An Analysis of Case-Based Value Function Approximation by Approximating State Transition Graphs

Thomas Gabel and Martin Riedmiller

Neuroinformatics Group
Department of Mathematics and Computer Science
Institute of Cognitive Science
University of Osnabrück, 49069 Osnabrück, Germany
{thomas.gabel|martin.riedmiller}@uni-osnabrueck.de

Abstract. We identify two fundamental points of utilizing CBR for an adaptive agent that tries to learn on the basis of trial and error without a model of its environment. The first link concerns the utmost efficient exploitation of experience the agent has collected by interacting within its environment, while the second relates to the acquisition and representation of a suitable behavior policy. Combining both connections, we develop a state-action value function approximation mechanism that relies on case-based, approximate transition graphs and forms the basis on which the agent improves its behavior. We evaluate our approach empirically in the context of dynamic control tasks.

1 Introduction

A key characteristic that has significantly contributed to the attractiveness of case-based reasoning (CBR) is that it allows for a *controlled* degree of inexactness during problem solving and, hence, can provide justifiable, though approximate solutions in situations where other approaches would fail. In this work, we consider learning agents that must solve some task in an unknown environment and that must adapt their behavior appropriately, solely on the basis of feedback concerning the suitability of actions taken that is obtained from the environment. Research in reinforcement learning (RL) has brought about a variety of learning algorithms for such problems. Most of them rely on learning a function that, given a situation, numerically expresses how appropriate each action is and that, logically, allows for choosing the right actions.

Aiming at the acquisition of such a value function, we will utilize the approximate nature of CBR methods in two stages. First, we will employ CBR in the obvious manner to represent the targeted value function by a finite number of instances distributed over a continuous state space. This utilization of case-based techniques represents a further development of our approach to state value function approximation using CBR [6] towards learning tasks where no model of the environment is available. Second, we utilize the CBR paradigm already prior to the application of an RL algorithm that aims at the determination of a value function. To distinguish and properly evaluate the costs and benefits of a certain action in some situation, we must compare its effects to the effects other actions might yield. Here, CBR comes into play: If no information for trading off different actions is available, then it can be approximated in a case-based manner by retrieving the effects of identical or similar actions in similar situations. Based on that principle, we develop an approach that constructs an *approximate transition graph* (ATG) which serves as an ideal input to an RL algorithm.

In Section 2, we briefly review some basic concepts of RL and introduce necessary notation, focusing in particular on model-free batch-mode RL techniques that are relevant in the scope of this paper. Section 3 presents our learning framework, including the two mentioned stages of using CBR, and Section 4 continues with a discussion of important modelling variants and extensions. The results of a first empirical evaluation of our approach are presented in Section 5.

2 Model-Free Batch-Mode Reinforcement Learning

The basic idea of reinforcement learning [15] is to have an adaptive agent that interacts with its initially unknown environment, observes outcomes of its actions, and modifies its behavior in a suitable, purposive manner. In each time step, the learner observes the environmental state $s \in S$ and decides on an action a from the set of viable actions A. By executing a, some immediate costs $c(s, a)$ may arise and, moreover, the agent is transferred to a successor state $s' \in S$. The goal of the agent, however, is not to always decide in favor of the "cheapest" actions, but to minimize its long-term expected costs.

The behavior of the agent is determined by its decision policy $\pi : S \to A$ that maps each state $s \in S$ to an action $a \in A$ to be performed in that state. Accordingly, the overall goal of a reinforcement learning algorithm is to acquire a good policy π dynamically, only on the basis of the costs the agent perceives during interacting within the environment.

2.1 Model-Free Reinforcement Learning Methods

The majority of RL methods[1] is centered around learning value functions which bear information about the prospective value of states or state-action pairs, respectively, and which can be used to induce the best action in a given state. Usually, this is done by formalizing the learning problem as a Markov decision process $M = [S, A, c, p]$ [12], where S is the set of environmental states, A the set of actions, $c : S \times A \to \mathbb{R}$ is the function of immediate costs $c(s, a)$ arising when taking action a in state s, and $p : S \times A \times S \to [0, 1]$ is a state transition probability distribution where $p(s, a, s')$ tells how likely it is to arrive at state s' when executing a in s. Throughout this paper, we use S, A, $c(\cdot)$, and $p(\cdot)$

[1] There are exceptions such as direct policy learning methods that are not considered in the scope of this paper.

to refer to the components of the Markov decision process corresponding to the considered environment. In particular, we assume $S \subset \mathbb{R}^n$ and A to be finite.

A state value function $V^\pi : S \to \mathbb{R}$ estimates the future costs that are to be expected when starting in s and taking actions determined by policy π:

$$V^\pi(s) = E\left[\sum_{t=0}^{\infty} \gamma^t c(s_t, \pi(s_t)) | s_0 = s\right] \quad (1)$$

where $\gamma \in [0, 1]$ is a discount factor that models the reduced trust in future costs. Assuming an optimal value function $V^\star$ that correctly reflects the cost and state transition properties of the environment is available, the agent may infer an optimal behavior policy by exploiting $V^\star$ greedily according to

$$\pi^\star(s) := \arg\min_{a \in A} \left(c(s, a) + \gamma \sum_{s' \in S} p(s, a, s') V^\star(s') \right) \quad (2)$$

