Abstracting Reusable Cases from Reinforcement Learning

Andreas von Hessling and Ashok K. Goel

College of Computing
Georgia Institute of Technology
Atlanta, GA 30318
{avh, goel}@cc.gatech.edu

Abstract. Reinforcement Learning is a popular technique for game-playing because it can learn an optimal policy for sequential decision problems in which the outcome (or reward) is delayed. However, Reinforcement Learning does not readily enable transfer of acquired knowledge to other instances. Case-Based Reasoning, in contrast, addresses exactly the issue of transferring solutions to slightly different instances of a problem. We describe a technique for abstracting reusable cases from Reinforcement Learning. We also report on preliminary experiments with case abstraction in a microworld.

1 Introduction

A game-playing agent may use Case-Based Reasoning (CBR) in several different ways. Firstly, an agent may use CBR for deciding on a specific plan in a given game situation by retrieving and adapting a plan in a similiar game situation, or an agent may use CBR for deciding on a specific action in a given game state by retrieving and adapting an action in a similar game state. For example, Aha, Molineaux and Ponsen[1] (2005) describe a game-playing agent that uses CBR for action selection in the real-time strategy game called Stratagus. Secondly, an agent may use a past case to initialize another method for plan or action selection. Although we are unaware of any work that actually uses CBR in this way, Gabel and Riedmiller[2] (2005) describe how cases can be used for approximating the state value function in Reinforcement Learning, and similarly West[3] (2005) reports on the potential use of cases for initializing the policy for action selection in Reinforcement Learning. Thirdly, cases may be used as abstractions of the results of playing a game by another method. For example, an agent may use Reinforcement Learning to learn about the right action in a given state, and then abstract and encapsulate action selection as a case for transfer to other, similar states. In this paper, we investigate this, the third, use of CBR in game playing.

Reinforcement Learning (RL) is a popular technique for game-playing because it can, without supervision, learn an optimal policy for sequential decision problems in which the outcome (or reward) is delayed. RL has been successfully used in learning to play games as complex as Backgammon[4]. A limitation of current RL techniques however is that they do not readily enable transfer of the acquired knowledge to other instances of game playing (even if the other instances are only slightly different), but instead require learning of an optimal policy for each instance. CBR, in contrast, addresses exactly the issue of transferring solutions to slightly different instances of a problem. The motivation for the work described in this paper, then, is to use cases for enabling transfer of RL; the goal is to develop a technique (i.e., a representation and an algorithm) for abstracting reusable cases from RL.

Abstracting reusable cases from RL raises several technical issues: (a) what might be the content of a case, (b) how might this case be represented, and (c) how might a case be abstracted from RL. In our present work, a case contains a problem-solution 2-tuple, where the problem is represented as a feature vector describing a world and the solution is represented as a decision tree (DT), which is a concise classification of the optimal action based on the values of the feature vector. The technique for abstracting cases from RL has the follwing steps: first, in a pre-processing phase, an abstraction of the world is modeled as a Markov Decision Process (MDP) and the optimal policy for a certain task in the world is generated by Q-learning, a specific version of RL. Then, in a training phase, for some selected states, the feature values in the world and the action in the states are recorded. Finally, in a post-processing phase, a decision tree (DT) is learned from the training instances. The abstracted case contains the feature vector describing the world and the learned DT representing the optimal actions.

The rest of this paper is organized as follows: first, we provide some background on Q-learning, DT-learning, and the microworld used in our experiments. Then, we explain the technique for abstracting cases in the form of DTs from RL. Finally, we present some preliminary conclusions.

2 Background

This section describes the microworld used in our experiments, and provides a brief overview of Q-learning and DT learning.

2.1 Domain Specification

Jones and Goel[5] (2004) and Ulam, Jones and Goel[6] (2004) describe our earlier work on game playing in the Freeciv domain. Freeciv is an interactive, turn-based, multi-player strategy game. The world in Freeciv is non-deterministic, partially observable, and has a huge state space. Because of the enormous complexity of Freeciv, we found that running meaningful experiments on case abstraction in this domain is very complicated and intricate. For this reason, in the present work we ran experiments in a much smaller and simpler microworld.

The microworld in the present work represents an office room: the room consists of map tiles arranged in a grid, each of which may have a certain color, decoration type, or exit sign associated with it. While the exit signs show the direction to the exit and are placed with purpose, the assignment of decoration and colors to tiles is random. Walls may be present in map tiles that restrict the ways in which a certain grid position may be reached from another one. Exits may be present in any non-wall tile.

