In Proceedings of the International Joint Conference on Neural Networks (IJCNN) 2013

On the Convergence of Techniques that Improve Value Iteration

Marek Grześ and Jesse Hoey

***Abstract*— Prioritisation of Bellman backups or updating only a small subset of actions represent important techniques for speeding up planning in MDPs. The recent literature showed new efficient approaches which exploit these directions. Backward value iteration and backing up only the best actions were shown to lead to a significant reduction of the planning time. This paper conducts a theoretical and empirical analysis of these techniques and shows new important proofs. In particular, (1) it identifies weaker requirements for the convergence of backups based on best actions only, (2) a new method for evaluation of the Bellman error is shown for the update that updates one best action once, (3) it presents the theoretical proof of backward value iteration and establishes required initialisation, (4) and shows that the default state ordering of backups in standard value iteration can significantly influence its performance. Additionally, (5) the existing literature did not compare these methods, either empirically or analytically, against policy iteration. The rigorous empirical and novel theoretical parts of the paper reveal important associations and allow drawing guidelines on which type of value or policy iteration is suitable for a given domain. Finally, our chief message is that standard value iteration can be made far more efficient by simple modifications shown in the paper.**

I. Introduction And Motivation

We consider the problem of finding an optimal policy in discrete time, finite state and action, discounted (by factor $\gamma < 1$) as well as undiscounted ($\gamma = 1$) Markov Decision Processes [1]. The optimal policy, π^*, can be sought by means of the optimal state, V^*, or state-action, Q^*, value function where $V^*(x)$ is the expected (discounted) reward when the execution of policy π^* starts in state, x, or $Q^*(x,a)$ the analogous reward when the execution starts in state, x, the first action is a and policy π^* is followed thereafter. The best policy can be determined as follows: $\pi^*(x) = \arg\max_a Q^*(x,a)$.

A large body of research in MDP planning is on algorithms which iteratively compute estimates $\hat{V}$ or $\hat{Q}$ of V^* or Q^* until the maximum difference between two successive iterations satisfies the termination condition, i.e., the difference, known as the Bellman error, is sufficiently small. For example, value iteration (VI) applies the following operator—defined as Bellman backup, backup or update for short—to all states in every iteration:

$$\hat{V}'(x) = \max_a \left\{ \hat{Q}(x,a) = R_x(a) + \gamma \sum_{x'} T_{x,a}(x') \hat{V}(x') \right\} \quad (1)$$

and stops when the Bellman error satisfies $||\hat{V}' - \hat{V}|| < \epsilon$. In this paper, we adhere to this practical approach and we assume that planning algorithms stop under the above condition, and we call the resulting $\hat{V}$ the ϵ-optimal value function. The value function, V^π, of the greedy policy, π, derived from ϵ-optimal $\hat{V}$ satisfies $||V^\pi - V^*|| < 2\epsilon\gamma/(1-\gamma)$ when $\gamma < 1$ [2], [3]. $R_x(a)$ is the immediate reward function and $T_{x,a}(x')$ the conditional probability $P(x'|x,a)$ [1].

The basic idea that has been motivating MDP research for decades already is that the rate of converge of value iteration depends on how states are ordered for their updates (using Equation 1 for every state) and in particular some states can be updated many times before other states are updated once. A number of directions were considered which include prioritised sweeping [4], [5], [6], RTDP [7] and its variations [8], and also methods which exploit the topology of the state space [9], [10]. It is important to note that some approaches, such as RTDP, focus on anytime performance and in effect they may be slower than value iteration in converging through all states. Bonet and Geffner [11] discuss these properties of RTDP in detail. RTDP is not considered in our paper because we investigate algorithms that return polices which do not depend on the initial state.

The ordering of backups can be crucial for achieving fast convergence, but it comes at a certain price. For example, prioritised sweeping methods have to maintain a priority queue whereas backward value iteration has to perform backward search from the goal state in every iteration. These operations yield additional computational cost. Thus, other ways of speeding up the planning process can prove themselves to be powerful. One such approach is based on reducing the number of actions updated in every backup of Equation 1. For example, policy iteration methods [12], [13] take advantage of this fact. This direction, in a different flavour, was recently exploited also in [14] where it allowed for an exponential scale-up of the R-max algorithm which requires frequent MDP replanning. In our work on general MDP planning, we found that since ordering may have its cost, action reduction techniques when applied to standard VI methods can turn out to be competitive even against sophisticated ordering methods. For this reason, the goal of this paper is to take a close, analytical look at these kinds of algorithms, show novel theoretical findings, deepen understanding and draw guidelines via rigorous empirical comparisons.

Our contributions can be summarised as follows: (1) it is shown that updating only best actions is valid under relaxed conditions compared to those required in our previous work [14] where their advantage was demonstrated, (2) a new method for evaluation of the Bellman error is shown for one of the updates that focuses on best actions, (3) the proof of backward value iteration that establishes required initialisation is derived, (4) it is shown that the default state ordering of backups in standard value iteration can

Marek Grześ and Jesse Hoey are with the School of Computer Science, University of Waterloo, Canada (email: {mgrzes, jhoey}@cs.uwaterloo.ca).

significantly influence its performance and hence should be reported in empirical comparisons in the literature; we also show how a good order can be derived in various conditions and (5) an empirical analysis is shown that demonstrates significant improvements on standard value iteration without considering any sophisticated prioritisation methods or referring to policy iteration.

