Reducing the Memory Footprint of Temporal Difference Learning over Finitely Many States by Using Case-Based Generalization

Matt Dilts, Héctor Muñoz-Avila

Department of Computer Science and Engineering, 19 Memorial Drive West,
Lehigh University, Bethlehem, PA 18015
{ mjd309,hem4 }@lehigh.edu

Abstract. In this paper we present an approach for reducing the memory footprint requirement of temporal difference methods in which the set of states is finite. We use case-based generalization to group the states visited during the reinforcement learning process. We follow a lazy learning approach; cases are grouped in the order in which they are visited. Any new state visited is assigned to an existing entry in the Q-table provided that a similar state has been visited before. Otherwise a new entry is added to the Q-table. We performed experiments on a turn-based game where actions have non-deterministic effects and might have long term repercussions on the outcome of the game. The main conclusion from our experiments is that by using case-based generalization, the size of the Q-table can be substantially reduced while maintaining the quality of the RL estimates.

Keywords: reinforcement learning, case similarity, case-based generalization

1 Introduction

Over the years there has been a substantial interest in combining case-based reasoning (CBR) and reinforcement learning (RL). The potential for integrating these two techniques has been demonstrated in a variety of domains including digital games [1] and robotics [2]. For the most part the integration has been aimed at exploiting synergies between RL and CBR that result in performance that is better than each individually (e.g., [3]) or to enhance the performance of the CBR system (e.g., [4]). Although researchers have pointed out that CBR could help to enhance RL processes [5], comparatively little research has been done in this direction, and the bulk of it has concentrated on tasks with continuous states [6,7,16,17].

In reinforcement learning [8], an agent interacts with its environment in a cyclic pattern. The agent first perceives its state and selects an action to execute. The environment updates the state to reflect changes caused by the agent's action and potentially other actors, and provides the agent with a numerical reward. The reinforcement learning problem is to develop a policy (a mapping from each state to an action that should be taken in that state) that will maximize the sum of rewards the agent will receive in the future.

In this paper we use CBR to address a limitation of temporal difference learning (TD learning), a widely used form of reinforcement learning [8]. One of the reasons why TD learning has achieved such a widespread use is because it allows the agent to act based on experience in the same episode from when it was learned. This characteristic of TD learning allows it to frequently converge rapidly to an optimal policy faster than Monte Carlo or Dynamic Programming methods [8].

Most implementations of TD learning maintain a Q-table, which is a mapping of the form:

$$\text{Q-table: States} \times \text{Actions} \rightarrow \text{Values}$$

That is, the Q-table associates with each state-action pair a value *v*, which represents the expected value of taking the corresponding action in the corresponding state. When an agent takes an action *a* while in an state *s*, the value of the corresponding entry in the Q-table *(s,a)* is updated according to the reward from executing *a*. A drawback of TD learning is that the Q-table can grow very large. For this reason people have suggested generalization methods that reduce the size of the Q-table. For example, neural networks have been used to allow the generalization of states across multiple Backgammon games [10].

In this paper we explore using case-based similarity metrics to reduce the size of the Q-tables when the set of possible states that the agent can visit is finite. In a nutshell, the basic idea is to use a similarity relation $SIM_{state}(s_1,s_2)$ that holds if s_1 and s_2 are very close. Instead of maintaining one entry in the Q-table for each state, the agent maintains one entry for each group of states that are similar enough according to SIM_{state}. Clearly, this will reduce the size of the Q-table. However, this might affect the performance of the TD learning process, possibly reducing the speed of convergence to an optimal policy or making this convergence impossible.

We hypothesize that case-based generalization can attain the reduction of the Q-table while still maintaining the performance of the TD learning process, and potentially even improving it as a result of the reduction in the space of possibilities that the TD learning algorithm must consider. We tested this hypothesis by performing experiments on a gaming testbed. Our experiments confirm our hypothesis pointing towards the potential of case-based similarity to generalize Q-tables while still achieving good performance.

The paper continues as follows: the next section describes our gaming testbed. Section 3 provides a brief overview of TD learning. Then we describe the case-based generalization of Q-tables in Section 4. Then we describe the empirical evaluation. Section 6 describes related work and Section 7 makes some final remarks.

2 Motivation Domain: The Descent Game

We performed experiments on our implementation of Descent, a tabletop, turn-based game where actions have non-deterministic effects that might have long term repercussions on the outcome of the game. This is the kind of game where one would expect temporal difference learning to perform well since episodes last long and, hence, the learning process could take advantage of using the estimates of the Q-values in the same episodes in which they occur.

Descent is our implementation of a tabletop game *Descent: Journeys in the Dark®* in which one to four players control four hero characters cooperating to defeat the overlord, which is controlled by another player [11]. Unlike games like Dungeons & Dragons® where the goal is for the players and the dungeon master to combine efforts to tell a riveting story, in this game the overlord's goal is purely to annihilate the heroes and, as such, he has a fully fleshed-out rule set in this game just like the heroes. Descent is a highly tactical, turn based, perfect information game (each player sees the complete board), and has non-deterministic actions (i.e., actions performed by a player might have multiple outcomes such as the attack action may or may not hit a monster). The entire game is set in a fantasy setting with heroes, monsters, treasures, dragons, and the like. We implemented a digital version of Descent that uses a subset of the original set of rules of the tabletop version, yet is self-contained (i.e., a complete game can be played without referring to rules not implemented).