Unfortunately, Eqn. 2 can only be applied if we are in possession of a state transition model p of the environment. For most learning tasks with relevance to real-world problems, however, there is no such model available, which is why, in this paper, we are focusing on *model-free* scenarios. Under these circumstances, we consider *state-action* value functions and Eqn. 1 can be written as

$$Q^\pi(s, a) = E\left[\sum_{t=0}^{\infty} \gamma^t c(s_t, a_t) | s_0 = s, a_t = a \text{ if } t = 0 \text{ and } a_t = \pi(s_t) \text{ else}\right]. \quad (3)$$

The crucial question now is, how to obtain an optimal state-action value function $Q^\star$. Q learning [16] is one of the most prominent algorithms used to acquire the optimal state-action value function for model-free learning problems. Q learning directly updates estimates for the values of state-action pairs according to

$$Q(s, a) := (1 - \alpha) Q(s, a) + \alpha(c(s, a) + \gamma \min_{b \in A} Q(s', b)) \quad (4)$$

where the successor state s' and the immediate costs $c(s, a)$ are generated by simulation or by interaction with a real process. For the case of finite state and action spaces where the Q function can be represented using a look-up table, there are convergence guarantees that say Q learning converges to the optimal value function $Q^\star$ under mild assumptions. Then again, it is easy to infer the best action for each state and hence, the optimal policy $\pi^\star$ by greedy exploitation of $Q^\star$ according to

$$\pi^\star(s) := \arg\min_{a \in A} Q^\star(s, a). \quad (5)$$

2.2 Offline Q Learning with Value Function Approximation

Recent research in RL has seen a variety of methods that extend the basic ideas of Q learning. Aiming at the applicability to situations where large and/or infinite state spaces must be handled, two necessities should be considered. The approach to learning a (near-)optimal behavior policy we are pursuing in this paper adheres to both of these requirements, as we will show subsequently.

Value Function Approximation To cover infinite state spaces, the value function must be represented using a function approximation mechanism. This means, we replace the optimal state-action value function $Q^\star(s,a)$ by an appropriate approximation $\tilde{Q}(s,a)$. For this, we will revert to case-based methods, building up an approximate state transition graph utilizing the CBR paradigm.

Exploitation of Experience Standard Q learning is an online learning method where experience is used only once to update the value function. By contrast, there has recently been much interest in offline (batch-mode) variants of Q learning, and a number of algorithms were proposed that are subsumed under the term *fitted Q iteration* (e.g. NFQ [13] or FQI with decision trees [5]). Here, from a finite set of transition tuples $\mathbb{T} = \{(s_i, a_i, c_i, s'_i)|i = 1, \dots, m\}$ that are made up of states, actions, immediate costs, and successor states, an approximation of the optimal policy is computed.

Aiming at fast and efficient learning, we will, on the one hand, store all transition tuples in an experience set as well, exploit it fully to construct an approximate state transition graph for the respective environment, and, on the other hand, employ k-nearest neighbor techniques to gain an approximated state-action value function for a continuous state space from which to infer a near-optimal policy.

3 Approximate Transition Graphs

Initially, the learning agent is clueless about state transitions that may occur when taking specific actions, as well as about the cost structure of the environment. During ongoing learning, however, it gains more experience and competence which it must utilize as smartly as possible in order to develop a good behavior policy. In this section, we present our learning approach by which the agent first creates an approximate transition graph (ATG) from its experience using case-based techniques and, second, performs RL on the ATG in order to finally induce a case-based policy that features near-optimal performance.

3.1 Basic Ideas of Approximate Transition Graphs

When the agent explores its environment by repeatedly acting within that environment, it steadily collects new pieces of experience that can be described as four-tuples (s, a, c, s') which the agent stores in its experience set $\mathbb{T} \subset S \times A \times \mathbb{R} \times S$. This set can also be interpreted as a partial transition graph.

Definition 1 (Partial Transition Graph).
Let $\mathbb{T} = \{(s_i, a_i, c_i, s'_i)|i = 1, \dots, m\}$ be the transition set containing the experience the learning agent has gathered while interacting within its environment. Then, $\mathbb{T}$ determines a directed partial transition graph $\mathcal{P} = (V, E)$ *whose set of nodes V is the union of all states s and s' that are components of elements from $\mathbb{T}$. Further, each transition $t \in \mathbb{T}$ is represented by an edge $(s, a, c, s') \in E$ pointing from s to s' that is annotated with the value of action a and costs c.*

As we assume the state space to be continuous, the probability that, during its lifetime, the agent enters one particular state $s_x \in S$ more than once is zero. So, in general there is only one single transition tuple $t_x = (s_x, a_x, c_x, s'_x) \in \mathbb{T}$ that relates to state s_x, implying that $\mathbb{T}$ contains no information about what happens when taking an action $a \in A \setminus \{a_x\}$ in state s_x. This precludes the application of a Q learning style learning algorithm: In order to perform an update on the value function according to the Q update rule (Eqn. 4), it is necessary to calculate the minimum over all actions, i.e. $\arg\min_{b \in A} Q(s'_x, b)$ must be evaluated. This, of course, is impossible if no information about $Q(s'_x, b)$ for most $b \in A$ is available.