The goal of the agent in the office room is to reach an exit. The agent must make deterministic movements in each time step. On any tile, the agent can access any other tile that is adjacent to the left, right or straight, relative to its current position and bearing. Figure 1 demonstrates one possible office world.

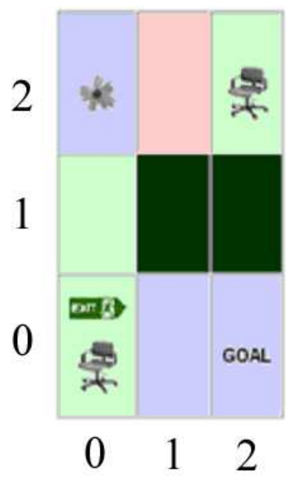

Fig. 1. An instance of the microworld. Map tiles may have different decoration such as colored carpets, chairs or walls. Walls are denoted by dark tiles. An exit sign points in the direction of the goal. Tiles are indexed by (row, column)

To see the relationship between the Freeciv and the above microworld, consider the task of capturing an enemy city in Freeciv. Again, the world consists of map tiles arranged in a grid, each of which may contain an enemy city, enemy soldiers or friendly soldiers. Again, the agent must make deterministic movements in each time step, and again the agent has a small number of actions to select from. Notice that the task of capturing an enemy city occurs repeatedly during the course of playing Freeciv, with only small differences in each instance.

2.2 Q-Learning

Q-learning[7] is a temporal difference machine learning method that addresses the temporal credit assignment problem: how can an agent trying to maximize its cumulative discounted reward in a sequential decision process determine the utility values of alternative actions in a particular state, if this reward is delayed? In other words, Q-learning can generate advice about which of the alternative actions should be performed in a certain state in order to maximize the reward over time. For this, Q-learning requires the following input: (a) a set of states, (b) available actions in the states, (c) transitions between these states, (d) a reward function, and (e) a discount factor. Given these definitions in a specific MDP representation of a world, Q-learning can generate an assignment of an action for every state ("policy") that is "optimal", meaning that it provides the greatest possible discounted reward over time. This assignment is created by selecting each state's action with the highest utility estimates, which are determined by executing the following update rule until a certain termination criterion is met:

$$Q\left(a,\ s\right)\ \leftarrow\ Q\left(a,\ s\right)\ +\ \alpha\Big(R\left(s\right)\ +\ \gamma\ \max_{a'}\ Q\left(a',\ s'\right)\ -\ Q\left(a,\ s\right)\Big)\,.$$

Here, $Q\left(a,\ s\right)$ denotes the utility estimate for action a in state s, $R\left(s\right)$ refers to the immediately perceived reward, γ is the discount factor for each time step and s' is the state that the agent is in after executing action a. The parameter γ signifies the learning rate and may be changed to adjust the speed of the learning process.

One of Q-learning's great advantages is that it does not require any prior domain knowledge to perform the task of finding the optimal policy. However, at the same time, it does not provide any generalization capability between instances of a problem class. For example, if the reward for just one transition is changed, the optimal policy may have changed and thus has to be re-computed for the modified state space. Thus, transfer of learned knowledge is not feasible.

2.3 DT Learning

DT learning[9] is a function approximation technique that provides classification capability after supervised training on offline data sets. DTs aim at providing generalization capability by finding patterns in the data that explain the goal variable. For this task, attribute tests are used to partition a set of instances into subsets. Information Gain is a measure describing the inverse of the decrease in entropy (randomness) for an attribute and can be used to rank the attribute tests in the tree according to their performance. The principle of picking the attribute that performs best when classifying the instances in the data set is a heuristic that is the basis of the inductive bias inherent in DT learning, which amounts to preferring shorter trees over longer trees.

As opposed to many other function approximation techniques such as Neural Networks, the advantage of DTs is that their restrictive bias (in particular, their output) is the class of sentences in propositional logic and thus amenable to human understanding.

3 Abstracting Cases from RL

We now describe our approach to transforming knowledge about an optimal policy into a classification of the optimal action in a state by this state's feature values. Figure 2 shows the task structure of the task of case abstraction. The following sections describe the tasks of pre-processing by Q-learning, training, and post-processing (the three left-most leaf nodes in the tree in the figure). Section 3.1 shows how standard Q-learning is applied to our example domain in order to determine an optimal policy. Then, in section 3.2, we explain how the optimal action and the state's feature values are recorded. Section 3.3 presents a description of how these records are used to generate concise classifications in DT learning.

