Using Information Theoretic Criteria to Discover Useful Options

Viktoria Kovecses
McGill University
viktoria.kovecses@mail.mcgill.ca

Doina Precup
McGill University
dprecup@cs.mcgill.ca

ABSTRACT

Reinforcement learning allows autonomous agents to compute optimal ways of behaving by interacting with an environment. Temporally extended actions, or options, allow algorithms to achieve faster convergence to an optimal policy. While options can be defined manually for some MDPs, this is difficult in general because it may be unclear what options are good without prior knowledge of the environment. Therefore, it is beneficial to find good options automatically. We propose a novel approach which creates random options and then sifts through them using an information theoretic policy search algorithm. The algorithm combines a measure of predictive information and rewards to guide the agent. Our approach shows promising empirical results on small navigation problems and is computationally efficient.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

General Terms

Theory

Keywords

finding options, information theory, predictive information

1. INTRODUCTION

People often use different levels of planning to execute their everyday actions. For instance, the action of making an appointment may consist of many sub-actions, such as picking up the phone, dialling the number, and speaking to the receptionist. The abstraction of sub-actions into larger more compact actions is therefore useful in everyday life. However, it may also be useful in the context of reinforcement learning.

Temporal abstraction of actions has proved to be a useful tool allowing for faster convergence of algorithms to an optimal policy in Markov Decision Processes (MDPs). These temporally abstracted actions are often referred to as options, and they in general consist of a set of states, a policy, and termination conditions.

Although options may be a very powerful tool once created, they are not always easy to instantiate manually. In some environments the state space may be simply too large, while in others it may not be evident what a good option would consist of. Therefore, it would be beneficial to be able to generate options programmatically for any given environment.

Previous work on finding useful options has shown that for at least some environments it is beneficial to define the option in such a way that the goal state of the subpolicy is a bottleneck (e.g. a hallway between rooms) [3]. However, this method fails if for instance the bottleneck happens to have a very negative reward. Another method for finding options is to look at solutions to several tasks in the same environment, and observe where the optimal actions overlap [1]. The states where the actions remain constant across tasks can then be used to make options.

In our project, instead of trying to understand what makes a good option by first observing interesting parts of the environment and then creating options, we instead create several random options and observe how an information theoretic driven policy search algorithm [4] chooses between options and actions. Essentially, we would like to examine whether using predictive information (i.e. $I[(A_t, X_t); X_{t+1}]$) as well as the return in the environment could be useful in finding decent options for any given MDP.

2. BACKGROUND

To begin we give a brief overview of reinforcement learning, options, and information theory, as well as important terminology and notation.

2.1 Reinforcement Learning

The reinforcement learning framework involves an agent learning about an environment by interacting with it across discrete time steps. To accomplish this the agent moves around by taking different actions, and accumulating rewards as it goes. The environment is often modelled as a Markov Decision Process (MDP) [2], which consists of a set of states $\mathcal{S}$, a set of actions $\mathcal{A}$, a transition probability matrix $\mathcal{P}$, and a

reward function $\mathcal{R}$. Each state represents a portion of the environment, while the actions can be applied to take the agent from one state to another. The transition probability matrix contains the probability that the agent ends up at state s_{t+1} if it takes action a_t in state s_t at time t. Finally, the reward function indicates the reward r_{t+1} that the agent obtains by taking action a_t in state s_t. In addition, the agent may act according to a policy π, which is a mapping of actions to states that indicates the action to take at each state. The goal of the agent now becomes to maximize the accumulated reward by finding an optimal policy in the environment.

2.2 Options

In many situations it may be beneficial to use temporal abstraction to allow the agent to plan using different levels in the environment. For example, if we wish the agent to walk straight until it reaches an obstacle in a continuous environment, we may not want to tell it what to do at each state but simply tell it to move in the same direction for a certain period of time. This would be an example of a temporally abstracted action.

By defining a fixed subpolicy for a group of states in an MDP before the agent begins exploration, one can achieve a faster convergence to an optimal policy. More formally, we may refer to these as options, which are made up of three parts: an initiation set, a policy, and termination conditions [5]. The initiation set, a subset of the state space, includes all the states where the option can be executed, the policy represents how the agent should act in each of these states, and the termination condition is the probability that the option will terminate at any given state [5].

2.3 Information Theory Aspect

Maybe give a different title to this subsection. Talk about the information theory aspect and its importance in the research. Link it to options.