Thus, our goal is to create an extension of a partial transition graph – which we will call approximate transition graph – that contains sufficient information about actions and their effects to allow for the application of Q learning. The key to deriving such an ATG is CBR: In Figure 1, we outline the building blocks involved in the construction of a value function approximation mechanism based on transition graphs that are approximated using case-based reasoning.

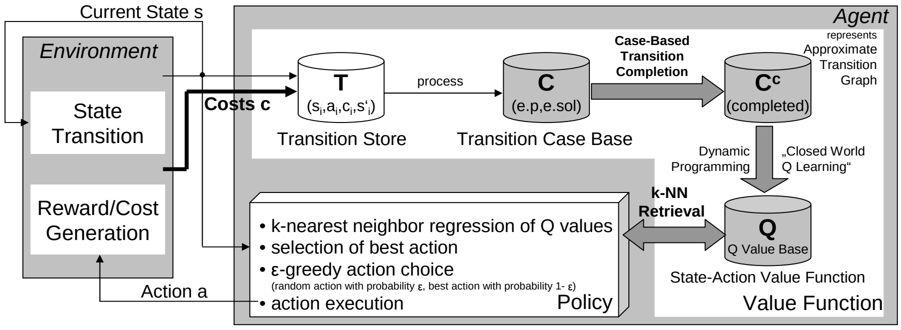

Fig. 1. Building Blocks of the Case-Based Approach to State-Action Value Function Approximation via Approximating State Transition Graphs: Descriptions for the individual components involved are given in Sections 3.2-3.4.

3.2 Case-Based Transition Completion

In a first processing step, we employ the agent's stored transitions to build up a transition case base $\mathbb{C}$.

Definition 2 (Transition Case Base).
Given an experience set $\mathbb{T} = \{(s_i, a_i, c_i, s'_i) | i = 1, \dots, m\}$*, the transition case base is defined as a collection of cases*[2] $\mathbb{C} = \{e = (e.p, e.sol)\}$ *where each case consists of a problem part* $e.p \in S$ *and a solution* $e.sol$ *that is a set of up to* $|A|$ *elements. Each element of the solution* $e.sol$ *is itself a three-tuple* (a_i, c_i, s'_i) *for some* $i \in \{1, \dots, m\}$ *with* $e.p = s_i$*. Further,*

[2] Differing from common usage, we denote a single case by e instead of c throughout this work to avoid confusion with costs.

- *for all* $(s_t, a_t, c_t, s'_t) \in \mathbb{T}$ *with* $s_t = s$ *there is only one case* $e \in \mathbb{C}$ *with* $e.p = s$
- *for all* $e \in \mathbb{C}$ *with* $e.p = s$ *there is at least one* $t \in \mathbb{T}$ *with* $s_t = s$.

So, we refer to the starting state s of a transition as the case's problem part and to triples of second to fourth component (action, costs, and successor state) as a case's solution part. Note that transitions are clustered with respect to identical starting states and hence are assigned to the same case[3]. For ease of notation, we will also allow a specific element of a case's solution to be accessed with the index operator $[\cdot]$, such that $e.sol[a_x]$ refers to $(a, c, s') \in e.sol$ with $a_x = a$.

As emphasized, it is not possible to apply a Q learning style algorithm if no estimations about the values of *all* actions that may be taken in one state are available, so that the *argmin* operator (Eqn. 4) can be evaluated. One approach to solving that problem is represented by fitted Q iteration algorithms which employ estimates of $Q(s, a)$ for all states and actions provided by some function approximator already at any intermediate point of learning. Our approach to solving that problem, however, relies on the CBR paradigm. In particular, we assume that in similar situations similar actions yield a similar effect. More concretely, we assume that for a given pair of state and action (s, a), a nearby state – starting from which that action has actually been executed – may provide a good indicator regarding what immediate costs arise and regarding which successor state is entered upon executing a in s. Naturally, we formalize the vague phrase of a nearby state by defining a similarity measure $sim_S : S \times S \to [0, 1]$ over the state space and by employing the principle of nearest neighbors.

With that assumption, we define case completion rules that are used to approximate the effects of yet unexplored actions via the nearest neighbor principle.

Definition 3 (Case-Based Transition Completion).
Let $\mathbb{C}$ *be a transition case base and* $sim_S : S \times S \to [0, 1]$ *be a similarity measure over the state space* S. *Further, denote by* $\mathbb{C}_{a_x} = \{(e.p, e.sol) \in \mathbb{C} | \exists (a_i, c_i, s'_i) \in e.sol : a_i = a_x\}$ *the subset of the case base* $\mathbb{C}$ *containing solution information about taking action* a_x. *Then, the case-based transition completion yields a new case base* $\mathbb{C}^c$ *where each* $e \in \mathbb{C}^c$ *results from applying the completion rule:*

$$\text{For all } a \in A\text{: If } \nexists (a_i, c_i, s'_i) \in e.sol \text{ with } a_i = a, \\ \text{then } e.sol := e.sol \cup NN^a(e).sol$$

where $NN^a(e)$ *is defined as the nearest neighbor of case* e *from the sub case base* $\mathbb{C}_a$, *i.e.* $NN^a(e) = \arg\max_{f \in \mathbb{C}_a} sim(e.p, f.p)$.

The key point regarding this completion method is that we have attached to each case a piece of information about the effects of taking any available action. Having started this section with the definition of a partial transition graph, we have now arrived at an approximate transition graph with completed actions that is represented by the completed transition case base $\mathbb{C}^c$ from Definition 3.