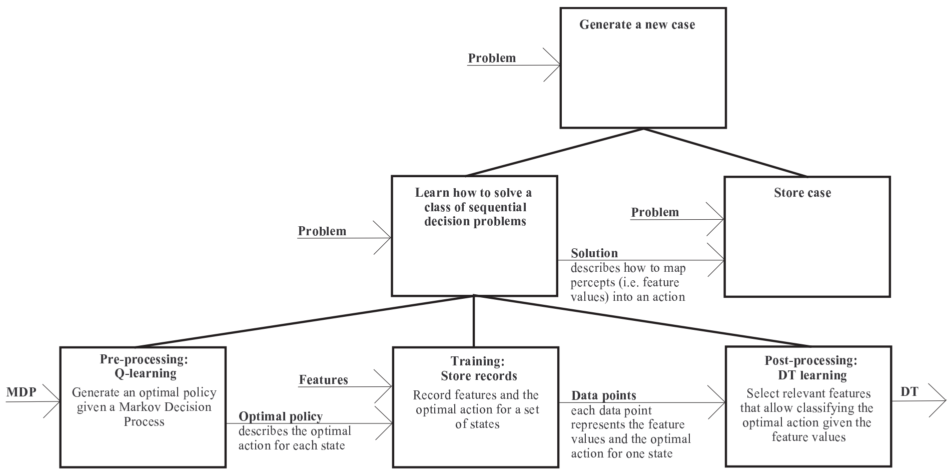
Generate a new case
Problem
Learn how to solve a class of sequential decision problems
Problem
Store case
Problem
Solution
describes how to map percepts (i.e. feature values) into an action
MDP
Pre-processing: Q-learning
Generate an optimal policy given a Markov Decision Process
Optimal policy
describes the optimal action for each state
Features
Training: Store records
Record features and the optimal action for a set of states
Data points
each data point represents the feature values and the optimal action for one state
Post-processing: DT learning
Select relevant features that allow classifying the optimal action given the feature values
DT

Fig. 2. The structure of the task of abstracting cases from RL.

3.1 Determining an Optimal Policy

In the initial pre-processing step, Q-learning is used in the office microworld in order to determine an optimal policy which contains the knowledge that is being transformed in subsequent steps. We define an agent's state to be a map tile and its potential successor states to be the map tiles adjacent to the North, East, South and West, excluding walls. Furthermore, a positive reward is set for transitions to the terminal exit tile, and 0 for any other transition. Notice that features existing on map tiles such as colors, decoration types or exit signs do not influence the operation of the Q-learning algorithm. Instead, Q-learning determines each state's action that yields the highest reward based solely on the states, actions, transitions, and rewards given by the problem instance.

3.2 Storing Records

Whereas Q-learning offers recommendations for choosing an action in a state that maximize the expected utility, it only views this optimal action in the context of the current state and the successor states. It does not explicitly take into account the world's feature values that may have caused this particular action to be preferred over other available actions. That is, it does not extract environmental patterns that may align with a particular action. In our approach, we assume the accessibility of feature values of the world in a given state. This enables transforming the knowledge represented in the temporal context to empirical knowledge about the association between values of the optimal action and world features. In this knowledge transformation step, a state's feature values and its optimal action are stored as a data point. As opposed to Q-learning, this recording step does not require information about rewards or possible transitions etc. The training process generates a data point in an (n+1)-dimensional space, where n is the number of features and where the goal variable is the optimal action. This data point can be viewed as a Production Rule: in a particular state, the reactive agent perceives certain world features which make it perform the given action. This terminology emphasizes the desire to extract decision-making knowledge as opposed to mere empirical knowledge. However, this requirement is not necessarily met in cases of irrelevant features, and thus we refer to the stored records as data points. The purpose of these data points is that they allow for the application of supervised learning techniques, as we will describe in the next section.