The goal of the game is for the heroes to defeat the last boss, ***Narthak***, in the dungeon while accumulating as many points as possible. The heroes gain 1200 points for killing a monster, lose 170 points for taking a point of damage, gain 170 points for removing a point of damage, and lose 850 points the hero's for dying. When a hero dies, he respawns at the start of the map with full health. Furthermore, the heroes lose 15 points per turn. This form of point entropy encourages players to finish the game as quickly as possible.

We hard-coded a competent version of the overlord and developed an API that allows an AI agent to control the hero characters, taking the place of the human hero player. This AI agent sends messages to the game server while receiving and evaluating incoming messages from the game server.

Each hero has a number of hit points called wounds, a weapon, armor, a conquest value, a movement speed, 1 special hero ability, and 3 skills (ranging from additional special abilities to basic additional stats). Heroes may move in any direction including diagonals by spending 1 movement point. They may move through their own allies, but may not move through monsters. It takes 2 movement points to open or close a door. Heroes may not move through obstacles (such as the rubble spaces that adorn the map). This means that the AI agent must make a complex decision considering multiple factors: whether to move and if so in what direction, whether it should move forwards and risk attack from a monster or wait for other players (which is always detrimental because of the loss of health per turn). To simplify the AI choices, in our implementation, every turn the heroes can take one of three actions: battle, advance, or run, each of which grants the heroes a different number of attacks and movement points. If the hero declares an advance or battle, it will move closer to the nearest monster and attack whenever possible. If the hero declares a run, it will retreat towards the start of the map.

After all of the heroes have taken their turn, the overlord's turn begins. The current hardcoded overlord AI is set to have each monster pick a random hero on the map and move towards that hero and attack the hero if he is within melee range. Monsters also have special abilities, move speeds (with same restrictions as the heroes), specific attack dice, armor values, and health values.

3 TD Learning

TD learning is a widely used form of reinforcement learning wherein an agent learns a policy which, for every state of the agent's world, maps an estimate of the value of taking each applicable action in that state; the goal of the agent is to maximize the sum of the future rewards it receives.

3.1 Q-Tables and Policies

TD learning algorithms maintain a ***Q-table*** of expected rewards for each state-action pair. A Q-table stores a value for each state-action *(s,a)* pair (*Q(s, a)* $\rightarrow$ *value*), where the table in this case has game states as row labels, and abstract game action names as column labels. Each entry in the Q-table is called a ***Q-value***.

Given a Q-table, a ***policy*** can be inferred by greedily selecting for each state the action with the highest Q-value. This is called a greedy policy, Π_{greedy}, and is defined as:

$$\Pi_{greedy}(s) = \arg\max_a Q(s,a)$$

3.2 TD Learning Updates

TD learning algorithms balance between exploiting the greedy policy from the current Q-table and exploring other alternative actions even when they do not correspond to the greedy policy. Exploration is done to avoid local minima in which the Q-values converge towards selecting an action *a* for a state *s* even though there is another action *a'* that over the long run will result in a higher Q-value for *s*. An strategy, called ***$\in$-greedy***, for balancing exploitation and exploration in TD learning is selecting the greedy action, $\Pi_{greedy}(s)$, for state *s* with probability 1-$\in$, where $\in$ is an input parameter in the range [0,1]. This parameter is usually set lower than 0.5 so that most of the time the greedy action for state *s* is selected. With probability $\in$ a random selection is made among the set of actions that can be applied in state s.

An alternative to $\in$-greedy is called ***softmax*** [8], whereby the probability of selecting an action a for state s is relative to its value Q(s.a). Hence, actions with high Q-values will be more likely to be selected while actions with low Q-values, including those that have a Q-value of 0, will still have a non-zero probability of been selected. The agents we use in our experiments perform a softmax selection.

Regardless of how the action is selected, TD learning uses bootstrapping, in which the agent updates the Q-values based on its own estimates of the Q-value. The following formula is used to update the Q-value Q(s,a) for the action *a* selected in state *s*:

$$Q(s,a) \leftarrow Q(s,a) + \alpha(R + \gamma Q(s',a') - Q(s,a)) \qquad (1)$$

Here R is the reward obtained after taking action a in state s, and α is the step-size parameter, which determines the extent of the update done to the Q-value; lower values will reduce the extent of the update while larger values will increase it. The value of γ, which is called the discount rate parameter, adjusts the relative influences of current and future rewards in the decision making process. The state s' is the state that was reached after taking action a in state s, and a' is the action that was taken after reaching state s'. Thus, the value of Q(s,a) is updated by looking one step ahead into the estimate of the subsequent state and action pair that the agent visited.

