Options with exceptions

Munu Sairamesh and Balaraman Ravindran

Indian Institute Of Technology Madras, India

Abstract. *An option is a policy fragment that represents a solution to a frequent subproblem encountered in a domain. Options may be treated as temporally extended actions thus allowing us to reuse that solution in solving larger problems. Often, it is hard to find subproblems that are exactly the same. These differences, however small, need to be accounted for in the reused policy. In this paper, the notion of options with exceptions is introduced to address such scenarios. This is inspired by the Ripple Down Rules approach used in data mining and knowledge representation communities. The goal is to develop an option representation so that small changes in the subproblem solutions can be accommodated without losing the original solution. We empirically validate the proposed framework on a simulated game domain.*

Keywords: Options framework, Transfer Learning, Maintenance of skills

1 Introduction

One of the main advantages of Hierarchical Reinforcement Learning(HRL) is that HRL frameworks allow us to learn *skills* required to solve a task and transfer that knowledge to solving other related problems. This allows us to considerably cut down on learning time and achieve better success rates in solving new problems. But the main difficulty in such transfer is that a problem is seldom encountered in exactly the same form. There are some changes in the task, such as a different obstacle configuration, or minor changes in the dynamics of the world. We need a framework where we can accommodate such changes without compromising the quality of the solution of the original sub-task that the skill was learned on. In fact we would like to cascadingly accommodate more such changes, without degrading any of the solutions learned along the way.

The problem is exacerbated when we look at spatial abstractions that are specific to the skill being learnt. There has been a lot of interest in recent times on deriving skill specific representations (MAXQ[**?**], VISA [**?**] and Relativized option [**?**]). Dietterich, while proposing the MaxQ framework for value function decomposition has discussed safe state abstraction conditions that would minimally affect the quality of the sub-task policies that are being learned. Jonsson and Barto have looked at the problem of jointly determining spatial and temporal abstractions in factored MDPs. Ravindran and Barto have proposed relativized options that use the notion of partial homomorphisms to determine lossless option specific state representations that satisfy some form of safe state abstraction conditions.

Such approaches yield more compact representations of the skills and allow for greater transfer since they ignore all aspects of the state representation not required for that sub-task. This also renders them more fragile since they can fail if there are small changes to the system dynamics that require that different spatial abstractions be used to represent the modified solutions. For example, in the simulated game used in [**?**], the introduction of an additional obstacle might require changing the policy only around the states with the obstacle. But this would require us to represent the location of the obstacle as well as the original information that we were considering.

In order to accommodate such scenarios we have proposed the notion of *options with exceptions* (OWE). The knowledge management community have long dealt with the problem of incremental maintenance of rule bases using Ripple Down Rule(RDR) [5]. Suppose that some new training instances contradict a compact rule that was derived from the original training data, the traditional approach would be to either accept the decreased performance or try to re-learn the rule base from scratch. The ripple down rules representation allows us to minimally modify the existing rule by adding exceptions that are derived from only the contradicting instances. Taking inspiration from this concept, we propose a framework that allows us to modify an option's policy only using instances where the existing policy seems to fail.

There are several key challenges to address here. How to represent the policy so that modifications can be made without affecting the base policy when we add the exceptions? In a stochastic domain, how do we even detect that the old policy has failed or does not perform up to expectations? In a deterministic environment this is somewhat easy to address, since we can form models of expected behavior of the world. How to compactly represent such models in stochastic environments? How to detect which of the new features of the state space should we add to the option representation in order to trigger the exception?

Our framework has several components that extend and augment the traditional option representation. The first is a more structured representation of the option policy that allows for easy addition of exceptions. The second is a network of *landmarks* or way points that acts as a compact model of the environment and quickly allows us to narrow down to the region where the option failed. The third is a path model for the original policy that allows us to pinpoint the features that we need to include in the exception representation.

We test the ability of the framework to identify exceptions and to learn new policies to address the exceptions. We also show that when the option with exception is applied to the task it was originally learnt on, there is no degradation of performance.

The next section describes the necessary notations and background work. Section 3 describes the landmark network and the transition time model, while Section 4 describes the identification of the actual exceptions and presents our experimental results. Section 5 concludes with some discussion and directions for future work.