[3] As argued, the probability of entering the same state twice in a continuous environment approaches zero. However, some state may be re-entered if the system allows putting the agent into specific (starting) states.

Definition 4 (Approximate Transition Graph).
Let $\mathcal{P} = (V_{\mathcal{P}}, E_{\mathcal{P}})$ be a partial transition graph for an experience set $\mathbb{T}$ and $\mathbb{C}^c$ be the corresponding completed transition case base. An approximate transition graph $\mathcal{A} = (V_{\mathcal{A}}, E_{\mathcal{A}})$ corresponding to $\mathbb{C}^c$ is a proper extension of $\mathcal{P}$ where $V_{\mathcal{P}} \subseteq V_{\mathcal{A}}$. Further, each component $(a_j, c_j, s'_j) \in e.sol$ of the solution of each case $e \in \mathbb{C}^c$ is represented by an edge $(e.p, a_j, c_j, s'_j) \in E_{\mathcal{A}}$ that points from $e.p$ to s'_j and that is annotated with the value of action a_j and costs c_j.

Figure 2 provides a simple example within a two-dimensional state space, two available actions (a_1 and a_2), the Euclidean distance as similarity measure, and an arbitrary set of transitions. By means of case-based transition completion, we have constructed an approximate transition graph (right part of the figure) from a partial transition graph (left), determined by the contents of the experience set $\mathbb{T}$. In the PTG, for example, there is no information about the effects of taking action a_2 in state s_5. Since the nearest neighbor of s_5 is s_3 and taking a_2 in s_3 yields a transition to s_4 with costs of zero, the approximate transition graph assumes that taking a_2 in s_5 leads the system to s_4 under zero costs, too.

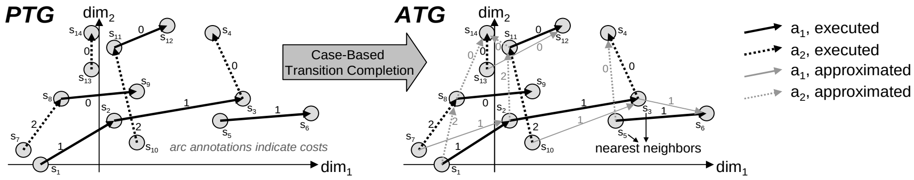

Fig. 2. From a Partial to a Completed, Approximate Transition Graph

3.3 Learning from a Completed Case Base

The output of the case-based transition completion is a case base $\mathbb{C}^c$ that is enriched by virtual state transitions and that represents an approximate state transition graph. Since $\mathbb{C}^c$ contains, in each of its cases $e \in \mathbb{C}^c$, solutions (i.e. tuples consisting of action, immediate costs, and successor state) referring to each available action $a \in A$, we are now in the position to calculate state-action values. The actual learning procedure we employ is termed closed world Q learning (CWQL), which is in tribute to the fact that this routine abstracts from the real system dynamics and considers the finite information in the completed case base only. CWQL operates like standard Q learning (Eqn. 4) on the finite set of points in state action space provided by $\mathbb{C}^c$ and is thus able to compute a value function that could be stored in a look-up table.

In our approach, the calculated state-action values are immediately attached to the case solution parts stored in $\mathbb{C}^c$, thus giving rise to a new, extended version of that case base which we subsequently call Q value base $\mathbb{Q}$ and whose individual state-action values for some state s and some action a can be accessed by:

$\mathbb{Q}(s,a) = e.sol[a].Q$ where $e \in \mathbb{Q}$ with $e.p = s \in S$ and $e.sol[a] = (a, c, s', Q) \in e.sol$. So, starting with $e.sol[a].Q = 0$ for all $e \in \mathbb{Q}$ and all $a \in A$, all state-action pairs are repeatedly updated according to the Q update rule, giving rise to

```
repeat
    for all $e \in \mathbb{Q}$ and all $a \in A$
        $f \leftarrow g \in \mathbb{Q}$ with $f.p = e.sol[a].s'$
        $e.sol[a].Q \overset{\alpha}{\leftarrow} e.sol[a].c + \gamma \min_{b \in A} f.sol[b].Q$        (6)
until $convergence$
```

We point to an important precondition that the realization of CWQL requires to work: For each state $e.p \in S$ and for each element (a_i, c_i, s'_i) of the corresponding solution $e.sol$, there must either exist at least one case $f \in \mathbb{C}^c$ with $f.p = s'_i$ or s'_i must be a goal state of the system. This condition is fulfilled for all $e.sol[a_i] = (a_i, c_i, s'_i)$ that were added to e in the scope of applying case completion rules (case-based transition completion, Section 3.2). For elements of case solutions that stem from real interaction with the environment (so, they are already included in $\mathbb{T}$ and $\mathbb{C}$, respectively), the requirement is fulfilled when the training data has been gathered along trajectories, which we assume for the remainder of this paper. Otherwise, prior to performing CWQL some sub case base of $\mathbb{C}^c$ fulfilling the requirement must be extracted.

