

# Using Expert Knowledge to Construct State-Action Aggregations for Reinforcement Learning

*Robby Goetschalckx      Jan Ramon*  
*Hendrik Blockeel      Maurice Bruynooghe*

*Report CW 445, May 2006*



Katholieke Universiteit Leuven  
Department of Computer Science  
Celestijnenlaan 200A – B-3001 Heverlee (Belgium)

# Using Expert Knowledge to Construct State-Action Aggregations for Reinforcement Learning

*Robby Goetschalckx      Jan Ramon*  
*Hendrik Blockeel      Maurice Bruynooghe*

*Report CW 445, May 2006*

Department of Computer Science, K.U.Leuven

## **Abstract**

This paper offers an approach to the problem of large state spaces for reinforcement learning by constructing a state-action pair aggregation (treating similar state-action pairs as if they were the same) with the use of domain knowledge. Arbitrary aggregation is known to give possibly very large errors. In this paper it is shown how, by using expert knowledge, a state-action pair aggregation can be constructed with an error bound which can be arbitrarily small. An algorithm for this approach is proposed, and experimental results on a number of different domains are given. Using this approach, only a limited number of episodes and a limited amount of memory are needed for convergence to a provably approximate solution.

**Keywords :** Reinforcement Learning

**CR Subject Classification :** I.2.6

## 1 Introduction

Reinforcement Learning is a standard way to solve Markov Decision Problems without a full model, but in large state spaces the standard algorithms (such as  $Q$ -learning [12]) need too many training episodes and require a lot of memory (for every state-action pair, a  $Q$ -value has to be stored) and may therefore be practically infeasible.

A standard way to handle this problem is to use state aggregation [7]. Using state aggregation, sets of states are treated as if they are the same. As Singh et al. showed in [10], state aggregation in general can give very large errors. However, for a lot of problem domains, domain experts can provide extra knowledge which can be used to construct an aggregation of states which gives a reasonable approximation.

In this paper, we first introduce a generalisation of state aggregation by aggregating *state-action pairs*. We show how, with certain domain knowledge, such a state-action pair aggregation can be constructed *online* for arbitrarily small error bounds (the aggregation depends on the chosen error bound). We prove this theoretically, illustrate the applicability of the approach on a number of problem domains, and demonstrate its practical feasibility experimentally. The experimental results illustrate how a limited amount of domain knowledge can give important reductions in the memory needs and number of needed episodes to reach a solution with limited error.

This paper is structured as follows: in Section 2 the kind of problems we try to solve are discussed. Also a brief summary of related work will be given here. In Section 3, we show how to use domain knowledge to construct a state-action pair aggregation with limited error and provide a proof of this bound. We introduce an algorithm which is based on this theory. Section 4 discusses the performance of this algorithm on a number of different experiment domains. We conclude in Section 5.

## 2 Background and Related Work

### 2.1 Background

We consider an agent acting in a Markov Decision Process (MDP)  $\langle S, A, R, T, \gamma \rangle$ . Here  $S$  is a set of *states*,  $A$  a set of *actions*,  $R : S \rightarrow \mathbb{R}$  the *reward* function (arriving in state  $s$  gives reward  $R(s)$ ),  $T : S \times A \times S \rightarrow [0, 1]$  the transition probabilities (performing action  $a$  in state  $s$  will lead to state  $s'$  with probability  $T(s, a, s')$ ), and  $0 \leq \gamma < 1$  the *discount factor*, which indicates the relative importance of future rewards with respect to immediate rewards. We assume the agent has *full observability* of the state of the environment. The problem is to find a *policy*, a mapping from states to actions  $\pi : S \rightarrow A$ , which gives the maximal expected discounted reward  $\mathbb{E} \sum_{i=0}^{\infty} \gamma^i R(s^{(i)})$  (from here on  $s^{(i)}$  indicates the state encountered at time step  $i$ ).

The *quality* function of a state-action pair  $(s, a)$  using a fixed policy  $\pi$  is defined as the single solution of the following set of recursive equations:

$$Q^\pi(s, a) = \sum_{s'} T(s, a, s') [R(s') + \gamma Q^\pi(s', \pi(s'))]$$

An optimal policy  $\pi^*$  of a MDP is a policy with highest  $Q$ -values. We will use the shorthand notation  $Q^*$  for  $Q^{\pi^*}$ .

Reinforcement Learning [11] is the solving of this problem without knowledge of the functions  $R$  and  $T$ . In  $Q$ -learning [12], estimates of the  $Q$ -values of each state-action pair are kept, and when a state-action pair  $(s, a)$  is visited for the  $i$ -th time, after which state  $s'$  is observed and a reward  $R(s')$  received, the  $Q$ -value of  $(s, a)$  is updated by:

$$Q(s, a) \leftarrow Q(s, a)(1 - \alpha_i) + \alpha_i \left[ R(s') + \max_{a'} Q(s', a') \right]$$

Here  $\alpha_i$  is the learning rate at time step  $i$ ,  $\sum_{i=1}^{\infty} \alpha_i = \infty$  and  $\sum_{i=1}^{\infty} \alpha_i^2 < \infty$ . This procedure is guaranteed to converge to the optimal values  $Q^*$ . However, it requires a large number of visits of each state-action pair, which in large state spaces is unacceptable. Worse, for continuous state spaces or action spaces this approach is not applicable.

