Using Common Sense for Decision Making in an Adventure Game

Deepak Ramachandran
Computer Science Dept.
University of Illinois at Urbana-Champaign
Urbana, IL 61801

Abstract

In this paper we apply common sense knowledge to the task of decision making in an adventure game. We show how knowledge about the underlying domain, induced from a common sense knowledge base, can be used to do *reward shaping* of the environment, enabling reinforcement learning agents to learn optimal behaviours faster. The lessons learnt from this experiment could be used as a first step to finding robust applications of common sense knowledge.

1 Introduction

Many complex decision making problems in stochastic domains can be cast into the setting of *Markov Decision Processes* (MDPs). In situations involving partial observability, Partially Observable MDPs can be used. Over the last 10 years, this paradigm and its associated algorithms (value iteration, reinforcement learning, policy trees etc.) have enjoyed enormous success in academic and real-world domains.

Despite the success of these techniques, there are challenges involved in scaling them up to work in large applications. One reason is that these methods fail to take into account relevant information about the specific nature of the problem. This could be in the form of domain knowledge given by an expert, or it could be common sense knowledge from a Knowledge Base (KB) such as CYC (Ramachandran, Reagan, & Goolsbey 2005) or ConceptNet (Liu & Singh 2004). In this work we attempt the task of incorporating common sense into reinforcement learning.

In particular, we apply common sense reasoning to the situation of an agent playing an *adventure game*. This domain is particularly well-suited for the application of our methods: They represent a limited form of real-world reasoning and their complexity can be gradually scaled as our confidence in our techniques grow. The common sense knowledge we use is extracted from ConceptNet and identifies features of the state space of the MDP and actions that are relevant to the goal. This knowledge is translated into a quantitative change in the behaviour of a reinforcement learning agent for the adventure game by *reward shaping* (changing the reward function to effect faster convergence). Our experiments show that using the correct knowledge to do reward shaping leads to substantial improvements in the learning rate.

Common sense is the central problem of Artificial Intelligence but the question of its practical use is poorly understood. Large Knowledge Bases such as CYC and ConceptNet have been constructed in recent years, yet their applications to real-world problems are notoriously brittle. There is always the danger that using a particular axiom from a KB in a certain context could lead to a false conclusion; or the KB might not be complete and relying on it to always give an answer is impossible. When used as a bias for reinforcement learning however, these problems are not as severe. In the absence of relevant knowledge the reinforcement learner works just as usual. While false beliefs could lead the agent astray, a carefully constructed learning algorithm will always recover, perhaps taking longer to converge. We propose that applying common sense to decision making could be a way to "bootstrap" the use of common sense in other areas of AI.

The rest of this paper is organized as follows: In section 2 we briefly present some technical preliminaries relating to Markov Decision Problems that are needed for the rest of the paper. In section 3 we describe adventure games and how they may be modeled as MDPs. In section 4 we present our algorithm for retrieving relevant common sense knowledge from ConceptNet. In sections 5, 6 and 7 we examine methods for using this extracted knowledge to speed up reinforcement learning. Finally in sections 8 and 9 we discuss related work and our conclusions respectively.

2 Markov Decision Problems

In this section we state the basic definitions and theorems relating to Markov Decision Problems and Reinforcement Learning.

A (finite) *Markov Decision Problem* is a tuple $(S, A, T_{sa}, \gamma, R)$ where

- S is a finite set of N *states*.
- $A = \{a_1, \ldots, a_k\}$ is a set of k *actions*.
- $T_{sa} : S \mapsto [0, 1]$ is a *transition probability distribution* over the next state s', given the current state s and action taken a.
- $\gamma \in (0, 1)$ is the *discount factor*.

- $R : S \times A \mapsto \mathbb{R}$ is a *reward function*, with absolute value bounded by R_{max}.