The paper is organised as follows: Section II reviews relevant concepts. After that our contributions are presented. First, Section III proves relaxed requirements for the convergence of backups that do not update all actions of every state in one iteration of the value iteration algorithm. In addition, Section IV shows that the calculation of the Bellman error for obtaining an ϵ-optimal value function has to be modified when one of the investigated types of updates from Section III is considered. These two sections are on improving Bellman backups. Next, Section V shows a new proof that guarantees the correctness of backward value iteration, and Section VI highlights the importance of the static order of states in value iteration that competes with more sophisticated algorithms such as backward value iteration discussed in Section V. These two sections are on improving the state order in value iteration. The core contribution of this paper should be sought in the theoretical results, however, Section VII shows additional empirical results that provide a deeper insights into analysed concepts and show evidence that they are useful. The paper concludes in the final section.

II. Background

In this section, relevant algorithms and definitions are reviewed.

A. Policy Iteration

The problem associated with Equation 1 is the need to evaluate all actions in every update. Policy iteration methods avoid this issue by maintaining a specific policy, π, and alternating between policy evaluation and policy improvement. The policy evaluation step can be done iteratively according to the formula:

$$\hat{V}'^{\pi}(x) = R_x(\pi(x)) + \gamma \sum_{x'} T_{x,\pi(x)}(x')\hat{V}^{\pi}(x'). \quad (2)$$

When the number of actions is large, this evaluation is naturally more efficient than the full backup of Equation 1. The policy iteration algorithm starts with a policy, π_0, (which has to be proper when stochastic shortest path (SSP) problems are considered, i.e., the goal state has to be reachable from any state with probability 1), evaluates the policy via computing $\hat{V}^{\pi}$ where evaluation is carried out until the Bellman error drops below ϵ, and then the algorithm tries to improve the policy and computes the next policy, π', using the formula:

$$\pi'(x) = \arg\max_a \left\{ R_x(a) + \gamma \sum_{x'} T_{x,a}(x')\hat{V}^{\pi}(x') \right\}. \quad (3)$$

The algorithm stops when $\pi' = \pi$.

The above is the original formulation of policy iteration (PI) which evaluates each policy until $||\hat{V}'^{\pi} - \hat{V}^{\pi}|| < \epsilon$ [12]. It was observed in the literature that the policy evaluation step does not have to be carried out until the above condition is satisfied. Instead, in every iteration, k, a specific fixed amount, m_k, of policy evaluation steps can be performed. This is known as the modified policy iteration (MPI) algorithm [13]. In our comparisons, MPI is used according to the pseudo-code in [1, pp. 186–187].

B. Review of Relevant Concepts

The following definitions from the existing literature [8] are considered:

Definition 1: Q is pessimistic if $Q(x,a) \leq Q^*(x,a)$ and optimistic if $Q(x,a) \geq Q^*(x,a)$.

Definition 2: Q is monotone pessimistic if $Q(x,a) \leq R_x(a) + \gamma \sum_{x'} T_{x,a}(x')V(x')$ and monotone optimistic if $Q(x,a) \geq R_x(a) + \gamma \sum_{x'} T_{x,a}(x')V(x')$ for all x and a, where $V(x) = \max_a Q(x,a)$.

Theorem 1: If Q is monotone pessimistic (optimistic), then Q is a lower (upper) bound on Q^*.

The proof can be found, for example, in [14] and is based on the intuitive fact that if the monotonicity condition is satisfied for all states, it will remain satisfied after all future Bellman backups until convergence to Q^*. The reader should note that this is a different property than the monotonicity lemma of the Bellman backup—Lemma 2.1 in [15] or the proof of Theorem 3 in [7].

III. Updating Best Actions Only

The deficiency of backups based on a straightforward application of Equation 1 is that all actions have to be backed up. Policy iteration methods avoid this by evaluating a particular, single policy. In this section, another method is considered which is based on a simple intuitive idea, that if all Q-values are optimistic then it is sufficient to keep updating only best actions. We exploited this idea recently in [14] where a particular version of this operation was proposed, named best actions only update (BAO). BAO, as specified in Algorithm 4 in [14], keeps updating the best actions of the current state *until the Bellman error is smaller than* ϵ and then moves to the next state in the same iteration. This approach in conjunction with several MDP planning algorithms allowed for exponential speed-up of the R-max algorithm [16] which performs frequent MDP replanning. An alternative idea to BAO is to update the best action (or all best actions) of a given state *once* and move to the next state immediately after one best action was updated. We call this simpler approach best action once (BAOnce) and analyse it jointly with the BAO method.

The proof of the BAO type of updates which was presented in [14] requires monotone optimistic initialisation according to Definition 2. We show below a new proof that a more general optimistic initialisation (Definition 1) is sufficient.

Theorem 2: Planning based on backups which, in every state, update one or all best actions only once (BAOnce) or keep updating all best actions until the Bellman error of best

actions is smaller than ϵ (BAO) converges to the optimal value function when the initial value function is optimistic.

Proof: First, we use the monotonicity lemma [15, Lemma 2.1] which shows that $V^0 \geq V^*$ guarantees $V^k \geq V^*$ where V^k is obtained by applying either Equation 1 or Equation 2 k times to V^0. The fact that the monotonicity lemma holds for Equation 2, allows us to consider updates of individual actions separately, and it shows that $V^k \geq V^*$ when in every of k state updates an arbitrary single action is updated and $V^0 \geq V^*$.

