Efficient Planning in R-max

Marek Grześ and Jesse Hoey
David R. Cheriton School of Computer Science, University of Waterloo
200 University Avenue West, Waterloo, ON, N2L 3G1, Canada
{mgrzes, jhoey}@cs.uwaterloo.ca

ABSTRACT

PAC-MDP algorithms are particularly efficient in terms of the number of samples obtained from the environment which are needed by the learning agents in order to achieve a near optimal performance. These algorithms however execute a time consuming planning step after each new state-action pair becomes known to the agent, that is, the pair has been sampled sufficiently many times to be considered as known by the algorithm. This fact is a serious limitation on broader applications of these kind of algorithms.

This paper examines the planning problem in PAC-MDP learning. Value iteration, prioritized sweeping, and backward value iteration are investigated. Through the exploitation of the specific nature of the planning problem in the considered reinforcement learning algorithms, we show how these planning algorithms can be improved. Our extensions yield significant improvements in all evaluated algorithms, and standard value iteration in particular. The theoretical justification to all contributions is provided and all approaches are further evaluated empirically. With our extensions, we managed to solve problems of sizes which have never been approached by PAC-MDP learning in the existing literature.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning; I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search

General Terms

Algorithms, Experimentation, Theory

Keywords

Reinforcement learning, Planning, MDP, Value Iteration

1 Introduction

The key research challenge in the area of reinforcement learning (RL) is how to balance the exploration-exploitation trade-off. One of the best approaches to exploration in RL, which has good theoretical properties, is so called PAC-MDP learning (PAC means Probably Approximately Correct). State-of-the-art examples of this idea are E^3 [9] and R-max [3]. PAC-MDP learning defines the exploration strategy which guarantees that with high probability the algorithm performs near optimally for all but a polynomial number of time steps (i.e., polynomial in the relevant parameters of the underlying process). This fact means that PAC-MDP algorithms are considerably efficient in terms of the number of samples which are needed during learning in order to achieve a near optimal performance. These algorithms however execute a time consuming planning step after each new state-action pair becomes known to the agent, i.e., the pair was sampled sufficiently many times to be considered as known by the algorithm, and this is a serious limitation against broader applications of these kind of algorithms [21].

This paper examines the planning problem in PAC-MDP learning. A number of algorithms are investigated with regard to planning in PAC-MDP RL (this includes value iteration, prioritized sweeping, and backward value iteration), and the contributions of this paper can summarized as follows: First, we show how the standard R-max algorithm can reduce the worst case number of planning steps from $|S||A|$ to $|S|$. Second, exploiting the special nature of the planning problem in considered RL algorithms, the new update operator is proposed which updates only the best action of each state until convergence within the given state. This approach yields significant improvements in all evaluated algorithms, and in standard value iteration in particular. Next, an extension is proposed to the prioritized sweeping algorithm which again exploits properties of planning problems in PAC-MDP learning. Specifically, only policy predecessors of each state are added to the priority queue in contrast to adding all predecessors as in the standard prioritized sweeping algorithm. Finally, we apply backward value iteration (BVI) to planning in R-max, and we show that the original algorithm from the literature [4] can fail on broad classes of MDPs. We show the problem, and after that our correction to the BVI algorithm is proposed for the general case. Then, our extensions to the corrected version of BVI which are again specific to planning in PAC-MDP learning are proposed. The theoretical justification to all contributions is provided and all approaches are further evaluated empirically on two domains.

Regardless which particular PAC-MDP algorithm is considered, the time consuming planing step is required after a new state-action pair becomes known. This problem applies also to other model-based RL algorithms which are not PAC-MDP, such as the Bayesian Exploration Bonus algorithm [10]. Our work is to improve the planing step of these kind of algorithms, and it applies to all existing flavours of PAC-MDP learning [16, 19]. In this paper, we are focusing on R-max, a popular example of PAC-MDP learning, and our work is equally applicable to other related model-based RL algorithms (including those which heuristically modify rewards [1]).

2 Background

The underlying mathematical model of the RL methodology is the Markov Decision Process (MDP). An MDP is defined as a tuple $(\mathbb{S}, \mathbb{A}, T, R, \gamma)$, where $s \in \mathbb{S}$ is the state space, $a \in \mathbb{A}$ is the action space, $T : \mathbb{S} \times \mathbb{A} \to \mathbb{S}$ is the transition function, $R : \mathbb{S} \times \mathbb{A} \times \mathbb{S} \to \mathcal{R}$

Cite as: Efficient Planning in R-max, Marek Grześ and Jesse Hoey, *Proc. of 10th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2011)*, Tumer, Yolum, Sonenberg and Stone (eds.), May, 2–6, 2011, Taipei, Taiwan, pp. XXX-XXX.

the reward function (which is assumed here to be bounded above by the value R_{max}), and $0 \leq \gamma \leq 1$ is the discount factor which determines how the long-term reward is calculated from immediate rewards [15]. The problem of solving an MDP is to find a policy (i.e., mapping from states to actions) which maximizes the accumulated reward. A Bellman equation defines optimality conditions when the environment dynamics (i.e., transition probabilities and a reward function) are known [2]. In such a case, the problem of finding the policy becomes a planning problem which can be solved using iterative approaches like policy and value iteration [2]. These algorithms take $(\mathbb{S}, \mathbb{A}, T, R, \gamma)$ as an input and return a policy which determines which action should be taken in each state so that the long term reward is maximized. In algorithms which represent the policy via the value function, $Q(s,a)$ reflects the expected long term reward when action a is executed in state s and $V(s) = \max_a Q(s,a)$.

The policy and value iteration methods require access to an explicit, mathematical model of the environment, that is, transition probabilities, T, and the reward function, R, of the controlled process. When such a model is not available, there is a need for algorithms which can learn from experience. Algorithms which learn the policy from the simulation in the absence of the MDP model are known as reinforcement learning [18, 2].