A *policy* is a map $\pi : S \mapsto A$ and the (discounted) *value* of a policy π at state $s_1 \in S$,denoted $V^\pi(s_1)$ is given by:

$$V^\pi(s_1) = R(s_1, \pi(s_1)) + E_{s_2,\ldots}[\gamma R(s_2, \pi(s_2)) + \ldots | \pi]$$

where $Pr(s_{i+1}|s_i, \pi) = T_{s_i \pi(s_i)}(s_{i+1})$ and $E[\cdot]$ means expectation. The goal of standard Reinforcement Learning is to find an *optimal policy* π^* such that $V^\pi(s)$ is maximized for all $s \in S$ by $\pi = \pi^*$. Indeed, it can be shown (see for example (Sutton & Barto 1998)) that at least one such policy always exists for ergodic MDPs.

To *solve* an MDP means to determine the optimal policy, given complete information about its transition and reward functions. There are algorithms (e.g. policy iteration and value iteration) that solve MDPs in time pseudo-polynomial in the number of states, but these can be exponential in the number of features that define the state.

Reinforcement Learning (RL) is the more general problem of learning an optimal policy when neither the transition model nor the reward function is known, but must be discovered by taking actions in the MDP and making observations. Most reinforcement learning algorithms involve a trade-off between exploration of the state-space and exploitation of the states already explored.

For the solution of MDPs, it is useful to define the following auxiliary *Q-function*:

$$Q^\pi(s, a) = R(s, a) + \gamma E_{s' \sim P_{sa}(\cdot)}[V^\pi(s')]$$

Finally, we state the following fundamental result concerning MDPs (Sutton & Barto 1998) :

Theorem 1 (Bellman Equations)**.** *Let a Markov Decision Problem $M = (S, A, T_{sa}, \gamma, R)$ and a policy $\pi : S \mapsto A$ be given. Then,*

1. For all $s \in S, a \in A, V^\pi$ and Q^π satisfy

$$V^\pi(s) = R(s, \pi(s)) + \gamma \sum_{s'} T_{s\pi(s)} V^\pi(s') \qquad (1)$$

$$Q^\pi(s, a) = R(s, a) + \gamma \sum_{s'} T_{sa}(s') V^\pi(s')$$

2. π is an optimal policy for M iff, for all $s \in S$,

$$\pi(s) \in \operatorname*{argmax}_{a \in A} Q^\pi(s, a) \qquad (2)$$

3 Adventure Games

The kind of problems we will be considering are those arising from *text-based adventure games*. In this popular subgenre of computer games, the user plays a character who is placed in a controlled environment and has certain objectives to achieve (collect treasure, defeat enemies etc.). The emphasis is on solving puzzles that require the use of complex decision-making and everyday reasoning. Adventure games are attractive testbeds for trying out various AI techniques because they represent a circumscribed environment where general ideas can be experimentally verified without facing the full complexity of the real world (Hlubocky & Amir 2004).

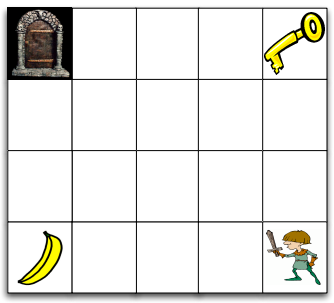


Figure 1: The Banana Game

The example that we will use to motivate this work will be the following (called the "banana game" for historical reasons) (Figure 1): Our agent is trapped in a dungeon, and he can only escape through a door in one of the cells. The door is locked but fortunately there is a key that can be found in one of the rooms of the dungeon. The agent must locate this key, pick it up and carry it to the room with the door, use it to open the door and exit to freedom. There is also a banana in one of the other rooms that the agent could try opening the door with (Though he wouldn't get very far).

Let us model this problem as an Markov Decision Process: The state representation will be a combination of the position of the agent and the items he possesses. We define the state-space through binary *features* such as `has-key` and `door-open`. It is crucial for our techniques that the feature names are descriptive and accessible to the algorithm. The available actions at each state are: moving, picking up or dropping items in the current room and attempting to open the door. These actions lead to probabilistic state transitions because each action has a small probability of failing even if its preconditions are satisfied.

In this setting various tools from the MDP literature can be used to solve and do reinforcement learning on the MDP. However on application to larger problems these methods become intractable. In the rest of this paper, we will be considering methods for improving the performance of these algorithms by using common sense knowledge about the task (in the above case, the knowledge that having the key is necessary to open the door). Observe that our game can be made arbitrarily complicated not only by increasing the number of rooms, but by adding extraneous objects to confuse and slow down the learner.