This problem can be solved with the use of *state-action pair aggregation*: State-action pair aggregation is a mapping from the set of state-action pairs  $S \times A$  to a set  $X$ . This gives a partitioning of  $S \times A$ . The MDP will be transformed into a POMDP (partially observable MDP). It is a POMDP because the Markov property (future behaviour is only dependent on the current state, not on the history) is lost.

Singh et al. showed [10] that  $Q$ -learning when using *state aggregation* converges to the solution of the following set of recursive equations:

$$\hat{Q}(x, a) = \sum_{s \in S} P(s|x, \pi_l) \sum_{s' \in S} T(s, a, s') \left[ R(s') + \gamma \hat{Q}(\mu(s'), \hat{\pi}(\mu(s'))) \right]$$

$$\hat{\pi}(x) = \underset{a}{\operatorname{argmax}} \hat{Q}(x, a)$$

Here  $P(s|x, \pi_l)$  is the probability that state  $s$  is the actual ground state, if aggregate state  $x$  is observed. This probability depends on the learning policy  $\pi_l$  that was used while training. If the ground states which belong to one aggregate are very different (in reward or transition probabilities), this result can have arbitrarily large errors [9]. However, if a state aggregation would be constructed where there are bounds on the differences in quality values for all states in an aggregate, better results can be attained.

In a lot of domains experts have useful information about similarities and differences between state-action pairs. We show how this knowledge can be used to construct a state-action pair aggregation where the differences in quality values for all state-action pairs in an aggregate are bound. For a number of domains, this will be illustrated in the experiments of Section 4, where a limited amount of extra knowledge is used to obtain better results than classical approaches.

## 2.2 Related Work

State Aggregation as a solution for reinforcement learning in large state spaces has been widely studied. In [7], Li et al. present a summary of a large part of this work, however they concentrate on exact methods, while our approach is an approximate method. They divide the different approaches, amongst other criteria, according whether the algorithm needs to have full knowledge of the MDP. Our approach uses *partial* knowledge about the MDP.

Using knowledge about domain topology and dynamics is also not a new concept. In [1, 5, 6] the authors argue that assumptions about the topology of the state space can be useful (or even necessary) for reinforcement learning to work in a reasonable way. Kakade et al. [4] discuss how a near-optimal algorithm can be constructed for reinforcement learning in MDP's with a natural metric on the state space. This algorithm requires the ability to construct local models for the reward and transition functions, and finds a near optimal policy in an amount of time which is a function of the *covering* number of the state space (the number of local models needed to cover the entire space). Our approach also works in an amount of time dependent on the covering number of the state space instead of the actual size (cardinality) of the state space; however it does not require local or global models.

In [2], a method is discussed which learns a partitioning of an N-dimensional state space online, using a method closely related to self-organising maps. Our approach will construct the partitioning online, but using knowledge from domain experts instead of experiment results (which might be noisy in stochastic domains). The advantage of our approach is that it is proven to find an approximately good solution, and that it can also be used in other types of worlds than N-dimensional state spaces.

## 3 Theory and Algorithm

In this section we will discuss what information is needed to construct a reasonable aggregation. We show how a distance function between state-action pairs or between states can be constructed from this information. We prove that the difference between the  $Q$ -values of state-action pairs is closely related to this distance function. We argue how a reasonable state-action pair aggregation can be constructed with this distance function. We prove that the optimal  $Q$ -values of ground state-action pairs are similar to the estimated  $Q$ -values of the aggregate states. We introduce an algorithm which constructs such an aggregation *online*.

### 3.1 Needed Domain Knowledge

We will represent the expert domain knowledge by two mappings  $\Delta R$  and  $\beta$ , which bound how much two episodes starting at given state-action pairs may diverge. We will describe the difference between two state-action pairs using elements of some partially ordered space  $\Theta, \leq_{\Theta}$ ; the exact form of  $\Theta$  is left open

for now. The difference function  $\theta : (S \times A) \times (S \times A) \rightarrow \Theta$  maps two state-action pairs onto their “ $\Theta$ -difference”.

We now define a function  $\Delta R : \Theta \rightarrow \mathbb{R}$  that maps a  $\Theta$ -difference onto an upper bound for the difference in expected immediate reward that two state-action-pairs with that  $\Theta$ -difference may have. In other words, whenever two state-action-pairs have a  $\Theta$ -difference of  $d$ , their expected immediate reward can be at most  $\Delta R(d)$ . More precisely,

$$\sum_{s'_1, s'_2} T(s_1, a_1, s'_1) \times T(s_2, a_2, s'_2) |R(s'_1) - R(s'_2)| \leq \Delta R(\theta(s_1, a_1, s_2, a_2)) \quad (1)$$

We next define  $\beta$  as follows. Given two state-action-pairs  $\langle s_1, a_1 \rangle$  and  $\langle s_2, a_2 \rangle$  with  $\Theta$ -difference  $d$ ,  $\beta(d)$  is an upper bound for the  $\Theta$ -difference between  $\langle s'_1, a \rangle$  and  $\langle s', a \rangle$ , where  $s'_1$  and  $s'_2$  can be any state to which  $\langle s_1, a_1 \rangle$  resp.  $\langle s_2, a_2 \rangle$  may lead. Formally,

$$\forall a : T(s_1, a_1, s'_1) \times T(s_2, a_2, s'_2) > 0 \Rightarrow \theta(s'_1, a, s'_2, a) \leq_{\Theta} \beta(\theta(s_1, a_1, s_2, a_2)) \quad (2)$$

We further impose the following monotonicity constraints on  $\Delta R$  and  $\beta$ :

$$\forall \theta_1, \theta_2 \in \Theta : \theta_1 \leq_{\Theta} \theta_2 \Rightarrow [\Delta R(\theta_1) \leq \Delta R(\theta_2) \text{ and } \beta(\theta_1) \leq_{\Theta} \beta(\theta_2)] \quad (3)$$

These functions and constraints will be illustrated in the examples in Section 4.

