Mixed Acceleration Techniques for Solving Quickly Stochastic Shortest-Path Markov Decision Processes

M. de G. García-Hernández*[1], J. Ruiz-Pinales[1], E. Onaindía[2], S. Ledesma-Orozco[1], J. G. Aviña-Cervantes[1], E. Alvarado-Méndez[1], A. Reyes-Ballesteros[3]

[1]University of Guanajuato, Comunidad de Palo Blanco s/n, C.P. 36885, Salamanca, Guanajuato, México
{garciag,pinales,selo,avina,ealvarad}@ugto.mx
[2]Universitat Politécnica de Valéncia, DSIC,
Camino de Vera s/n, 46022, Valencia, España, onaindia@dsic.upv.es
[3]Electrical Research Institute, Reforma 113, C.P. 62490, Temixco, Cuernavaca, Morelos, México, areyes@iie.org.mx

ABSTRACT
In this paper we propose the combination of accelerated variants of value iteration mixed with improved prioritized sweeping for the fast solution of stochastic shortest-path Markov decision processes. Value iteration is a classical algorithm for solving Markov decision processes, but this algorithm and its variants are quite slow for solving considerably large problems. In order to improve the solution time, acceleration techniques such as asynchronous updates, prioritization and prioritized sweeping have been explored in this paper. A topological reordering algorithm was also compared with static reordering. Experimental results obtained on finite state and action-space stochastic shortest-path problems show that our approach achieves a considerable reduction in the solution time with respect to the tested variants of value iteration. For instance, the experiments showed in one test a reduction of 5.7 times with respect to value iteration with asynchronous updates.

Keywords: Markov decision processes, acceleration techniques, prioritization.

RESUMEN
En este documento proponemos la combinación de variantes aceleradas del algoritmo de iteración de valor combinadas con el algoritmo de barrido priorizado mejorado para la rápida solución de los procesos de decisión de Markov de ruta estocástica más corta. Iteración de valor es un algoritmo clásico para resolver a los procesos de decisión de Markov, pero este algoritmo y sus variantes son lentos para resolver problemas considerablemente grandes. Con el objeto de mejorar el tiempo de solución de este algoritmo, en este documento se han explorado técnicas de aceleración tales como actualizaciones asíncronas, priorización y barrido priorizado. Un algoritmo de reordenamiento topológico también fue comparado con uno de reordenamiento estático. Los resultados experimentales obtenidos en un problema de ruta estocástica más corta con espacios de estados-acciones finitos; muestran que nuestro enfoque logra una considerable reducción en el tiempo de solución con respecto a las variantes de iteración de valor probadas. Por ejemplo, los experimentos mostraron en una prueba una reducción de 5.7 veces con respecto a iteración de valor usando actualizaciones asíncronas.

1. Introduction

In planning under uncertainty, the planner's objective is to find a policy that optimizes some expected utility. Most approaches for finding such policies are based on decision-theoretic planning [1]. Despite their general applicability and mathematical soundness, the task of generating optimal policies for large problems is computationally challenging. Markov decision processes (MDPs) [2, 3] have successfully solved decision problems in process control, decision analysis and economy. Fast solution of MDPs is necessary for real-world applications such as process control, where many variables may change unexpectedly because of the operation of devices (valves, equipment switches, etc.) or the occurrence of exogenous (uncontrollable) events, resulting in changes of the parameters of the MDP model.

However, the computational complexity of those processes is significant for the case of continuous or high dimensionality domains, making an intractable solution time for very large problems [4]. In order to address this issue, many general MDP-based solution methods have been developed: state abstraction and aggregation techniques, feature extraction methods, value-function approximations, heuristic and greedy search, simulation-based techniques, envelope-based methods and prioritization-based methods. State aggregation and abstraction techniques reduce the search space by grouping similar states [5, 6, 7]. For instance, the search space can be partitioned based on a reward function. Feature extraction-based methods combine dynamic programming with compact representations that involve an arbitrarily complex feature extraction stage [8]. In the value function approximations approach, a dynamic programming cost-to-go function can be fitted by a linear combination of pre-selected basis functions [9]. In heuristic and greedy search, a state is labeled as solved when the heuristic values, and the greedy policy defined by them, have converged over that state [10, 11]. Simulation-based techniques use an adaptive sampling algorithm for approximating the optimal value for a finite horizon MDP [12]. In envelope-based methods, world dynamics can be represented by a compact set of rules related with an envelope of states [13]. For instance, rules can be logical sentences, whose STRIP scheme contains the action name, precondition and a set of probabilistic effects [14]. Priority-based methods can reduce the number of needed backups considerably by prioritizing backup operations, but they may incur in excessive overhead, and may fail to scale to general MDPs. Improved prioritized sweeping (IPS) [15] is a priority-based method that was originally conceived as an extension of the classical Dijkstra's algorithm for the solution of stochastic shortest-path MDPs (with positive rewards and a single goal). IPS reduces to Dijkstra's algorithm for the deterministic case (acyclic MDPs). In addition, IPS is a single-pass algorithm, which means that it is one of the fastest algorithms for the deterministic case. Unfortunately, for the case of stochastic shortest-path MDPs, the convergence of IPS is not guaranteed [16]. Another priority-based method is improved topological value iteration (iTVI) [17]; it performs backups in a static order which can be computed by using the Dijkstra's algorithm. However, the topological order may not be the optimal update order besides that the computation of the topological order may result in a considerable overhead.