3.3 TD Learning to Control Descent Agents

One of the main challenges of using TD learning for controlling Descent agents is the large number of potential states. It would be impossible to generate and populate a full state table with values given the amount of time it takes to run a single game of Descent. For example, if we assume we would need a different state for each possible monster and hero positioning and a different state for each combination of hero and monster health amounts, given a 26×26 map, 16 monsters, and 4 heroes (and ig noring heroes' health), we would need 4.0×10^{56} states. Because of this, state abstractions are needed to lower the number of possible states; this is a common practice when using reinforcement learning in games [12].

Each state is represented by the following abstraction: the hero's distance to the nearest monster, the number of monsters within 10 (moveable) squares of the hero, the estimated damage those monsters would inflict if they were to all attack the hero, and the hero's current health. In general, the distance to the nearest monster is no more than 20 movable squares. The number of monsters within range is usually no more than 6, the estimated damage taken is typically no more than 18, and the most health any hero has is 12. This reduces our 55 million states problem down to 6500 for each hero. While the reduction is substantial, heroes will visit only dozens of states in an average game. Hence, some form of state generalization is needed.

4 Case-based Generalization of Q-tables

Frequently, the Q-tables are pre-generated and reside in memory. That is, the agent allocates a memory footprint of the order of O(|S|×|A|), where S is the set of possible states that the agent can visit and A is the set of possible actions that the agent can take.[1] Borrowing ideas from CBR, rather than generating a large table and filling it in with exploration and exploitation choices, what we propose instead is to begin with a blank Q-table and slowly fill it in with new cases, which we view as entries in the Q-table, as the agent encounters them. Furthermore, we propose using a case similarity function to encompass many possible different entries in the Q-table. For example, standing near a monster with 5 health is not much different than standing near a monster with 4 health, so the agent will consider those two to be essentially the same state when generating and using the Q-table. Consider a 2-dimensional map where each point on the map represents a state. Initially there is a completely empty Q-table and the map is not covered at all. When the agent visits the

[1] Actions do not need to be applicable in every state; if an action a is not applicable in an state s, its corresponding Q-Value, Q(s,a), can be initialized with a special value, such as -1, to represent this fact.

first state, a new entry is made to the Q-table. The state can be thought to cover an area in the map as shown in Figure 1 (left); as usual, the point in the middle of the circle represents the state the agent is currently in and the circle around that point represents the similarity function's coverage of similar states. After visiting 5 different states, the map could be covered as shown in Figure 1 (right).

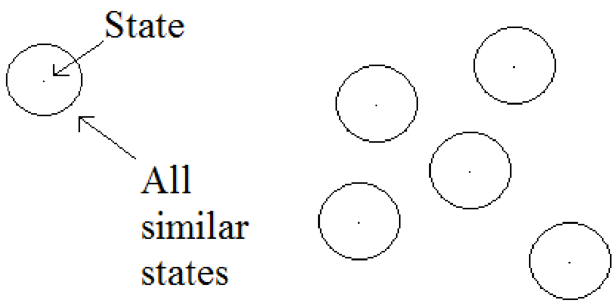

Figure 1. Graphic depiction of case base coverage

Each circle in Figure 1 (right) represents all states which are close enough to the state s_{first} first visited. Hence when visiting any state s_{new} that is similar to s_{first}, the agent does not need to add a new entry in the Q-table. Instead, s_{first} acts as a proxy for s_{new}. This has the following two consequences:

- For selecting which action to choose from state s_{new}, we do a softmax selection based on the Q-values for the actions in s_{first}, which will result in the selection of an action a.
- For doing the update of the Q-values, the agent updates the entry for $Q(s_{first},a)$ as indicated in Formula 1.

In other words s_{new} and s_{first} are considered to be the same state for the purpose of determining our policy and for the purpose of updating the Q-table. Overlap in the table is guaranteed since similarity does not take action choice into effect. So there will be multiple different state similarity blocks that use different actions. Furthermore, it is possible to generate a state near an already existing state, causing overlap. When overlap occurs the agent is essentially considered to be in the same "state" just with multiple different action choices. Below we present the algorithm, SIM-TD, that takes into account the notion of case-based similarity into the standard temporal difference algorithm. It initializes the Q-table Q with an empty table and runs n episodes, each of which calls the procedure $\text{SIM-TD}_{episode}$, which updates Q.

SIM-TD(α, γ, n)
Input: α: step-size parameter, γ: discount factor, n: number of episodes
Output: Q: the Q-table

```
Q $\leftarrow$ []  // the Q-table is initially empty; no memory allocated for  it
S $\leftarrow$ [] //current list of states represented in Q
k $\leftarrow$ 1
```

```
while (k ≤ n) do
    Q ← SIM-TD_episode(α, γ, Q, S)
    k ← k + 1
end-while
return Q
```

The procedure SIM-TD$_{episode}$ is shown below. The crucial difference with standard temporal difference occurs at the beginning of each iteration of the while loop. Namely, it checks if there is a state s' similar to the most recently visited state s_{new}. In such a case, s' is used for the selection of the next action and for the TD update of the Q-table Q. If no such a similar state s' exists, then a new entry for state s_{new} is added to the table.