These functions give a bound on the difference in optimal  $Q$ -values of state-action pairs:

**Lemma 1.** *If two state-action pairs  $\langle s_1, a_1 \rangle$  and  $\langle s_2, a_2 \rangle$  are both followed by a fixed sequence of actions  $\mathbf{a} = a^{(1)}, a^{(2)}, a^{(3)}, \dots$ , the difference in expected cumulated discounted reward  $\left| \sum_{i=0}^{\infty} \gamma^i R(s_1^{(i+1)}) - \sum_{i=0}^{\infty} \gamma^i R(s_2^{(i+1)}) \right|$  is at most  $\sum_{i=0}^{\infty} \gamma^i \Delta R(\beta^i(\theta(s_1, a_1, s_2, a_2)))$ , where  $\beta^i$  denotes applying the function  $\beta$   $i$  times.*

*Proof.* It is easy to prove by induction that, for every two state-action pairs  $(s_1, a_1)$  and  $(s_2, a_2)$ :  $\forall i : \beta^i(\theta(s_1, a_1, s_2, a_2)) \geq_{\Theta} \theta(s_1^{(i)}, a^{(i)}, s_2^{(i)}, a^{(i)})$ . For  $i = 0$ , this is obvious as  $\beta^0(\theta) = \theta$ . For  $i > 0$ , this follows from the fact that  $\beta$  is monotonous and an overestimation of the  $\theta$ -difference of the next state-action pairs.

From the monotonicity of  $\Delta R$  then follows

$$|R(s_1^{(i+1)}) - R(s_2^{(i+1)})| \leq \Delta R(\theta(s_1^{(i)}, a^{(i)}, s_2^{(i)}, a^{(i)})) \leq \Delta R(\beta^i(\theta(s_1, a_1, s_2, a_2))) \quad (4)$$

for all  $i > 0$  and therefore

$$\sum_{i=0}^{\infty} \gamma^i |R(s_1^{(i+1)}) - R(s_2^{(i+1)})| \leq \sum_{i=0}^{\infty} \gamma^i \Delta R(\beta^i(\theta(s_1, a_1, s_2, a_2))) \quad (5)$$

The fact that

$$\left| \sum_{i=0}^{\infty} \gamma^i R(s_1^{(i+1)}) - \sum_{i=0}^{\infty} \gamma^i R(s_2^{(i+1)}) \right| \leq \sum_{i=0}^{\infty} \gamma^i |R(s_1^{(i+1)}) - R(s_2^{(i+1)})| \quad (6)$$

concludes the proof.  $\square$

In what follows, we use the shorthand notation  $B(\theta) := \sum_{i=0}^{\infty} \gamma^i \Delta R(\beta^i(\theta))$ . From Lemma 1, we can prove the following theorem:

**Theorem 1 (Upper bound on difference in  $Q$ -values).** *The difference between the optimal  $Q$ -value of two state-action pairs  $\langle s_1, a_1 \rangle$  and  $\langle s_2, a_2 \rangle$  is given by  $B(\theta(s_1, a_1, s_2, a_2))$ :*

$$\forall s_1, a_1, s_2, a_2 : |Q^*(s_1, a_1) - Q^*(s_2, a_2)| \leq B(\theta(s_1, a_1, s_2, a_2)) \quad (7)$$

*Proof.* Without loss of generality, take  $Q^*(s_1, a_1) \geq Q^*(s_2, a_2)$ . Take an action sequence  $a^{(1)}, a^{(2)}, \dots$  that is optimal after state-action pair  $(s_1, a_1)$ . From Lemma 1 it follows that performing the same sequence after  $(s_2, a_2)$  gives an expected cumulated discounted reward  $\text{REW} \geq Q^*(s_1, a_1) - B(\theta(s_1, a_1, s_2, a_2))$ ; at the same time  $Q^*(s_2, a_2) \geq \text{REW}$  because of the optimality of  $Q^*$  and  $Q^*(s_1, a_1) \geq Q^*(s_2, a_2)$  by assumption. This proves the result.  $\square$

Any function  $B$  which overestimates the difference in  $Q$ -values could be used. The approach using the functions  $\theta$ ,  $\beta$  and  $\Delta R$  is a useful method to construct such a function for many domains. We used this approach for the experiment domains of Section 4. Any overestimating function could be used, but using tighter bounds will lead to more optimal behaviour. In the extreme case, we could take  $\theta = 0$  if the state-action pairs are equal, and  $\theta = 1$  otherwise, and  $\Delta R$  the difference between the highest and lowest possible reward. In this case the result would be that all state-action pairs are stored. On the other hand, if we have full knowledge of the entire MDP, we could calculate the exact differences in optimal  $Q$ -values and use these as  $\theta$ .

### 3.2 Constructing the Aggregation

We now will show how to construct a state-action pair aggregation with limited error.