2 Notation and Background

2.1 Notation

A trajectory is a sequence of states through which an agent moves. The length of the trajectory is denoted by l. Let s_t and a_t be the state and action at time t. Let η_s be the total number of times that a state s has occurred in all the trajectories. A transition instance is the tuple $\langle s_t, a_t \rangle$ and a transition chain is an ordered sequence of transition instances.

2.2 Option

An option [?] is represented as $\langle I, \pi, \beta \rangle$, where I is the initiation set of the option, i.e., the set of states in which the option can be invoked, π is the policy according to which option selects the actions and β is the termination function where $\beta(s)$ gives the probability that the option terminates in state s. An option can be thought of as representing a skill.

2.3 Policy representation

To explicitly represent the policies of the option, we choose a specific representation. Suppose, the policies are represented using Q values, reusing them in another task would result in changes in the Q values. This could modify the original policy adversely. Hence a representation is required in which changes in these values do not alter the existing policy. For example, in order to accommodate exceptions, this can be achieved if the representation of the new policy is conditioned on the feature that caused the exception. Also, the exception may affect the policy in its neighboring states, hence requiring the use of a small amount of history in representing the new policy. This can be achieved using a suffix tree, which divides the history space.

Suffix trees are used to build a discrete internal state action representation e.g. U-Tree. The internal nodes of the suffix tree are from the set $\{ s_{t-1}, a_{t-1}, s_{t-2}, a_{t-2} \ldots s_{t-H}, a_{t-H} \}$, where H is the history index. It denotes the number of time steps in the past from the current time. The Suffix tree is trained using a transition chain.

3 Transition Time model

One of the chief components of *Option With Exceptions* (OWE) framework is a transition time model that records the time of transition between certain distinguished states or *landmarks*.

Landmarks are places that are visited often and are used to find the way back or through an area. Using the same notion we define a landmark for a region as a state that is visited on most of the paths (successful or unsuccessful) through that region. A sub-goal may be a landmark whereas a landmark need not be a sub-goal. To build an efficient transition time model, landmarks need to be uniformly distributed throughout the state space.

The landmark network is a weighted *Directed Acyclic Graph(DAG)*, where nodes are landmarks and an edge $i \rightarrow j$ indicates that node j is followed by node i in a majority of the sampled trajectories, with the edge weight denoting average transition time between i and j. The agent, constrained with the topology of the landmark network has to follow a sequence of landmarks(l_{sequence}) starting with the start landmark and ending with the terminal landmark. If the landmark in the l_{sequence} is not visited within the transition time stored in the time model then a failure is ascertained. Failure leads to learning a new path between the landmarks or learning a new average transition time between existing paths of the landmarks. This new transition time is added as an element to already existing set of average transition times.

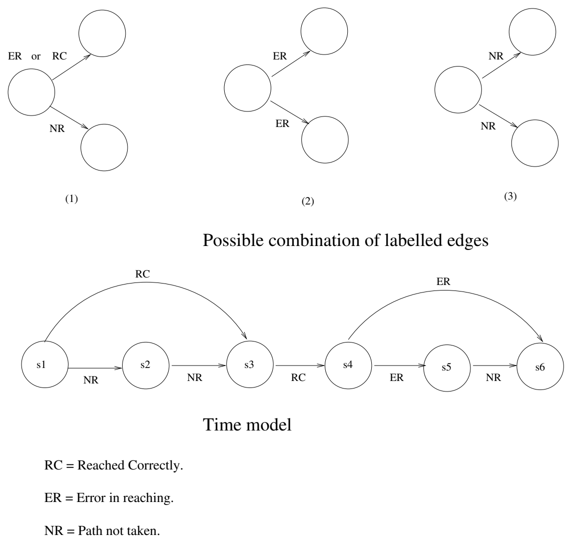
3.1 Identification of landmark

We propose two heuristics to identify landmarks :