SIM-TD$_{episode}$(α, γ, Q, S)
Input: α: step-size parameter, γ: discount factor, Q: the current Q-table, S: states
Output: Q: the updated Q-table

```
start-episode(G)   //for our experiment G will be one run of the Descent game
s ← null; a ← null;
s_new ← initialState(G)
while not(end-of-episode(G)) do
    s' ← similarState(S, s_new)  //finds a state s' in S similar to s_new
    if (s' = null) then        // no such an s' exists currently in S
        s' ← s_new
        S ← S ∪ {s'}
        make-entry(Q,s')  // creates a new row in Q for state s' and
                          // Q(s',a) is initialized randomly for each action a
    end-if
    a' ← softmax-action-selection(Q,s')
    if (a ≠ null and s ≠ null) then //avoids doing the update in the first iteration
        Q(s,a) ← Q(s,a) + α(R + γQ(s',a') – Q(s,a))   // Same as Formula (1)
    end-if
    a ← a'
    s ← s'
    (R, s_new) ← take-action(a',G) // reward R obtained and the state s_new visited after
                                   // executing action a'
end-while
return Q
```

While we do expect that using case-based generalization will reduce the memory footprint of temporal difference, there is a potential danger: that precision will be lost; this is a common difficulty with generalization techniques. In our context this could result in updates made to wrong entries of the Q-table (e.g., when two conceptually different states are combined into the same entry in the Q-table). This could have a negative effect in the performance of the agent that is using the Q-table. For example, the updates might pull the agent in opposing choices for some crucial state, making it incapable of converging to an optimal policy or even learning a "good" policy. In the next section we present some experiments we performed evaluating both the reduction in memory requirements for temporal difference and the effect of the generalization on the performance of an agent using these case-based generalization techniques.

5 Experimental Evaluation

We performed an experiment to evaluate the effectiveness of our similarity based approach to temporal difference. Specifically we wanted to validate the following hypothesis: the size of the Q-table for the similarity-based temporal difference as implemented in SIM-TD is reduced compared to the size of the Q-table needed for standard temporal difference while still preserving comparative levels of performance. We simulate the standard temporal difference by using SIM-TD with the identity similarity (indicating that two objects are similar only if they are the same). Hence, every new state that is visited will create a new entry in the Q-table.

5.1 Performance Metric

The performance metric is the score of the game is computed formulas follows:

$$\text{Score} = \omega_k * \text{kills} + \omega_h * \text{health-gain} - \omega_d * \text{deaths} - \omega_h * \text{health-lost} - \omega_L * \text{length}$$

Kills refers to the number of monsters killed by the heroes, health-gain is the health that the heroes gain (which can only be gained when the hero performs a run action, in which case they gain roughly 30% of their missing health back), deaths is the number of heroes' deaths (every time a hero dies, he respawns at the starting location), health lost by the heroes during the game and length, which indicates the length of the game (i.e., measured as the number of turns; each turn includes each of the 4 heroes' movements plus the overlord). We ran the experiments on two maps, a small one and a large one. The ranges of these attributes, for each map, are shown in Table 1. The attributes health-gain and health-loss are map independent. The asterisk in front of the ranges indicates that the ranges are unbounded to the right. For example, heroes can die any number of times. Health gain/loss range is 0-12* because each hero has a maximum of 12 health. However, a hero might lose/gain a lot of health. For example, a single hero might lose 60 health in one game because he would lose 12 health, die, lose 12 more health, die again, and so forth. The range is shown for illustration purposes. The ranges for hero's death are per kill; certain heroes are worth more negative points than others upon death.

Table 1: Attributes contributing to scoring formula

Attribute	**Range Small map**	**Range large map**	**Points earned**
Kills	0 to 9	0 to 23	1,200 (per kill)
Health-gain	0 to 12*	0 to 12	170 (per point)
Deaths	0 to 4*	0 to 4*	-1700 to -3400
Health-lost	0 to 12*	0 to 12	-170 (per point)
Length	0 to 15*	0 to 25	-60 (per turn)

5.2 Similarity Metric

We define a similarity relation that receives as input a case's state C and the current state S and returns a Boolean value indicating if C and S are similar or not. Each hero maintains its own case base to account for the different classes of heroes. States are defined as 4-tuples: (distance to monster, monsters in range, expected damage, health). Distance to monster refers to the Manhattan distance to the nearest monster from the hero's position, monster in range indicate the total number of monsters that can be reached by the hero within one turn (different heroes might have different movement ranges), expected damage is computed based on the maximum damage that the hero can take from the monsters that can reach him within one turn; if no monster is within range the value is set to 0. For the small map, there can be at most 9 monsters in range (13 for the large map) and on average these monsters will do 31 damage (41 for large map). Health is the current health of the hero. This information is sufficient to determine which action to apply. The solution part of the case is the action that the hero must take, which, as detailed in Section 2, is battle, advance, or run.