2.3.1 Predictive Information

In information theory mutual information is often defined as the information that a random variable X carries about a random variable Y, and vice versa. In other words, it tells us how well knowing one of the random variables can help us predict the other, and is calculated as follows:

$$I(X;Y) = \sum_{x \in X} \sum_{y \in Y} p(x,y) log(\frac{p(x,y)}{p(x)p(y)})$$

The predictive information is then defined to be the information that the current state and action carry about the state the agent will end up in next (i.e. $I[(A_t, X_t); X_{t+1}]$). It can also be thought of as a measure of the ability of the agent to predict where it will move if it takes an action at its current state.

2.3.2 Kullback-Leibler Divergence

The Kullback-Leibler divergence between two probability distributions is defined as follows:

$$D_{KL}(p||q) = \sum_{x \in X} p(x) log(\frac{p(x)}{q(x)})$$

It essentially measures the difference between two probability distributions p and q. For instance, if the two probability distributions are identical, the divergence would be 0, and if they are very different then the divergence may go to infinity.

In order to maximize the predictive information, as the information theoretic aspect of the algorithm aims to do, we do not directly calculate the predictive information using the mutual information formula, but instead use the Kullback-Leibler divergence between the transition probabilities (i.e. the true model) and the empirically calculated steady state distribution: $D_{KL}(p(X_{t+1}|a, x_t)||p^{\pi}(X_{t+1}))$. If we consider the steady state distribution $p^{\pi}(X_{t+1})$ to be an approximation of the transition probability distribution, then we can use the divergence as a measure of the predictive information. For example, if the steady state distribution is very different from the transition probability distribution, then what the agent observes about the environment is not yet close to what the agent expects to see. Therefore, more exploration is required to reduce the divergence and hopefully bring the steady state distribution closer to the transition probability distribution if possible.

2.3.3 Algorithm

The algorithm used in our research was taken from previous work completed by Doina Precup and Suzanne Still [4]. It is a policy search algorithm which takes into account both the predictive information in the environment (i.e. $I[(A_t, X_t); X_{t+1}]$), as well as the return in order to compute an optimal deterministic policy which may be tuned to encourage a certain amount of exploration in the environment. The policy is computed at each time step using the following equation:

$$q_{opt}(A_t = a|X_t = x) = \frac{p_t(a) e^{\frac{1}{\lambda} * (D(x,a) + \alpha * Q(x,a))}}{Z(x)}$$

where $D(x, a)$ represents the Kullback-Leibler divergence between the true model and an estimated steady state distribution, or how well the estimated model predicts the true model; and $Q(x, a)$ represents the action values as in usual reinforcement learning.

2.4 Experiments

Here we give an overview of the environments in which the algorithm was tested.

2.4.1 Grid Environments

For the experiments, different variations of a simple 10 x 10 grid world were created. They all contained at a start and a goal state, and the dynamics of the environment remained constant across variations. The agent would always start from the start state and try to move towards the goal which had a reward of +1. The reward everywhere else was 0. The action space contained four primitive actions which the agent could take, namely $up, down, left$, and $right$; and

these actions simply moved the agent towards the corresponding adjacent square. For any primitive action taken, there was a 0.7 probability that the action would succeed, and in the event of failure a different action would be chosen randomly.

2.4.2 Interesting area

In order to test whether the information theoretic criteria had any influence in the agent's exploration, an area of interest was added to some of the grid environments (see figure). This area consisted of a set of states in which either the reward or the probability of an action succeeding would vary based on a Gaussian or Uniform probability distribution, respectively. In the case that the reward was varied, the average reward remained 0, and therefore going through this area to reach the goal ultimately made no difference in reward for the agent compared to other paths.

2.5 Results

Before attempting to remove useless options or chose good options from the environment, a series of experiments were conducted on the grid world environments, mentioned above, in order to determine how the algorithm would choose between both good and bad options and actions, and the nature of the policies that would be produced.

In Figure 1, the algorithm is used on a grid world with a large random option, composed of every state except the subgoal where the option terminates. The policy found only includes the option if it moves the agent closer to the goal from most states, and leads to improved performance when it is used. This shows the ability of this technique to sift through options and potentially make it easier to find good options without making any assumptions about the environment.