The first major approach to RL is to estimate the missing model of the environment using, e.g., statistical techniques. The repeated simulation is used to approximate or average the model. Once such an estimation of the model is available, standard techniques for solving MDPs are again applicable. This approach is known as model-based RL [17]. This paper investigates a special type of model-based RL which is known as PAC-MDP learning.

An alternative class of approaches to RL which are not considered in this paper does not attempt to estimate the model of the environment, and because of that is called model-free RL. Algorithms of this type directly estimate the value function or a policy [13] from repeated simulation. The standard examples of this approach constitute Q-learning and SARSA algorithms [18].

PAC-MDP learning is a particular approach to exploration in RL and is based on optimism in the face of uncertainty [9, 3]. Like in standard model-based learning, in PAC-MDP model-based algorithms, the dynamics of the underlying MDP are estimated from data. If a certain state-action pair has been experienced enough times (parameter m controls this in R-max), then the estimated dynamics are close to the true values. The optimism under uncertainty plays a crucial role when dealing with state-action pairs which have not been experienced m times. For such pairs, the algorithm assumes the highest possible value of their Q-values. State-action pairs for which $n(s,a) < m$ are named unknown and known when $n(s,a) \geq m$ where $n(s,a)$ is the number of times the state-action pair was experienced. When a new state action pair becomes known, the existing approximation, $\hat{M}$, of the true model, M^*, is used to compute the corresponding optimal policy for $\hat{M}$ which when executed will encourage the algorithm to try unknown actions and learn their dynamics. Such an exploration strategy guarantees that with high probability the algorithm performs near optimally for all but a polynomial number of steps (i.e., polynomial in the relevant parameters of the underlying MDP).

The prototypical R-max algorithm uses the standard Bellman backup (see Algorithm 1) and value iteration to compute the policy, $\hat{\pi}$, for the model $\hat{M}$, where the policy $\hat{\pi}(s)$ is defined in Equation 1.

$$\hat{\pi}(s) = \arg\max_a \hat{Q}(s,a) \tag{1}$$

Summarizing, the R-max algorithm works as follows: It acts greedily according to the current $\hat{V}$. Once a new state-action pair becomes known, it performs planning with the updated model (i.e., a model with a new known state-action pair), and again acts greedily according to updated $\hat{V}$. A natural and the most efficient approach to planning in this scenario is to use the outcome of the previous planning process as the initial value function for new planning, which we refer in the paper to as *incremental planning*. This is assumed for all algorithms and experiments of this paper.

Algorithm 1 Backup(s): Bellman backup for state s

$old\_val \leftarrow \hat{V}(s)$
$\hat{V}(s) = \max_a \left\{ \hat{Q}(s,a) = \hat{R}(s,a) + \gamma \sum_{s'} \hat{T}(s,a,s') \hat{V}(s') \right\}$
return $|old\_val - \hat{V}(s)|$

The proofs and the theoretical analysis of PAC-MDP algorithms can be found in the relevant literature [8, 16]. In our analysis one specific property of such algorithms is advocated: the optimism under uncertainty which guarantees that inequality $\hat{V}(s) \geq V^*(s)$ is always satisfied during learning, where $V^*(s)$ is the optimal value function which corresponds to the true MDP model M^*.

3 Known States in R-max

The focus of this paper is how to perform the planning step in R-max efficiently. In original R-max, the planning step is executed every time a new state-action pair becomes known [3] (this is also the case in known implementations [1]). While investigating the range of planning algorithms which are discussed below, we found that the efficiency of planning in R-max can be improved by taking into account the fact that *the value of a given state does not change until all its actions become known*. This is because if all unknown state-action pairs are initialized with V_{max} (as is the case in R-max), where $V_{max} = R_{max}/(1-\gamma)$ when $\gamma < 1$ and $V_{max} = R_{max}$ if $\gamma = 1$, then $V(s) = V_{max}$ as long as at least one action remains unknown in state s. If the R-max algorithm executes the planning algorithm after the pair (s,a) becomes known, whereas there still exists at least one action which is unknown in s, then only one Q-value will change its value, i.e., the value of the pair (s,a). If, after the update, $Q(s,a) < V(s) = V_{max}$, the value of s will not change. Action a will not be executed next time in state s, and another action will be used. In this way, unknown actions are correctly explored by policy $\hat{\pi}$ from Equation 1, but we observe here that the update is useless. Our novel improvement, which comes from the above observation, is to extend the notion of known state-action pairs by a notion of a known state, where $known(s) = true$ iff $\forall a\ known(s,a) = true$. With this extension, our approach is to execute the planning step in R-max only when a new state, s, becomes known (i.e., $known(s)$ becomes true). The only issue now is that the action selection according to Equation 1 has be changed in order to deal properly with states for which $known(s) = false$. This can be addressed by selecting actions using Algorithm 2 instead of Equation 1. As explained above, this procedure will not change the exploration of the R-max algorithm when ties are broken randomly. Normally, when

Algorithm 2 GetAction(s): a modified action selection method

if $known(s)$ **then**
  return $\hat{\pi}(s)$ {see Equation 1}
else
  return any action a for which $known(s,a) = false$
end if

the planning step is executed after learning each new state-action pair, its Q-value is $Q(s,a) \leq V(s) = V_{max}$ when there exists at leat one unknown action. When ties are broken randomly (this is for the case when $Q(s,a) = V_{max}$ for updated known action a), this is equivalent to postponing planning and executing another action which is still unknown when $known(s) = false$.