Second, we have to justify that non-best actions do not have to be updated before currently better actions are updated. Let's assume that a is a non-best action of a particular state x. The update of a could make its Q-value become higher than its current value (this could happen because we do not require Definition 2 to be satisfied). However, the above argument showed that since the current Q-value of that action is definitely higher than its Q^*, there is no reason in performing the backup which would increase its value even further. If, on the other hand, the update of a would make its value lower then there is no reason to update that action as well, because there is another action already better than a. Both cases show that the non-best action a does not have to be updated. ■

BAOnce was applied in [11], however it was not mentioned in that paper explicitly, and one can read this fact from the pseudo-code of the algorithm which updates only one best action in every backup of a particular state. Their approach satisfies our above theorem because RTDP uses optimistic initial values. Another approach which has similar flavour to BAOnce updates is found in Sampled RTDP in [17] where only a subset of actions that are 'likely' to be the best is sampled and updated. Since, the best actions cannot be identified certainly in that case (the concurrent MDP with an exponential growth of parallel actions is considered), Mausam and Weld [17] framed their solution as an approximate approach. If, for every update, they could identify the best greedy action, their algorithm would satisfy the above theorem and would become exact. In [14], we proposed an extension to this approach that we named BAO and since it yielded a significant improvement when planning in R-max, it was important to analyse it deeper in this paper.

Value iteration with BAO and BAOnce updates can be seen as a special case of modified policy iteration, in which every backup is performing a continuous evaluation of the policy that is greedy according to the current $\hat{Q}$. Full backups of policy iteration never happen since the policy update step is implied by selection or memorization of $\arg\max_a Q(x,a)$. Thus VI with BAO or BAOnce is like MPI with implicit but continuous policy updates without the use of full backups, and continuous evaluation of the greedy policy induced by $\hat{Q}$. This indicates that there may be domains where VI with BAO can work better than policy iteration since the latter has to do full backups in its policy update step.

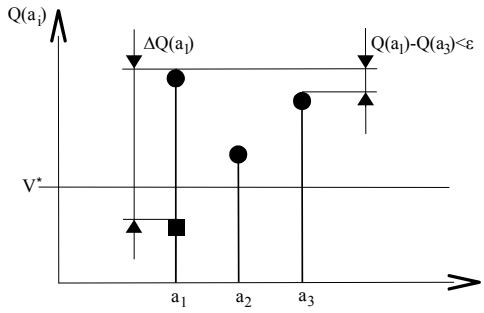

Fig. 1. The diagram is for one state, s, of the MDP. The horizontal axis represents actions and the vertical axis Q-values of actions. Current values of actions are represented with bold lines that end with a circle. The square point in the value of action a_1 represents the new value after the update of action a_1.

IV. Performance Bounds when Updating Best Actions Once

The previous section has shown that updates that focus on best actions guarantee convergence when initial values are optimistic. As noted in the introduction, value iteration or related methods are usually stopped when the Bellman error is within arbitrarily small ϵ. The resulting value function is named ϵ-optimal, and the value function, V^π, of the greedy policy, π, derived from such a value function satisfies $||V^\pi - V^*|| < 2\epsilon\gamma/(1-\gamma)$ when $\gamma < 1$ [2], [3]. The problem is that in order to apply this bound to BAOnce updates, a slightly modified stopping condition is needed or more precisely a different way of computing the Bellman error is required. The problem is explained using Fig. 1. In the figure, initially $Q(a_1) - Q(a_3) < \epsilon$, hence, if a_1 is the only updated action when the considered state, s, is updated in one iteration of value iteration, then, even though the value of a_1 can change by $\Delta Q(a_1) >> \epsilon$, the Bellman error can wrongly indicate convergence in the state because $\Delta V(s) = |V(s) - V(s')| = |Q(a_1) - Q(a_3)| < \epsilon$. Such an error (that we refer to here as ΔV because it is computed as $|V(s) - V'(s)|$) cannot be used for stopping the algorithm because as the image shows the values in state s have not converged after update of only action a_1. This situation requires a modified notion of an error for updates that update only one action in every update of the state. In order to address this requirement, we introduce a new error:

$$\Delta V^u(s) = \max\left[|V(s) - V'(s)|, \max_a(|Q(s,a) - Q'(s,a)|)\right] \quad (4)$$

that is an upper bound on ΔV where $\max_a$ in the equation is for those actions that were updated. Since ΔV^u is an upper bound on ΔV, it will cope with problems exposed in Fig. 1 when only one action of a state is updated in every iteration of the value iteration algorithm. This analysis is important from the theoretical point of view as it has never been investigated according to the best of our knowledge. It is also required in practice if one wishes to obtain an ϵ-optimal value function and not to update all actions in every state.

We indicate further that this requirement does not concern

BAO updates which can still use error ΔV, because they update all best actions until the best action does not change more than ϵ, hence the problem depicted in Fig. 1 does not apply.

V. Backward Value Iteration

The prioritization of Bellman backups without the priority queue was investigated in [9] where backward value iteration (BVI) was proposed. The BVI algorithm is characterised as follows: it performs iterations of state updates where, in every iteration, states are traversed starting from the goal state and the policy predecessors of every state are visited after the state is backed up. Each node is visited and updated exactly once in each iteration.

It was recently shown in [14] that the original BVI algorithm can fail in computing a correct value function. If, after any iteration of the BVI algorithm, the policy induced by the current value function has a loop (which means that the policy is not proper), the states involved in the loop will not be updated in any of the future iterations. This will leave at least the states involved in the loop with incorrect values. This can happen because states in the loop have only loop states among their policy successors, and the goal state is not reachable from the loop using policy actions (the policy is not proper). The problem is discussed in [14] and the example is provided which shows how this situation can arise.

The above problem indicates that the following 'loop invariant' [18] for the main loop of BVI should be considered in the analysis of correctness of this algorithm: *after every iteration, the policy induced by the current value function is proper*. Below we show the BVI proof which identifies required initialisation of the value function which guarantees that the above loop invariant is satisfied after every iteration.