- ☐ Spiked state: While traveling most of the people try to associate landmarks with some junctions, where they want to leave the road and move onto another road. Spiked state heuristic is based on this idea and the junction of many trajectories is called a spiked state. Here the spiked state has highest frequency of visits across trajectories as compared to other neighboring states. Once a policy is learnt, trajectories are sampled with random start states. For a given trajectory let s_{t-1}, s_t, s_{t+1} be state that occurred at successive time instances. If $\eta_{s_{t-1}} < \eta_{s_t}$ and $\eta_{s_t} > \eta_{s_{t+1}}$ then s_t is a spiked state. If two states s and s' spiked in the same trajectory, they are called co-occurring spiked states. From this set of spiked states we choose the set of states which spiked in maximum number of trajectories. Since we need landmarks to be uniformly distributed in the domain, we define a lower bound on the count i.e., number of times state spiked in all the trajectories. This bound can be estimated empirically as it depends upon the domain.
  One of the problems with this heuristic is that it requires the domain to be highly connected so that there are various paths through the states which enables us to find the pattern required to identify the spiked states.

- ☐ Mean-to-variance ratio: This heuristic uses a simpler approach as compared to the spiked state heuristic. Trajectories are sampled as in spiked state heuristic, but here all trajectories start from the same state. A state is said to have been accessed at time t if it is encountered in the trajectory at step count t. The time step at which each state is accessed in the trajectory is extracted for all states in the trajectories and its mean and variance. The states are sorted in decreasing order of mean-to-variance ratio. Here variance is considered in order to avoid the effect of states closer to initial states and the terminal states in the trajectory as these states will always be accessed. The analysis is done for the states which appears to occur in maximum number of trajectories. Distance between the landmarks is defined as the average number of time steps taken to traverse between the landmarks. Landmarks are selected sequentially from the sorted list of states so that a criteria of minimum distance between the landmarks is satisfied.

Fig. 1: Identification of exception landmark.



3.2 Construction and updation of Transition Time model

Transition time model stores the transition time between the source and destination landmarks. This is estimated from sampled trajectories for all landmark pairs of the landmark sequence(l_{sequence}), if they are part of the same trajectory. Here $l_{sequence}$ is a topological ordering of landmarks in the landmark network. Let the old average transition time between the landmarks p and q stored in the transition model be called as t^{pq}_{old}.

In order to find a failure, a new instance of the transition time model is created. It is populated with the average new transition time, extracted from the new sampled trajectories. These new trajectories are sampled in batches using the old policies. If there is no significant change in the transition probability of the states in the batch, sampling is stopped. Let t^{pq}_{new} be a new average transition time between the landmark p and q stored in the new instance of transition model. Percentage increase in the average transition time between any two landmark, p and q is called Change In Transition Time($CITT$).

$CITT = \frac{t^{pq}_{\text{new}} - t^{pq}_{\text{old}}}{t^{pq}_{\text{old}}} \times 100$

Failure is ascertained if $CITT$ exceeds the empirically estimated threshold percentage. Since each source and destination landmark may store the set of average transition times. This enforces computation of $CITT$ between t_{new}^{pq} and the set of t_{old}^{pq}s. Hence minimum $CITT$ for each edge is compared with the threshold percentage, leading to edges being labeled as shown in Figure **??**. There are three types of scenarios which usually occur in the transition time model. In the first scenario shown in Figure **??**, one of the destination landmarks is reached from the source landmark. If at least one of the $CITT$ value is within the threshold percentage, the edge is labeled as "*RC*" (Reached Correctly), else it is labeled as "*ER*"(Error In Reaching). The rest of the edges leading to other landmarks are labeled as "*NR*"(Path Not Taken). An edge is labeled as "*NR*" only if the source and destination landmark pair does not occur simultaneously within the specified threshold count of trajectories. There cannot be a node with two out edges labeled as "*RC*" and "*ER*".