This improvement is particularly useful for planning algorithms which do the systematic update of the entire Q-table as value iteration does, because when $known(s) = false$ the entire planning process changes Q-values only of those actions which have just become known and there are no changes in Q-values of any other states, whereas value iteration will iterate and perform (useless) Bellman updates for all sates. Experimental validation of our extension is in the experimental section of the paper. Since, this improvement yielded a considerable speed-up and represents a more efficient implementation of R-max, if not stated otherwise, we use this extension in all experiments presented in the paper. The main goal of this paper is to speed up the R-max algorithm with regard to planning, and our approach presented here reduces the number of executions of the planner (regardless which planner is used) from $O(|S||A|)$ to $O(|S|)$.

4 Best-actions Only Updates

From this point, we are looking at ways of improving planning algorithms. The first extension which is introduced in this section is applicable to all algorithms investigated in the paper. However in order to make the presentation easier to understand by the reader and to explain the intuition which is behind this extension, we show firstly how it applies to value iteration. Its application to other planning approaches is discussed in detail in subsequent sections.

Lets assume the standard scenario of R-max learning when value iteration is used as a planning method, together with the incremental approach indicated at the end of Section 2. This means that the initial value function at the beginning of planning is always optimistic with regard to the value which is the result of planning. Additionally, under conditions specified below, the value function after each Bellman backup is also optimistic with regard to the value function after the previous Bellman backup (in R-max, values are successively decreased to reflect the change in the model which made the model less optimistic when a new state became known). The intuition which motivates Algorithm 3 is that *the change of $V(s)$ in a given iteration can be triggered only by the change of the Q-value of the best action of s* because all $Q(s,a)$ are always optimistic with regard to the optimal value function and to the value after succeeding Bellman backups, and we argue here that in each state the action which has highest $Q(s,a)$ should be updated first. This can be explained as follows. If the value of the best action will not change after its update, which means that $V(s)$ will not change in the current iteration, then all other remaining actions can be skipped in this iteration because they have lower values and they will not influence $V(s)$ (this explains why the for loop in Algorithm 3 can backup only the best actions). If the value of the best action changes after the update on the other hand, then another action may be the best and it is reasonable to update currently the best action of the same state again (this explains why the external loop of Algorithm 3 makes sense). We recall here that in the standard Bellman backup (see Algorithm 1) all actions are updated. Our idea here is that it is profitable to focus Bellman backups only on the best action of each state instead of performing updates of all actions when optimistic initialization satisfies conditions defined below. This concept is named best-actions only update (BAO) and is captured by Algorithm 3.

The two formal arguments below prove that Algorithm 3 is valid.

Algorithm 3 BAO(s): best-actions only backup of state s

```
old_val ← V(s)
repeat
  best_actions = all a in s st. |Q(s,a) − max_i Q(s,i)| < ε
  δ = 0
  for each a in best_actions do
    old_q = Q(s,a)
    Q(s,a) = R(s,a) + γ Σ_s' T(s,a,s') max_a' Q(s',a')
    if |old_q − Q(s,a)| > δ then
      δ = |old_q − Q(s,a)|
    end if
  end for
until δ < ε
return |old_val − V(s)|
```

DEFINITION 1. *Optimistic initialization with one step monotonicity (OOSM) is the special case of optimistic initialization of the Q-table which satisfies the following property:* $Q(s,a) \geq R(s,a) + \gamma \sum_{s'} T(s,a,s')V(s')$.

The property of OOSM initialization is satisfied, e.g., in any MDP as long as all Q-values are initialized with V_{max}. It will be shown in what follows that planning in R-max satisfies the OOSM requirement as well.

In order to prove Algorithm 3, we first prove the following lemma:

LEMMA 1. *If all $Q(s,a)$ are initialized according to optimism with one step monotonicity (OOSM), then after each individual $t+1$-st Bellman backup of the Q-table, the following inequality is satisfied: $\forall_{s,a} Q_t(s,a) \geq Q_{t+1}(s,a)$, where Q_t is the value function after the previous, t-th, Bellman backup.*

PROOF. We prove this lemma by induction on the number of performed Bellman backups of Q-values. To prove the base case, we show that the lemma is satisfied after the first Bellman backup. This is satisfied directly by the definition of optimism with one step monotonicity (see Definition 1). After proving the base case, we assume that the statement holds after t Bellman backups, and we will show that it holds after $t+1$ backups using the following argument:

$$Q_t(s,a) = R(s,a) + \gamma \sum_{s'} T(s,a,s')V_{t-1}(s')$$

$$\geq R(s,a) + \gamma \sum_{s'} T(s,a,s')V_t(s') = Q_{t+1}(s,a),$$

The first Bellman equation shows that the update of $Q_t(s,a)$ in the backup t is based on values of all next states, s', after $t-1$ backups, and the third Bellman equation is analogous for the backup $t+1$. The second step is from the induction hypothesis which assumes that $V_{t-1}(s') \geq V_t(s')$. □

The following corollary results from Lemma 1:

COROLLARY 1. *Q-values converge monotonically to $Q^*(s,a)$ when all $Q(s,a)$ entries are OOSM initialized in value iteration.*

THEOREM 1. *Value iteration with best-actions only updates of Algorithm 3 converges to the same value as standard value iteration with the Bellman backup of Algorithm 1 when the value function is OOSM initialized, i.e., when the optimistic initialization satisfies Definition 1.*

PROOF. In order to prove this theorem, it is sufficient to show that non-best actions do not have to be updated. Lets assume that a is a non-best action of a particular state s, i.e., an action st. $Q(s,a) < \max_i Q(s,i)$. Because all Q-values are initially OOSM optimistic, we know from Lemma 1 that $Q(s,a)$ cannot be made higher than its current value in any of the future iterations of value iteration. It means that $Q(s,a)$ cannot be made higher than $\max_i Q(s,i)$ by updating $Q(s,a)$, and the only way to make $Q(s,a)$ the best action in s is to reduce the value of $\max_i Q(s,a)$ which may happen only by updating action i which satisfies $\max_i Q(s,i)$. This shows that if the value function is initialized with OOSM optimism, it is sufficient to update the best actions only. Additionally, if $\Delta \max_i Q(s,i) < \epsilon$, $V(s)$ cannot change in the current iteration of value iteration (within given precision ϵ) and the algorithm can move to updating other states of this iteration. □

This proof makes BAO updates applicable to general value iteration planning with OOSM optimistic initialization. As mentioned before, OOSM is naturally satisfied in any MDP as long as all values are initialized with V_{max}. This requirement is rather weak and easy to satisfy and in this way applicability of BAO is substantial.