Theorem 3: In the backward value iteration algorithm specified in [9], the policy induced by the current value function is proper after every iteration when:

1) the initial value function is monotone pessimistic, i.e., the conditions of Definition 2 are satisfied
2) the initial policy is proper, i.e., at least one goal state is in the policy graph of each state

Proof: Since the initial/previous policy is proper, all states will be updated in the current iteration. We need to show that the policy remains proper in all states which were updated within the current iteration and that this will hold also after remaining states—not updated till now—will be updated. In this way, the policy will remain proper after all states are updated. Hence, the proof is by induction on the number of backups within one iteration. The base case holds after the backup of the first state because the first state has the goal state as its successor and the policy action of the first state has to lead to the goal state since its remaining—non-goal—successors are initialised pessimistically. Next, we assume satisfiability after updating k states. In the $k+1$ update, state x is updated. The next states, x', reachable from x by the new greedy action of x may or may not have been updated before x. (1) If they have been updated then we already know that x is on the path to the goal state in the next iteration because by the induction assumption its next states are (and will remain) in the policy graph of the goal state. (2) If any of the next states x' of x has not been updated in the current iteration then we have to show that all such x' have to have their values (after their forthcoming update of the same iteration) higher than x, and in this way x cannot become the only successor of x'. This will be indeed the case because $V(x')$ is (monotone) pessimistic and the forthcoming update can make $V(x')$ only higher or in the worst case the update can leave the current value the same in the current iteration[1]. The current—monotone pessimistic—$V(x')$ has already made x' to be the successor of x (after the current $k+1$st update of x), and the current $V(x')$ is better (and can become even better after the forthcoming update of x') than the value which would be obtained by x' having x as its only successor. This means, that x cannot become a successor of x' when x' will be updated, the policy loop will not appear (i.e., the policy will remain proper), and x is guaranteed to have the goal state in its policy graph after update $k+1$ and after updates of its all successors x' which still need to be updated in the same iteration. This shows that the policy is proper for all states which were updated, will remain proper after other states are updated, and that the policy remains proper after the iteration ends. ■

When the loop invariant of Theorem 3 is satisfied, the BVI algorithm is valid because it implements standard value iteration where the only extension is a special ordering of updates, and the satisfiability of the invariant will guarantee that all states are backed up in each iteration until the Bellman error meets the convergence criterion on all states.

The monotone pessimistic initialisation required by Theorem 3 is more challenging to obtain than arbitrary pessimistic initialisation. This difficulty was investigated in detail in [8] where an algorithm to find monotone pessimistic initialisation was proposed and is named Dijsktra Sweep Monotone Pessimistic Initialisation (DS-MPI). Since this algorithm guarantees monotonicity, we suggest that it could be used in order to obtain an initial value function for BVI and the combination of these two algorithms will be used in our empirical evaluation. Another way of satisfying our new requirements of the BVI algorithm is to use non-monotone pessimistic initialisation, and apply single-sided backups [19]. These backups compute a new value which is then ignored when the current value is higher—monotonicity is enforced and values can only increase.

The previous sections showed that BAO updates could be considered a special case of MPI. BVI however does not have this property when full backups are used. The current policy is used only for determining the order in which states are updated. Since the policy implied by the

[1]We note that this is not the case when initialisation is non-monotone pessimistic or optimistic. We cannot guarantee in the optimistic case that x' after its update cannot be a policy predecessor of x, and we cannot exclude the policy loop—this is exactly what is happening in the example in our previous work in [14] and our proof agrees with that example.

Q-function is never explicitly improved or checked, BVI has requirements of proper policies indicated in Theorem 3 and the corresponding loop invariant. This observation suggests one improvement to BVI, which could make it applicable with any initialisation of its value function. The idea is to add, like in policy iteration, the policy check/improvement step using a full backup on *all states* after BVI ends and repeat BVI when the Bellman error of the policy improvement step does not satisfy $||\hat{V}' - \hat{V}|| < \epsilon$. This algorithm could be easily proven by observing the fact that the post-BVI error check is checking all states present in the state space, and the algorithm will terminate only when ϵ is satisfied for all states. The reader should note that such a stopping condition would be equivalent to the one which is in VI.

VI. The Order of Backups in Value Iteration

Algorithms, such as backward value iteration [9], topological value iteration [10], prioritised sweeping [4], or heuristic search value iteration [20] in partially observable MDPs aim at improving the order of state updates. As long as these methods explicitly and dynamically adjust the order of state updates, the importance of another implicit static ordering is usually left unmentioned in existing empirical comparisons of these kinds of algorithms. In this paper, we bring to the attention of the community the fact that when standard Gauss-Seidel value iteration (GSVI) is applied, there exists also some *default (i.e., implied by particular implementation)*, static state ordering that, when easily adjusted, can have a huge impact on the speed of convergence of GSVI. We indicate that in the case of domains with a terminating goal state, the states of the domain can be ordered beforehand by applying breath-first search (BFS) from the goal state. Intuitively, states closer to the goal state should be updated first and the breath-first order is particularly convenient because it can additionally deal—to some extend—with unintended stochastic outcomes of actions (i.e., outcomes that do not lead towards the goal state). The order in which states are visited during BFS represents a static order of updates that, as shown in our experiments below, is very competitive against more sophisticated dynamic ordering methods that are time consuming because they require additional resources to adjust the order dynamically.

VII. Experimental Results

Having presented our theoretical results, we now show experiments on five domains that have different properties: the first two are discounted with two actions in every state, the third one has many actions, the fourth one has interesting realistic properties that allow for determining a better upper bound on initial values of states, and the fifth one represents a class of stochastic shortest path problems where an informed admissible bound can be easily computed. The purpose of experimental results is to demonstrate the significance of those concepts that were investigated theoretically but also to give methodological guidance on how these kinds of algorithms could be compared in the literature in the future.