In this paper we explore a different approach based on the combination of priority-based methods and value iteration improved with acceleration techniques for the solution of large MDPs. First, we code all the state transitions with non-zero probability as a list of transitions consisting of initial state, next state, action and the corresponding transition probability. Then, we study the combination of state-of-the-art acceleration techniques for solving the resulting MDP. Next, we study the combination of a prioritized method for accelerating a variant of value iteration.

This paper is organized as follows: We begin with a brief introduction to MDPs, followed by a description of a classical algorithm for solving MDPs (value iteration) and its improvements. Next, we present the formulation of MDPs in terms of a list of non-null transitions, followed by a description of acceleration techniques and their mixed forms. Finally, experimental results and conclusions are presented.

2. Markov decision processes

MDPs provide a mathematical framework for modeling sequential decision problems under uncertainty dynamic environments [18].

Formally, a MDP is a four-tuple (S, A, T, R), where S is a finite set of states $\{s_1, \ldots, s_n\}$, A is a finite set of actions $\{a_1, \ldots, a_n\}$, $T : S \times A \times S \to [0,1]$ is the state transition function. So, the probability to achieve the state s', if one applies the action a in the state s, is given by $T(a, s, s')$. $R : S \times A$ is the reward function and $R(s, a)$ is the reward obtained if one operates the action a in the state s. A policy is defined to be a function $\pi : S \to A$. The problem is to find a policy π to maximize the expected total reward. The expected utility of a policy π for an initial state s can be written as

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

where $\gamma \in [0,1]$ is a discount factor, which may be used for decreasing exponentially future rewards. For the case of discounted rewards $(0 < \gamma < 1)$, the utility of an infinite state sequence is always finite. So, the discount factor expresses that future rewards have less value than current rewards [19]. For the case of additive rewards $(\gamma = 1)$ and infinite horizon, the expected total reward may be infinite and the agent must be guaranteed to end up in a terminal state. Value functions $U(s)$ are used to represent the expected reward of policies. So the optimal policy π^* is a one that maximizes its value function.

The optimal value function is given by

$$U^*(s) = \max_{\pi} U^{\pi}(s) \quad (2)$$

It is well known that the optimal value function $U_t^*(s)$ in stage t satisfies the Bellman equation:

$$U_t^*(s) = \max_{a}\left\{R(s,a) + \gamma \sum_{s'} T(s,a,s') U_{t+1}^*(s')\right\} \quad (3)$$

Value iteration, policy iteration and linear programming are three of the most well-known techniques for finding the optimal value function $U^*(s)$ and the optimal policy π^* for infinite horizon problems [20].

Last, for the case of large MDPs with sparse transition matrices, memory savings can be obtained by using *sparse* representations [21] in which all the state transitions with non-zero probability are coded. In this way, it is possible to handle larger problems than those that can be solved otherwise (mainly in highly sparse MDPs).

3. Acceleration techniques for value iteration

Policy iteration and linear programming are computationally expensive techniques when dealing with problems with large state spaces. This is mainly because both require the solution (in each iteration) of a linear system of the same size as the state space. In contrast, value iteration avoids this problem by using a recursive approach from dynamic programming.

Starting from an initial value function, value iteration applies successive updates to the value function for each $s \in S$ by using:

$$\hat{U}(s) = \max_{a}\left\{R(s,a) + \gamma \sum_{s'} T(s,a,s') U(s')\right\} \quad (4)$$

Let $\{U_n | n = 0,1,\ldots\}$ be the sequence of value functions obtained by value iteration. Then, it can be shown that every value function satisfies $|U_n - U^*| \le \gamma^n |U_0 - U^*|$. Thus, by using the Banach fixed point theorem [22], it can be inferred that value iteration converges to the optimal value function U^*. The power of value iteration (for the solution of large-scale MDP problems) comes from the fact that the value functions obtained can be used as bounds for the optimal value function [23].

The computational complexity of one update of value iteration is $O(|n_s|^2 |n_a|)$, where n_s is the number of states and n_a is the number of actions. However, the number of required iterations can be very large. Fortunately, it has been shown in [24] that an upper bound for the number of iterations (n_{it}) required by value iteration to reach an ε-optimal solution is given by

$$n_{it} \le \frac{b + \log\left(\frac{1}{\varepsilon}\right) + \log\left(\frac{1}{1-\gamma}\right) + 1}{1-\gamma} \quad (5)$$

where $0 < \gamma < 1$, b is the number of bits used to encode rewards and state transition probabilities, and is the approximation error. *The Bellman error* is given by

$$B_t(s) = \max_{a \in A} \left\{ R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') U_t(s') \right\} - U_t(s) \quad (6)$$

The convergence of value iteration may be quite slow for γ close to one. For this reason, several improvements to value iteration have been proposed. For instance, common techniques may improve the convergence rate, reduce the time taken per iteration and/or use better stopping criteria.

One of the easiest ways to improve the convergence rate is to update the value functions as soon as they become available (also known as asynchronous updates). For instance, Gauss-Seidel value iteration uses the following update equation (7).