The other problem for which we will show experimental results in this paper is the "flashlight game". Briefly, a certain room needs to be lighted up. Scattered around the environment are a flashlight and a set of batteries. They must be picked up (in either order) and then combined together in the obvious way to illuminate the room. Note the similarity of this problem with partial order planning.

4 Common Sense

What constitutes common sense knowledge is hard to pin down precisely, but nevertheless it is believed to be vital for true human level AI. An intelligent agent in our previous

example would know that given that his goal is to open the door and escape the dungeon, a key would probably be of more use than a banana. Our aim then is to use common sense knowledge to determine which features of the state space are relevant to the task at hand.

There have been a number of initiatives to construct large common sense knowledge bases. At present the largest and most comprehensive of these is Cycorp Inc's CYC (Ramachandran, Reagan, & Goolsbey 2005). The representation language for CYC is a higher-order predicate logic that includes support for contexts, modal extensions and non-monotonicity. An example of a common sense axiom in CYC relevant to the door example above is:

```
(implies (and
(agent-Possesses ?Agent ?Key)
(isa ?Key Key)
(isa ?Door Door)
(opensWith ?Agent ?Door ?Key ?T1)
(immediatelyAfter ?T1 ?T2))
(holdsIn ?T2 (isOpen ?Door)))
```

The ideal way to exploit this knowledge for the problem in section 3 is to perform *abduction* on the above axiom and the goal condition:

```
(holdsIn END (isOpen Dungeon-Door)))
```

One of the conclusions of this abduction would be the statement:

```
(agent-Possesses Agent Dungeon-Door-Key)
```

This can now be exploited by the agent when it solves the MDP or performs reinforcement learning as discussed in the following sections.

However the above idea, although intuitively the correct thing to do, is difficult to implement. Translating between the state representation used in our MDP to the logical axioms required by CYC to do inference is a difficult problem. In contrast the semantic network representation of ConceptNet (Liu & Singh 2004) is much easier to work with. Knowledge in ConceptNet is less structured than in CYC, and is closer to natural language. An example of ConceptNet's representation (for the concept 'key') is shown in figure 3.

Our algorithm for determining relevant features of the state space from ConceptNet is shown in figure 2. Given the underlying MDP, we first find the features of the goal states that define the goal condition. Our method (algorithm `getGoalFeatures`) chooses features that are likely to be "flipped" by the goal-achieving action e.g. the `open-door` feature. More precisely, we choose features that have a high probability of being turned on by a transition with high reward. The algorithm then calls ConceptNet's `project-consequences` method on each feature f in the state-space. This returns a set of concepts from ConceptNet that are the possible consequences of f (e.g. the door is opened by the presence of the key). If one of these consequences include a goal feature f' then we output f as being relevant with a score equal to the score of f' (from `getGoalFeatures`) times the relevance score between f and f' returned by ConceptNet. We also add f to the list of goal features with this score. Thus the search for relevant features is done *recursively*.

Algorithm **FindRelevantConcept** (MDP $M = (S, A, T, \gamma, R)$)
1. $GF :=$ `getGoalFeatures`(M)
2. For each feature f of S not in GF
3. For $(f', t') \in GF$
4. If $(f', t') \in$ `ConceptNet.project-consequences`(f) and $tt' > 0.01$
5. $RF := RF \cup (f, tt')$
6. $GF := GF \cup (f, tt')$
7. Return RF

Algorithm **getGoalFeatures** (MDP $M = (S, A, T, \gamma, R)$)
1. $GF := \emptyset$
2. For $(s, a) \in S, A$ s.t. $R(s, a) > 0.5$
3. For each feature f of S
4. If $s.f = 0$ and $E_{s' \sim T_{sa}}[s'.f] > 0.5$
5. $GF := GF \cup (f, E_{s' \sim T_{sa}}[s'.f])$
6. return GF