3.4 Deriving a Decision-Making Policy

Being provided with the current state s of the system, the decision-making policy's task is to select one of the viable actions for execution. We determine the best (greedy) action with k-nearest neighbor regression, given the Q value base $\mathbb{Q}$, where the value of taking action a in state s is determined by

$$\tilde{Q}_k(s,a) = \frac{\sum_{e \in NN_k(s)} sim_S(e.p, s) \cdot \mathbb{Q}(s,a)}{\sum_{e \in NN_k(s)} sim_S(e.p, s)} \tag{7}$$

where NN_k is the set of k nearest neighbors of s in case base $\mathbb{Q}$.

Given Eqn. 7 in combination with a suitable similarity measure sim_S defined over the state space S, the best action in state s can be derived according to $\pi(s) = \arg\min_{b \in A} \tilde{Q}_k(s,b)$. During learning, the agent pursues an ε-greedy policy to encourage exploration of the environment. With probability ε, a random action is chosen, whereas with probability $1-\varepsilon$, the greedy action $\pi(s)$ is selected. During the evaluation of the learning results a purely greedy policy is applied.

The main computational burden of our algorithm lies in the case-based transition completion. Since for all cases and for all untried actions the nearest neighbors from $\mathbb{C}$ must be determined, the complexity is $O(|A||\mathbb{C}|^2)$ when linear retrieval is performed. Note that our actual implementation of the case-based ATG learning scheme realizes a policy iteration style learning algorithm, i.e. after having experienced one episode (terminated by reaching a goal state or by a time-out), the agent adds all transitions to $\mathbb{T}$ and initiates the processes of transition case base creation, case-based transition completion, and CWQL to obtain a new approximation $\tilde{Q}$ of the optimal value value function $Q^\star$. Then, the next episode is sampled by ε-greedily exploiting this new, improved Q function.

4 Modelling Variants

In Section 3, we have presented a method to approximate value functions for RL problems with CBR techniques based on the core idea of creating approximate, yet completed state transition graphs. The basic approach leaves much space for extensions and improvements, two of which shall be discussed in more detail below. Moreover, we clarify the connections to relevant related work.

4.1 Efficient Exploration

Choosing actions ε-greedily during learning means that the agent picks an arbitrary action a_{expl} with probability ε. Instead of doing that, the selection of a_{expl} may also be guided by the experience the agent has made so far. Here, we suggest the use of an efficient exploration strategy that is conducive for the construction of an approximate transition graph. This implies that we must foster a good performance of the case-based transition completion method from which an ATG (represented by case base $\mathbb{C}^c$) results.

We grant the policy access to the transition case base $\mathbb{C}$ to retrieve the nearest neighbor $NN(s)$ of the current state s. Then, the explorative action is determined as follows: Let $E = \{a \in A | \exists (a, c, s') \in NN(s).sol\}$ be the set of actions already explored in the nearest neighbor state of s. If $E = A$, then a purely random action is chosen, otherwise, however, the agent picks a_{expl} randomly from $A \setminus E$. This way, generally those actions are favored for which very little information about their effects is so far available. As a consequence, obtaining experience about taking differing actions in the neighborhood of s is fostered.

4.2 Transformational Analogy

The method of case-based transition completion (see Section 3.2) as the central step in creating an ATG performs null adaptation. As illustrated in Figure 3, the transition case base is searched for a similar case e where some action has in fact been executed, and the solution from that case is taken to solve, i.e. to complete the case considered, without any modifications (Definition 3).

Transformational analogy (TA) means that the solution of the similar case is transformed into a new solution for the current problem [4]. This basic idea can easily be integrated into case-based transition completion: Instead of adopting a solution $(a, c, s') \in NN(s).sol$ of a nearest neighbor case $NN(s)$, that solution – in particular, the solution's successor state s' – is adapted with respect to the shift[4] between s and $NN(s)$. Thus, we define:

[4] Recall that $S \subset \mathbb{R}^n$, which is why the vectorial shift $s_1 - s_2 \in \mathbb{R}^n$ between two states can easily be calculated.

Definition 5 (Case-Based Transition Completion Using Transformational Analogy). *Let all preconditions be as in Definition 3. Then, case-based transition completion using transformational analogy yields a new case base $\mathbb{C}^c$ where each $e \in \mathbb{C}^c$ results from applying the completion rule:*

$$\begin{aligned} &\textit{For all } a \in A\textit{: If } \nexists (a_i, c_i, s'_i) \in e.sol \textit{ with } a_i = a, \\ &\qquad \textit{then } e.sol := e.sol \cup \{\ T(e.p,\ NN^a(e).p,\ (a_j, c_j, s'_j)) \mid \\ &\qquad\qquad (a_j, c_j, s'_j) \in NN^a(e).sol\} \end{aligned}$$

where $NN^a(e)$ denotes the nearest neighbor of case e from sub-case base C_a.

The transformation operator $T : S \times S \times (A \times \mathbb{R} \times S) \to (A \times \mathbb{R} \times S)$ is defined as $T(s, s_{nn}, (a_{nn}, c_{nn}, s'_{nn})) = (a_{nn}, c_{nn}, s'_t)$ with $s'_t = s'_{nn} + s_{nn} - s$.