It is well known that policy iteration converges in less number of iterations than value iteration does, but requires solving a system of linear equations for each iteration. Value iteration is slower than policy iteration but it does not require the solution of any linear system of equations. A combined approach (modified policy iteration) can exploit the advantages of both. In this way, modified policy iteration uses a partial policy evaluation step based on value iteration. Another method to reduce the time taken per iteration is to identify and eliminate suboptimal actions [25]. For instance, bounds of the optimal value function can be used to eliminate suboptimal actions. The advantage of this approach is that the action set is progressively reduced with the consequent reduction in time. Otherwise, the number of iterations can be slightly reduced by using improved stopping criteria based on tighter bounds of the Bellman error (see Equation 6). For instance, a stopping criterion would be to stop value iteration when the span of the Bellman error falls below a certain threshold. Last, other way of improving the convergence rate as well as the iteration time is prioritization. This approach is based on the observation that, in each iteration, the value function usually changes only for a reduced set of states. Thus, by restricting the computation to only those states, a reduction of the iteration time is expected. It has been outlined that for acyclic problems the ordering of the states such that the transition matrix becomes triangular may result in a significant reduction in time.

4. Mixing acceleration techniques

For the fast solution of stochastic shortest-path MDPs, we have modified the classical value iteration (VI) in terms of a list of non-null transitions of the form (s_k, s'_k, a_k, p_k) where s_k is the initial state, s'_k is the final state, a_k is the applied action and p_k is the transition probability. Let $L = \{l_k | l_k = (s_k, s'_k, a_k, p_k)\}$ be the set of transitions, R be the state rewards, n be the number of states, γ be the discount factor and epsilon be the maximum error. For the following, we assume that all transitions in L are stored such that they are in ascending order of their initial state. The resulting value iteration algorithm is shown in Algorithm 1.

$$U^t(s) = \max_a \left\{ R(s,a) + \gamma \sum_{s'<s} T(s,a,s') U^t(s') + \gamma \sum_{s' \geq s} T(s,a,s') U^{t-1}(s') \right\} \quad (7)$$

Algorithm 1. Value iteration.

function $(S, A, R, L, \gamma, \varepsilon)$

$(\forall s \in S) U^0(s) = \max_a R(s,a)$

repeat

$(\forall s \in S, a \in A) J(s,a) = R(s,a)$

for $k = 1$ to $|L|$ do

$s = l_k.s$

$s' = l_k.s'$

$a = l_k.a$

$p = l_k.p$

$J(s,a) = J(s,a) + \gamma p U^{t-1}(s')$

end for

$(\forall s \in S) U^t(s) = \max_a J(s,a)$

until $|U^t - U^{t-1}| \leq \varepsilon$

$(\forall s \in S) \pi(s) = \arg\max_a J(s,a)$

return π

Even though we have not applied yet any accelerating methods (as those shown in a previous section) in this first algorithm, we expect this approach to be faster than the classic value iteration. In order to improve this algorithm, we have formulated variants of VI by the application of state-of-the-art acceleration techniques such as asynchronous updates, prioritization and improved prioritized sweeping.

The first variant of VI (called SVI) is almost the same as Algorithm 1 but it updates asynchronously the value function. The second variant of VI (called SVI2), shown in Algorithm 2 is almost the same as SVI but it updates asynchronously the states that change the value function between iterations.

Algorithm 2. SVI2. Asynchronous VI with updates of only those states that change between iterations and improved termination criterion.

function $(S, A, R, L, \gamma, \varepsilon)$

$(\forall s \in S)U(s) = \max_a R(s,a)$

$(\forall s \in S)B(s) = U(s)$

$changed = \{s | B(s) > \varepsilon\}$

repeat

for all $s \in$ changed do

$(\forall a \in A)J(s,a) = \sum_{k|l_k.s=s, l_k.a=a} l_k.pU(l_k.s')$

$B(s) = \max_a \{R(s,a) + \gamma J(s,a)\} - U(s)$

$U(s) = B(s) + U(s)$

end for

for all $s \in \text{neighbors}(\text{changed}) - \text{changed}$ do

$J(s,a) = \sum_{k|l_k.s=s, l_k.a=a} l_k.pU(l_k.s')$

$B(s) = \max_a \{R(s,a) + \gamma J(s,a)\} - U(s)$

$U(s) = B(s) + U(s)$

end for

$\text{changed} = \{s | s \in \text{changed} \cup \text{neighbors}(s), B(s) > \varepsilon\}$

until $\max_{s \in \text{changed}} |B(s)| \leq \varepsilon$

$(\forall s \in S)\pi(s) = \arg\max_a \{R(s,a) + \gamma J(s,a)\}$

return π

The third variant of VI (called SVI3), shown in Algorithm 3, also updates asynchronously the value function but it updates only those states (as well as their neighbors) whose value function changed in the previous iteration and it uses a static reordering of the states in decreasing order of maximum reward. This is because it is better to use a good static ordering instead of a random ordering. Thus, state reordering is performed only once (during initialization). In this case, sorting is performed using the sort method of the Array *Java class*, which has a complexity of $O(n \log n)$. Note that variable reordering is only effective when using asynchronous updates.