A short explanation is required on why in R-max OOSM is satisfied. In our approach, each new planning step starts with the value function of the previous planning step (incremental planning). The new MDP model is different from the previous one just in having one more known state. Thus, all states which were known in the previous model satisfy OOBC with equality, and the state which has just become known still has its $V(s) = V_{max}$ which cannot be made higher, which satisfies OOBC as well.

Due to the nature of the BAO updates, this method is expected to yield particularly significant improvements in domains with larger numbers of actions in each state. It also has a great potential to improve planning in domains with continues actions, because only a limited number of continuous actions should be updated.

5 Prioritized Sweeping for R-max

Prioritized sweeping (PS) has been popular for improved empirical convergence rate but the theoretical convergence was only expected by [12] to be provable based on the convergence results in asynchronous dynamic programming (ADP) by observing that PS is an ADP algorithm. The first formal proof for general PS was recently presented by [11], and the PS algorithm of [12] was also proved as a special case under rather a restrictive condition that initially all states have to be assigned non-zero priority. This is a rather restrictive assumption with regard to incremental planning which is found in R-max because in R-max usually not all states require being updated even once. In what follows, we prove that PS converges when used for planning in R-max without those restrictive assumptions. This holds also for our extension to basic PS (shown in Algorithm 4), which is based on the idea that it is sufficient to add to the priority queue only policy predecessors s' of state s, defined as

$$PolicyPred(s) = \{s' | T(s', \pi(s'), s) > 0\}, \quad (2)$$

(see Line 6 in Algorithm 4) instead of all predecessors, defined as

$$Pred(s) = \{s' | \exists a T(s', a, s) > 0\}, \quad (3)$$

as it is the case in standard PS [12].

LEMMA 2. *The prioritized sweeping algorithm specified in Algorithm 4 drives Bellman errors to 0 (with a required precision ϵ) when executed for a newly learned state, s_k, in R-max, and initializing the value function using the value function of the previous planning step in which s_k was not known.*

Algorithm 4 PS-PP(s_k): prioritized sweeping with policy predecessors for incremental planning in R-max after state s_k becomes known

```
1: PQ ← s_k
2: while PQ ≠ ∅ do
3:    s ← remove the first element from PQ
4:    residual(s) ← Backup(s)
5:    if residual(s) > ε then
6:       for all s' ∈ PolicyPred(s) do
7:          priority ← T(s', a, s) × residual(s)
8:          if s' ∉ PQ then
9:             insert s' into PQ according to priority
10:         else
11:            update s' in PQ if the new priority is higher
12:         end if
13:      end for
14:   end if
15: end while
```

PROOF. Let $\mathbb{F} \subset \mathbb{S}$ be the set of states which do not have s_k in their policy graph. Since, the value of s_k can only decrease in the current planning process (because in the previous planning process it was unknown with $V(s_k) = V_{max}$, and now it becomes known and its $V(s_k) \leq V_{max}$), state s_k will not appear in the *optimal* policy graph of any state in $\mathbb{F}$, therefore current values of all states in $\mathbb{F}$ are correct, do not require updates, and their Bellman error is already 0. This argument proves that states in $\mathbb{F}$ do not have to be updated, and only states in $\mathbb{S} \setminus \mathbb{F}$ should be updated, that is, policy predecessors of s_k. This proves that backward expansion of policy predecessors in Line 6 is correct, and constitutes our extension to the standard PS algorithm [12] for planning in R-max.

Let $\mathbb{S}_{s_k}$ be $\mathbb{S} \setminus \mathbb{F}$. Since s_k is the only state in $\mathbb{S}_{s_k}$ which changes its dynamics, s_k is the only state from which the modified value function should be back-propagated. The argument of the previous paragraph showed that this back-propagation can keep updating only policy predecessors of state s_k, therefore the last condition to prove is that the predecessor s' of state s should be visited only when $residual(s) > \epsilon$. We do this by showing that if for all s which can be reached when any action a is executed in s', $residual(s) \leq \epsilon$, then $residual(s') \leq \epsilon$. This means that if all successors of s' change less than ϵ, s' does not have to be backed up given precision ϵ. This can be derived as follows:

$$\begin{aligned} residual(s') &= \max_a |R(s',a) + \gamma \sum_s T(s',a,s)[V(s) \\ &\quad + \Delta V(s)] - R(s',a) - \gamma \sum_s T(s',a,s) V(s)| \\ &= \max_a |\gamma \sum_s T(s',a,s) \Delta V(s)| \leq \max_a \gamma \sum_s T(s',a,s) |\Delta V(s)| \\ &= \max_a \gamma \sum_s T(s',a,s) \times residual(s) \\ &\leq \max_a \gamma \sum_s T(s',a,s) \epsilon = \gamma \epsilon \leq \epsilon. \end{aligned}$$

The first equation is the definition of $residual(s')$ where current $V(s')$ was computed from $V(s)$, and new $V(s')$ is for $V(s) + \Delta V(s)$ for each successor s of s'. Next steps are simple algebraic operations, and inequalities are from $|a+b| \leq |a| + |b|$, $residual(s) \leq \epsilon$, and $\gamma \leq 1$. Backward search from s_k in Algorithm 4 will not expand state s' only when all successors of s' for a given policy action a have $residual(s) \leq \epsilon$ (s' will be visited if at least for one s $residual(s) > \epsilon$). This ends the proof that $V(s')$ is

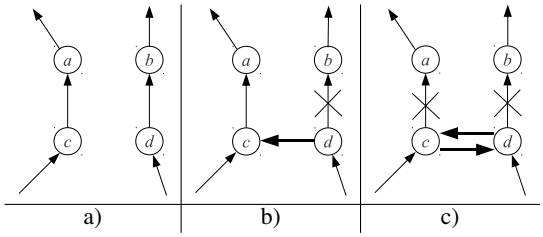

Figure 1: An example when the original backward value iteration fails on the loop

within required precision ϵ when the algorithm terminates. □

Algorithm 4 would normally use the $Backup(s)$ method of Algorithm 1 in Line 4. The proof of Theorem 1 extends to Algorithm 4 with OOSM initialization as well, and the BAO procedure presented in Algorithm 3 can be also used in Algorithm 4 by replacing, in Line 4, $Backup(s)$ with $BAO(s)$.