The transition time model with the labeled edges is used to find the exception. If the task is successfully completed then there exists a unique path consisting of a sequence of edges labeled as "*RC*" between the start and terminal landmarks. This indicates that an exception has not occurred. Failure is ascertained when at least one of the outgoing edges from the current landmark is labeled as "*ER*" or all the outgoing edges are labeled as "*NR*" as shown in second scenario in Figure **??**. The current landmark is then called an exception landmark. For example, in Figure **??**, s_4 is the exception landmark in the sequence of landmarks s_1, s_2, s_3, s_4, s_5 and s_6

Topological ordering of landmarks(l_{sequence}) helps to identify a set of landmarks called *potential destination landmarks*, that come topologically after the exception landmark. A new path is learnt between the exception landmark and some potential destination landmarks using Q-learning. The transition time model is updated to include the new transition time between the landmarks. While updating, the newly learnt path might add edges between landmarks which were not connected earlier or it might add new task times to existing set of task times between already connected landmarks. The landmark network is topologically reordered to reflect the new ordering of the landmarks.

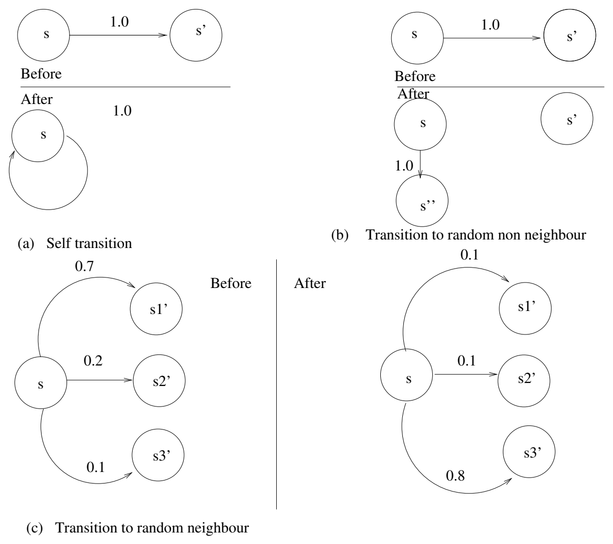
4 Identification of exception state

An exception happens when the policy fails. The state in the which failure first happens is called the exception state. Failure can happen because of changes in the dynamics of the domain. The reasons for this to happen is either introduction of a new object in the domain and a change in the values that features of the domain can take. In either case, there are some features of the exception state which reflect the change in the domain. The new policy is represented in the suffix tree by conditioning on these features.

In order to identify the exceptions, transition probabilities of all the state action pairs need to be stored. But rather than storing it for all the actions, a model called as *minimum model* can be used to store only the optimal policy action for all the states. Exception state is found by comparing the minimum model learnt before and after the failure is ascertained by the time model. Rather than comparing for all the states in the trajectory, exception landmarks can be used as reference point from where comparison could start. Exception landmark is the landmark after which an exception occurs. Figure **??** shows basic types of exceptions, which change the dynamics of the domain. These are self transitions, transitions to random neighboring states and transitions to random non-neighboring states . Other forms of exceptions that are possible are combinations of these basic types of exceptions.

After the exception policy is learnt, the model is updated to reflect the changes in transition probabilities of exception states.

Fig. 2: Different types of exceptions.

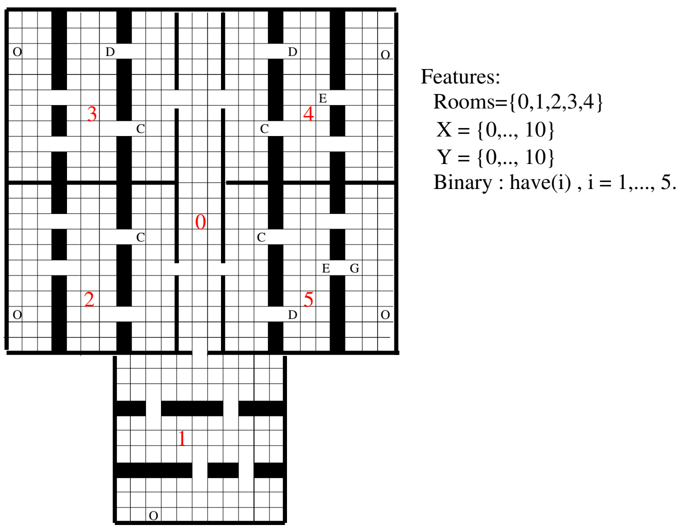


5 Experiment and Results

In Figure 3, the agent's goal is to collect the diamond(o) in every room by occupying the same square as the diamond. Each of the rooms is a 11 by 11 grid with certain obstacles in it. The actions available to the agent are N, S, E, W. An action would result in transition to the expected state with probability 0.9. It results in a transition in any one of the unintended directions with a probability 0.1.