Table 2 shows the 3 similarity relations we used in our experiments: major similarity, which allows more pairs of states to be similar, minor similarity, which is more restrictive than major similarity, and no similarity, which considers two states to be similar only if they are identical. The rows are for the same attributes indicated in the previous paragraph. They indicate the minimum requirements for two states to be considered similar. Two states are similar if the absolute difference of the attributes is smaller or equal than each of the corresponding entries in the table below. For example, (6,2,5,10) is similar to (3,1,8,5) relative to the major similarity but not relative to the minor similarity. The values in parenthesis in the Major similarity show the ranges for the large and small maps. The current health is independent of map and, hence, only one value is shown. For the minor similarity we consider special values of the attributes that supersede the attribute comparison criteria. For example, if a hero has maximum health, then the case we are comparing against must also have maximum health. We have analogous criteria in place for the other attributes. This makes the minor similarity a much more restrictive criterion than the major similarity.

Table 2: Boundaries for similarity metrics

Attribute	**Major similarity**	**Minor similarity**	**No similarity**
Distance to monster	4 (0-22; 0-29)	4*	0
Monster in range	3 (0-9; 0-13)	3*	0
Expected damage	7 (1-31; 141)	6*	0
Current health	5 (0-12)	4*	0

5.3 Experimental Setup

We ran three variants of SIM-TD: SIM-TD with (1) non similarity (our baseline), (2) minor similarity, and (3) major similarity. We refer as **agents** to any of these three variants, which as explained before, are used to control each hero in the game (i.e., each hero maintains its own Q-table and chooses the actions based on softmax selection of the table). We created two maps. The first map is the original map in the actual Descent board game. The second map is a smaller version of the original map with half the map sawed off.

The two different maps were used to test the different effects of similarity with different scenarios. The smaller map tends to have a much smaller Q-table since the set of situations the heroes can find themselves in is much smaller than with a large map. The large map on the other hand has a much larger set of possible states. For example, the large map has more monsters on it than the small map. Because of this, there is a much wider variance on the Monster in range and Expected damage fields. Also, since the large map is larger it makes sense that the ‘closest monster’ field could potentially be much larger as well. Using different sets of games can show us different possible results with experimentation. In both maps the boss is located behind a wall from which the boss cannot exit. This is to ensure that the games do not end early by chance because the boss wanders towards the heroes. Since the hardcoded section of the hero AI always attacks the nearest monster and the monster AI is always to run straight for the hero, it is impossible for the hero to kill the last boss before any other monster ensuring that a game does not end abruptly by chance.

For both the small and the large maps, trials of games were run until within each trial the games were fluctuating around a certain score. For the small maps score fluctuated around eight thousand points after 8 games. For the large maps score fluctuated around 9 thousand points after 4 games. Because of this, we ran trials of 8 games each for the small map and 4 games for the large map. The large maps also took a much larger amount of time to run than the small maps. Running each experiment took almost an entire day with a human operator starting each game. Also, while a single trial had multiple games in it to observe the effect of the score increasing over time, with multiple games, multiple trials needed to be run to obtain a reliable estimate of the average score over time. For each set of games, we ran a set of five trials. This was largely a time constraint decision. The game’s scoring system tends to fluctuate a lot since combat has a random factor influencing the outcome and other factors such as early decision by a hero to explore instead of attack.

5.4 Results

Figure 2 shows in the y-axis the average number of entries in the Q-table per trial. This table shows the expected effect in regard to the size of the Q-table. Using major similarity, the Q-table had a much smaller number of entries in the end; for a total of about 100 entries (or 25 per hero). For minor similarity, about twice as many were seen, about 225. And for no similarity, about twice as many again were seen, in the 425 range. This shows that case similarity can reduce the size of a Q-table significantly over the course of several games. The no similarity agent used almost five times as many cases as the major similarity agent. The small difference between the number of cases captured in the smaller and in the larger map for each type of similarity is due to the state abstraction explained in Section 3.3, which makes the number of states relatively independent of the size of the map.

Figures 3 and 4 show the scores at the end of each game. Overall with either the major or minor similarity it had a better performance than without similarity on both maps, aside from the game # 3 in the large map, where major similarity performed worst. But in general, the agent performed better with some form of similarity. Even during the first game, the agent managed to learn some strategies that performed better than the other two agents. The anomaly at game #3 can be explained by the multiple random factors which results in a lot of variation in the score. This randomness is observable even with five game trials. The smaller map had many more trials, so there is less variation. We can draw the same conclusions as in the large map. Again the major similarity agent was better than the other two, occasionally dipping below the minor similarity agent. The no similarity agent performed worse than the other two similarity agents. Even with the fluctuations in the graph, it never surpassed either of the similarity agents past the first game. This shows once again that the notion of similarity helped to make the reinforcement learning agents learn a better solution much faster than without similarity. However, again the Major Similarity Agent was competitive and beat out the Minor Similarity agent at the start and did roughly about as well towards the end.