In another experiment, a similar grid world was used, with this one now containing a large square obstacle in the center. This left for two paths of equal length to the goal, each of which contained a hand-crafted option leading the agent down the path and towards the goal (see Figure 2). The sole difference between the two paths was that the left-most path contained a subset of states, called the "interesting area", in which the probability of an action succeeding varied from 0 to 1 uniformly randomly. The interest of the experiment was to see how the options would be selected along either path, and if the area where actions succeed randomly would affect the policy. The results showed that along both paths the options were initially preferred, and quickly increased the overall return. However, in the states examined the algorithm traded the option for the optimal primitive action, which also happened to be the same action the option was executing in that state. The policy thus switched to consist mainly of primitive actions, though there was no significant change in the return. This shows that the useful options are incorporated into the policy, and only get traded for the optimal primitive action, which is expected as this leads to an improved and simpler policy in most cases. It can also be noted that the policy selected the optimal primitive action very early on, and this action remained dominant until the completion of the algorithm. The reason for this is most likely that the Q value for the chosen action quickly became higher than the Q values for all other actions, even surpassing the options, thus giving this action a significant advantage. The same results were found when observing states in the interesting area; however, in these states there were slightly greater fluctuations in the policy, most likely due to the high uncertainty of actions succeeding.

3. CONCLUSIONS

Summarize results and there significance, talk about future work that could be done.

More work is still needed to see whether this method would work well in different settings, and to compare it to other option finding algorithms. Furthermore, using different aspects of information theory, such as entropy instead of predictive information may be interesting to explore as well.

4. ACKNOWLEDGMENTS

This section is optional; it is a location for you to acknowledge grants, funding, editing assistance and what have you. In the present case, for example, the authors would like to thank Gerald Murray of ACM for his help in codifying this *Author's Guide* and the **.cls** and **.tex** files that it describes.

5. REFERENCES

[1] M. Pickett and A. G. Barto. Policyblocks: An algorithm for creating useful macro-actions in reinforcement learning. In *ICML*, volume 2, pages 506–513, 2002.

[2] M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1994.

[3] A. Solway, C. Diuk, N. Corova, D. Yee, A. Barto, Y. Niv, and M. Botvinick. Optimal behavioral hierarchy. *PLoS Computational Biology*.

[4] S. Still and D. Precup. An information-theoretic approach to curiosity-driven reinforcement learning. *Theory in Biosciences*, 131(3):139–148, 2012.

[5] R. S. Sutton, D. Precup, and S. Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1):181–211, 1999.

APPENDIX

A. HEADINGS IN APPENDICES

The rules about hierarchical headings discussed above for the body of the article are different in the appendices. In the **appendix** environment, the command **section** is used to indicate the start of each Appendix, with alphabetic order designation (i.e. the first is A, the second B, etc.) and a title (if you include one). So, if you need hierarchical structure *within* an Appendix, start with **subsection** as the highest level. Here is an outline of the body of this document in Appendix-appropriate form:

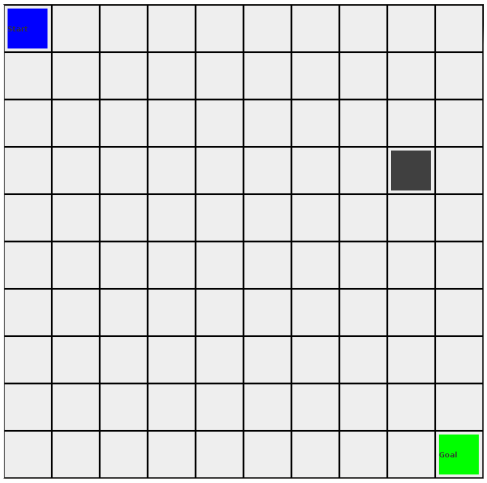
A.1 Introduction

A.2 The Body of the Paper

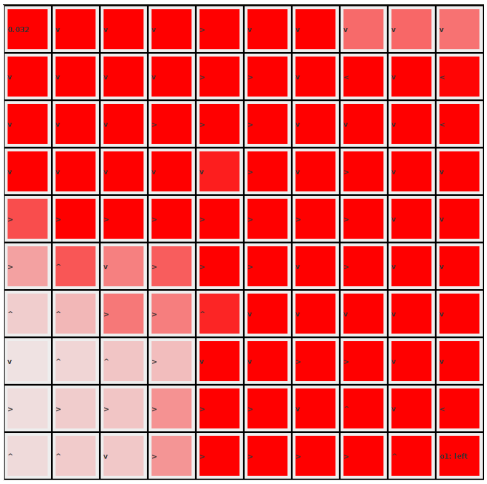
A.2.1 Type Changes and Special Characters