The following algorithms are considered:

Nr	Time [ms]		Backups		Algorithm
1	6562.8	± 19.4	7594965.0	± 9269	VI-V(0)-random
2	4814.6	± 14.4	6324696.0	± 0	VI-V(0)-BFS
3	1855.2	± 6.7	2073292.2	± 5330	VI-V(1)-random
4	1331.8	± 5.1	1687770.0	± 0	VI-V(1)-BFS
5	**769.0**	± **1.7**	942630.0	± 0	**VI-V(1)-BAO-BFS**
6	6581.5	± 16.5	5542988.0	± 0	PS-V(0)
7	1036.8	± 1.0	871662.0	± 0	PS-V(1)
8	2913.9	± 11.9	3407518.8	± 11859	MPI(15)-V(0)-BFS
9	1135.2	± 13.0	1073066.4	± 11063	MPI(5)-V(1)-BFS
10	8425.1	± 14.6	6269990.0	± 0	BVI-V(0)-SS
11	1406.3	± 444.7	942374	± 298004.8	LBVI-V(1)-BAO

TABLE I

Results on the mountain car.

- VI: standard Gauss-Seidel value iteration [21]
- MPI(k): modified policy iteration [13] where k is the constant number of iterations in policy evaluation
- PI: policy iteration [12]
- PS: prioritised sweeping with priority based on the Bellman error [4]
- BVI-SS: BVI with single sided updates [19]
- BVI-DS-MPI: BVI with monotone pessimistic initialisation using DS-MPI [8]
- BVI-PC: BVI with policy check
- LBVI: BVI with backward search to all predecessors as introduced in [14]

If BAO or BAOnce are applicable, they are used as one of the options and added to the name of the algorithm in the results. $V(i)$ and V_{max} mean that the value function of a particular algorithm was initialised with i or $R_{max}/(1-\gamma)$ correspondingly. All flavours of BVI are applicable exclusively to domains with a terminating state. The fact whether the domain is discounted or not does not matter as long as there is a goal state or states from which BVI can start. When applicable, BFS state ordering was evaluated and compared against random ordering, and random ordering used when BFS did not apply. Each domain was evaluated 10 times, for every randomly generated domain 10 instances were generated and solved, the precision ϵ was 10^{-5}, and the standard error of the mean (SEM) is shown in the results which display the planing time and the number of performed backups. In all cases, the final result was verified whether the Bellman error indeed satisfied $\epsilon = 10^{-5}$ when the algorithm terminated. For all algorithms, either the result with the best parameter configuration is presented, or several results are shown when the influence of different factors is investigated. All algorithms were implemented in `C++` and executed on the same machine.

One could argue that some of the domains evaluated below are relatively small. We clarify that all algorithms are thoroughly evaluated on the same set of domains and the difference between algorithms would be comparable and particular values scaled up appropriately when compared on larger domains with the same properties. We believe that this fact does not diminish the importance of properties that we show in our comparisons.

The first set of experiments is on the **mountain car (MCar)** and **single arm pendulum (SAP)** where the exact

Nr	Time [ms]		Backups		Algorithm
1	27924.0	± 57.1	26584000.0	± 29333	VI-V(0)-random
2	**7842.0**	± **22.7**	9360000.0	± 0	**VI-V(0)-BFS**
3	33366.2	± 105.5	30992000.0	± 48000	VI-V(1)-random
4	10059.0	± 21.4	11680000.0	± 0	VI-V(1)-BFS
5	10486.6	± 25.7	12690283.0	± 0	VI-V(1)-BAO-BFS
6	26273.5	± 83.4	18218230.0	± 0	PS-V(0)
7	18684.2	± 41.2	12855608.0	± 0	PS-V(1)
8	**6256.7**	± **20.8**	5260000.0	± 13663	**MPI(5)-V(0)-BFS**
9	**6867.2**	± **21.3**	4719884.0	± 0	**BVI-V(0)-SS**
10	13061.3	± 15.8	9359768.0	± 0	LBVI-V(0)
11	18012.9	± 49.1	12689973.0	± 0	LBVI-V(1)-BAO

TABLE II

RESULTS ON THE SINGLE ARM PENDULUM.

Nr	Time [ms]		Backups		Algorithm
1	3545.9	± 147.0	7526000.0	± 310506	VI-V(0)
2	3024.4	± 127.4	6305000.0	± 255679	VI-Vmax
3	**170.9**	± **4.6**	172349.5	± 5251	**VI-Vmax-BAO**
4	**166.3**	± **2.7**	127223.5	± 1900	**VI-Vmax-BAOnce**
5	6958.2	± 142.7	7819750.0	± 155515	PS-Vmax
6	1963.9	± 72.2	96840.0	± 3460	MPI(2)-V(0)
7	431.8	± 14.2	98630.0	± 3279	MPI(10)-V(0)
8	250.6	± 6.8	102980.0	± 2862	MPI(20)-V(0)
9	**101.1**	± **4.8**	209310.0	± 10885	**MPI(500)-V(0)**
10	**111.4**	± **5.4**	251550.0	± 12444	**PI-V(0)**