**Definition 1 ( $\epsilon$ -bounded aggregation).** *Consider a set  $C \subset (S \times A)$  of centers, and a mapping  $\text{AGG} : S \times A \rightarrow C$ , such that  $\forall s, a : B((s, a), \text{AGG}(s, a)) \leq \epsilon$ . The state-action pairs  $(s, a)$  with  $\text{AGG}(s, a) = c$  belong to the center  $c$ .*

**Theorem 2.** *For an  $\epsilon$ -bounded aggregation, any two state-action pairs which belong to the same center, have a difference in optimal  $Q$ -values bound by  $2\epsilon$ :  $\forall s_1, s_2, \forall a_1, a_2 : \text{AGG}(s_1, a_1) = \text{AGG}(s_2, a_2) \Rightarrow |Q^*(s_1, a_1) - Q^*(s_2, a_2)| \leq 2\epsilon$*

*Proof.* From Theorem 1, we can see that the difference in optimal  $Q$ -values between a center and the state-action pairs which belong to it is bound by  $\epsilon$ , and

$$|Q^*(s_1, a_1) - Q^*(s_2, a_2)| \leq |Q^*(s_1, a_1) - Q^*(c)| + |Q^*(s_2, a_2) - Q^*(c)| \leq \epsilon + \epsilon$$

$\square$

We will create a variant of the  $Q$ -learning algorithm in an MDP with  $\epsilon$ -bounded state-action pair aggregation. The  $Q$ -learning algorithm will store a  $Q$ -value for each center. If a state  $s$  is encountered, and the action  $a$  is used, and the next-state is  $s'$ , the  $Q$ -value of  $\text{AGG}(s, a)$  will be updated by the maximum of possible next  $Q$ -values:

$$Q(\text{AGG}(s, a)) \leftarrow (1 - \alpha_i)Q(\text{AGG}(s, a)) + \alpha_i \left[ r(s') + \gamma \max_{a' \in A} Q(\text{AGG}(s', a')) \right] \quad (8)$$

Using a proof following closely the proof in [9], this can be shown to converge to a solution of

$$\begin{aligned} \hat{Q}(c) &= \sum_{s,a} P(s, a|c, \pi_l) \sum_{s' \in S} T(s, a, s') \left[ R(s') + \gamma \hat{Q}(\text{AGG}(s', \hat{\pi}(s'))) \right] \quad (9) \\ \hat{\pi}(s) &= \underset{a}{\operatorname{argmax}} \hat{Q}(\text{AGG}(s, a)) \end{aligned}$$

*Proof.* Consider a variation on  $Q$ -learning called semi-batch  $Q$ -learning, where the updates to the  $Q$ -values are stored for batches of size  $M$ , where  $M$  is large enough so the average number of occurrences that state  $s$  and  $a$  are actually encountered when the aggregate  $\text{AGG}(s, a)$  is observed, is close enough (closer than some  $\varepsilon$ ) to the real  $P(s, a|\text{AGG}(s, a), \pi_l)$  with probability  $1 - \varepsilon$ . This version of semi-batch  $Q$ -learning will behave analogous to the version in [9]:

Let  $M_k(c)$  be the number of times the aggregate state-action pair  $c$  was executed within the  $k^{\text{th}}$  batch of size  $M$ ,  $n_k(s, a|c)$  be the number of times the actual underlying state-action pair was  $\langle s, a \rangle$  when the aggregate  $c$  was executed, and  $n_k(s'|c)$  the number of times the next-state after executing  $c$  was  $s'$ . The policy used during learning is the policy  $\pi_l$ . Then the  $Q$ -value of  $c$  after the  $k^{\text{th}}$  batch is given by:

$$\begin{aligned} Q_{k+1}(c) &= (1 - M_k(c)\alpha_k(c))Q_k(c) \\ &\quad + M_k(c)\alpha_k(c) \sum_{s'} \frac{n_k(s'|c)}{M_k(c)} \left[ R(s') + \gamma \max_{a'} Q_k(\text{AGG}(s', a')) \right] \end{aligned}$$

Denote  $\hat{Q}(c)$  the solution of (9). Let

$$F_k(c) = \sum_{s'} \frac{n_k(s'|c)}{M_k(c)} \left[ R(s') + \gamma \max_{a'} Q_k(\text{AGG}(s', a')) \right] - \hat{Q}(c)$$

then

$$\begin{aligned} F_k(c) &= \sum_{s,a} P(s, a|c, \pi_l) \sum_{s'} \\ &\quad R(s') \times \left[ \frac{n_k(s'|c)}{M_k(c)} - T(s, a, s') \right] \\ &\quad + \gamma \left[ \frac{n_k(s'|c)}{M_k(c)} \max_{a'} Q_k(\text{AGG}(s', a')) - T(s, a, s') \max_{a'} \hat{Q}(\text{AGG}(s', a')) \right] \end{aligned}$$