6 Backward Value Iteration with Loops

Backward value iteration (BVI) is an algorithm for planning in general MDPs with a set of terminal states [4]. This algorithm traverses the transpose of the policy graph using breath- or dept-first search which starts from the goal state, and checks for duplicates so that each state is updated only once in the same iteration. States are backed up in the order they are encountered during search. Before applying this algorithm for planning in R-max and propose our extensions, we show that the original version of the algorithm can fail in computing the correct value function. Let's assume the original version of the BVI algorithm from [4] and summarized above, and the use of this algorithm in planning in the domain whose four states are shown in Figure 1. First, in Figure 1a, current policy actions are shown before any updates of the current iteration of BVI. Figure 1b shows policy actions after performing backups on states b and d after which the policy action of state d changed (the new action is highlighted using a think style). Figure 1c shows updates of states a and c after which the best action of state c changed (again the thick style shows a new action). After these updates, there is a loop which involves states c and d, and the BVI algorithm will not update these states in the current iteration again because each state is updated only once, and the algorithm will also never update these two states again in any of the future iterations, because *policy* actions of all states in the loop do not lead to any state outside of the loop (so neither c nor d will be the previous state - according to a policy action - of any state outside of the loop). This situation can happen in a broad class of MDPs in which states are revisited, as in our testing domains, and applies also to stochastic actions when all actions of all states in the loop lead to states in the loop only. It is worth noting that in [4] where the BVI algorithm was introduced, all domains require many steps to revisit the state (actions are not easily reversible due to velocity in the state space). Our example shows, that the standard version of the BVI algorithm can fail by encountering the loop in a broad class of MDPs. This problem of the standard BVI algorithm was found empirically during our experimentation in this research, in which the R-max agent was getting stuck in such a loop. It is worth recalling here that the PS-PP algorithm of the previous section expands only policy predecessors, however it will not suffer from the same problem because PS-PP guarantees that s' will be visited if at least for one s $residual(s) > \epsilon$, thus states which constitute the loop will be updated as well and they will converge to proper values. The BVI algorithm with policy predecessors and updating each state once in each iteration will fail in this as indicated in Figure 1.

The brief analysis of Figure 1c indicates one simple solution to the presented problem of the standard BVI algorithm. Since states which are in the loop have other non-policy actions which lead to states outside of the loop (e.g., state d has a non-policy action which leads to state b), the straightforward solution to the loop problem is to perform backward search on *all predecessors* of a given state s as opposed to *policy predecessors* as it is the case in the original BVI algorithm. This is the first extension to BVI which is proposed in this paper, and the BVI algorithm modified in this way is named LBVI which stands for BVI with loops. The LBVI algorithm with this modification is applicable to general MDP planning. Our additional extensions to the LBVI algorithm are specific to incremental planning in R-max which is studied in this paper. The complete algorithm is presented in Algorithm 5. This is the standard version of the BVI algorithm with the following extensions: (1) all predecessors are used in the state expansion in Line 13 (to deal with the problem of Figure 1), (2) residual is checked in Line 12 (to prune the state expansion when possible), and (3) the BAO backup is applied in Line 8.

Algorithm 5 LBVI(s_k): backward value iteration for incremental planning in R-max after state s_k becomes known

```
 1: repeat
 2:    ∀_s appended(s) ← false
 3:    LargestResidual ← 0
 4:    FIFOQ ← s_k
 5:    appended(s_k) ← true
 6:    while FIFOQ ≠ ∅ do
 7:       s ← remove the first element from FIFOQ
 8:       residual(s) ← Backup(s)
 9:       if residual(s) > LargestResidual then
10:          LargestResidual ← residual(s)
11:       end if
12:       if residual(s) > ε then
13:          for all s' ∈ Pred(s) do
14:             if appended(s') == false then
15:                append s' to FIFOQ
16:                appended(s') = true
17:             end if
18:          end for
19:       end if
20:    end while
21: until LargestResidual < ε
```

LEMMA 3. *The backward value iteration algorithm specified in Algorithm 5 drives Bellman errors to 0 (with a required precision ϵ) when executed for a newly learned state, s_k, in R-max, and initializing the value function using the value function of the previous planning step in which s_k was not known.*

PROOF. Let $\mathbb{E} \subset \mathbb{S}$ be the set of states from which state s_k cannot be reached using any policy and non-policy actions. Since state s_k is not reachable from any state in $\mathbb{E}$ and s_k is the only state whose dynamics change, none of the states in $\mathbb{E}$ requires being updated, hence Bellman error of all states in $\mathbb{E}$ is already 0.