Figure 2: Extracting Relevant Features

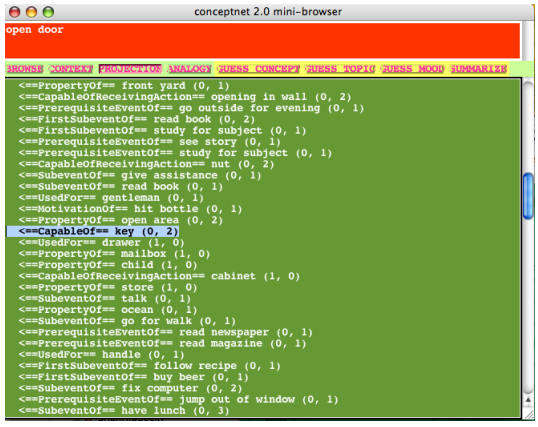


Figure 3: A projection of the ConceptNet term "open door" finds the concept "key" for the binary relation "CapableOf".

An alternative to using algorithm `getGoalFeatures` is to have annotated action names as well (such as `pick-key`) and simply use these as the goal terms.

For the banana game example, the action of opening the door in a state where the agent possesses the key gives high positive reward, and these actions transition from states where `open-door` is 0 to states where it is 1. Therefore, `open-door` is a goal feature with score 1. When `project-consequences` is performed on the feature `have-key` we get `open-door` as one of the consequences (Fig. 3) with relevance score 0.38. Thus we return `have-key` as a relevant feature with score 0.38.

5 Value Initialization

We will first attempt to use common sense to solve the simpler (but still theoretically intractable in the worst case) problem of solving the adventure game MDP i.e. determining the optimal value function and policy given the transition

model and reward functions. We assume that we have complete knowledge of the environment including the transition model. For the dungeon example, we know where the key, the banana and the door is located. We also know the effects our actions will have on the state. We can compute the optimal value function V by an iterative algorithm such as value or policy iteration. The running time of these procedures are pseudo-polynomial in the number of states but exponential in the number of state features. The value iteration algorithm is shown in figure 4. The initial value (V_0) is typically set to 0 for all states in the standard version of the algorithm.

We attempted to use common sense knowledge to speed up the convergence of value iteration by biasing the initial V values of each state depending on the knowledge we have about how desirable each state is. The idea is that the initial V_0 value assigned to each state should be a function of the set of relevant features (Section 4) that are currently active. In particular we set it to be equal to the sum of the scores of each relevant and active feature returned by algorithm `GetRelevantFeatures`. That is,

$$V_0(s) = \sum_{f \in F} f(s) \cdot \text{score}(f)$$

where F is the feature set. It is also possible to multiply this expression by a scaling factor proportional to the maximum possible value, $V_{max} = \frac{R_{max}}{1-\gamma}$.

Unfortunately, our experiments showed little improvement in the convergence times using this improved form of value iteration. In figure 5(a) we show the number of iterations needed for convergence of value iteration in the banana game, for both the biased initial values case and the uniformly zero initial values case for various state-space sizes (number of rooms)[1]. A similar plot is shown for the flashlight game in 5(b). The common sense knowledge seems to make little difference in either case.

Studying the value iteration process on this game revealed an interesting phenomenon that explains why the common sense knowledge did not help in solving the MDP. Essentially, during the initialization, we bumped up the values for "useful" states prematurely i.e. before there was a "reason" for giving them a high value or more precisely, before the bellman backup (i.e. iterative update) from the goal had reached this state. Hence, within a few iterations, the value of these bumped up states become indistinguishable from those obtained by the uniformly initialized algorithm.

Our conclusion is that for common sense knowledge of the value of a state to be useful for an MDP algorithm, it must be persistent over iterations. This is an issue that we will address in the next section.