The state is described by the following features: the room number the agent is in, with 0 denoting the corridor, the x and y co-ordinates within the room or corridor and boolean variables *have*(i), i = 1,..., 5, indicating possession of

Fig. 3: A simple room domain with different objects. The task is to collect the diamond, o in the environment



diamond in room i. The goal is any state of the form $\langle 0, ..1, 1, 1, 1, 1\rangle$, where o indicates the corridor and 1 indicates possession of diamonds in the respective rooms. The state abstraction for the option is $\langle x, y, have\rangle$. Exceptions are caused by the objects C, F (self transition) , D (transition to random neighbor) and E (transition to random non-neighbor).

The agent used a learning rate of .05, discount rate of 1.0 and ϵ- greedy exploration, with an ϵ of 0.1. The result were averaged over 25 independent runs. The trials were terminated either on completion of the task or after 1000 steps. The landmarks were selected by the mean to variance ratio. States that occur in more than 80% of the trials only were considered as candidate landmarks. Three sets of experiments were performed with different desired minimum distance (in time) between the landmarks, resulting in a different number of landmarks for each experiment. The distance between the landmarks is 18 steps in experiment 1, 8 steps for experiment 2 and 4 steps for experiment 3. The selected landmarks are shown in table 1. To compare the transition time between the landmarks, threshold value is taken as 25% for experiment 1, 30% for experiment and 35% for experiment 3. A landmark is considered as not reachable(“NR”) if the number of trajectories is less than 75%.

The objects were introduced in the domain in the order C, D, E and F. These objects can be introduced randomly in the domain but for the purpose of learning nested OWE, they were introduced in the specific order. Exception caused by the object “D” is not identified in most of the experiments because the maximum change in the number of steps caused by the exception is not more than 1.5 on an average. If the threshold value is adjusted to account for

Table 1: Selected Landmarks

Experiment		
1	2	3
(10, 5, 0), (0, 3, 1), (10, 5, 1)	(10, 5, 0),(5, 5, 0),(1, 2, 0) (4, 5, 1),(10, 5, 1)	(10 ,5, 0),(7 ,7 ,0),(3 ,5, 0) (1, 3, 0),(0, 3, 1),(1, 5, 1) (4, 5, 1),(5, 7, 1),(9, 7, 1) (10, 5, 1)

this small change, it leads to detection of exceptions caused by the stochasticity of the environment. Hence exception caused due to D in the domain remains unidentified. Whereas, if the same object is placed in the grid (8,7), exception is identified because the most of trajectories do not go through (8,6).

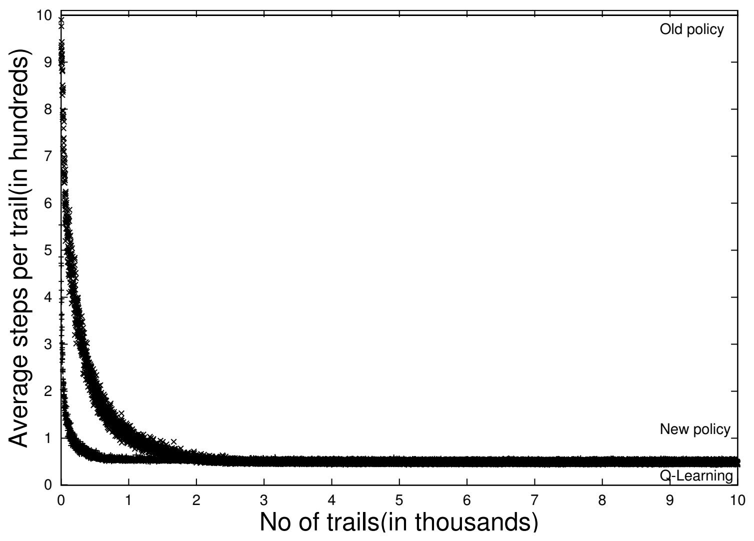


Fig. 4: Comparison of old policy, new policy and policy learnt using Q-learning where the exception is caused by the object C.