A.2.2 Math Equations

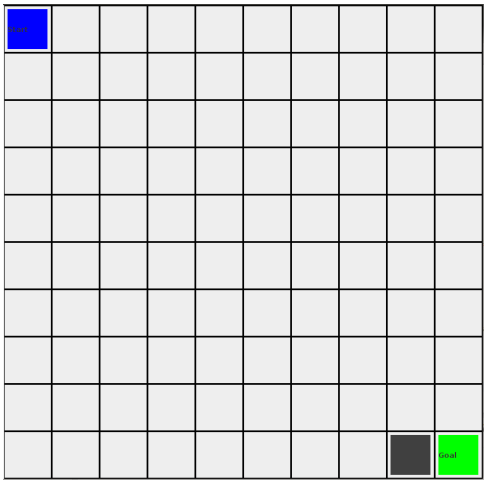
Inline (In-text) Equations



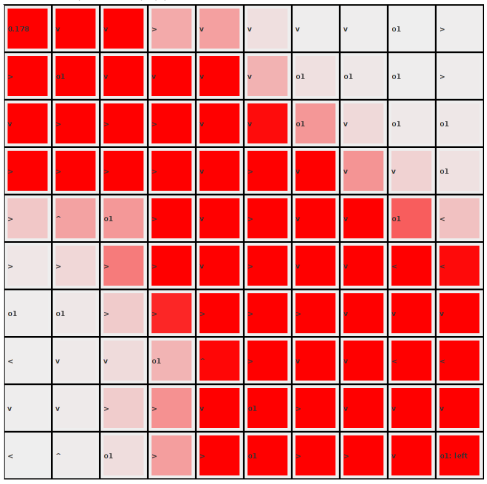
(a) Option which leads most states away from the goal



(b) Option (from (a)) is not selected in any of the states

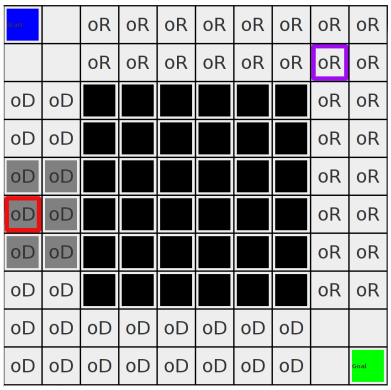


(c) Option which leads most states towards the goal

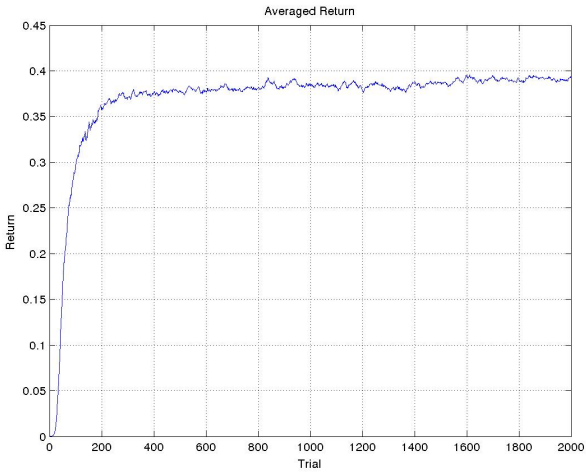


(d) Option (from (c)) is selected in several states

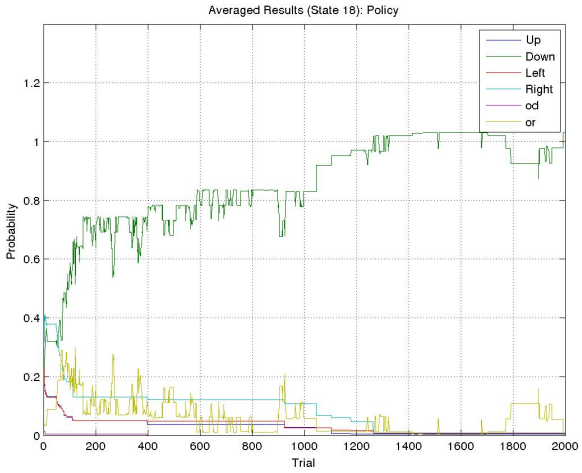
Figure 1: Results of testing the algorithm on an obstacle-free environment with one large random option. In (a) and (c) the blue tile represents the starting state, the green tile represents the goal, and the black tile is the subgoal for the option. In (b) and (d), a darker red represents states that were visited more frequently.