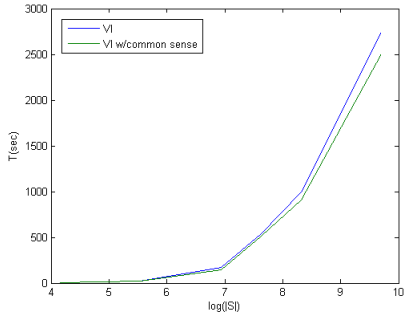
6 Reward Shaping

Reward shaping, first described in (Cohen & Hirsh 1994), is the process of restructuring the reward function of an MDP so that an agent using reinforcement learning will converge to an optimal policy faster. For example, in the banana game

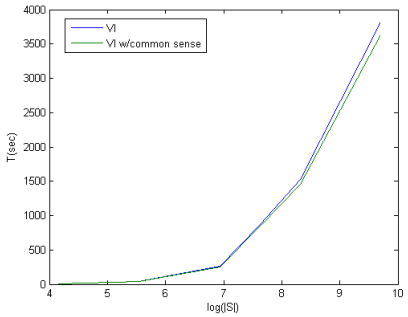
[1]Note that we have plotted the logarithm of the number of rooms on the x-axis.

Algorithm **Value-Iteration** (MDP $M = (S, A, T, \gamma, R)$, Initial Value Function V_0)
1. $V = V_0$.
2. For $t = 1, 2 \ldots$ until convergence of V_t
3. For each $s \in S$
4. $V_t(s) := \max_{a \in A}[R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V_{t-1}(s')]$
5. Return V_t

Figure 4: Value Iteration



(a) Banana Game



(b) Flashlight Game

Figure 5: Value iteration experiments

we could reshape the reward function so that the agent gains a small positive reward for states where he possessed the key. He would then learn to perform this action without needing a full backup of the bellman update to the goal state.

Reward Shaping is especially useful in environments with *delayed rewards*, where the agents must execute a complex sequence of actions, moving through a number of states before obtaining a reward. MDPs with such sparse reward structures are notoriously difficult for reinforcement learning, sometimes requiring exponentially many steps before convergence.

The danger with this method is that indiscriminate reward shaping can change the optimal policy in unforeseen ways. For example, if the agent gets a very high reward for picking up the key, it might then become optimal to simply keep picking up and dropping the key repeatedly. Reward shaping needs to be done in such a way that faster convergence is obtained but the optimal policy remains invariant.

(Ng, Harada, & Russell 1999) first characterized the set

of reward functions that satisfies this policy-invariance property in a slightly different setting. We give a similar result for our MDP framework.

Definition 1. *Let any $S, A, \gamma < 1$ and a function $F : S \times A \mapsto \mathbb{R}$ be given. We say F is a* potential-based shaping function *if there exists a real-valued function $\Phi : S \mapsto \mathbb{R}$ (called the* potential function*) such that for all $s \in S, a \in A$,*

$$F(s, a) = \gamma E_{s' \sim P_{sa}}[\Phi(s')] - \Phi(s)$$

The idea is that F is a shaping function that is added to the normal reward function R for an MDP. The potential-based property ensures that the optimal policy does not change between R and $R+F$. In fact, any such policy-invariant shaping function has to be of this form as the following theorem shows:

Theorem 2. *Let $M = (S, A, T, \gamma, R)$ be an MDP. Let $F : S \times A \mapsto \mathbb{R}$ be some shaping function. Then,*

1. *If F is potential-based, then every optimal policy for $M' = (S, A, T, \gamma, R + F)$ will also be an optimal policy for M (and vice versa).*
2. *If F is not potential-based, then there exists a transition function T and a reward function R such that no optimal policy for M' is optimal for M.*

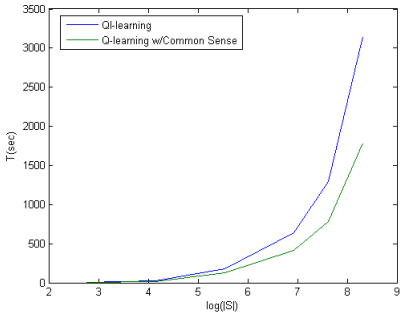
Proof. See appendix. □

The intuition behind choosing a potential-based shaping function is that the agent cannot get a net positive reward by travelling in a cycle of states $s_1 \rightarrow s_2 \rightarrow \ldots \rightarrow s_1$.