Algorithm 3. SVI3. Asynchronous VI with updates of only those states that change between iterations, improved static ordering of states and improved termination criterion.

function $(S, A, R, L, \gamma, \varepsilon)$

$sort(R, L)$ // compute static ordering

$(\forall s \in S) U(s) = \max_a R(s,a)$

$(\forall s \in S) B(s) = U(s)$

$changed = \{s | B(s) > \varepsilon\}$

repeat

for all $s \in$ changed do

$(\forall a \in A) J(s,a) = \sum_{k|l_k.s=s, l_k.a=a} l_k.pU(l_k.s')$

$B(s) = \max_a \{R(s,a) + \gamma J(s,a)\} - U(s)$

$U(s) = B(s) + U(s)$

end for

for all $s \in \text{neighbors}(\text{changed}) - \text{changed}$ do

$J(s,a) = \sum_{k|l_k.s=s, l_k.a=a} l_k.pU(l_k.s')$

$B(s) = \max_a \{R(s,a) + \gamma J(s,a)\} - U(s)$

$U(s) = B(s) + U(s)$

end for

$\text{changed} = \{s | s \in \text{changed} \cup \text{neighbors}(s), B(s) > \varepsilon\}$

until $\max_{s \in \text{changed}} |B(s)| \leq \varepsilon$

$(\forall s \in S) \pi(s) = \arg\max_a \{R(s,a) + \gamma J(s,a)\}$

return π

The forth variant of VI (called SVI4) uses the same acceleration techniques as SVI3 but it uses a different static ordering obtained by means of the modified topological ordering algorithm proposed in [25]. For special cases of acyclic MDPs, the use of a topological ordering on the states can yield an optimal ordering of states. For other cases, a good possibility could be to reorder the states to make the transition matrix "nearly triangular". Unfortunately, real world problems involve highly cyclic MDPs and obtaining a topological ordering is nearly impossible.

Last, the proposed algorithm (SVI5), shown in Algorithm 4, is the same as SVI3 but it computes a static ordering by using a multi-goal state/multi-start state version of IPS. Since IPS is very fast, we also profit to obtain an initial value function (usually suboptimal [26]) by means of IPS and then we obtain the optimal policy by means of SVI3. The advantage of using IPS for computing a static ordering is its speed (because IPS is a single-pass algorithm).

5. Experiments

For evaluating our approach we chose the sailing domain [27] which involves highly cyclic MDPs. This is a finite state and action-space stochastic shortest path problem (*e.g.* a race), where a sailboat has to find the shortest path between two points of a lake under fluctuating wind conditions.

The details of the problem are as follows: the sailboat's position is represented as a pair of coordinates on a grid of finite size. The controller has eight actions giving the direction to a neighboring grid position. Each action has a cost (required time) depending on the direction of the action and the wind. For the action whose direction is just the opposite of the direction of the wind, the cost must be high. For example, if the wind is at 45 degrees measured from the boat's heading, we say that the boat is on *an upwind* tack. On such a tack, it takes four seconds to sail from one waypoint to one of the nearest neighbors. But, if the wind is at 90 degrees from the boat's heading, the boat moves faster through the water and can reach the next waypoint in only three seconds. Such a tack is called a *crosswind* tack. If the wind is a quartering tailwind, we say that the boat is on a downwind tack; such a tack takes two seconds. Finally, if the boat is sailing directly *downwind*, we say that it is on an *away* tack (only one second is required). Otherwise, if the wind is hitting the left side of the sails we say that the boat is on a *port* tack. If the wind is on the right-hand side, we say that it is a *starboard* tack. If the boat is heading directly into the wind or directly away from the wind, then it is on neither a starboard nor a port tack. When changing from a port to a starboard tack (or vice versa), we assume that our sailor wastes three seconds *(delay*) for every such change of tack. To keep our model simple, we assume that the wind intensity is constant but its direction can change at any time. The wind could come from one of three directions: either from the same direction as the old wind or from 45 degrees to the left or to the right of the old wind. Table 1 shows the probabilities of a change of wind direction.

Each current state s comprises a position of the boat (x, y), a tack $t \in \{0,1,2\}$ and a current wind direction $w \in \{0,1,\ldots,7\}$. When the heading is along one of the diagonal directions, the time is multiplied by $\sqrt{2}$ to account for the somewhat longer distance that must be traveled.

All the experiments were performed on a 2.66 GHz Pentium D computer with 2 GB RAM. All algorithms were implemented in the *Java* language under a robotic planning environment [28]. The initial and

	N	NE	E	SE	S	SW	W	NW
N	0.4	0.3						0.3
NE	0.4	0.3	0.3					
E		0.4	0.3	0.3				
SE			0.4	0.3	0.3			
S				0.4	0.2	0.4		
SW					0.3	0.3	0.4	
W						0.3	0.3	0.4
NW	0.4						0.3	0.3

Table 1. Probabilities of change of wind direction. First column indicates old wind direction and first row indicates new wind direction.

maximum size of the stack of the *Java* virtual machine was set to 800 MB and 1536 MB, respectively.