Let $\mathbb{S}_{s_k}$ be $\mathbb{S} \setminus \mathbb{E}$. Since s_k is the only state in $\mathbb{S}_{s_k}$ which changes its dynamics, s_k is the only state from which the modified value function should be back-propagated. Since the backward search process expands all predecessors of each state and starts from s_k, all states which reach state s_k (using both policy and non-policy

actions) will be updated. Therefore the last condition to prove is that the predecessor s' of state s should be visited only when $residual(s) > \epsilon$. In the prof of Lemma 2, it has been already shown that if for all s which can be reached from s', $residual(s) \leq \epsilon$, then $residual(s') \leq \epsilon$. Backward search from s_k in Algorithm 5 will not expand state s' only when all successors of s' have $residual(s) \leq \epsilon$ (s' will be visited if at least for one s $residual(s) > \epsilon$). This ends the proof that when the algorithm terminates, $V(s)$ is within required precision ϵ. □

Algorithm 5 would normally back up state s in Line 8 using the Bellman backup shown in Algorithm 1. The proof of Theorem 1 extends to Algorithm 5 as well, and the BAO procedure presented in Algorithm 3 for backing up state s can be also used in Algorithm 5 by replacing, in Line 8, $Backup(s)$ with $BAO(s)$.

7 Empirical Evaluation

This section presents empirical evaluation of proposed approaches to incremental planning in R-max. Planning time is the measure that one wishes to minimize in R-max.

7.1 Algorithms

The first experiment evaluates the extension to the R-max algorithm introduced in Section 3. Specifically, the standard R-max with value iteration and action selection according to Equation 1 is compared against modified R-max with our predicate $known(s)$ and the action selection rule specified by Algorithm 2 instead of using Equation 1.

The goal of the main empirical evaluation is to check how different extensions to standard planning algorithms improve the time of planning, and for this reason all proposed extensions are evaluated also separately to see their individual influence. Therefore, the following configurations are evaluated in the empirical study of the paper:

- VI: standard value iteration
- VI-BAO: value iteration with BAO updates
- PS: standard prioritized sweeping [12]
- PS-PP: standard prioritized sweeping with policy predecessors
- PS-BAO: standard prioritized sweeping with BAO updates
- PS-PP-BAO: prioritized sweeping with policy predecessors and BAO updates
- LBVI: backward value iteration which copes will loops (backward search to all predecessors)
- LBVI-RES: LBVI with residual check (Line 12 in Algorithm 5)
- LBVI-BAO: LBVI with BAO updates
- LBVI-RES-BAO: LBVI with residual check and BAO updates

All algorithms were implemented in C++, and the goal was to provide the same amount of optimization to each algorithm. With this in mind, the crucial element of prioritized sweeping algorithms was the priority queue. Since, the operation of increasing the priority of the element in the priority queue is required (in Line 11 in Algorithm 4), the trinomial heap was used because it supports this operation in constant time [20]. In the implementation of the queue used in LBVI, memory buffers were reused in order to have fast operations on the FIFO queue.

As mentioned before, if not stated otherwise, all algorithms use the modified treatment of unknown states as specified in Algorithm 2 in Section 3, which significantly reduces the number of times the planners are executed. In all experiments, the R-max parameter m was set to 5, and the planning precision ϵ was 10^{-4}. Experiments on the maze domain present the average value of 30 runs, and the hand washing domain of 10 runs. The standard error of the mean (SEM) is shown both in graphs and in the table.

7.2 Domains

The first domain is the version of the navigation maze task which can be found in the literature. In our implementation a scaled up version of such a maze from [1] is used and it contains 25×25 grid positions. The second domain is a simplified model of a situated prompting system that assists multiple persons with dementia to complete activities of daily living (ADL) more independently by giving appropriate prompts when needed. Such a situation arises in a shared space, e.g. a 'smart' long-term care facility, or 'smart home' with multiple residents in need of assistance. Prompting for each ADL-resident combination can be done using a (PO)MDP [6], but the situation is more complex when multiple residents are present, as prompts can interfere across ADL and between residents. The optimal solution (pursued here) is to model the complete joint space of all residents and ADL, although approximate distributed solutions are also possible [5]. Our specific implementation follows the description in [14]. In our case, each MDP has 9 states and there are 3 prompts (do nothing or issue one of the two prompts specific to the current plan step) for each state. When prompting many clients at the same time, prompts of one client can influence other clients, whereas other prompts cannot be executed for more than one client at a time, e.g., audio prompts. For example, the domain with 4 clients has $9^4 \times 3^4$ $Q(s, a)$ entries in its Q-table. Other sizes can be calculated analogously.

7.3 Results

The first test was to evaluate the improvement of our modified notion of states being known to the R-max algorithm as introduced in Section 3. As specified in the first paragraph of Section 7.1, two versions of the R-max algorithm were evaluated on the maze domain. These two versions of R-max were executed 30 times and the user time was compared. The version of the algorithm with our approach to distinguish known and unknown states (from Section 3) was 2.3 times faster than the original version. The applicability of this extension does not depend on the planning algorithm and all succeeding experiments use this modification to standard R-max.

Next experiments evaluate the major contributions of this paper. Figures 2 and 3 show the evaluation of all 10 algorithms specified in Section 7.1 on the maze domain. These algorithms determine how planning is done, and in principle the R-max algorithm should be able to explore in exactly the same way regardless which planning algorithm is used. In order to verify this, the obtained results are compared with regard to the asymptotic convergence of the R-max algorithm, and the average cumulative reward as a function of the episode number is presented in Figure 4. This figure shows that exploration was the same, and this can be seen as an empirical proof, that all planning algorithms where returning the same exploration policy at their output.

The BAO approach to updating states shows substantial improvement in all three algorithms. In particular, value iteration which is traditionally slower than, for example, prioritized sweeping significantly reduced its planning time and the number of backups. This result is particularly significant not only to planing in R-max, but also to general value-based planning in MDPs when initialization satisfies the requirement of Definition 1 which uniform optimistic initialization with V_{max} does. With our BAO approach, value iteration can be done much faster in a straightforward way.