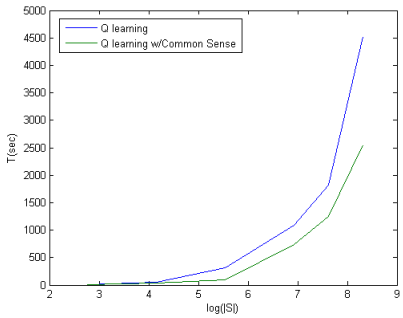
We have thus chosen to perform reward shaping with a potential-based shaping function to ensure that the optimal policy remains invariant. The potential function Φ that we use will be the same as the initial value function defined in section 5, i.e. the sum of the relevancy scores for each active feature of the state space:

$$\Phi(s) = \sum_{f \in F} f(s) \cdot \text{score}(f)$$

In contrast to value initialization, the reinforcement gained by the agent through reward shaping with common sense knowledge is persistent across iterations because the underlying MDP is now fundamentally different. This has a much more significant effect on the convergence rates. As shown in figure 6, there is roughly a 60% improvement in the convergence times. The reinforcement learner in these experiments implemented a Q-learning algorithm (see (Sutton & Barto 1998)) with a Boltzmann exploration strategy.



(a) Banana Game



(b) Flashlight Game

Figure 6: Reward shaping experiments

7 Further Ideas

Here we discuss some more methods of incorporating common sense knowledge into probabilistic decision making:

1. **Value Function Approximation :** In domains where the state space is so large that the value function cannot be efficiently represented in tabular form, a *linear basis function* is used to compactly represent an approximate value function. An LBF is a weighted sum of basis functions of state variables. Solution techniques exists for determining the weights by gradient descent, but the choice of the basis functions themselves must be made by the domain expert. Common sense knowledge can be used to deduce a relevant sets of basis functions for the value function from the specific nature of the problem e.g. a basis function that measured the distance from each item in the dungeon.

2. **Exploration Strategy :** Reward shaping is a strategy for a teacher. The reinforcement learning agent has no control itself over the reinforcement given by the environment. However one aspect of the system that it does control is the exploration strategy. Typically, agents try to find a balance between greedy exploitation strategies and exploration strategies that perform sub-optimal actions in the hope of eventually discovering higher value regions of the state-space. Common sense could guide the exploration strategy used by the agent e.g. such a strategy would prescribe attempting the action `pick-key` more often than `pick-banana`.

3. **Factoring MDPs :** Recently there has been a lot of interest in factored MDPs (Guestrin, Koller, & Parr 2001). These are formalisms that decompose the state representations into sets of independent features and do reinforcement learning on these sub-spaces semi-independently. Common sense information could be used to determine how to decompose large state-spaces e.g. pairs of features that have low relevance scores (like `have-banana` and `open-door` could be factored into separate MDPs.)

8 Related Work

To our knowledge, there has been no work so far on using common sense knowledge for improved decision making in stochastic domains, but there has been work on reward shaping by explanation-based learning (Laud & DeJong 2002). The idea of using domain knowledge for traditional planning problems has been explored in relatively more depth. For example TALPlanner (Kvarnström & Doherty 2001) is a forward-chaining planner that utilizes domain-dependent knowledge to control search in the state space.

Other related ideas we plan to explore to obtain deeper insights into the problem, is the work on Relational MDPs (Mausam & Weld 2003) and First-Order MDPs (Boutilier, Reiter, & Price 2001). While both these formalisms seem to be better suited to the task of exploiting knowledge (especially in logical form) we did not use them for this work, because as of now neither of them have been developed to the point where useful implementations exist and there has not been sufficient work on reinforcement learning or reward shaping for them.

9 Conclusions

Our results demonstrate that when used judiciously, it is possible to exploit common sense knowledge to improve reinforcement learning algorithms. The challenge is to do it in sufficient generality that it can be usefully re-applied to a number of different problems. One issue that we have finessed for now is the *reference* problem : how to connect the features in the state-based representation in the MDP to the language of the common sense KB. Here we have assumed that both representations use the same vocabulary.