As the sample probabilities converge to  $P(s, a | \text{AGG}(s, a), \pi_l)$  with an error of maximally  $\varepsilon$  with probability  $(1 - \varepsilon)$ , the same must go for the probabilities  $P(s' | c)$ , and the first part of this formula can be bounded by

$$\sum_{s'} R(s') \times \left[ \frac{n_k(s' | c)}{M_k(c)} - \sum_{s, a} P(s, a | c, \pi_l) T(s, a, s') \right] \leq \sum_{s'} R(s') \times \varepsilon$$

with probability  $(1 - \varepsilon)$ . For the second part of the formula, using an analogous reasoning, we can see that it is bounded by

$$\begin{aligned} & \gamma \sum_{s'} \left[ \frac{n_k(s' | c)}{M_k(c)} \max_{a'} Q_k(\text{AGG}(s', a')) \right. \\ & \quad \left. - \sum_{s, a} P(s, a | c, \pi_l) T(s, a, s') \max_{a'} \hat{Q}(\text{AGG}(s', a')) \right] \\ & \leq \gamma \times \varepsilon \times \max_{c'} |Q_k(c) - \hat{Q}(c')| \end{aligned}$$

with probability  $(1 - \varepsilon)$ . We have that  $\forall \varepsilon : \exists M$  such that the probabilities have converged to within  $\varepsilon$  with probability  $(1 - \varepsilon)$ , because of Perron's theorem [8]. The rest of the proof is equal to the proof of the convergence of  $Q$ -learning in POMDP's of [9]. The semi-batch result can be extended to the online case by using the analysis of [3].  $\square$

Now we only have to show that the estimated values  $\hat{Q}(c)$  will have an error which is bound by a function of  $\varepsilon$ .

**Definition 2** ( $Q^{(i)}$  and  $Q_{(i)}$ ).

$$Q^{(0)}(s, a) = Q^*(s, a) \tag{10}$$

$$Q_{(i)}(c) = \sum_{(s, a) : \text{AGG}(s, a) = c} P((s, a) | c, \pi_l) Q^{(i)}(s, a) \tag{11}$$

$$Q^{(i+1)}(s, a) = \sum_{s' \in S} T(s, a, s') \left( r(s') + \gamma \max_{a' \in A} Q_{(i)}(\text{AGG}(s', a')) \right) \tag{12}$$

These functions define  $Q^{(i)}$  and  $Q_{(i)}$  recursively.  $Q_{(i)}$  computes the aggregated value of the center (taking the average over the ground states) and  $Q^{(i)}$  computes the quality value of a state-action pair. Note that the computation switches to the optimal policy after  $i$  time steps (when  $Q^{(0)}$  is reached). Below, we show that this converges to values which have an error which is bound by a function of  $\varepsilon$  when  $i$  goes to infinity.

**Lemma 2.** *If the parameter  $i$  of  $Q^{(i)}$  and  $Q_{(i)}$  is increased by 1, the increase in error is bound by a factor  $\gamma$  resp. an addition of  $2\varepsilon$ .*

1.  $\forall s, a : |Q^*(s, a) - Q^{(i)}(s, a)| \leq \Delta \Rightarrow \forall s, a : |Q^*(s, a) - Q_{(i)}(\text{AGG}(s, a))| \leq \Delta + 2\varepsilon$

$$2. \forall s, a : |Q^*(s, a) - Q_{(i)}(\text{AGG}(s, a))| \leq \Delta \Rightarrow \forall s, a : |Q^*(s, a) - Q^{(i+1)}(s, a)| \leq \gamma\Delta$$

*Proof.* 1. Assume  $\forall s, a : |Q^*(s, a) - Q^{(i)}(s, a)| \leq \Delta$ . Consider a specific state-action pair  $(s_1, a_1)$ . From Theorem 2, we know that for each  $(s, a)$  for which  $\text{AGG}(s_1, a_1) = \text{AGG}(s, a)$ ,  $|Q^*(s_1, a_1) - Q^*(s, a)| \leq 2\epsilon$ . This means that for any probability distribution on the state-action pairs in this same aggregate:

$$\sum_{(s,a)} P((s, a) | \text{AGG}(s_1, a_1), \pi_l) |Q^*(s_1, a_1) - Q^*(s, a)| \leq 2\epsilon \quad (13)$$

Because of the assumption, we know that

$$\sum_{(s,a)} P((s, a) | \text{AGG}(s_1, a_1), \pi_l) |Q^*(s, a) - Q^{(i)}(s, a)| \leq \Delta \quad (14)$$

Using the triangle inequality, we get that

$$\sum_{(s,a)} P((s, a) | \text{AGG}(s_1, a_1), \pi_l) |Q^*(s_1, a_1) - Q^{(i)}(s, a)| \leq \Delta + 2\epsilon \quad (15)$$

Using the definition of  $Q_{(i)}(c)$ :

$$\forall s, a : |Q^*(s, a) - Q_{(i)}(\text{AGG}(s, a))| \leq 2\epsilon + \Delta \quad (16)$$

2. Assume  $\forall s, a : |Q^*(s, a) - Q_{(i)}(\text{AGG}(s, a))| \leq \Delta$ . Because of the definition of  $Q^{(i+1)}(s, a)$ , we know that:

$$\left| Q^*(s, a) - Q^{(i+1)}(s, a) \right| \quad (17)$$

$$\leq \sum_{s' \in S} T(s, a, s') \gamma \left| \max_{a'} Q^*(s', a') - \max_{c \in \mathcal{N}_\epsilon(s')} Q_{(i)}(c) \right| \quad (18)$$

We know that  $|Q^*(s', a') - Q_{(i)}(\text{AGG}(s', a'))| \leq \Delta$  (otherwise they would not be in the same aggregate). Combining this with (18) gives the desired result.  $\square$

**Corollary 1.** *If the aggregation is used infinitely, the error (with respect to optimal  $Q$ -values) of the predicted  $Q$ -values for each state-action pair is bound by  $\frac{2\epsilon}{1-\gamma}$ .*

