Acceleration of association-rule based markov decision processes

Ma. de G. García-Hernández*[1], J. Ruiz-Pinales[2], A. Reyes-Ballesteros[3], E. Onaindía[4], J. Gabriel Aviña-Cervantes[5], S. Ledesma[6]

[1,2,5,6] Universidad de Guanajuato, Comunidad de Palo Blanco s/n, C.P. 36885, Salamanca, Guanajuato, México {garciag,pinales,avina,selo}@salamanca.ugto.mx
[3] Instituto de Investigaciones Eléctricas, Reforma 113, C.P. 62490, Temixco, Morelos, México, areyes@iie.org.mx
[4] Universidad Politécnica de Valencia, DSIC, Camino de Vera s/n, 46022, Valencia, España, onaindia@dsic.upv.es

ABSTRACT
In this paper, we present a new approach for the estimation of Markov decision processes based on efficient association rule mining techniques such as Apriori. For the fastest solution of the resulting association-rule based Markov decision process, several accelerating procedures such as asynchronous updates and prioritization using a static ordering have been applied. A new criterion for state reordering in decreasing order of maximum reward is also compared with a modified topological reordering algorithm. Experimental results obtained on a finite state and action-space stochastic shortest path problem demonstrate the feasibility of the new approach.

Keywords: Markov decision processes, association rules, acceleration procedures.

RESUMEN
En este documento se presenta un nuevo enfoque para la estimación de procesos de decisión de Markov basado en técnicas eficientes de minería de reglas de asociación tal como Apriori. Para la más rápida solución del resultante proceso de decisión de Markov basado en reglas de asociación, han sido aplicados varios procedimientos de aceleración tales como actualización asíncrona y priorización usando reordenamiento estático. Un nuevo criterio para el reordenamiento de estados es también comparado con un algoritmo modificado de reordenamiento topológico. Los resultados experimentales obtenidos en un problema estocástico de ruta más corta, con un número finito de acciones y estados, demuestran la viabilidad del nuevo enfoque.

Palabras clave: Procesos de decisión de Markov,reglas de asociación, procesos de aceleración.

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, 2, 3]. Despite their general applicability and mathematical soundness, the task of generating optimal policies for large problems is computationally challenging. For instance, in a real-world process control problem many variables change dynamically because of the operation of devices (valves, equipment switches, etc.) or the occurrence of exogenous (uncontrollable) events. If the control system does not consider the possibility of fault, then it will surely not make intelligent actions in the event of a fault occurrence. This problem is very complex and uncertainty plays an important role during the search for solutions.

Since the addition of new capabilities to a planner, heuristic search has shown limitations for the case of non integer data as well as in the use of additive graphs in the solution of real world problems [4]; thus they are frequently solved by using Bayesian representations and inference [5], or Markov decision processes (MDPs) [2]. The latter have successfully solved decision problems in process control, decision analysis and economy. However, the computational complexity of those processes is a significant one for the case of continuous or high dimensionality domains, making an intractable solution time for very large problems [6]. The different approaches can be broadly classified as: state abstraction and aggregation techniques [7, 8, 9], feature extraction methods [10], value function approximations [11], heuristic and greedy search [12, 13], simulation-based techniques [14] and envelope-based methods [15]. State aggregation and abstraction techniques reduce the search space by grouping similar states [8]. 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 [10]. In value function approximations, the dynamic programming cost-to-go function can be fitted by a linear combination of pre-selected basis functions [11]. 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 [12]. Simulation based techniques use an adaptive sampling algorithm for approximating the optimal value for a finite horizon MDP [14]. In envelope based methods, world dynamics can be represented by a compact set of rules related with an envelope of states [15]. For instance, rules can be logical sentences, whose STRIP scheme contains the action name, precondition and a set of probabilistic effects [16].

In this paper, we explore a different approach for the solution of problems involving large state spaces by means of MDPs. First, we propose a method for the estimation of MDPs based on efficient association rule mining techniques. Then, we study the application of state-of-the-art acceleration techniques to the resulting association-rule-based MDP and provide an improved static ordering technique.

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, a brief survey of association rules mining is introduced, followed by a description of the new approach to the estimation and solution of MDPs and, finally, results and conclusions are presented.