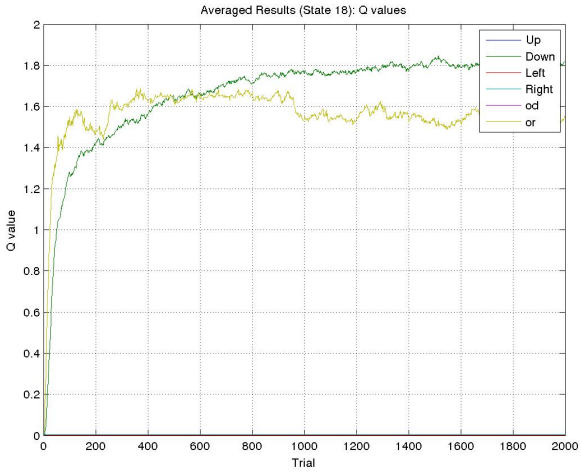
(a) Environment with square obstacle, and two large options leading towards the goal (oD and oR). Blue square: start, Green square: Goal, Grey squares: interesting area, Purple outline: State 18, Red outline: State 50



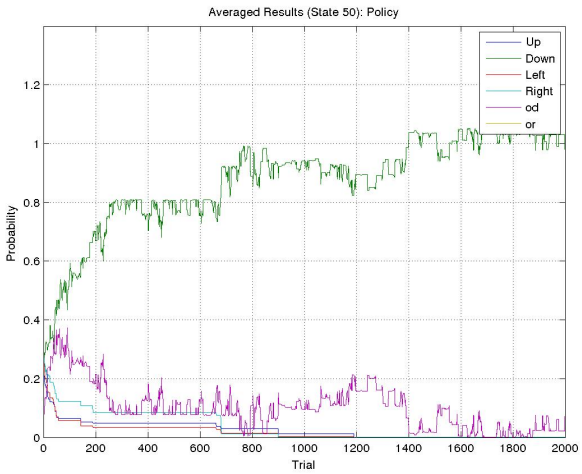
(b) Average return accumulated over time



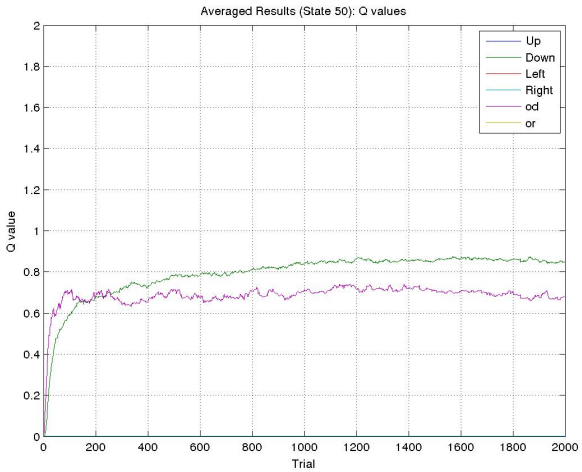
(c) Policy over time at state 18. Green line: Probability of taking down action, Yellow line: Probability of taking option oR



(d) Q values over time at state 18. Green line: Q value for down action, Yellow line: Q value for option oR



(e) Policy over time at state 50. Green line: Probability of taking down action, Purple line: Probability of taking option oD



(f) Q values over time at state 50. Green line: Q value for down action, Purple line: Q value for option oD

Figure 2: Results of running the algorithm on an environment with one large obstacle and two hand-crafted options. The results are averaged over 20 full runs of the algorithm.

This section is inserted by LaTeX; you do not insert it. You just add the names and information in the `\additionalauthors` command at the start of the document.

Generated by bibtex from your .bib file. Run latex, then bibtex, then latex twice (to resolve references) to create the .bbl file. Insert that .bbl file into the .tex source file and comment out the command `\thebibliography`.

The acm_proc_article-sp document class file itself is chockfull of succinct and helpful comments. If you consider yourself a moderately experienced to expert user of LaTeX, you may find reading it useful but please remember not to change it.