*Proof.* The maximal error of  $Q_{(0)} = 2\epsilon$ , the maximal error of  $Q_{(1)} = 2\epsilon(1 + \gamma)$ , the maximal error of  $Q_{(2)} = 2\epsilon(1 + \gamma + \gamma^2)$ , and so on. In general: the bound on the error of  $Q_{(i)} = 2\epsilon \sum_{j=0}^i (\gamma^j)$ . For  $i \rightarrow \infty$ , meaning the aggregation is used infinitely, the maximal error will converge to  $\frac{2\epsilon}{1-\gamma}$ .  $\square$

**Theorem 3.** *A policy which chooses in each state an action for which the approximate  $Q$ -value of the corresponding center is highest will have an error which is bound by  $\frac{2\epsilon}{1-\gamma}$ .*

*Proof.* The approximate  $Q$ -values correspond to the expected cumulated rewards, estimated using the learning policy  $\pi_l$ . However, the error for *any* policy is bound by  $\frac{2\epsilon}{1-\gamma}$ , therefore the expected cumulated reward when using the derived policy will also have an error which is bound by this value.  $\square$

### 3.3 The SAGA Algorithm

We implemented an algorithm SAGA, which requires a distance function between states or state-action pairs (as introduced in subsection 3.2) and an error bound  $\epsilon$ , and which constructs an  $\epsilon$ -bounded aggregation. While constructing this aggregation, the algorithm also estimates the  $Q$ -values of every aggregate state-action pair. As follows from Theorem 3, the result will be an approximation of the optimal  $Q$ -values which has an error no worse than  $\frac{2\epsilon}{1-\gamma}$ .

SAGA starts with an empty set of stored state-action pairs. Whenever a state-action pair is encountered, SAGA checks if there is a stored state-action pair nearby (closer than  $\epsilon$ ). If this is not the case, the new state-action pair will be stored as a new center. If there were state-action pairs nearby, those will be updated using the  $Q$ -update-rule as in standard  $Q$ -learning.

## 4 Experiments

In this section we describe some experimental domains where we applied the algorithm, we discuss the information we used about the domain and compare the result with other classical state abstraction approaches.

A first experiment domain is illustrated in Figure 1(a). The state space is  $\mathbb{R}$ , the actions are  $\{+0.01, -0.01, +0.00\}$ . The reward received at a state  $x$  is  $e^{-|x-t|}$ , with  $t$  in the interval  $[0, 10]$  – the value of  $t$  was chosen randomly at the start of every experiment (figure 1(a) has  $t \approx 8.1$ ). The value of  $\gamma$  is 0.9. We ran our algorithm with three different error bounds (0.2, 1, and 5). In the distance function, we included knowledge of the (absolute value) of the gradient at the states, how much this gradient can change at each timestep, and the absolute difference between the two states. We did not aggregate actions, only states. In particular, we used the following functions for  $\theta$ ,  $\Delta R$  and  $\beta$ , ( $R'(x)$  denotes the gradient of the reward function at position  $x$ ):

$$\begin{aligned} \theta(s_1, a_1, s_2, a_2) &= \langle |s_1 - s_2|, \max(|R'(s_1)|, |R'(s_2)|) \rangle \quad \text{if } a_1 = a_2 \\ &= \infty \quad \text{otherwise} \\ \Delta R(\langle D, G \rangle) &= D \times G, \Delta R(\infty) = \infty \\ \beta(\langle D, G \rangle) &= \langle D, G \times e^{0.01} \rangle, \beta(\infty) = \infty \\ \langle D_1, G_1 \rangle \leq_{\theta} \langle D_2, G_2 \rangle &\Leftrightarrow D_1 \leq D_2 \quad \text{or} \quad G_1 \leq G_2 \end{aligned}$$

(The  $\infty$  is used to aggregate only states, not actions.)

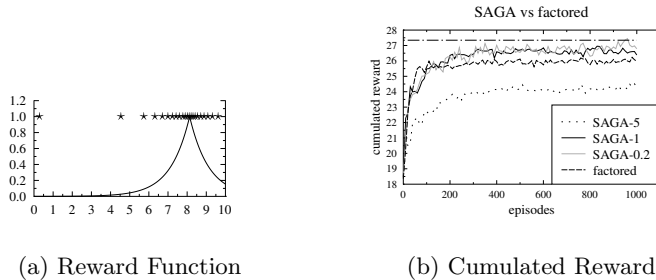
For this choice of  $\theta$ ,  $\beta$  and  $\Delta R$ ,  $\leq_{\theta}$ , we can prove that the constraints from Section 3 hold:

- (1) :  $\Delta R(\theta(s_1, a, s_2, a)) = |s_1 - s_2| \times \max(|R'(s_1)|, |R'(s_2)|) \geq |R(s_1 + a) - R(s_2 + a)|$  because of the local concavity of the reward function.
- (2) :  $\beta(\theta(s_1, a, s_2, a)) = |s_1 - s_2|, e^{0.01} \times \max(|R'(s_1)|, |R'(s_2)|)$ . Indeed, if we perform the same action  $a$  in both  $s_1$  and  $s_2$ , the distance between the next-states will not increase. The maximum relative increase of the gradient  $= |R'(t - s - 0.01)| = \left| \frac{\delta e^{t-s-0.01}}{\delta s} \right| = e^{t-s-0.01} \leq 0.01 \times e^{t-s}$ .