TABLE III

RESULTS ON NON-TERMINATING MDPS AND $\gamma = 0.99$

implementation was taken from the source code accompanying publication of [22]. For the description of these domains the reader is referred to Sections 3.1 in [22]. We used the discretisation into 100×100 states in MCar, and 200×200 in SAP. γ was set to 0.99. Both tasks have two actions in every state, and the reward of 1 only upon entering the goal state, hence $V_{min} = 0$ and $V_{max} = 1$. The detailed evaluation is in Tables I and II, and we discuss the key insights: **(1)** the comparison of lines 1 and 2 in both tables shows that default/static state ordering has an important influence on the performance of VI and significant savings can be achieved with the use of the BFS order as we indicated in Section VI. Additionally in Table II, VI with random order (line 1) is slower than PS (lines 6 and 7), but is faster than PS when the BFS state order is used in line 2 (NB: the time to initialise the BFS order before VI starts is counted). **(2)** the comparison of line 1 against line 3 or line 2 against line 4 in both tables shows the impact of the initialisation on performance of VI. Specifically, initialisation with 0 increases the planning time of VI over 3.5 times in Table I. The impact in Table II is reversed where initialisation with 0 yields faster planning. **(3)** the previous dependency extends to all algorithms within each table, which means that in Table I algorithms initialised with 1 are faster, whereas in Table II initialisation with 0 is better. This explains the performance of BVI algorithms because LBVI-BAO requires initialisation with a value of 1 (optimistic), hence it is relatively good in Table I and BVI requires initialisation with a value of 0 (pessimistic), hence it is very good in Table II. **(4)** the two previous comments explain why BAO was the best on the MCar in Table I, and not the best one in Table II. This is because optimistic initialisation is very good in MCar and pessimistic is better in SAP whereas BAO cannot use pessimistic initialisation. **(5)** the comparison of lines 5 and 11 in Table II confirms our initial hypothesis that algorithms such as BVI or PS may spend a significant amount of time on arranging the prioritisation/search through the state space. LBVI-BAO performs almost exactly the same number of backups as VI-BAO however the time of LBVI is two times longer than that of VI-BAO because VI has a fixed, BFS in that case, order of states, determined only once before planning starts, and can focus on doing actual updates. VI static state ordering was critical for this improvement. This is an empirical evidence of our claim in Section VI and also shows that the literature should report how states are ordered in standard value iteration in order to make fair comparisons against more sophisticated methods.

Since the savings due to policy iteration or BAO updates can be more evident when **the number of actions is high**, in the second experiment, the methodology from [1] was used and non-terminating MDPs with 100 states and 100 actions in each state were randomly generated. Each action could lead to three randomly selected states with a probability sampled from a uniform distribution. Rewards were sampled from a uniform distribution [1-100]. The result is in Table III. The key observation is that value iteration enhanced with BAO updates (line 3 in Table III) can perform almost as well as policy iteration methods, which by definition work well on domains with many actions, with the best tuning of their parameters (the importance of this feature of VI is stressed in conclusion). There is no terminating state in these discounted MDPs so prioritised sweeping had to assign initially non-zero priority to all states as required in [23] and its performance turned out to be weak. Also the main gain here is achieved by selecting good actions to update instead of prioritising states.

Car replacement is a scenario with realistic properties where many actions are required (there are 41 states and 41 actions in this domain). Results are in Table IV and are for $\gamma = 0.97$ which is is taken from [12] where it is justified as having a real meaning of around 12% annual interest rate. Rewards have a high variance in this domain, but this time there is another property which strongly influences the performance of evaluated algorithms. Specifically, a short horizon policy is very sub-optimal when compared with a long horizon policy, because actions which yield high instantaneous reward are very sub-optimal in the long term (selling a good car now and buying a very cheap one may result in getting money now but is not good in the long term). Hence, BAO first learns actions which seem promising in short term and then has to unlearn them. Similar problems are encountered by MPI. Specifically, when k is small, MPI is slower than with higher values of k. With sufficiently large k, policies are evaluated 'almost' exactly, and this helps avoiding short horizon policies. This also explains why MPI with lowest k is even slower than BAO because MPI applies full backups during policy improvement. An explanation is required why $V(0)$ could be used to initialise the value

Nr	Time [ms]		Backups		Algorithm
1	126.3	± 6.2	500601.8	± 23357	VI-$V(0)$
2	112.4	± 0.9	172302.0	± 1095	VI-BAO-V_{max}
3	47.5	± 0.7	78500.1	± 1172	VI-BAO-$V(0)$
4	25.8	± 0.5	46239.7	± 998	VI-BAO-V^+
5	108.7	± 0.6	158153.4	± 284	VI-BAOnce-V_{max}
6	42.5	± 0.5	63849.3	± 439	VI-BAOnce-$V(0)$
7	23.4	± 0.3	36588.4	± 412	VI-BAOnce-V^+
8	245.8	± 2.0	590875.6	± 2190	PS-$V(0)$
9	66.9	± 1.8	14645.2	± 348	MPI(2)-V_{min}
10	51.3	± 1.9	11184.8	± 420	MPI(2)-$V(0)$
11	41.0	± 1.4	9003.6	± 313	MPI(2)-V^-
12	30.7	± 1.0	15420.1	± 483	MPI(5)-V_{min}
13	22.0	± 1.0	11275.0	± 521	MPI(5)-$V(0)$
14	20.4	± 0.9	10541.1	± 423	MPI(5)-V^-
15	16.9	± 0.4	15379.1	± 388	MPI(10)-V_{min}
16	12.7	± 0.3	11562.0	± 257	MPI(10)-$V(0)$
17	13.0	± 0.6	11586.6	± 409	MPI(10)-V^-
18	13.4	± 0.2	16465.6	± 226	MPI(15)-V_{min}
19	10.5	± 0.3	12509.1	± 328	MPI(15)-$V(0)$
20	9.9	± 0.5	12349.2	± 579	MPI(15)-V^-
21	11.2	± 0.4	16145.8	± 448	MPI(20)-V_{min}
22	9.2	± 0.3	13763.7	± 470	MPI(20)-$V(0)$
23	9.2	± 0.3	13771.9	± 382	MPI(20)-V^-
24	19.7	± 0.5	74013.2	± 2712	MPI(500)-V_{min}
25	19.3	± 0.5	72832.4	± 2140	MPI(500)-$V(0)$
26	18.5	± 0.3	70044.4	± 1477	MPI(500)-V^-
27	20.5	± 0.6	77982.0	± 2065	PI-V_{min}
28	21.1	± 0.7	79318.6	± 2131	PI-$V(0)$
29	19.4	± 0.6	74669.2	± 2402	PI-V^-

TABLE IV

RESULTS ON CAR REPLACEMENT

function in BAO. This is the result of the observation that in this domain there is never a positive long term reward because the possession of a car always incurs costs. With this knowledge, BAO can be competitive even on this challenging domain. If the bound can be improved, BAO gains further speed-up. Thus, $V(0)$, V_{max}, and V^+ yields optimistic initialisation required by BAO.