A closer analysis of PS performance indicates that both policy predecessors and BAO updates yield improvement when applied individually, and further improvement is gained when both techniques are used together. Overall with our extensions, PS when used for incremental planning in R-max is narrowing its gap to BVI which was shown in [4] to outperform PS in the standard case due

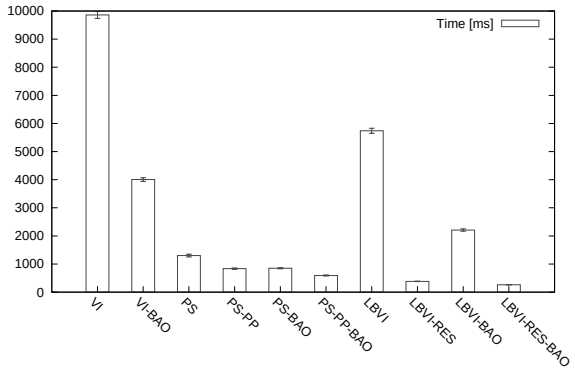

Figure 2: Planning time in the maze experiment

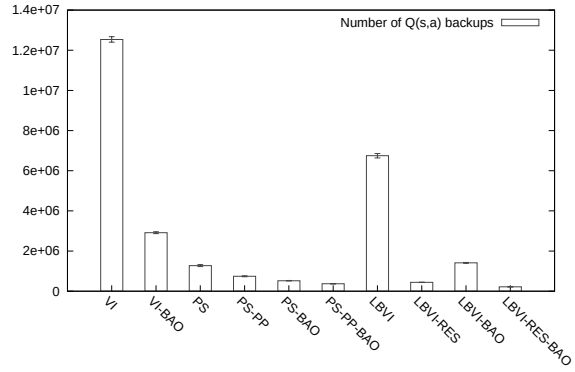

Figure 3: Number of Q(s,a) backups in the maze experiment

to the overhead of maintaining the priority queue.

The LBVI algorithm was evaluated with residual checking and with BAO updates. Here, these extensions yield improvements when applied individually, and additional gains are obtained when they are used together. The fastest planning algorithm in this experiment was LBVI with both residual check and BAO updates.

In our implementation, BVI is used with our modification which updates all predecessors instead of policy predecessors, since this was shown to be a straightforward solution to the loop problem of the standard BVI algorithm as discussed in Section 6. This leads however to an increase in the number of state expansions, but our extensions proved to be sufficient in order to guarantee fast planning of the modified BVI algorithm. We acknowledge that there is another direction to improve the performance of BVI by still using policy predecessors, however the solution has to be found on how to avoid loops which are reported in Section 6. This loop problem is detrimental for R-max agents because the agent gets stuck in such a loop during exploration.

Results on the hand washing domain are in Table 1. The rank of each algorithm is the same as in the maze domain above. The significance of our improvements, BAO in particular, becomes more evident when the state and action spaces are bigger. It is worth noting that in the last two instances (4 and 5 clients), we were able to do off-line planning in R-max with 5.3×10^5 and 1.4×10^7 state-action pairs in the Q-table! Experiments in which it was infeasible to wait for their completion are indicated with '-'.

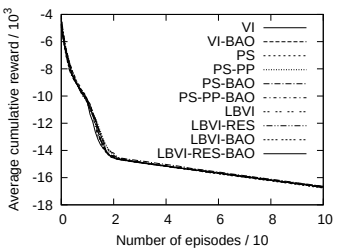

Figure 4: The cumulative reward of the learning agent

8 Related Work

The fact that planning is a bottleneck of PAC-MDP learning has been recently emphasized also in [21] where Monte Carlo on-line planning algorithms for PAC-MDP learning were proposed. These algorithms are interesting because their complexity does not depend on the number of states. This is achieved by sampling C times from each state (which limits the branching factor) and the horizon is additionally limited by the discount factor. In this way, it is sufficient to do Monte Carlo sampling only in the limited neighbourhood of a given state. The disadvantage of these algorithms is that they require the entire process to be repeated for each action selection. Our algorithms which are proposed in this paper also make use of the fact that when new state becomes known, mostly only its neighbourhood needs to be updated, which is reflected very well in our results. Our conjecture here is that the algorithms which we propose in this paper, could be proven to have complexity dependent only on the close neighbourhood of the state which triggers the planning process. The rational for this theoretical future work is indicated by our results in this paper. In [21] authors report results with Monte Carlo planning on a flag domain with 5×5 grid and 6 flags possibly appearing, where VI did not succeed. In our experiments of this paper, we are reporting results on large domains where even though VI was very inefficient or did not work at all, our extensions to VI-based planning were proven to be successful. Such off-line algorithms require planning only once for each known state and once planning is done, the policy can be used very fast, whereas Monte Carlo methods plan for every step. Our methods could further scale the off-line methods up when used with factored planners for MDPs [7]. We are additionally not aware of any PAC-MDP results with off-line planning on domains as large as those solved in this paper.

9 Conclusions

PAC-MDP algorithms are particularly efficient in terms of the number of samples which are needed by the learning agents in order to achieve a near optimal performance. These algorithms however execute a time consuming planning step after each new state-action pair (or a new state according to our extension) becomes known to the agent. This fact is a serious limitation on broader applications of these kind of algorithms. This paper examines the planning problem in PAC-MDP learning, and seeks ways of shortening the duration of the planning step. The contribution of this paper can be summarized as follows:

- The number of executions of the planner can be reduced when planning is triggered by a new state becoming known as introduced in Section 3
- The new update operator, BAO, was proposed which, instead of updating all actions of a given state once, updates only the best action of each state but continues this updating until convergence within the given state. This approach yields significant improvements in all evaluated algorithms, and standard value iteration in particular. This approach is also applicable beyond planning in R-max, since optimistic initialization with V_{max} can be easily applied in general value-based MDP planning, and this contribution has potential to bear an impact on the field