In the office microworld, converting the knowledge is done by placing the agent in a random location in the office space and allowing it to continuously repeat its perception-action loop, which makes it store the data point in this particular state and then executing the recommended action, until it has reached the goal. Figure 3 shows an example run in a hand-crafted instance of the office scenario. The red line shows the path the agent takes following the optimal policy from its starting position to the exit, perceiving any of its current position's color, decorations and exit signs. An exemplary data point that was recorded in state (2,2) of the map in Figure 3 is given by the following pseudo-code:

```
((color = "green"),
 (decoration = "chair"),
 (exitSign = "points left"),
 (optimalAction = "turn left and go ahead"))
```

3.3 Applying DT Learning: Selecting Relevant Features for the Solution

The process of generating data points does not perform any abstraction functionality by itself. That is, the pure memorizing of features with the corresponding taken action does not necessarily generate knowledge that allows for generalization. A post-processing step is needed to determine the relevance of features for the actions. This step is performed by DT learning, using the optimal action as the goal variable. The DT learner generalizes the learned knowledge exploiting its function approximation functionality. The purpose of this is not only to find

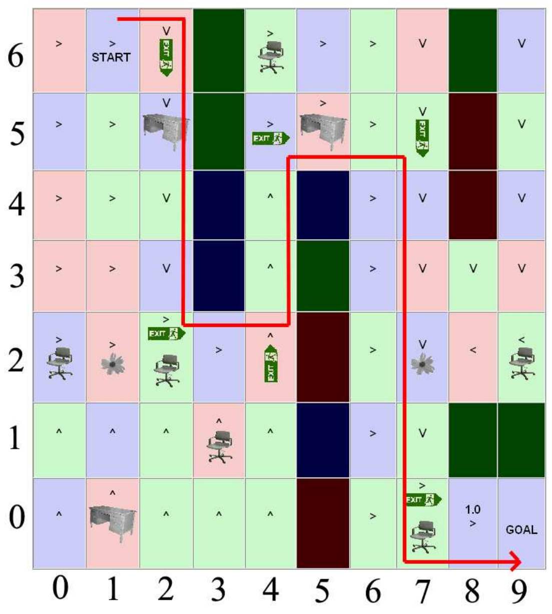
Fig. 3. An example path an agent takes when placed in tile (6,1). The agent finds its way indicated by the red line to the goal at (0,9), according to the arrows in each tile that represent the optimal action in that state

”meaningful” patterns in the environmental features in conjunction with certain actions, but also to separate these patterns from irrelevant noise in the data. In particular, the hope here is to find causal relations between the features and the action - relations that are functions of features and produce an action that ”makes sense”. The Information Gain measure fits the needs for this separation, as it measures the decrease in randomness when a certain feature is applied to the data set. That is, features with high Information Gain are more likely to be important than others, and fewer of them are needed to describe the situation than less informing features.

For illustrating this DT learning step in our office scenario, we use the data points that were recorded during the agent’s execution along the path shown in figure 3. Applying an ID3[9] implementation on this data set and using the optimal action as the target variable, we obtained the following output (graphically depicted by figure 4):

```
if (exitSign == "N/A" (Information Gain = 0.818))
  optimalAction = "go straight ahead";
else if (exitSign == "points right" (Information Gain = 0.818))
  optimalAction = "turn right and go ahead";
else if (exitSign == "points left" (Information Gain = 0.818))
  optimalAction = "turn left and go ahead";
```

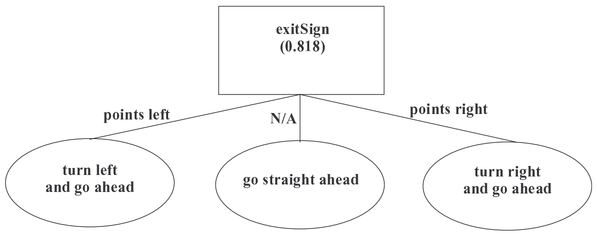

Fig. 4. A prototypical DT that classifies the optimal action only on the basis of the exitSign feature

This result indicates that DT learning abstracted from irrelevant features such as chairs or colors - it can completely describe the knowledge-based rules used to exit the building with the direction the exit sign is pointing to. Notice that the rules are not only very short and human-readable, but are also based on empirical correlation (or, in the prototypical case, the rules are based on knowledge describing how to make good decisions) as opposed to the knowledge generated by the Q-learner. The generated DT represents the solution generated for the given problem of exiting a building. Obviously the scenario used here was hand-crafted to yield this prototypical finding. If the world was less perfect, such as when the exit sign at position (2,2) would be missing, the answers also get less clear (figure 5 depicts the following output in a graphical manner):

```
if (exitSign == "N/A" (Information Gain = 0.818))
  if (decoration == "N/A" (Information Gain = 0.244))
    optimalAction = "go straight ahead";
  else if (decoration == "desk" (Information Gain = 0.244))
    optimalAction = "go straight ahead";
  else if (decoration == "plant" (Information Gain = 0.244))
    optimalAction = "go straight ahead";
  else if (decoration == "chair" (Information Gain = 0.244))
    optimalAction = "turn left and go ahead";
else if (exitSign == "points right" (Information Gain = 0.818))
  optimalAction = "turn right and go ahead";
else if (exitSign == "points left" (Information Gain = 0.818))
  optimalAction = "turn left and go ahead";
```