For all the experiments, we set $\varepsilon = 10^{-7}$ and $\gamma = 1$. The last value is due that we are dealing with an undiscounted MDP where convergence is not guaranteed by the Banach fixed point theorem and the bound of the number of iterations given by Equation (5) no longer holds. Fortunately, the presence of absorbing states (states with null reward and 100% probability of staying in the same state) allow the algorithm to converge [29].

The lake size was varied from 55296 to 940896 states. We repeated each run ten tmes and then we calculated the average and standard deviation of the solution time.

6. Results

Figure 1 shows the solution time as a function of the number of states for all the tested algorithms. We can see that SVI5 (the one that obtains an initial value function by means of IPS) is significantly faster than the other algorithms. We can see in Table 2 that, for 940896 states, SVI5 was 2.3 times faster than SVI3, 4 times faster than SVI2, 5.7 times faster than SVI, 7.2 times faster than VDP and 15.5 times faster than SVI4. As we can see, iTVI was not tested for more than 400000 states because of its high memory requirements. Even the first variant of VI (SVI) is approximately 1.5 times faster than VDP.

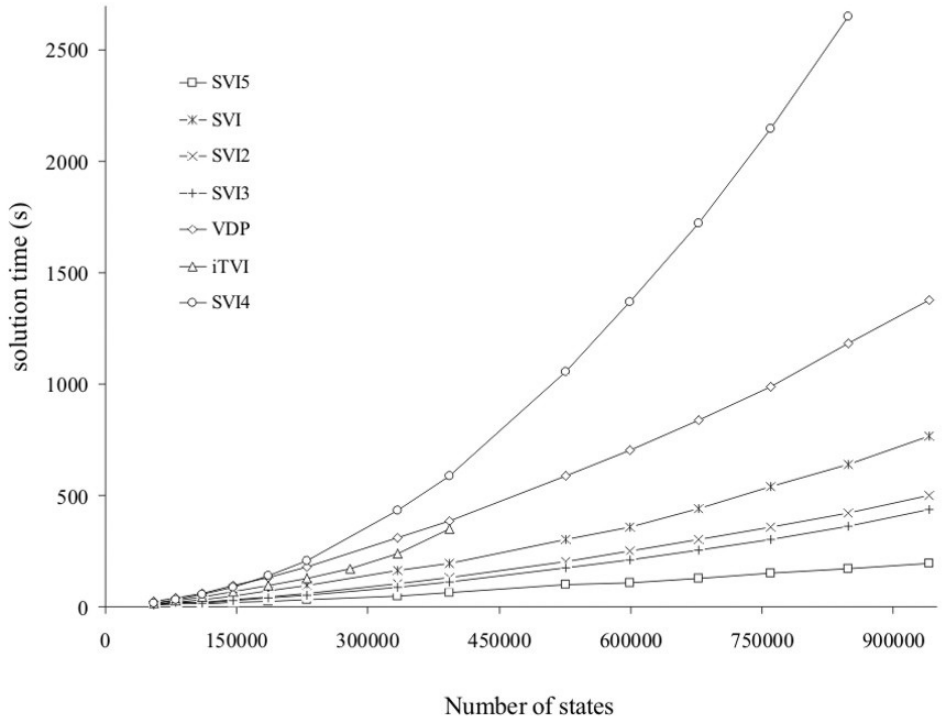

Figure 1. Solution time as a function of the number of states for all the algorithms tested. iTVI was not tested for more than 400000 states because it exhausted the memory resources.

Algorithm	Solution time (ms)	Relative solution time
SVI5	192891.0	1.0
SVI3	436302.9	2.3
SVI2	766124.9	4.0
SVI	1095706.3	5.7
VDP	1376572.0	7.2
SVI4	2980789.0	15.5
iTVI	-	-

Table 2. Summary of results in terms of solution time for all the algorithms tested (the number of states was 940896). Relative solution times are calculated with respect to the solution time of SVI5.

In the same figure we can see that SVI3 is significantly faster than SVI4, even when they differ only in the way states are sorted. This shows clearly that the modified topological reordering algorithm used by SVI4 was very slow in comparison with the ordering algorithm used by SVI3 (in decreasing order of maximum reward). In this case, the use of topological reordering does not reflect in a better solution time because of the high overhead incurred to find the topological ordering. An alternative to this modified topological ordering algorithm is to remove the smallest set of transitions that render the MDP acyclic (also known as feedback arc set problem) and to use linear complexity algorithms for acyclic graphs based on depth-first search. Unfortunately, it turns out that the feedback arc set problem is known to be NP-complete [30]. Another possibility to find a topological ordering would be to apply a strongly connected components algorithm as in [16]. Anyway, preliminary results obtained in an experiment in which we used a strongly connected component algorithm indicated that the maximum reward reordering was still faster.

In all cases, SVI5 yielded the smallest solution time. This implicates that, at least in the sailing strategies problem, the combination of acceleration techniques and IPS can result in a very fast algorithm. Other experiments in the sailing strategies problem (stochastic shortest-path problem) indicated that the use of prioritization (excepting static ordering in decreasing value of maximum reward) resulted in excessive overhead. This may be caused by the cyclic nature of the resulting MDPs as shown in [25] for the SysAdmin problem.