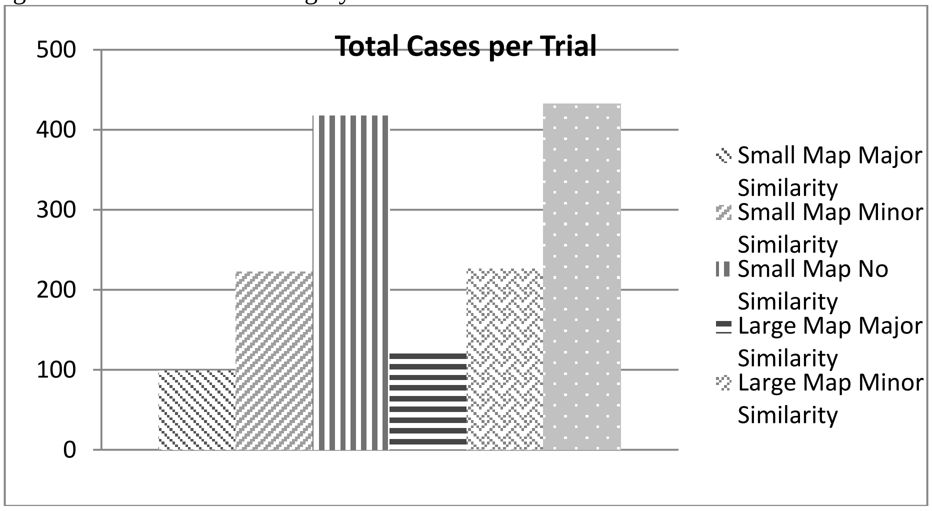

Figure 2. Total (average) number of cases for small and large maps

We believe that the reason why there is a better performance with some form of generalization than without any generalization is a reflection of the particular case based generalization used working well in this particular domain. Thus, whereas in the non-generalized situation a state s must be visited ns times before it is able to find a good approximation to the value of its actions, in the generalized situation any visit to a similar but not necessarily identical state s' will update the value of the actions. Therefore it will be able to find good values faster.

We performed statistical significance tests with the Student's t-test on the score results obtained. The difference between minor and no-similarity is significant for both maps. The difference between the major and the no-similarity is significant for the small map but not so for the large map (the t-test score was 93.9%). The difference between the major and the minor similarities was significant for the small map but not significant for the large map. The main conclusion from this study is that by using case-based generalization, the size of the Q-table can be substantially reduced while still maintaining at least as good as the performance without case-based generalization, and can even become significantly better.