Algorithm	1 Client	2 Clients	3 Clients	4 Clients	5 Clients
VI	7.9 ± 0.48	955.8 ± 23.30	273698.8 ± 3053.90	-	-
VI-BAO	2.7 ± 0.26	86.5 ± 3.87	12721.9 ± 70.20	1671388.3 ± 6827.52	-
PS	5.1 ± 0.46	76.5 ± 3.07	7151.5 ± 98.77	788296.8 ± 2318.42	-
PS-PP	3.7 ± 0.45	45.7 ± 1.93	2394.0 ± 27.96	154282.2 ± 792.65	-
PS-BAO	1.3 ± 0.15	14.5 ± 1.16	1006.5 ± 6.11	79717.0 ± 271.98	-
PS-PP-BAO	1.4 ± 0.34	13.8 ± 1.02	602.5 ± 5.29	28601.8 ± 157.43	11956396.5 ± 194255.47
LBVI	5.3 ± 0.30	168.5 ± 9.60	24066.6 ± 202.85	-	-
LBVI-RES	4.3 ± 0.30	83.6 ± 1.29	6182.4 ± 51.74	666335.2 ± 1498.05	-
LBVI-BAO	1.4 ± 0.16	16.5 ± 1.00	1183.1 ± 5.79	90647.5 ± 407.24	-
LBVI-RES-BAO	1.6 ± 0.27	11.4 ± 0.64	562.0 ± 7.35	28941.5 ± 128.09	11480025.3 ± 367755.46

Table 1: Planning times [ms] for different sizes of the hand washing domain

- An extension to the prioritized sweeping algorithm was proposed which exploits properties of planning problems in PAC-MDP learning. Specifically, only policy predecessors of each state are added to the priority queue in contrast to adding all predecessors as in the standard prioritized sweeping algorithm
- It was shown that the original backward value iteration algorithm from the literature - which updates each state exactly once in each iteration - can fail on a broad class of MDP domains. The problem and one straightforward correction were shown. Then, our extensions to the corrected version of BVI which are specific to planning in PAC-MDP learning were proposed. Specifically, it was shown that the predecessor state does not have to be expanded in a given iteration when all its successors have their residuals smaller than precision ϵ
- The instances of the hand washing domain with large state spaces were solved, which extends applicability of the PAC-MDP paradigm considerably beyond existing PAC-MDP evaluations which can be found in the literature
- All presented in the paper algorithms are equally applicable to goal-based as well as infinite horizon RL problems, because both in prioritized sweeping and backward value iteration, planning starts from a specific state, and it does not matter whether the domain has a goal state or not

The theoretical justification to all contributions was provided and all approaches were further evaluated empirically.

Regardless of the more specific details of the empirical evaluation, a particularly substantial contribution of this work is that the standard value iteration algorithm can be made considerably faster by the straightforward application of the BAO update rule which was proposed in this paper.

10 Acknowledgements

This research was sponsored by American Alzheimer's Association grant number ETAC-08-89008.

11 References

[1] J. Asmuth, M. L. Littman, and R. Zinkov. Potential-based shaping in model-based reinforcement learning. In *Proceedings of AAAI*, 2008.

[2] D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1996.

[3] R. I. Brafman and M. Tennenholtz. R-max - a general polynomial time algorithm for near-optimal reinforcement learning. *JMLR*, 3:213–231, 2002.

[4] P. Dai and E. A. Hansen. Prioritizing Bellman backups without a priority queue. In *Proceedings of ICAPS*, 2007.

[5] J. Hoey and M. Grześ. Distributed control of situated assistance in large domains with many tasks. In *Proc. of ICAPS*, 2011.

[6] J. Hoey, P. Poupart, A. von Bertoldi, T. Craig, C. Boutilier, and A. Mihailidis. Automated handwashing assistance for persons with dementia using video and a partially observable markov decision process. *Computer Vision and Image Understanding*, 114(5), May 2010.

[7] J. Hoey, R. St-Aubin, A. Hu, and C. Boutilier. SPUDD: Stochastic planning using decision diagrams. In *Proceedings of UAI*, pages 279–288, 1999.

[8] S. M. Kakade. *On the Sample Complexity of Reinforcement Learning*. PhD thesis, Gatsby Computational Neuroscience Unit, University College, London, 2003.

[9] M. Kearns and S. Singh. Near-optimal reinforcement learning in polynomial time. *Machine Learning*, 49:209–232, 2002.

[10] J. Z. Kolter and A. Ng. Near-Bayesian exploration in polynomial time. In *Proceedings of ICML*, 2009.

[11] L. Li and M. L. Littman. Priorioritized sweeping converges to the optimal value function. Technical report, Rutgers University, 2008.

[12] A. W. Moore and C. G. Atkenson. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning*, 13:103–130, 1993.

[13] A. Y. Ng and M. Jordan. PEGASUS: A policy search method for large MDPs and POMDPs. In *In Proceedings of Uncertainty in Artificial Intelligence*, pages 406–415, 2000.

[14] P. Poupart, N. Vlassis, J. Hoey, and K. Regan. An analytic solution to discrete Bayesian reinforcement learningbell. In *Proceedings of ICML*, pages 697–704, 2006.

[15] M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1994.

[16] A. L. Strehl and M. L. Littman. An analysis of model-based interval estimation for Markov decision processes. *Journal of Computer and System Sciences*, 74:1309–1331, 2008.

[17] R. S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of ICML*, pages 216–224, 1990.

[18] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[19] I. Szita and C. Szepesvári. Model-based reinforcement learning with nearly tight exploration complexity bounds. In *Proceedings of ICML*, pages 1031–1038, 2010.

[20] T. Takaoka. Theory of trinomial heaps. In *Proceedings of the International Conference on Computing and Combinatorics*, LNCS, pages 362–372, 2000.

[21] T. J. Walsh, S. Goschin, and M. L. Littman. Integrating sample-based planning and model-based reinforcement learning. In *Proceedings of AAAI*, 2010.