It is well known that the computational complexity of value iteration and variants can be computed as the complexity of an update of one state multiplied by the number of updates performed until convergence. Since the complexity of an update of one state was the same for all the tested algorithms (e.g. it is equal to the number of actions), it is possible to make a comparison by using only the number of updates performed by each algorithm to reach convergence. So, we performed a series of experiments aimed to compute different the number of updates performed by the different algorithms to reach convergence.

Figure 2 shows the number of updates performed as a function of the number of states (n) by the different algorithms. We can see that SVI5 performs the lowest number of updates whereas VDP performs the highest number up updates. We can also see that SVI, SVI2 and SVI4 perform the same number of updates. It seems that the only difference among these algorithms is their overhead. We also see that the curve for iTVI and SVI3 overlaps. However, iTVI demands a lot more memory and has a higher overhead than SVI3. The number of updates performed by SVI5 is equal to $n^{1.35}$, for SVI3 it was $n^{1.45}$, for SVI, SVI2 and SVI4 it was $n^{1.46}$, and for VDP it was $n^{1.47}$. Thus, even without taking into account the overheads, SVI5 had the lowest computational complexity.

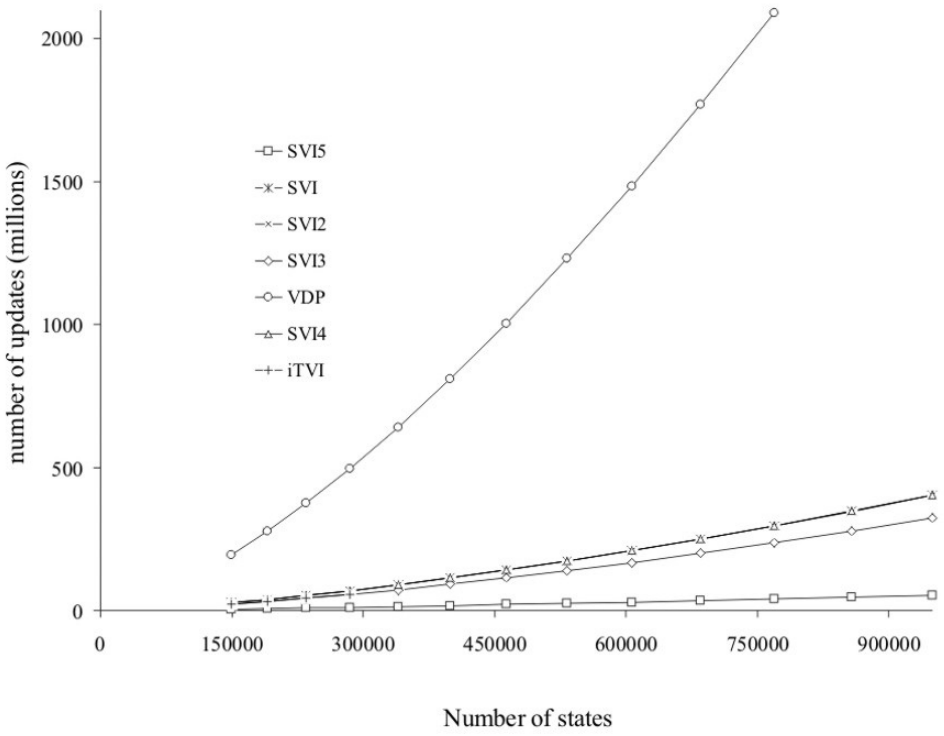

Figure 2. Number of updates of one state as a function of the number of states for all the algorithms tested. Let us note that the curves for SVI, SVI2 and SVI4 are overlapped and iTVI was not tested for more than 400000 states because it exhausted the memory resources.

Table III shows a comparison in terms of the number of updates for the different algorithms tested when the number of states was 940896. As we can see, SVI5 performed the lowest number of updates whereas SVI3 performed the second lowest number of updates and VDP performed the highest number of updates. On the other hand, we can see that SVI5 required 6 times less updates than SVI3, 7.5 times less updates than SVI, SVI2 and SVI4, 28 times less updates than VDP.

Algorithm	Number of updates	Relative number of updates
SVI5	53034256	1.0
SVI3	318022848	6.0
SVI	397999008	7.5
SVI2	397999008	7.5
SVI4	397999008	7.5
VDP	1487405248	28.0
iTVI	-	-

Table 3. Summary of results in terms of number of updates for all the algorithms tested (the number of states was 940896). Relative numbers of updates are calculated with respect to the number of updates of SVI5.

7. Conclusions

In this paper, we have successfully tested different combinations of state-of-the-art acceleration techniques such as asynchronous updates, prioritization and improved prioritized sweeping, in order to obtain the faster solution time in large MDPs. We also compared two methods using static reordering: sorting in decreasing value of maximum reward and a modified topological reordering algorithm proposed in [25]. In addition, other two algorithms were tested: the improved topological value iteration and a dynamic programming approach. In general, the use of prioritization, excepting static ordering in decreasing value of maximum reward, resulted in excessive overhead. This may be in part because of the cyclic nature of MDPs resulting from the sailing strategies problem (as shown before [25]). At least in the sailing domain, the combination of asynchronous updates and prioritization using a static reordering computed by using improved prioritized sweeping yielded the lowest solution time. The experiments showed in one test a reduction of 5.7 times with respect to value iteration with asynchronous updates and 15.5 times with respect to the variant of value iteration which uses a topological reordering. Future work will focus on the evaluation of the proposed method in real-world problems such as a power plant operator assistant [31].