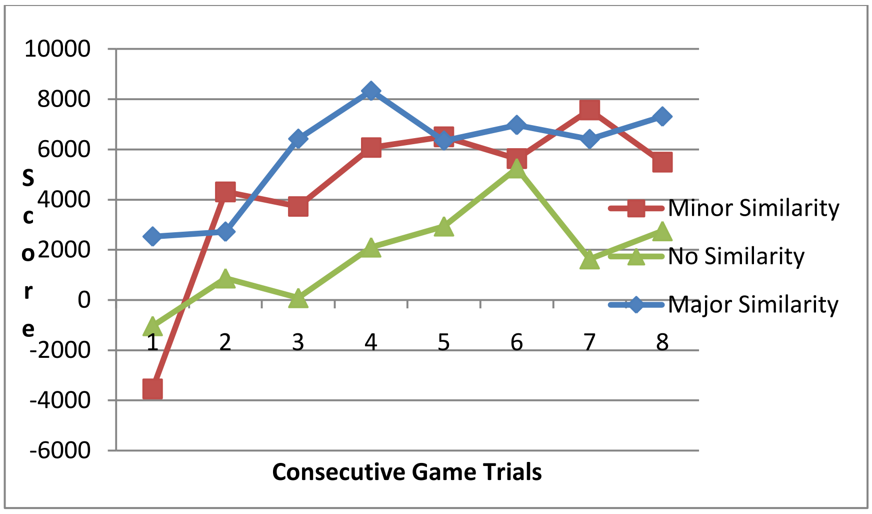

Figure 3. Average scores after each game for small map

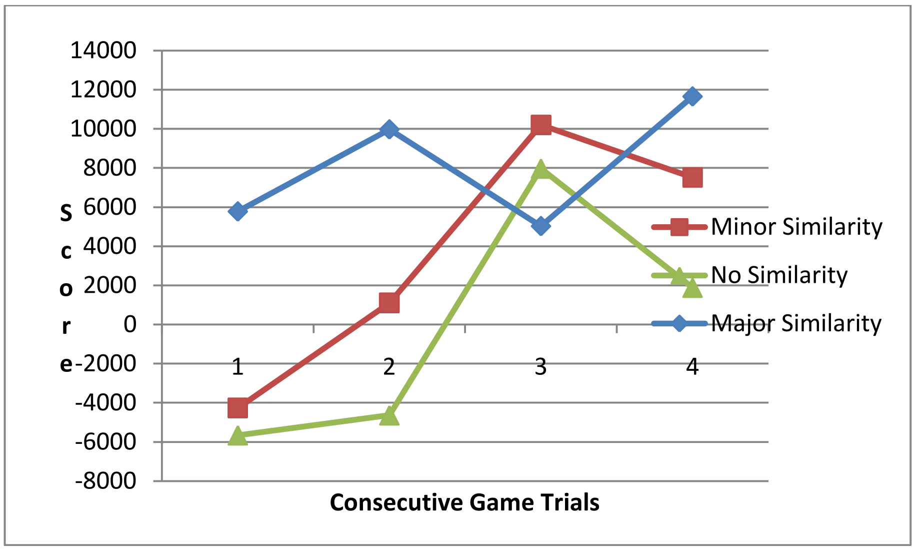

Figure 4. Average scores after each game for large map

6 Related Work

We also explored using other reinforcement learning methods such as dynamic programming and Monte Carlo methods. It is feasible that case-based generalization could have similar positive effects to those we demonstrated for Temporal Difference. However, for this particular testbed both were unfeasible to use. Dynamic programming requires that the agent knows the transition probabilities for the actions to be chosen and the expected rewards from those actions. This would require running extensive games to obtain these values. Monte Carlo methods perform the function approximation updates after the episodes ends. This will likely require it to play many more games before it learns capable policies. Games in our testbed are fairly long lasting 15 minutes on the short map and 25 on the larger one. One run of the experiment was lasting one day. Under our time constraints it was not feasible for us to run such experiments.

Researchers have investigated other approaches for reducing the memory footprint requirements of reinforcement learning. TD-Backgammon used neural networks for this purpose [10]. Clustering algorithms have been proposed to group states that are clustered together [13,19]. This requires the system to either know beforehand all states that can be visited or wait until a large sample of states have been visited. In contrast our approach is grouping states as they are visited, which is the classical lazy learning approach in CBR. However, similar to work integrating lazy and non lazy learning approaches [14], one could use our CBR approach until enough states have been visited and at that point run a clustering algorithm. Other works combine gradient-descent methods with RL [9]. Instance-based learning has been used to reduce the number of states needed and showcases with continuous states [18]. The crucial observation here is that the agent does not know the state granularity apriori. Instance-based learning methods allow the agent to refine the granularity as needed. These ideas have been studied in the context of case-based reasoning systems in [16], which also surveys instance-based and case-based reasoning approaches for continuous tasks. As per this survey, our work can be classified as a coarse-coded (since one entry in the table represent multiple states), case-based (since it maintains the Q-value for all actions in that state as a row in the Q-table) function approximation approach.

There has been a large interest in combining CBR and RL over the last years. These include Derek Bridge's ICCBR-05 invited talk where he described potential synergies between CBR and RL [5], the SINS system which performs problem solving in continuous environments by combining case-based reasoning and RL [15], and CBRetaliate, which stores and retrieves Q-tables [3]. Most of these works pursue to improve the performance of an agent by exploiting synergies between CBR and RL or enhance the CBR process by using RL (e.g., using RL to improve the similarity metrics). In contrast, in our work we are using CBR principles to address a well-known limitation of reinforcement learning. Bianchi et al. uses cases as a heuristic to speedup the RL process [7] and Gabel and Riedmiller uses cases to approximate state value functions in continuous spaces [6,17].

7 Conclusions

In this paper we presented an approach for reducing the memory footprint requirement of temporal difference learning when the agent can visit a finite number of states. We use case-based similarity to group the states visited during the reinforcement learning process. We follow a lazy learning approach: cases are grouped in the order in which they are visited. Any new state visited is assigned to an existing entry in the Q-table provided that a similar state has been visited before. Otherwise a new entry is added to the Q-table. We performed experiments on our implementation of Descent, a turn-based game where actions have non-deterministic effects and might have long term repercussions on the outcome of the game. This is the kind of game where one would expect temporal difference learning to perform well since episodes last long and, hence, the learning process could take advantage of using the estimates of the Q-values in the same episodes in which they occur. The main conclusion from this study is that by using case-based generalization, the size of the Q-table can be substantially reduced while improving the performance compared to without case-based generalization.

As discussed in the related work section, there are a number of closely related works in the literature, CBR-based and otherwise, to tackle RL's memory footprint problem. We used a simple similarity-based approach to tackle this problem and obtained significant gains in the context of a relatively complex game. It is conceivable that the use of recent advances in CBR research, such as case-based maintenance (e.g., [20]), can be used to formulate a robust CBR solution to this problem that can be demonstrated across a wider range of applications domains. It is worthwhile to point out that, as of today, there is no application of RL to a modern commercial game unlike other AI techniques such as induction of decision trees [21] and AI planning [22]. We speculate that part of the reason is the lack of robust generalization techniques for RL that allow rapid convergence towards good policies.