Admissible, i.e., optimistic heuristics are the main driving force of informed search methods [24]. In some cases, such heuristics which upper bound V^* can be identified in MDP domains. The last experiment aims at evaluating considered algorithms when such **an admissible heuristic is available**. A 50×50 maze was used where each position is blocked with probability 0.15, there is one terminating state, and there are 8 actions in every state. An action has its expected effect with probability 0.8 and slips to one of the 4 adjacent transitions with the remaining mass of probability uniformly distributed among those transitions. All actions yield reward of -1 and γ is 1, hence this is a standard stochastic shortest path problem. The Euclidean distance to the goal was used to derive an informed upper bound on V^*, and, when used, $V(Eucl)$ was added to the name of the algorithm in results in Table V. The key observations: **(1)** again, standard VI improves dramatically when BFS instead of random state ordering is used together with informed Euclidean initialisation (line 4 in Table V). **(2)** enhanced with BAO updates, VI was the best among all algorithms (line 5). NB: Euclidean initialisation is valid with BAO updates due to Theorem 2. **(3)** Prioritized sweeping, for which initialisation -100 (a lower bound on V^*) was the best, was worse than all other algorithms on this domain. **(4)** MPI did not beat VI

Nr	Time [ms]		Backups		Algorithm
1	889.8	± 2.1	1186660.8	± 2254	VI-V(0)-random
2	862.6	± 1.8	1183280.0	± 0	VI-V(0)-BFS
3	648.9	± 3.9	867175.2	± 3608	VI-V(Eucl)-random
4	323.8	± 3.8	422600.0	± 0	VI-V(Eucl)-BFS
5	**163.1**	± **1.2**	202274.0	± 0	**VI-V(Eucl)-BAO-BFS**
6	295.1	± 0.5	345545.0	± 0	VI-V(Eucl)-BAOnce
7	1493.6	± 4.0	1529072.0	± 0	PS-V(-100)
8	211.6	± 0.5	69729.0	± 0	MPI(2)-V(Eucl)-BFS
9	200.5	± 0.5	107763.0	± 0	MPI(5)-V(Eucl)-BFS
10	228.5	± 0.3	175379.0	± 0	MPI(10)-V(Eucl)-BFS
11	405.9	± 0.8	464437.4	± 282	MPI(100)-V(Eucl)-BFS
12	869.2	± 1.4	1289352.6	± 1035	MPI(500)-V(Eucl)-BFS
13	565.0	± 1.1	456200.0	± 0	BVI-V(-100)-SS
14	611.5	± 0.9	544074.0	± 0	BVI-V(DS-MPI)
15	819.4	± 1.4	680856.0	± 0	BVIPC-V(-100)
16	528.3	± 2.1	422408.0	± 0	LBVI-V(Eucl)
17	366.8	± 2.0	202122.0	± 0	LBVI-V(Eucl)-BAO

TABLE V

RESULTS ON THE NAVIGATION MAZE

with BAO updates as well, even with its best combination of parameters and initialisation. **(5)** BVI methods were better than the most naïve VI, but lost against VI enhanced with BAO updates as well as with BFS ordering and Euclidean initialisation.

An important lesson learned form our experiments is that the state ordering of VI (MPI as well) should always be reported in evaluations that involve this algorithm. As shown in our experiments, it would be easy to put VI in a disadvantaged position through specific state orderings during planning. Our experiments show that more intelligent, such as BFS, state orderings reduce the gap between standard VI and methods such as prioritised sweeping or backward value iteration. Also the gap between MPI and VI is significantly reduced on a challenging case with many actions.

BAOnce was reported only in Table III (line 4) and Table IV because this was the only experiment where it outperformed BAO. In all other experiments, BAO was better. A general observation (also from experiments not shown in the paper) was that BAO is more robust against different domain properties. This can explain why BAOnce, even though found in the earlier literature, never displayed more pronounced improvements. BAO exploits the advantages of this idea further, which, for general MDPs, were shown in our experiments.

VIII. CONCLUSION

The research presented in this paper had shown that standard value iteration is able to perform much faster with better static state ordering and more efficient backups. The specific contributions are:

- We proved that updates of only best actions can be applied when initialisation is optimistic, which is easier to meet than monotone optimistic initialisation, and allows using bounds derived from admissible heuristics which displayed significant speed-up in our experiments
- We showed that in order to obtain an ϵ-optimal value function, the error evaluation of updates that update best actions only *once* has to be modified

- We identified the loop invariant of the main loop of the BVI algorithm and derived the proof and the initial conditions which guarantee that the BVI algorithm will converge
- We argued that state ordering should be reported in empirical evaluations of value and policy iteration algorithms, especially when they are compared against prioritisation methods, and we showed empirical evidence that better ordering can make value iteration very competitive or better than more sophisticated methods such as prioritised sweeping or backward value iteration. We proposed the BFS static state ordering in VI and showed its advantage empirically. This creates an important methodological guidance
- The initialisation of the value function was shown to be another factor which improves performance of standard algorithms, and (admissible) heuristics should be used in the same vain as in informed heuristic search. This observation has again methodological significance, because the initialisation can speed up standard value iteration and this fact should be considered in empirical evaluations
- The importance of BAO updates was shown in general MDP planning, whereas the existing work has shown results in reinforcement learning which is based on constant replanning of a slightly changing MDP

Many similar MDPs are usually solved when inverse reinforcement learning (IRL) is considered [25]. By arranging the search process of the IRL algorithm in a special way (i.e., preserving optimism), BAO updates could yield very significant speed-up for the same reason as in R-max [14]. Our theoretical analysis in this paper increases understanding of such improvements.