In Figure 4,5 and 6, the new policies are plotted for various obstacles. The new policy is a concatenation of the path given by the suffix tree from the start state to exception state or terminal state, followed by the path given by Q-learning from exception state to potential terminal landmarks, followed by a path given by the suffix tree from potential terminal landmark to terminal state. Graphs have been plotted for Experiment 3. Due to lack of space, we have not plotted the graphs for other experiments although the results agree with our expectations.

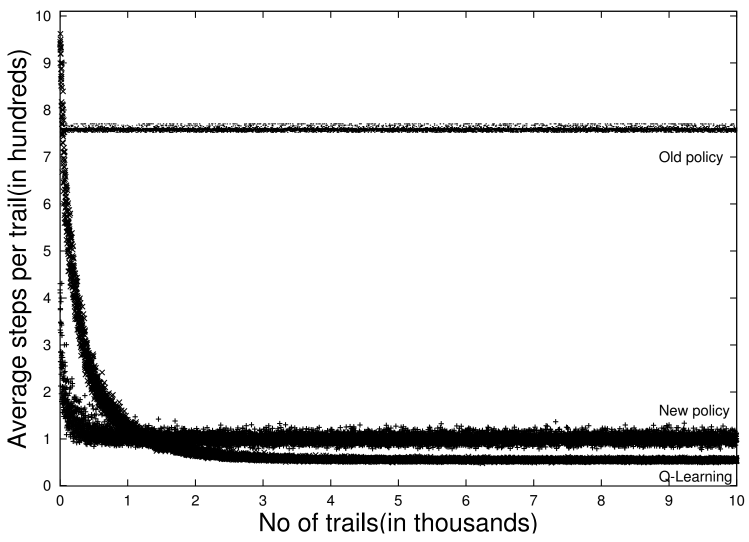


Fig. 5: Comparison of old policy, new policy and policy learnt using Q-learning where the exception is caused by the object E.

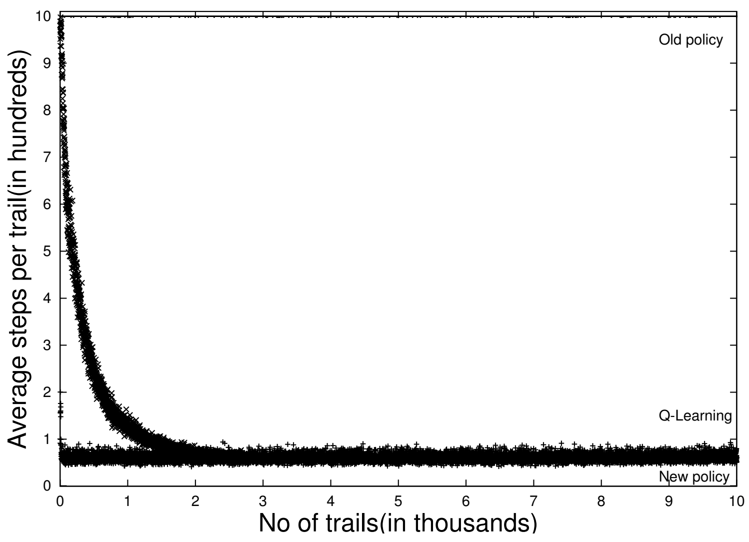


Fig. 6: Comparison of old policy, new policy and policy learnt using Q-learning where the exception is caused by the object F.

When the OWE policies were used in the old domain, i.e., without exceptions, the average number of steps taken to complete the task does not change appreciably, as shown in Table 2. The average was taken over 1000 trails. In Table 2, the rows indicate transfer of policies to old domain after particular object was introduced in the domain and columns indicate different objects present in the domain at that time.