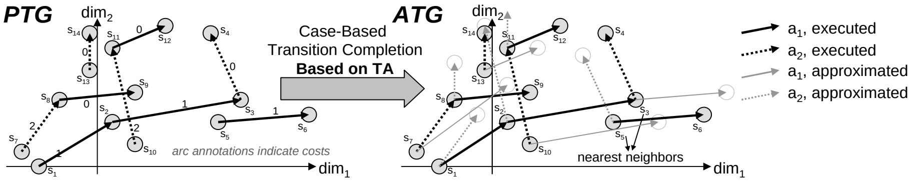

Fig. 3. Transformational Analogy During Case-Based Transition Completion

Using case-based transition completion with TA bears a disadvantage that can also be observed in Figure 3. The virtual successor states s' added to the cases' solution parts during completion do, in general, not correspond to goal states or to states for which there is an $e \in \mathbb{C}$ with $e.p = s'$. Accordingly, the precondition required for the execution of CWQL (see Section 3.3) is violated. To solve that problem, we now must apply a real fitted Q iteration algorithm where the "fitted" part, i.e. providing state-action values for states s' not covered by $\{e.p|e \in \mathbb{Q}\}$, is done by case-based estimation. We can achieve this, if we replace Equation 6 by

$$e.sol[a].Q \overset{\alpha}{\leftarrow} e.sol[a].c + \gamma \min_{b \in A} \tilde{Q}_k(e.p, b) \tag{8}$$

where $\tilde{Q}_k$ is defined according to Equation 7 and an intermediate version[5] of the Q value case base $\mathbb{Q}^t$ is forming the basis of the evaluation of $\tilde{Q}_k$. This change clearly increases the computational complexity of the algorithm since the evaluation of $\tilde{Q}_k$ requires the determination of the nearest neighbors of all $e.p$. We emphasize that this case-based Q iteration (CBQI) algorithm as determined by Equation 8 is also guaranteed to converge since the value function approximation it provides can be characterized as a contraction mapping [7].

[5] Intermediate refers to the fact that the learning process has not converged yet, i.e. that $\mathbb{Q}^t$ at iteration t of looping over all states and actions does not contain final state-action values.

4.3 Related Work

The idea of using instances of stored experience to represent value functions in the context of reinforcement learning is not new. For example, in [1] different versions of the k-NN algorithm are compared with the goal of teaching a robot to catch a ball. Several other control tasks and the use of different case-based methods with the focus on locally weighted regression are reviewed in [2]. In [6], we have analyzed the usability of case-based state value function approximation under the assumption that a transition model p is available and have evaluated our approach in the context of robotic soccer. By contrast, in this paper we exclusively focus on the model-free case and apply our algorithms to control and regulatory tasks. Highly related to ours is the work of Peng [10], where a memory-based dynamic programming (MBDP) approach is presented, that tries to learn a Q function [16] represented by finitely many experiences in memory, and uses k-nearest neighbor prediction to determine state-action values, as we do. Our work differs from MBDP insofar as we aim at the construction of a completed, approximate transition graph which is used as the starting point to acquire a state-action value function. Moreover, we also cover the issues of efficient exploration. A comprehensive article addressing the comparison of several instance- or memory-based methods to (value) function approximation is the one by Santamaria et al. [14].

The aspect of an agent learning from scratch has also been considered from a more CBR-centered perspective. For example, Macedo and Cardoso [9] focus in depth on the issue of efficient CBR-based exploration. While the agent uses a case base of entities encountered in the environment to generate expectations about missing information, our approach to efficient exploration is aimed at improving the accuracy of the value function approximation represented by an ATG. Powell et al. [11] introduce automatic case elicitation (ACE), an RL-related learning technique where the CBR system initially starts without domain knowledge and successively improves by interacting with the environment. While they focus on exact situation matching and develop a specialized action rating mechanism, our interest in this paper lies in domains with continuous states and on combining value function-based model-free RL with case-based methods.

5 Empirical Evaluation

For the purpose of evaluating our case-based approach to state-action value function with ATGs, we turn to two classical reinforcement learning benchmarks, the pole and the cart pole benchmark problems (see Figure 4).

The former represents a two-dimensional problem where the task is to swing up a pole, whose mass is concentrated in a mass point, from different starting situations. The learning agent controls a motor in the center that can apply left and right torques (-4N and +4N) to the pole. A swing-up episode is considered successful, if the agent has managed to bring the pole to a state $s = (\theta, \omega)$ with $|\theta| < 0.1$. The cart pole benchmark represents a more challenging, four-dimensional problem, where the state vector (θ, ω, x, v) consists of the pole's

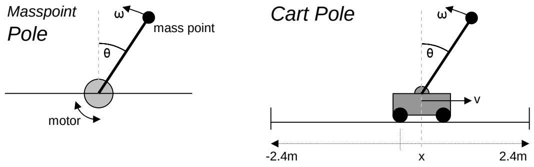

Fig. 4. Benchmark Problem Systems

angle θ and angular velocity ω as well as of the cart's position x and velocity v. Here, the task is to prevent the pole from falling down by accelerating the cart into the left/right direction ($\pm 10N$). A cart pole balancing episode is successful, if the agent manages to balance the pole for at least 1000 time steps and, to complicate the problem, the cart is positioned near the center of the track at the end ($|x| < 0.05m$). The agent fails if it hits the track boundaries ($|x| > 2.4$) or if the pole's angle exceeds the vertical position too much ($|\theta| > 1.0rad$), in which case the balancing episode is aborted. The starting states that system is initialized with during training, as well as during testing, are taken from $\mathcal{S} = \{(\theta, 0, 0, 0)| - 0.2rad \leq \theta \leq 0.2rad\}$. The dynamics for the physical simulation of this system are taken from [3].