2. Markov decision processes

Markov decision processes or MDPs provide a mathematical framework for modeling sequential decision problems in uncertain dynamic environments [17, 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, which takes an action, the current state and the next state and gives the probability of this transition. The transition-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 often defined to be a function $\pi : S \to A$, which yields an action for each state. The problem is to find a policy π to maximize the expected total reward [18]. The value function of a policy π and an initial state s is given by:

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

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.

The optimal value function is given by [18]

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

It is well known that the optimal value function $U_t^*(s)$ in stage t is given by the Bellman equation [2, 18]:

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

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].

3. Value iteration

Policy iteration and linear programming are computationally expensive techniques when dealing with problems with large state spaces. Mainly because they 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 [20].

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

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

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 $|U_n - U^*| \leq \gamma^n |U_0 - U^*|$ Thus, by using the Banach fixed point theorem, 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 [21].

The computational complexity of one update of value iteration is $O(|S|^2|A|)$. However, the number of required iterations can be very large. Fortunately, it has been shown in [22] that an upper bound for the number of iterations required by value iteration to reach an e-optimal solution is given by

$$n_{it} \leq \frac{B + \log\left(\frac{1}{\varepsilon}\right) + \log\left(\frac{1}{1-\gamma}\right) + 1}{1-\gamma} \tag{5}$$

where $0 < \gamma < 1$, B is the number of bits used to encode rewards and state transition probabilities, and e is the threshold of the *Bellman error* [18]:

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

The convergence of value iteration may be quite slow for g close to one. For this reason, several improvements to value iteration have been proposed [18]. 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 [18]:

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

It is well known that policy iteration converges in less 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 [18].

Other ways of improving the convergence rate as well as iteration time are prioritization and partitioning [23]. 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 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. Last, there exists another method that uses heuristics for prioritizing backups that do not require a priority queue [24].

Also, another method to reduce the time taken per iteration is to identify and eliminate suboptimal actions [18]. 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.

Last, the number of iterations can be slightly reduced by using improved stopping criteria based on tighter bounds of the Bellman error [18]. For instance, a stopping criterion would be to stop value iteration when the span of the Bellman error falls below a certain threshold.

4. Association rules

Association rules [25] represent an important tool in data mining applications. An association rule is a rule of the form $X \Rightarrow Y$, where X and Y are disjoint sets of items *(itemsets).* This implicates that if we find all *items* in X, it is likely that we also find all the items in Y.

A typical application of mining association rules is to discover associations between articles in market basket analysis [26]. These associations can offer useful information to retail managers for article collection decisions [27], personalized article recommendations [28], and implementation of promotional activities [29].

Association rules are usually selected from the set of all possible rules using measures of statistical significance and interestingness. Support is a measure of significance, which is defined as the percentage of instances that contain all the items in a rule [25] and it is given by

$$\text{supp}(X \Rightarrow Y) = \text{supp}(X \cup Y) = \frac{|X \cup Y|}{m} \qquad (8)$$

where $|X \cup Y|$ represents the number of instances that contain all the items in X or Y, and m is the number of instances in a specific database. Consequently, a minimum support threshold is used to select the most frequent (and hopefully the most important) item-sets [26].

Confidence is a measure of interestingness and it represents the maximum percentage of instances covered by a rule [25, 26]:

$$\text{conf}(X \Rightarrow Y) = \frac{\text{supp}(X \cup Y)}{\text{supp}(X)} = P(Y|X) \qquad (9)$$

Apriori is one of the most well known algorithms for mining frequent association rules in a database [25]. It exploits the property that any subset of a large item set is also large. Apriori (see Algorithm 2) starts by counting item occurrences in order to find the most frequent itemsets L_1. The subsequent passes, say pass k, consist of two phases. In the first phase, the frequent itemsets L_{k-1} found in the previous pass are used to generate the candidate itemsets C_k, using the Apriori candidate generation procedure. Next, the database is scanned and the support of candidates in C_k is counted. The set of candidate itemsets is subjected to a pruning process to ensure that all the subsets of the candidate sets are already known to be frequent itemsets. The intuition behind the candidate-generation procedure is that if an itemset X has minimum support, so do all subsets of X. The pruning step eliminates the extensions of $(k-1)$ -itemsets which are not found to be frequent. The Apriori algorithm moves downward in the lattice starting from level 1 till level k, where no candidate set remains after pruning.

Some of the advantages of Apriori are easy implementation and easy parallelization. Improved versions such as AprioriHybrid [30] have shown to scale linearly with the number of instances.

