A. Jadbabaie. Small world phenomenon, rapidly mixing markov chains, and average consensus algorithms. In *Decision and Control, 2007 46th IEEE Conference on*, pages 276–281, 2007.
- [13] M. E. Taylor and P. Stone. Transfer Learning for Reinforcement Learning Domains: A Survey. *Journal of Machine Learning Research*, 10:1633–1685, 2009.
- [14] S. Thrun and A. Schwartz. Finding Structure in Reinforcement Learning. In *Advances in Neural Information Processing Systems 7*, pages 385–392, 1995.

## APPENDIX

### A. SMALL WORLDS

In this section we will tackle the proof of the main theorem in [Section 3](#),

**THEOREM 2.** *Let  $f : V \rightarrow \mathbb{R}$  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  $0 \leq c_2 \leq \frac{c_1}{2}$ . Let  $M_f$  be the global maxima of  $f$ . Let  $\mathcal{GA}_\epsilon$  be an  $\epsilon$ -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.  $N(u) = \operatorname{argmin}_v \|f(v) - f(M_f)\|$ .*

*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  $O((\log |V|)^2)$  steps to reach  $M_f$ .*

**PROOF.** This result is a simple extension of Kleinberg's result in [\[5\]](#), and follows the proof presented there, albeit with the somewhat cleaner notation and formalism of [\[7\]](#). 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  $f : V \rightarrow \mathbb{R}$  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  $B_l^f(M_f)$ .

**LEMMA 3.** *The inverse normalised coefficient for  $p(u, v)$  is  $c_u = \Theta(\log n)$ , and  $p(u, v) = \|u - v\|^{-r} \Theta(\log n)^{-1}$ .*

**PROOF.**

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

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

The second part of the lemma follows as  $p(u, v) = \frac{\|u - v\|^{-r}}{c_u}$ .  $\square$

We are now ready to prove that  $\mathcal{GA}_\epsilon$  takes  $O((\log |V|)^2)$  decisions. Let a node  $u$  be in phase  $j$  when  $u \in B_{2^{j+1}}^f \setminus B_{2^j}^f$ . 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,

$$\begin{aligned} P(u, B_{2^j}^f) &\geq \frac{(1-\epsilon)}{\Theta(\log n)} \left(\frac{2^j+c_2}{\kappa_2}\right)^r \times \left(\frac{2^{j+2}+2c_1}{\kappa_1}\right)^{-r} \\ &\geq \frac{(1-\epsilon)}{\Theta(\log n)} \times \left(\frac{\kappa_1}{4\kappa_2}\right)^r \times \left(\frac{1+\frac{c_2}{2^j}}{1+\frac{c_1}{2 \times 2^j}}\right)^r \\ &\geq \frac{(1-\epsilon)}{\Theta(\log n)} \times \left(\frac{\kappa_1}{4\kappa_2}\right)^r \times \left(\frac{1+c_2}{1+\frac{c_1}{2}}\right)^r. \end{aligned}$$

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

$$\begin{aligned} E[X_j] &\leq \sum_{i=0}^{\infty} \left(1 - P(u, B_{2^j}^f)\right)^i \\ &\leq \frac{1}{P(u, B_{2^j}^f)} \\ &\leq \Theta(\log n) \frac{1}{(1-\epsilon)} \left(\frac{4\kappa_2}{\kappa_1}\right)^r \left(\frac{1+\frac{c_1}{2}}{1+c_2}\right)^r \\ &\leq \Theta(\log n). \end{aligned}$$

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