References

[1] Boutilier, C., Dean, T. and Hanks, S., Decision-theoretic planning: structural assumptions and computational leverage, Journal of Artificial Intelligence Research, 11, 1999, pp 1-94.

[2] Bellman, R. E., The theory of dynamic programming, Bull. Amer. Math. Soc., 60, 1954, pp 503-516.

[3] Puterman, M. L., Markov Decision Processes, Wiley Editors, New York, USA, 1994.

[4] Kuter, U., Hu, J., Computing and Using Lower and Upper Bounds for Action Elimination in MDP Planning, Proceedings of the Symposium on Abstraction, Reformulation and Approximation, SARA, 2007.

[5] Dean, T., Kaelbling, L. P., Kirman, J. and Nicholson, A., Planning under Time Constraints in Stochastic Domains, Artificial Intelligence, 76 (1-2), July 1995, pp 35-74.

[6] Boutilier, C., Dearden, R. and Goldszmidt, M., Stochastic Dynamic Programming with Factored Representations, Artificial Intelligence, 121 (1-2), 2000, pp 49-107.

[7] Givan, R., Dean T. and Greig, M., Equivalence Notions and Model Minimization in MDPs, Artificial Intelligence, 147 (1-2), 2003, pp 163-233.

[8] Tsitsiklis, J. N. and Van Roy, B., Feature-based methods for large-scale dynamic programming, Machine Learning, 22, 1996, pp 59-94.

[9] De Farías, D. P. and Van Roy, B., The linear programming approach to approximate dynamic programming, Operations Research, 51 (6), 2003, pp 850-865.

[10] Bonet, B. and Geffner, H., Labeled RTDP: Improving the Convergence of Real-Time Dynamic Programming, International Conference on Automated Planning and Scheduling, ICAPS, 2003, pp 12-21, Trento, Italy.

[11] Hansen, E. A. and Zilberstein, S., LAO: A Heuristic Search Algorithm that finds solutions with Loops, Artificial Intelligence, 129, 2001, pp 35-62.

[12] Chang, H. S., Fu, M. C., Hu, J. and Marcus, S. I., An Adaptive sampling algorithm for solving MDPs, Operations Research, 53 (1), 2005, pp 126-139.

[13] Gardiol, N. and Kaelbling, L. P., Envelope-based Planning in Relational MDP's, Neural Information Processing Systems NIPS, 16, 2003, Vancouver, B. C.

[14] Gardiol, N., Relational Envelope-based Planning, PhD Thesis, MIT, MA, USA, February 2008.

[15] McMahan, H. B. and Gordon, G., Fast Exact Planning in Markov Decision Processes, 15th International Conference on Automated Planning and Scheduling (Monterey, CA, USA, 2005a).

[16] Dai, P. and Goldsmith, J., Topological Value Iteration Algorithm for Markov Decision Processes, 20th International Joint Conference on Artificial Intelligence, IJCAI, 2007, pp 1860-1865, Hyderabad, India

[17] Dibangoye, J. S., Chaib-draa, B., Mouaddib, A., A Novel Prioritization Technique for Solving Markov

Decision Processes, 21st International FLAIRS Conference, Association for the Advancement of Artificial Intelligence, Florida, USA, 2008.

[18] Puterman, M. L., Markov Decision Processes, Wiley Interscience Editors, New York, USA, 2005.

[19] Russell, S., Artificial Intelligence: A Modern Approach, 2nd Edition, Making Complex Decisions (Ch-17), Pearson Prentice Hill Ed., USA, 2004.

[20] Chang, I. and Soo, H., Simulation-based algorithms for Markov decision processes, Communications and Control Engineering, Springer Verlag London Limited, 2007.

[21] Agrawal, S. and Roth, D., Learning a Sparse Representation for Object Detection, Proc. 7th European Conference on Computer Vision (Copenhagen, Denmark, 2002), pp. 1-15.

[22] Kirk, W. A., Khamsi, M. A., An Introduction to Metric Spaces and Fixed Point Theory, John Wiley, New York, USA, 2001.

[23] Tijms, H. C., A First Course in Stochastic Models, Wiley Ed., Discrete-Time Markov Decision Processes (Ch-6), UK, 2003.

[24] Littman, M. L., Dean, T. L. and Kaelbling, L. P., On the Complexity of Solving Markov Decision Problems, 11th International Conference on Uncertainty in Artificial Intelligence, 1995, pp 394-402, Montreal, Quebec.

[25] Wingate, D. and Seppi, K. D., Prioritization Methods for Accelerating MDP Solvers, Journal of Machine Learning Research, 6, 2005, pp 851-881.

[26] Li, L., A Unifying Framework for Computational Reinforcement Learning Theory, PhD Thesis, The State University of New Jersey (New Brunswick, NJ, USA, October 2009).

[27] Vanderbei, R. J., Optimal Sailing Strategies, Statistics and Operations Research Program, University of Princeton, USA, (http://orfe.princeton.edu/~rvdb/sail/sail.html), 1996.