Policy iteration in a distributed fashion is still a challenge [26] whereas VI (also with BAO updates) easily satisfies requirements for convergence when implemented in a parallel or distributed architecture [27], yet yielding performance competitive with policy iteration methods as was shown in our comparisons even in a challenging case with many actions.

Planning in MDPs can be improved using another orthogonal approach that eliminates sub-optimal actions using lower and upper bounds or other metrics computed for the purpose of eliminating actions [1], [28]. One disadvantage of those action elimination procedures is that (in most cases) they require two copies of the value function. On the positive side, they can permanently eliminate sub-optimal actions. In the case, when such methods need to maintain an upper bound, our BAO backup could lead to improvements of those methods and it would be interesting to investigate this interrelationship in future work.

ACKNOWLEDGMENT

This work was supported by the Ontario Ministry of Research and Innovation, Toronto Rehabilitation Institute and the Alzheimer's Association grant ETAC-10-173237.

REFERENCES

[1] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., 1994.
[2] R. Williams and L. C. Baird, "Tight performance bounds on greedy policies based on imperfect value functions," Northeasterm University, Tech. Rep. NU-CCS-93-14, 1993.
[3] S. Singh and R. Yee, "An upper bound on the loss from approximate optimal-value functions," *Machine Learning*, vol. 16, no. 3, pp. 227–233, 1994.
[4] A. W. Moore and C. G. Atkenson, "Prioritized sweeping: Reinforcement learning with less data and less time," *Machine Learning*, vol. 13, pp. 103–130, 1993.
[5] D. Andre, N. Friedman, and R. Parr, "Generalized prioritized sweeping," in *Proc. of NIPS*, 1997, pp. 1001–1007.
[6] H. B. McMahan and G. J. Gordon, "Fast exact planning in markov decision processes," in *Proc. of ICAPS*, 2005.
[7] A. G. Barto, S. J. Bradtke, and S. P. Singh, "Learning to act using real-time dynamic programming," *Artificial Intelligence*, vol. 72, no. 1-2, pp. 81–138, 1995.
[8] H. B. McMahan, M. Likhachev, and G. J. Gordon, "Bounded real-time dynamic programming: RTDP with monotone upper bounds and performance guarantees," in *Proc. of ICML*, 2005, pp. 569–576. [Online]. Available: http://doi.acm.org/10.1145/1102351.1102423
[9] P. Dai and E. A. Hansen, "Prioritizing Bellman backups without a priority queue," in *Proc. of ICAPS*, 2007.
[10] P. Dai and J. Goldsmith, "Topological value iteration algorithm for markov decision processes," in *Proc. of IJCAI*, 2007, pp. 1860–1865.
[11] B. Bonet and H. Geffner, "Labeled RTDP: Improving the convergence of real-time dynamic programming," in *Proc. of ICAPS-03*, 2003, pp. 12–21.
[12] R. A. Howard, *Dynamic Programming and Markov Processes*. Cambridge: MIT Press, 1960.
[13] M. L. Puterman and M. C. Shin, "Modified policy iteration algorithms for discounted Markov decision problems," *Management Science*, vol. 24, pp. 1127–1137, 1978.
[14] M. Grześ and J. Hoey, "Efficient planning in R-max," in *Proc. of AAMAS*, 2011.
[15] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-Dynamic Programming*. Athena Scientific, 1996.
[16] R. I. Brafman and M. Tennenholtz, "R-max - a general polynomial time algorithm for near-optimal reinforcement learning," *JMLR*, vol. 3, pp. 213–231, 2002.
[17] Mausam and D. S. Weld, "Solving concurrent markov decision processes," in *Proc. of AAAI*, 2004.
[18] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson, *Introduction to Algorithms*, 2nd ed. McGraw-Hill Higher Education, 2001.
[19] S. Singh and V. Gullapalli, "Asynchronous modified policy iteration with single-sided updates," University of Massachusetts, Tech. Rep., 1993.
[20] T. Smith and R. G. Simmons, "Heuristic search value iteration for POMDPs," in *Proc. of UAI*, 2004.
[21] D. P. Bertsekas, *Dynamic Programming and Optimal Control (2 Vol Set)*. Athena Scientific, 3rd ed., 2007.
[22] D. Wingate and K. D. Seppi, "Prioritization methods for accelerating MDP solvers," *Journal of Machine Learning Research*, vol. 6, pp. 851–881, 2005.
[23] L. Li and M. L. Littman, "Priorioritized sweeping converges to the optimal value function," Rutgers University, Tech. Rep., 2008.
[24] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach (2nd Edition)*. Prentice Hall, 2002.
[25] A. Y. Ng and S. J. Russell, "Algorithms for inverse reinforcement learning," in *Proc. of ICML*, 2000, pp. 663–670.
[26] D. P. Bertsekas and H. Yu, "Distributed asynchronous policy iteration in dynamic programming," in *Proc. of Allerton Conference on Communication,Control, and Computing*, 2010.
[27] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and distributed computation: numerical methods*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1989.
[28] N. M. A. Jaber, "Accelerating successive approximation algorithm via action elimination," Ph.D. dissertation, University of Toronto, 2008.