5. Our approach

Reinforcement learning is a variant of optimal control; however, optimal control involves only planning whereas reinforcement learning involves both learning (model learning) and planning. Reinforcement learning methods can be broadly classified as: direct (model-free) and indirect (model-based) methods [31]. In contrast with model-free methods which compute an optimal policy directly from experience data, model-based methods first estimate the underlying MDP and then use standard techniques, such as value iteration [3], to compute an optimal policy. An advantage of model-based methods is that they are very data efficient.

A usual method for the estimation of MDP parameters (transition probabilities and rewards) is based on maximum-likelihood. *The maximum likelihood estimator* is the value of the parameter which maximizes the likelihood of the data. Let $n(s,a)$ be the number of times the agent has

```
Function $\text{Apriori}(D, \text{minsup})$
$L_1 = \{\text{large} \cdot 1 - \text{itemsets}\}$;
for $(k = 2; L_{k-1} \neq \emptyset; k++)$ do
    $C_k = \text{apriorigen}(L_{k-1})$; // new candidates
    for all transactions $t \in D$ do
        $C_t = \text{subset}(C_k, t)$; // candidates contained in $t$
        for all candidates $c \in C_t$ do
            c.count++;
        end
    end
    $L_k = \{c \in C_k | \text{c.count} \geq \text{minsup}\}$
end
return $\bigcup_k L_k$

function $\text{apriorigen}(L_{k-1})$
insert into $C_k$
select $\text{p.item}_1, \text{p.item}_2, \ldots, \text{p.item}_{k-1}, \text{q.item}_{k-1}$
from $L_{k-1}\text{p}, L_{k-1}\text{q}$
where $\text{p.item}_1 = \text{q.item}_1, \ldots, \text{p.item}_{k-2} = \text{q.item}_{k-2}, \text{p.item}_{k-1} < \text{q.item}_{k-1}$;
for all itemsets $c \in C_k$ do
    for all $(k-1)-\text{subsets}$ $s$ of $c$ do
        if $s \notin L_{k-1}$ then
            delete $c$ from $C_k$;
    end
end
return $C_k$
```

Algorithm 1. Apriori

taken action a in state s. Let $n(s,a,s')$ be the number of times it arrives in state s' after having taken action a in state s. Let $\sum R(s,a)$ be the cumulative amount of rewards it receives when taking action a in state s. The maximum likelihood estimators of the transition probabilities and rewards are given by [32]:

$$\hat{T}(s,a,s')=\frac{n(s,a,s')}{n(s,a)} \tag{10}$$

and

$$\hat{R}(s,a)=\frac{\sum R(s,a)}{n(s,a)} \tag{11}$$

respectively.

In our approach, we start by computing, from experience data gathered by an agent, a set of association rules of the form $\{s,a\}\Rightarrow\{s'\}$ where s is a current state and s' is the resulting state after applying action a. Next, we use the resulting rules to build an Association-Rule-based Value Iteration (ARVI, see Algorithm 2) algorithm, where each decision rule is an association rule. Then, we assign to each association rule a transition probability and a reward.

One advantage of the use of association rules is that they can be computed by using efficient data mining algorithms such as Apriori [25] and FP-growth [33]. Another advantage is that the confidence measure of each rule can be used directly as transition probability. Rewards can be computed by using Equation (11) at the same time. For large datasets, the number of passes through the dataset may render some mining algorithms unfeasible. For instance, Apriori requires k (3 for our case) dataset scans whereas FP-growth only requires two. Thus, we may prefer the use of FP-growth for cases where the dataset is large. However, a sampling based algorithm (e.g., FPMax [34]) may be used to obtain successive approximations of the set of association rules. From these approximations, an approximate MDP can readily be obtained and solved at the same time the agent gathers experience data.