[28] Reyes, A., Ibarguengoytia, P., Sucar, L. E. and Morales, E., Abstraction and Refinement for Solving Continuous Markov Decision Processes, 3rd European Workshop on Probabilistic Graphical Models, 2006, pp 263-270, Prague, Czech Republic.

[29] Hinderer, K. and Waldmann, K. H., The critical discount factor for Finite Markovian Decision Processes with an absorbing set, Mathematical Methods of Operations Research, Springer Verlag, 57, 2003, pp 1-19.

[30] Garey, M. R. and Johnson, D. S., Computers and Intractability, A Guide to the Theory of NP-Completeness, Appendix A: List of NP-Complete Problems, W. H. Freeman, 1990.

[31] Reyes, A., Sucar, L. E., Ibargüengoytia, P., Power Plant Operator Assistant, Bayesian Modeling Applications Workshop in the 19th Conference on Uncertainty in Artificial Intelligence UAI-2003, August 2003.

Authors´ Biographies

Ma. de Guadalupe GARCÍA-HERNÁNDEZ

Dr. García-Hernández is a researcher and full-time professor in the Electronics Department at Universidad de Guanajuato. She graduated in chemical engineering from Universidad de Guanajuato in 1985 and obtained her master's degree in mechanical engineer from the same university in 1992. Dr. García-Hernández is phD candidate in form recognition and artificial intelligence from Universitat Politècnica de València, Spain. She has published papers in international and national scientific events, and in indexed and national journals. She has been a member of the Spanish Association for Artificial Intelligence since January, 2006.

José RUIZ-PINALES

Dr. Ruiz-Pinales received the BA degree in electronics and communications engineering and the MSE degree in electrical engineering from Universidad de Guanajuato (University of Guanajuato) in 1993 and 1996. He received the PhD degree in computer science from Ecole Nationale Supérieure des Télécommunications de Paris in 2001. He joined the department of Electronics Engineering of the Universidad de Guanajuato as a professor in 2001. His research interests are in computer vision and artificial intelligence including face recognition, handwriting recognition, neural network models, support vector machines, models of the human visual system, instrumentation, communications and electronics. He has coauthored more than 32 research papers.

Eva ONAINDIA

Dr. Onaindia is an assistant computer science professor at Universidad Técnica de Valencia (Technical University of Valencia). She received her Ph.D. in computer science from the Universitat Politècnica de València (Polytechnic University of Valencia) in 1997. She currently leads the group of Reasoning on Planning and Scheduling where she conducts research in temporal and classical planning, development of integrated techniques of planning and scheduling and replanning. She is currently collaborating in the Agreement Technologies project where she is working on the application of negotiation techniques to planning. She has led national research projects (MCyT, MEC) as well as sat on various scientific committees in her field (IJCAI, ICAPS, ECAI, etc.). She has published about 50 articles in specialized conferences and scientific journals related to topics of Planning in AI.

Sergio LEDESMA-OROZCO

Dr. Ledesma-Orozco got his M.S. from Universidad de Guanajuato while working on the setup of Internet in Mexico. In 2001, he got his Ph.D. from Stevens Institute of Technology in Hoboken, New Jersey. After graduating he worked for Barclays Bank as part of the IT-HR group. He has worked as a software engineer for several years, and he is the creator of the software Neural Lab and Wintempla. Neural Lab offers a graphical interface to create and simulate artificial neural networks. Neural Lab is free, and the latest version can be downloaded from Wikipedia. Wintempla provides a thin layer of encapsulation to ease program design and implementation for business and research. Currently, he is a research professor at Universidad de Guanajuato in Mexico. His areas of interests are artificial intelligence and software engineering.

J. Gabriel AVINA-CERVANTES

Dr. Avina-Cervantes received the engineering degree in electronics and communications from Universidad de Guanajuato in 1998, the M.S. degree in electrical engineering from the same university in 1999, the Ph.D. in informatics and telecommunications from the Institut National Polytechnique de Toulouse and the LAAS-CNRS, France, in 2005. His research interests include artificial vision for outdoor robotics, pattern recognition, object classification and image processing. He is currently a researcher and a full-time professor in the Engineering Division of Universidad de Guanajuato, Campus Irapuato-Salamanca.

E. ALVARADO-MÉNDEZ

Dr. Alvarado-Méndez is a professor in the Electronics Department of Universidad de Guanajuato. He received his Ph.D. in sciences in 1997. He currently leads the group of Optoelectronics where he conducts research on nonlinear optics, optical sensors, fiber optics, tweezers optics and optical image processing techniques. He has published about 25 articles and 70 specialized conferences related to those topics.

Alberto REYES-BALLESTEROS

Dr. Reyes-Ballesteros is a researcher at Instituto de Investigaciones Eléctricas (Electrical Research Institute), IIE, in Mexico and a part-time professor at Instituto Tecnológico y de Estudios Superiores de Monterrey (Technological Institute of Higher Education of Monterrey), ITESM, campus Mexico City. His research interests include decision-theoretic planning and machine learning, and their applications in robotics and industrial processes. He received a PhD in computer science from ITESM, campus Cuernavaca, and he was involved in a postdoctoral program at the Instituto Superior Técnico (Technical Higher Education Institute), IST, in Lisbon. He is a member of the National Research System in Mexico.