During all experiments, we measure similarity between states as an equally weighted amalgamation $sim_S(s_1, s_2) = \sum_{i=1}^{n} w_i sim_i(s_1(i), s_2(i))$ of local similarities where each sim_i refers to the similarity with respect to one single dimension: $sim_i(x, y) = (1 - \frac{|x-y|}{max_i - min_i})^2$ with max_i and min_i as the maximal/minimal value of the respective dimension (e.g. $max_x = 2.4$ and $min_\theta = -\pi$). Further, the exploration rate during all experiments was fixed to $\varepsilon = 0.1$.

5.1 Proof of Concept

We use the pole swing-up task to provide a proof of concept for the case-based ATG function approximator. For training, as well as for testing, we employ a set $\mathcal{S}$ of start situations equally distributed over the state space ($|\mathcal{S}| = 100$), and we set $k = 1$ for k-nearest neighbor determination. Actions that do not lead to a goal state incur costs of $c = 1.0$, otherwise $c = 0.0$; no discounting is used. Figure 5 shows that the ATG-based agent is able to learn extremely quickly. The agent is able to swing up the pole for each start situation already after 5 training episodes; by then, the case base contains 485 cases. After about 60 training episodes (circa 1100 cases in memory), the performance has become very good and, during ongoing learning, continues to improve. Finally, the learning agent comes very near to the theoretical optimum (which can be calculated brute force for this problem) of 8.9 steps on average to swing up the pole for the considered set of test situations.

5.2 Results

For the cart pole benchmark it is not possible to provide some kind of optimal solution as in the case of the pole swing-up task. However, it is known that

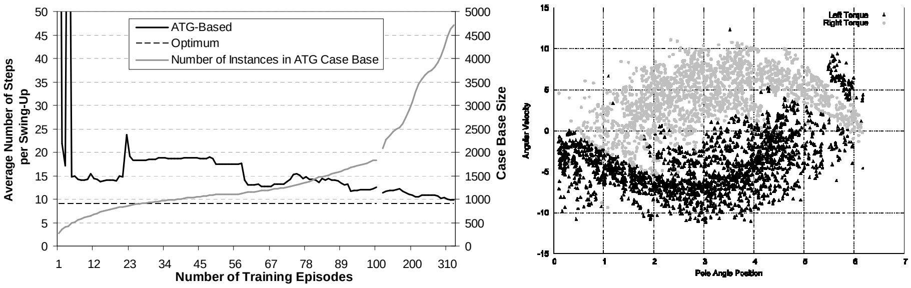

Fig. 5. Learning curve for the pole swing-up task and case base size (left). The right chart visualizes what happens during greedy exploitation of the resultant Q value base: It is shown which actions (left/right force) are considered best in which regions of the state space.

good policies which balance the pole for an arbitrary time can be learned within 200 episodes. In the following, we show that achieving that goal is also possible using an ATG-based function approximation mechanism. For a comparison to the performance of other learning techniques, we refer to the RL benchmarking website [8].

The agent is punished with very high costs if it leaves its working area ($c = 1000$) by letting the pole fall down or leaving the track. Each action incurs immediate costs of 10, except for transitions in the target area (pole nearly upright with $|\theta| < 0.3rad$ and cart within $x \in [-0.05m, 0.05m]$) are free of costs. The discounting rate is set to $\gamma = 0.98$.

In Figure 6 (left), we compare the version of the ATG-based learning agent employed in Section 5.1 to 4 variants, where we incremented the value of k and utilized the efficient exploration mechanism suggested in Section 4.1, respectively. Performance curves are shown for the evaluation of the learned policies on the set of test situations. The empirical results suggest that choosing the number of nearest neighbors to be considered during retrieval must be larger than $k = 1$ in order to obtain satisfying results. Then, the resulting function approximator features better generalization capabilities and represents the value function in a smoother way. Moreover, the efficient exploration mechanism is capable of speeding up the learning process. We note that, during testing, we aborted an episode after $t = 1000$ steps (during learning after $t = 200$), as we observed that the agent is generally able to keep the cart pole in the target area for an arbitrarily longer time if it already manages to balance for 1000 steps. Thus, in our charts an episode time of 1000 steps represents an upper limit and corresponds to a zero failure rate.

Since the combination of using $k = 3$ in conjunction with efficient exploration yields best performance (100% from about the 135th episode onward; by then, there are nearly $20k$ instances in the Q value case base), we performed further experiments on top of that base configuration (see right part of Figure 6). The use of transformational analogy during the process of case-based transi-

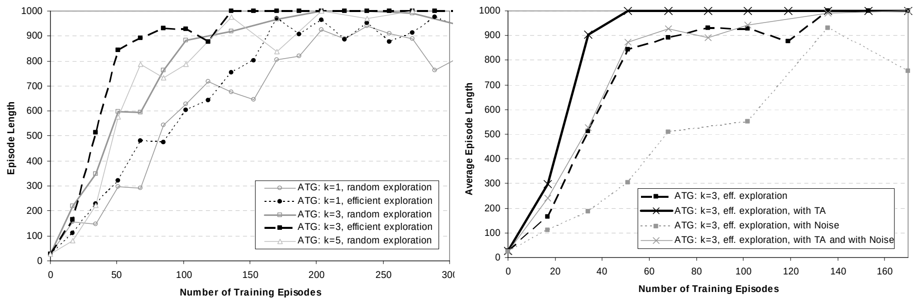

Fig. 6. Learning Curves for the Cart Pole Task.

tion completion (cf. Section 4.2) is computationally more expensive, but boosts the learning performance: Here, the agent attains a policy that yields no more failures already after about 50 training episodes.