(3) : evident

We computed the function  $B$  from this knowledge as a closed-form formula in  $D$  and  $G$ :  $B(G, D) = \frac{G \times D}{1 - \gamma e^{0.01}}$ . We also tested a classical aggregation using no extra domain knowledge, which partitions the state space in equal-sized intervals, with a number of intervals comparable to the number of stored state-action pairs using SAGA with error bound 1. We repeated the experiment ten times and took the average of the results. The average cumulated rewards after increasing numbers of episodes are shown in Figure 1(b). Here the horizontal line at 27.4 represents the average expected cumulated reward if the optimal policy would be used. We see from the graph that both SAGA-1 and SAGA-0.2 outperform the classical approach, but SAGA-5 performs worse. We can assume that SAGA-0.2 will improve even more after a larger number of episodes. The classic algorithm converges faster to a solution which is worse than the solution found by SAGA-1, with the same number of stored state-action pairs. This is what we expected, as this approach does not use extra knowledge and will not use a higher resolution in the more interesting parts. The stored states of SAGA-1 are marked in Figure 1(a) as stars. From this graph is clear that the resolution of stored states for SAGA is concentrated at the most interesting region. More results can be seen in Figure 1(c). Here is shown the average computing time in seconds, and the average number of stored states (the number of stored state-action pairs is equal to three times this number). From this table we can see that SAGA-0.2 performs slower and needs more state-action pairs than SAGA-1. The classic algorithm performs somewhat faster than SAGA-1, but uses slightly more memory and has worse performance. For SAGA-5, we see that the number of stored states is very low, but still convergence is slow. This can be explained by the higher variance in rewards due to the aggregation of dissimilar states: it will take longer before accurate averages are found as approximations of the  $Q$ -values.

A second experiment was performed on a problem which we named *colorstacks*. It is roughly based on the *blocks-world*. Consider a table on which are a set  $\Sigma$  of stacks of colored blocks. A subset  $N$  of the colors of those blocks are considered *needed*. This is illustrated in Figure 2(a). The possible actions are removing the top block of one of the stacks. If this block's color is in  $N$ , that color is removed from  $N$ . The problem is to find the solution requiring the smallest number of moves. This problem subsumes the set-cover problem, which implies it is an NP-complete problem. The knowledge we used incorporates both an upper bound  $U$  and a lower bound  $L$  on the number of moves required, both of which can be computed in time linear in the number of blocks times the number of needed colors:  $U(s) = \sum_{c \in N} \min_{t \in \Sigma} \text{DEPTH}(c, s)$ ,  $L(s) = \max_{c \in N} \min_{t \in \Sigma} \text{DEPTH}(c, s)$ . The distance between two state-action pairs  $\langle s_1, a_1 \rangle$  and  $\langle s_2, a_2 \rangle$ , giving next-states  $s'_1$  and  $s'_2$ , is then defined as  $\max(\gamma^L(s'_1) - \gamma^U(s'_2), \gamma^L(s'_2) - \gamma^U(s'_1))$ . We compared the performance of the SAGA-algorithm, using an error bound of 0.01, with a purely table-based approach (every state-action pair is stored). To make the comparison more fair towards the table-based version, we used some simplifications (which are implicitly also used in the SAGA distance measure). Unneeded colors of blocks were just labeled blank, to reduce the number of



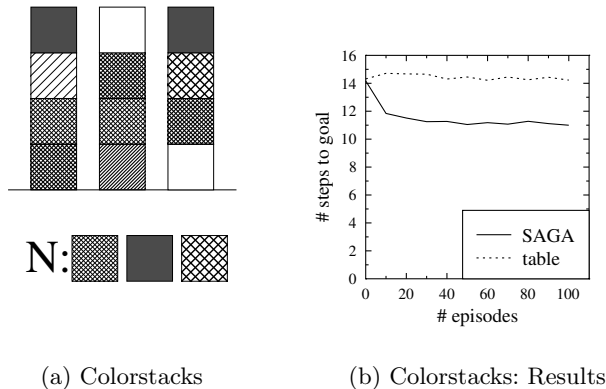
algorithm	time	stored
Classic	11260 s	21.8
SAGA 5	7948 s	5.2
SAGA 1	15347 s	20.4
SAGA 0.2	61159 s	89.0

(c) Time and Memory Needs

**Fig. 1.** One-dimensional problem

essentially equivalent states which would have to be stored. States which were equivalent up to a mapping of the needed colors were also considered equal. This would, however, still give approximately  $2.5 \times 10^{29}$  different state-action pairs if we put  $\#N = 3$ ,  $\#\Sigma = 5$ ,  $\text{STACK-HEIGHT} = 10$ . This number is exponential in  $\#\Sigma \times \text{STACK-HEIGHT}$ , which represents the number of blocks in an initial state. SAGA only kept about 30 state-action pairs in memory. The performance is shown in Figure 2(b). From this figure, it is clear that SAGA can reduce the number of steps needed to reach the goal with only a limited number of stored state-action pairs. The table-based approach is guaranteed to find the optimal solution for every situation, but it will take too much time and memory to be of real use.

A third experiment domain consists of a 2-dimensional world (Figure 3(a)). There is a reward-hill (similar to our first experiment domain), and the agent can take steps to the North, East, South and West, or stay put. The effect of the steps depends on the location: there is a varying friction in the world. At one spot, the friction is high and reduces the size of the steps. We ran SAGA using only state-aggregation, using domain knowledge consisting of the Euclidean distance between states, the difference in friction in both states and the maximal gradient of the reward function in both states. To speed up the process, we used *indexing*. This uses a equally-spaced grid on the state space, and only states within the same square of the grid will be compared.