The advantage that using common sense for decision making has over most other uses of common sense is its robustness to error and incompleteness. We might not find any useful common sense at all in which case our reinforcement learning algorithm will work as usual. Making the wrong conclusions from the KB might cause the learning agent to go astray. But if the knowledge is used carefully (e.g. with potential-based reward shaping), it will always be able to recover and still find the optimal policy. Thus the common sense would hopefully be beneficial most of the time or it might slow down the reinforcement learner sometimes, but it will not cause a catastrophic failure. It might turn out that supporting decision making might be the first truly useful application for common sense knowledge bases that are large but incomplete i.e. not comprehensive enough to cover all of common sense knowledge.

A Proof of Theorem 2

The optimal Q function for M, Q^*_M satisfies the Bellman equations (1),

$$Q^*_M(s,a) = R(s,a) + \gamma E_{s' \sim P_{sa}}[\max_{a' \in A} Q^*_M(s',a')]$$

Subtracting $\Phi(s)$ and rearranging terms on the right hand side:

$$\begin{aligned} Q^*_M(s,a) - \Phi(s) &= R(s,a) + \gamma E_{s' \sim P_{sa}}[\Phi(s')] - \Phi(s) \\ &\quad + \gamma E_{s' \sim P_{sa}}[\max_{a' \in A} Q^*_M(s',a') - \Phi(s')] \end{aligned}$$

Let $\hat{Q}_{M'}(s,a) \triangleq Q^*_M(s,a) - \Phi(s)$.

$$\hat{Q}_{M'}(s,a) = R'(s,a) + \gamma E_{s' \sim P_{sa}}[\max_{a' \in A} \hat{Q}_{M'}(s',a')]$$

So $\hat{Q}_{M'}$ satisfies the Bellman equation for M' and must be the optimal Q function. Therefore the optimal policy for M' satisfies:

$$\begin{aligned} \pi^*_{M'}(s) &\in \operatorname*{argmax}_{a \in A} Q^*_{M'}(s,a) \\ &= \operatorname*{argmax}_{a \in A} Q^*_M(s,a) - \Phi(s) \\ &= \operatorname*{argmax}_{a \in A} Q^*_M(s,a) \end{aligned}$$

and so π^* is also optimal for M. For the other direction, we can simply interchange the roles of M and M'.

For the proof of part 2 see (Ng, Harada, & Russell 1999).

References

Boutilier, C.; Reiter, R.; and Price, B. 2001. Symbolic dynamic programming for first-order MDPs. In *Proc. Seventeenth International Joint Conference on Artificial Intelligence (IJCAI '01)*, 690–700.

Cohen, W. W., and Hirsh, H., eds. 1994. *Reward functions for accelerated learning*. CA: Morgan Kauffman.

Guestrin, C.; Koller, D.; and Parr, R. 2001. Multiagent planning with factored mdps. In *In 14th Neural Information Processing Systems (NIPS-14)*.

Hlubocky, B., and Amir, E. 2004. Knowledge-gathering agents in adventure games. In *AAAI-04 workshop on Challenges in Game AI*.

Kvarnström, J., and Doherty, P. 2001. Talplanner: A temporal logic based forward chaining planner. *Annals of Mathematics and Artificial Intelligence (AMAI)* Volume 30:pages 119–169.

Laud, A., and DeJong, G. 2002. Reinforcement learning and shaping: Encouraging intended behaviors. In *Proceedings of the 19th International Conference on Machine Learning (ICML-02)*, 355–362.

Liu, H., and Singh, P. 2004. Conceptnet: A practical commonsense reasoning toolkit. *BT Technology Journal* 22.

Mausam, and Weld, D. S. 2003. Solving relational MDPs with first-order machine learning. In *Proc. ICAPS Workshop on Planning under Uncertainty and Incomplete Information*.

Ng, A. Y.; Harada, D.; and Russell, S. 1999. Policy invariance under reward transformations: theory and application to reward shaping. In *Proc. 16th International Conf. on Machine Learning*, 278–287. Morgan Kaufmann, San Francisco, CA.

Ramachandran, D.; Reagan, P.; and Goolsbey, K. 2005. First-orderized researchcyc: Expressivity and efficiency in a common-sense ontology. In *Papers from the AAAI Workshop on Contexts and Ontologies: Theory, Practice and Applications*.

Sutton, R. S., and Barto, A. G. 1998. *Reinforcement Learning*. MIT Press.