All experiments described so far were performed in a deterministic environment. To also examine non-deterministic environments, we added uniformly distributed noise to all state transitions. So, in *each* step the states $s = (\theta, \omega, x, v)$ resulting from taking an action were distorted subject to noise according to $(\theta \pm 1^\circ, \omega \pm \frac{1^\circ}{s}, x \pm 0.5cm, v \pm 0.5\frac{cm}{s})$. In the base configuration with added noise the learner now needs more than 200 training episodes to acquire a faultless policy for the first time (not included in the chart), whereas when using TA that goal is reached already after approximately 150 episodes.

6 Conclusion

We have presented an approach to state-action value function approximation for reinforcement learning agents that systematically relies on and employs case-based methods. Our approximate transition graph function approximation scheme utilizes case-based reasoning to replenish its transition experience and also represents the actual Q function in a case-based manner. Our empirical evaluations in the context of dynamic control tasks focused on established reinforcement learning benchmark problems and revealed that ATG-based function approximation employed by a Q learning agent brings about quick success and stable learning results.

A challenging issue for future work represents case base management. Currently, we make no attempts to discard redundant transitions gathered by the agent, which is why the case base's growth is not limited and retrieval times increase continuously. For an application scenario where, for example, the number of steps till reaching some terminal state is inherently large, using our approach in its current form may become infeasible. We also expect that the performance of an ATG-based value function approximator can significantly be improved when knowledge-intensive similarity measures are employed and when background knowledge about the respective application domain (e.g. the exploitation of symmetries) is utilized. Another topic for future work is the investigation of knowledge transfer from an ATG-based function approximator to another, preferentially more compact or computationally less demanding one.

Acknowledgment

This research has been supported by the German Research Foundation (DFG) under grant number Ri-923/2-3.

References

1. D. Aha and S. Salzberg. Learning to Catch: Applying Nearest Neighbor Algorithms to Dynamic Control Tasks. In P. Cheeseman and R. Oldford, editors, *Selecting Models from Data: Artificial Intelligence and Statistics IV*. 1994.
2. C. Atkeson, A. Moore, and S. Schaal. Locally Weighted Learning for Control. *Artificial Intelligence Review*, 11(1-5):75–113, 1997.
3. A. Barto, R. Sutton, and C. Anderson. Neuronlike Adaptive Elements that Can Solve Difficult Learning Control Problems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13(5):835–846, 1983.
4. J. Carbonell. Learning by Analogy: Formulating and Generalizing Plans from Past Experience. In R. Michalski, J. Carbonell, and T. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*. 1983.
5. D. Ernst, P. Geurts, and L. Wehenkel. Tree-Based Batch Mode Reinforcement Learning. *Journal of Machine Learning Research*, 2005.
6. T. Gabel and M. Riedmiller. CBR for State Value Function Approximation in Reinforcement Learning. In *Proceedings of the 6th International Conference on Case-Based Reasoning (ICCBR 2005)*, pages 206–220, Chicago, USA, 2005. Springer.
7. G. Gordon. Stable Function Approximation in Dynamic Programming. In *ICML 1995*, pages 261–268, San Francisco, USA, 1995. Morgan Kaufmann.
8. Neuroinformatics Group. Reinforcement Learning Benchmarking Site, `www.ni.uos.de/index.php?id=930`, 2007.
9. L. Macedo and A. Cardoso. Using CBR in the Exploration of Unknown Environments with an Autonomous Agent. In *Proceedings of the 7th European Conference on CBR (ECCBR 2004)*, pages 272–286, Madrid, Spain, 2005. Springer.
10. J. Peng. Efficient Memory-Based Dynamic Programming. In *Proceedings of the Twelfth International Conference on Machine Learning (ICML 1995)*, pages 438–446, Tahoe City, USA, 1995. Morgan Kaufmann.
11. J. Powell, B. Hauff, and J. Hastings. Evaluating the Effectiveness of Exploration and Accumulated Experience in Automatic Case Elicitation. In *Proceedings of ICCBR 2005*, pages 397–407, Chicago, USA, 2005. Springer.
12. M. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley-Interscience, USA, 2005.
13. M. Riedmiller. Neural Fitted Q Iteration – First Experiences with a Data Efficient Neural Reinforcement Learning Method. In *Machine Learning: ECML 2005, 16th European Conference on Machine Learning*, Porto, Portugal, 2005. Springer.
14. J. Santamaria, R. Sutton, and A. Ram. Experiments with Reinforcement Learning in Problems with Continuous State and Action Spaces. *Adaptive Behavior*, 6(2):163–217, 1998.
15. R. S. Sutton and A. G. Barto. *Reinforcement Learning. An Introduction.* MIT Press/A Bradford Book, Cambridge, USA, 1998.
16. C. Watkins and P. Dayan. Q-Learning. *Machine Learning*, 8, 1992.