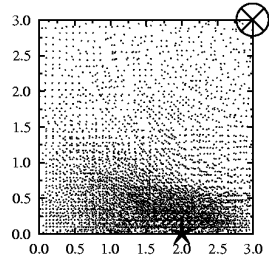
**Fig. 2.** Colorstacks

In Figure 3(a) we can see that the distribution of stored-state action pairs is higher in the neighbourhood of the top of the reward-hill (marked with a  $\star$ ), but lower in the higher-friction area (marked with  $\otimes$ ). The reason for this is that in the high-friction area, the steps are smaller. This gives a lower variance of  $Q$ -functions in this area. In Figure 3(b) the cumulated rewards (averaged over 5 runs of the experiment) is shown. We compared SAGA to an approach which divided the state space in an equally-spaced grid. From this graph it is clear that SAGA will need less episodes to converge to a reasonable solution. The number of stored states is about 4475 for the classic approach and 4425 for SAGA.

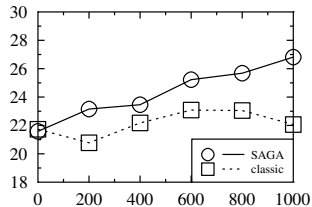
## 5 Conclusions

In this paper we introduced how expert knowledge can be used to construct a state-action pair aggregation with proven error bound. The advantage of this approach is that the number of stored state-action pairs can be greatly reduced while still approximating the optimal solution up to an arbitrarily small bound. With a smaller state-action space, the number of episodes needed to reach a reasonable solution can be expected to be much smaller. Compared to other classic approaches to reduce the number of states, the use of expert knowledge should give a denser resolution in the more difficult regions of the state space, and because of this, the error should be smaller.

With the experiments of Section 4 we verified these suppositions. The algorithm decreases the number of needed state-action pairs and the number of episodes needed for convergence, even though the variance on obtained rewards per state-action aggregate is higher with the SAGA-approach (the amount of variance depends on the value of the allowed error). Compared to classic ap-



(a) 2-dim Problem



(b) 2-dim Results

**Fig. 3.** 2-Dimensional Domain

proaches which use an equal number of stored state-action pairs, we obtain a solution with smaller error.

There are also some disadvantages to our approach. As the learned  $Q$ -values are only approximations, the inferred policies might be totally different from the optimal policies. SAGA is not guaranteed to give an exact optimal solution. Also, if the distance function  $B$  is complicated and computing it is time-consuming, the processing of each episode will take a long time, as every state-action pair has to be compared to all other state-action pairs which were already stored. However, as in the final experiment, an indexing function can be used to speed up the process, without increasing the error.

Planned future work includes using a *weighted* variant of SAGA, where a weighted average of stored state-action pairs is used for predictions instead of those within the error bound. Also, we will investigate the use of *dynamic* error bounds: start with a large bound, then gradually decrease it. Preliminary experiments with this were not entirely satisfactory. There is much tuning involved: when to decrease the bound, when to stop decreasing, how much to decrease it. Also it is not clear whether increasing the bound at some point in time might be useful or necessary.

## Acknowledgements

Jan Ramon and Hendrik Blockeel are postdoctoral fellow of the Fund for Scientific Research (FWO) of Flanders.

## References

1. R. Glaubius and W. Smart. Manifold representations for value-function approximation in reinforcement learning. Technical Report WUCSE-2005-19, Department of Computer Science and Engineering, Washington University, St. Louis, 2005.
2. D. Hougen, M. Gini, and J. Slagle. Partitioning input space for reinforcement learning for control. In *Proceedings of the IEEE International Conference on Neural Networks (ICNN-97)*, 1997.
3. T. Jaakkola, M. Jordan, and S. Singh. On the convergence of stochastic iterative dynamic programming algorithms. *Neural Computation*, 6:1185–1201, 1994.
4. S. Kakade, M. Kearns, and J. Langford. Exploration in metric state spaces. In *Proceedings of the 20th International Conference on Machine Learning*, 2003.
5. T. Lane and W. Smart. Why (PO)MDPs lose for spatial tasks and what to do about it. In *Proceedings of the ICML 2005 Workshop on Rich Representations for Reinforcement Learning*, 2005.
6. T. Lane and A. Wilson. Toward a topological theory of relational reinforcement learning for navigational tasks. In *Proceedings of the Eighteenth International Florida Artificial Intelligence Research Society Conference (FLAIRS-2005)*, pages 461–467, 2005.
7. L. Li, T. Walsh, and M. Littman. Towards a unified theory of state abstraction for MDPs. In *Proceedings of the Ninth International Symposium on Artificial Intelligence and Mathematics (AIMA06)*, Ft. Lauderdale, FL, 2006.
8. C. R. MacCluer. The many proofs and applications of Perron’s theorem. *SIAM Rev.*, 42(3):487–498, 2000.
9. S. Singh, T. Jaakkola, and M. Jordan. Learning without state estimation in partially observable environments. In *Proceedings of the Eleventh International Conference on Machine Learning*, 1994.
10. T. Singh, T. Jaakkola, and M. I. Jordan. Reinforcement learning with soft state aggregation. In *Advances in Neural Information Processing Systems 7*, Cambridge, MA, 1994. MIT Press.
11. R. Sutton and A. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, Cambridge, MA, 1998.
12. C. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8(3):279 – 292, 1992.