In the column "Object C,D,E", the average number of steps for experiment 3 is higher than the others because the new policy is constrained to go through extra landmarks which requires 3 extra steps in comparison to that required for experiment 1 and experiment 2. Therefore for experiment 3, all the values under the columns are greater than corresponding values of the other experiments.

Table 2: Transfer back of the policy to old domain(Avg No of steps)

		No Object	Object C	Object C,D	Object C,D,E
Experiment 1	No Object	44.43			
	After Object C	44.07	52		
	After Object D	44.32	52.29	55.12	
	After Object E	44.55	52.71	55.02	63.12
	After Object F	44.58	52.36	55.21	63.87
Experiment 2	No Object	44.45			
	After Object C	44.23	52.16		
	After Object D	44.14	52.30	54.42	
	After Object E	43.90	52.40	54.84	63.4
	After Object F	43.62	52.55	54.69	63.54
Experiment3	No Object	44.33			
	After Object C	44.39	53.19		
	After Object D	43.86	53.20	55.30	
	After Object E	44.76	53.78	55.17	66.56
	After Object F	44.27	53.19	55.60	66.24

6 Conclusion

In this work we draw attention to the problem of efficient incremental maintenance of library of skills. We propose a framework in which exceptions can be added to already learnt options without disturbing the original policies. While the framework succeeds in identifying small changes required to the options, we still have some distance to go before building a complete skill management system. We are currently working on the problem of automatically deriving the minimal set of features required to represent exceptions. We also want to automatically identify when an exception would suffice and when a new option is warranted. While we can think of several heuristics, based on the size of the suffix tree for instance, we would like a more principled approach.

References

1. Andrew G. Barto, Sridhar Mahadevan, Recent Advances in Hierarchical Reinforcement Learning, Discrete Event Dynamic Systems Volume 13 Issue 1-2, (2003)
2. Matthew E. Taylor and Peter Stone, Transfer Learning for Reinforcement Learning Domains: A Survey, Journal of Machine Learning Research, volume 10 pp 1633–1685 (2009)
3. Thomas G. Dietterich , Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition, Journal of Artificial Intelligence Research, volume 13 pp 227–303 2000
4. A. K. McCallum: Reinforcement Learning with Selective Perception and Hidden State, Ph.D. Thesis, Department of Computer Science, The College of Arts and Science, University of Rocheater, USA(1995)
5. Mehran Asadi and Manfred Huber, Autonomous Subgoal Discovery and Hierarchical Abstraction Learned Policies, FLAIRS Conference, pp 346-350, (2003)
6. Gaines, B.R and Compton, P, Induction of Ripple-Down Rules Applied to Modeling Large Database, Knowledge Acquisition 2(3), pp 241–258, (1995)
7. A. McGovern, Autonomous Discovery of Temporal Abstraction from Interaction with An Environment, Ph.D. Thesis, Department of Computer Science, University of Massachusetts, Amherst, USA, (2002)
8. Doina Precup, Temporal Abstraction in Reinforcement Learning, Ph.D. Thesis, Department of Computer Science, University of Massachusetts, Amherst, USA, (2000)
9. Amy McGovern and Andrew G. Barto, Automatic Discovery of Subgoals in Reinforcement Learning using Diverse Density, Proc. 18th International Conf. on Machine Learning, Morgan Kaufmann, San Francisco, CA, pp 361–368, (2001)
10. Steven J. Bradtke and Michael O. Duff, Reinforcement Learning Methods for Continuous-Time Markov Decision Problems, Advances in Neural Information Processing Systems, volume = "7", The MIT Press, G. Tesauro and D. Touretzky and T. Leen, pp 393–400, (1995)
11. Richard S. Sutton and Doina Precup, Intra-option learning about temporally abstract actions, In Proceedings of the Fifteenth International Conference on Machine Learning, Morgan Kaufman, pp 556–564, (1998)
12. Kaelbling, L. P, Hierarchical learning in stochastic domains: Preliminary results. In Proceedings of the Tenth International Conference on Machine Learning, pp 167–173, (1993)
13. Balaraman Ravindran and Andrew G. Barto , Relativized Options: Choosing the Right Transformation , Proceedings of the Twentieth International Conference on Machine Learning , pp 608–615 , (2003)