There is a difficulty with our approach that we will like to discuss. As we explained before, when visiting a state s_{new} the agent first checks if there is an entry in the Q-table for a similar state s_{first}. In such a situation, the action *a* to take is selected based on the Q-values for s_{first}. It is possible that the action *a* selected might not be applicable in s_{new}. This situation does not occur with the Descent agents because all actions are applicable in all states. One way to address this is to check if s_{new} and s_{first} have the same applicable actions and if not then make them dissimilar, so that each will have its own entry in the Q-table.

Acknowledgements. This work was supported in part by NSF grant 0642882.

References

1. Sharma, M., Holmes, M., Santamaria, J.C., Irani, A., Jr., C.L.I., Ram, A.: Transfer learning in real-time strategy games using hybrid CBR/RL. *Proceedings of the 20th Int. Joint Conf. on AI (IJCAI-07).* pp. 1041-1046. AAAI Press (2007)
2. Karol, A., Nebel, B., Stanton, C., Williams, M.A.: Case based game play in the robocup four-legged league part I the theoretical model. RoboCup. Volume 3020 of Lecture Notes in Computer Science. pp. 739-747. Springer (2003)
3. Auslander, Bryan, Lee-Urban, Stephen, Hogg, Chad, and Munoz-Avila, Hector. Recognizing The Enemy: Combining Reinforcement Learning with Strategy Selection using Case-Based Reasoning. Proceedings of the 9th European Conference on Case-Based Reasoning (ECCBR-08). pp. 59–73. Springer (2008)
4. Juell, P., Paulson, P.: Using reinforcement learning for similarity assessment in case-based systems. IEEE Intelligent Systems, 60 - 67 (2003)

5. Bridge, D.: The virtue of reward: Performance, reinforcement and discovery in case-based reasoning. *Proceedings of the 6th International Conference on Case-Based Reasoning (ICCBR-05)*. p. 1. Springer (2005)
6. Gabel, T., Riedmiller, M.A.: CBR for state value function approximation in reinforcement learning. *Proceedings of the International Conference on Case-Based Reasoning (ICCBR-05)*. pp 206-221. Springer (2005)
7. Bianchi, R., Ros, R., and Lopez de Mantaras, R.; Improving Reinforcement Learning by using Case-Based Heuristics. *Proceedings of the International Conference on Case-Based Reasoning (ICCBR-09)*. pp 75 – 89. Springer (2009)
8. Sutton, R.S., Barto, A.G.: *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA (1998)
9. Sutton, R.S. Learning to predict by the methods of temporal differences. *Machine Learning*, 9 - 44 (1988)
10. Tesauro, G.: Temporal difference learning and TD-Gammon. *Communications of the ACM* 38(3), 58 - 68 (1995)
11. http://en.wikipedia.org/wiki/Descent:_Journeys_in_the_Dark. Last checked: February (2010).
12. Vasta, M., Lee-Urban S. & Munoz-Avila, H. RETALIATE: Learning Winning Policies in First-Person Shooter Games. *Proceedings of the Innovative Applications of Artificial Intelligence Conference*. pp 1801-1806. AAAI Press (2007)
13. Fernández, F., and Borrajo, D. VQQL. Applying vector quantization to reinforcement learning. *RoboCup-99*. pp 292 – 303. Springer (2000)
14. Auriol, E., Wess, S., Manago, M., Althoff, K.-D. & Traphöner, R. INRECA: A Seamlessly Integrated System Based on Induction and Case-Based Reasoning. *Proceedings of the Int. Conf. on CBR*. pp. 371-380. Springer (1995).
15. Ram, A., Santamaria, J.C.. Continuous case-based reasoning. *Artificial Intelligence*, 25 - 77 (1997)
16. Santamaria, J.C, Sutton, R. S., Ram, A. Experiments with Reinforcement Learning in Problems with Continuous State and Action Spaces. *Adaptive Behavior*, 163 - 217 (1998)
17. Gabel, T. and Riedmiller, M. An Analysis of Case-Based Value Function Approximation by Approximating State Transition Graphs. In *Proceedings of the international Conference on Case-Based Reasoning (ICCBR- 2007)*. pp 344 – 358. Springer (2007)
18. McCallum, R. Andrew. *Instance-Based State Identification for Reinforcement Learning, Advances in Neural Information Processing Systems (NIPS 7)* (1995)
19. Molineaux, M., Aha, D.W., & Sukthankar, G. Beating the defense: Using plan recognition to inform learning agents. *Proceedings of the Twenty-Second International FLAIRS Conference*. pp 257–262. AAAI Press (2009)
20. Cummins, L., Bridge, D. Maintenance by a Committee of Experts: The MACE Approach to Case-Base Maintenance. In *Proceedings of the Int. Conference on Case-Based Reasoning (ICCBR- 2009)*. pp 120 – 134. Springer (2009)
21. Evans, Richard. Varieties of Learning. *AI Game Programming Wisdom*. Charles River Media, 567-578 (2002).
22. Jeff Orkin. Applying Goal-Oriented Action Planning to Games. *AI Game Programming Wisdom 2*. Charles River Media, 217-228 (2003)