Here, the algorithm must include decoration as a feature because the environmental patterns do not align as clearly with the optimal action as they did in the previous example. In this case, the agent would go left only when the current state happens to contain a chair, given it does not perceive an exit sign. However, the inclusion of many features in the DT may mean overfitting to the training data and can be addressed by pruning. Similarly, noise can occur due to (a) incorrect data, such as an exit sign pointing to the wrong direction, (b) an insufficient amount of data, (c) a non-deterministic behavior of the world generating the data, or (d) if the features are not capable of describing the concept

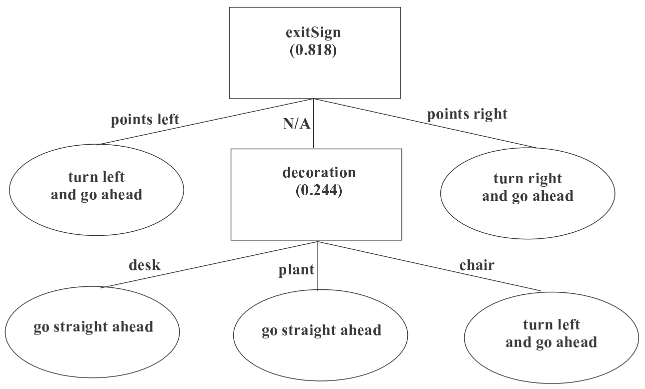

Fig. 5. In order to describe the decision knowledge, this DT that has to take into account the relatively irrelevant feature decoration (Information Gain: 0.244) because this world is less perfect than the previous one in that one exit sign was removed

of an "optimal action"[8]. It is the subject of future research to address these issues and and their impact on the scalability of this approach to agents that play computer games involving highly complex state spaces.

4 Conclusions

Reinforcement Learning is a popular technique for game-playing because it can learn an optimal policy for sequential decision problems in which the outcome (or reward) is delayed. However, Reinforcement Learning does not enable transfer of acquired knowledge to other instances. CBR, in contrast, addresses exactly the issue of transferring solutions to slightly different instances of a problem. In this paper we have presented a method for abstracting cases from Q-learning. This abstraction is achieved by combining the world's feature values and the optimal action in each state into one data point for DT learning.

References

1. Aha, D.W., Molineaux, M., & Ponsen, M. (2005). Learning to win: Case-based plan selection in a real-time strategy game. To appear in Proceedings of the Sixth International Conference on Case-Based Reasoning. Chicago, IL: Springer.
2. Gabel, T., & Riedmiller, R. (2005) CBR for State Function Approximation in Reinforcement Learning. To appear in Proceedings of the Sixth International Conference on Case-Based Reasoning. Chicago, IL: Springer.
3. West, G. Representations and Approaches for Continent Exploration in the Freeciv Domain. Internal Project Report, Design Intelligence Group, College of Computing, Georgia Institute of Technology.
4. Tesauro, G. Temporal Difference Learning and TD-Gammon Communications of the ACM, March 1995 / Vol. 38, No. 3.

5. Jones, J., & Goel, A.K. Revisiting the Credit Assignment Problem. In Fu, D. & Orkin, J. (Eds.) Challenges of Game AI: Proceedings of the AAAI'04 Workshop (Technical Report WS-04-04). San Jose, CA: AAAI Press.
6. Ulam, P., Jones, J., & Goel, A. K. Model-Based Reflection in Game Playing. In Fu, D. & Orkin, J. (Eds.) Challenges of Game AI: Proceedings of the AAAI'04 Workshop (Technical Report WS-04-04). San Jose, CA: AAAI Press.
7. Watkins, C., & Dayan, P. Technical note: Q-learning. PhD thesis, 1992.
8. Russell, S. and Norvig, P. (2003). Artifical Intelligence: A Modern Approach, Second Edition, Prentice Hall.
9. Quinlan, J.R. (1979). Discovering rules by induction from large collections of examples, D. Michie (ed.), Expert Systems in the Microelectronic age.