For the solution of the resulting association-rule-based MDP, we have modified value iteration in terms of association rules. Let $L=\{L_k | L_k=(s_k,s'_k,a_k)\}$ be the set of association rules with a given maximum support and confidence (obtained by Apriori or other association rule mining technique), R be the state rewards, T be the state transition probabilities of each rule and n be the number of states, γ be the discount factor, ε be the maximum error and $numit$ be the maximum number of iterations. Our resulting value iteration algorithm (ARVI) is shown in the next figure.
13

Even though we have not applied any accelerating methods yet (as those shown in a previous section) in this first algorithm, we expect this approach to be faster than classic value iteration (which uses an expensive 3-D transition probability matrix). In order to improve this first algorithm, we have formulated several variants of our ARVI by the application of state-of-the-art acceleration procedures such as asynchronous updates [18] and prioritization by using a static reordering of states [23]. The first variant of our ARVI

(ARVI2), shown in Algorithm (3), only uses asynchronous updates [18]. The second variant (ARVI3), shown in Algorithm (4), updates synchronously only those states (as well as their neighbors) whose value function changed in the previous iteration [23] (in order to focus computation in regions of the problem which are expected to be maximally productive, and

```
function $\mathrm{ARVI}(R, L, T, \gamma, \varepsilon, numit)$
$U^0(s) = \max_a R(s,a)$ for $s = 1,2,\ldots,n$
$t = 1$
do
    $J(s,a) = R(s,a)$ for $s = 1,2,\ldots,n$ and $a = 1,2,\ldots,m$
    for $k = 1$ to $|L|$
        $a = \mathrm{action}(L(k))$
        $s = \mathrm{initialstate}(L(k))$
        $s' = \mathrm{finalstate}(L(k))$
        $J(s,a) = J(s,a) + \gamma T(k) U^{t-1}(s')$
    end
    $U^t(s) = \max_a J(s,a)$ for $s = 1,2,\ldots,n$
    $t = t + 1$
while $t < numit$ and $\left\| U^t - U^{t-1} \right\|^2 > \varepsilon$
$\pi(s) = \arg\max_a J(s,a)$ for $s = 1,2,\ldots,n$
return $\pi$
```

Algorithm 2. ARVI (synchronous updates).

simultaneously avoid useless backups. In this case, the order of evaluation of the states is not modified. The third variant (ARVI4), shown in Algorithm (5), updates asynchronously only those states (as well as their neighbors) whose value function changed in the previous iteration [23]. The forth variant of our ARVI (ARVI5), shown in Algorithm (6), uses the same acceleration procedures as ARVI4 but 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 good dynamic ordering. Thus, state reordering is performed only once (during initialization) such that, for each sweep, they are updated in an approximately optimal order. 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 [23]. The fifth variant of our ARVI (ARVI6) uses the same acceleration procedures as ARVI5 but uses the modified topological ordering algorithm shown in Algorithm (7). The use of a priority queue for all states of the model may result in an excessive overhead. For special cases of acyclic MDPs the use of a topological sort on the states yields an optimal ordering of states. For other cases, a good possibility could be to reorder the states to make the transition matrix "nearly triangular" [23]. Another option is to compute the topological order of the states. For acyclic MDPs, at least one topological order exists; and usual algorithms for topological sorting with linear running time in the number of nodes and the number of edges can be used. Unfortunately, real world problems involve cyclic MDPs and making a topological sorting is impossible. However, in this case, a modified topological ordering method can still be used [23].

```
function $\mathrm{ARVI2}(R, L, T, \gamma, \varepsilon, numit)$
$U(s) = \max_a R(s,a)$ for $s = 1,2,\ldots,n$
$t = 1$
do
        $J(s,a) = R(s,a)$ for $s = 1,2,\ldots,n$ and $a = 1,2,\ldots,m$
        $mxerr = 0$
        $k = 1$
        do
                $s = \mathrm{initialstate}(L(k))$
                while $k \le |L|$ and $\mathrm{initialstate}(L(k)) = s$
                        $s' = \mathrm{finalstate}(L(k))$
                        $a = \mathrm{action}(L(k))$
                        $J(s,a) = J(s,a) + \gamma T(k) U(s')$
                        $k = k + 1$
                end
                $mx = \max_a J(s,a)$
                $err = |mx - U(s)|$
                $mxerr = \max\{err, mxerr\}$
                $U(s) = mx$
        while $k \le |L|$
        $t = t + 1$
while $t < numit$ and $mxerr > \varepsilon$
$\pi(s) = \arg\max_a J(s,a)$ for $s = 1,2,\ldots,n$
return $\pi$
```

Algorithm 3. ARVI2 (asynchronous updates and improved termination criterion).

```
function $\mathrm{ARVI3}(R, L, T, \gamma, \varepsilon, numit)$
$U^0(s) = \max_a R(s,a)$ for $s = 1,2,\ldots,n$
$B^0(s) = \left|U^0(s)\right|$ for $s = 1,2,\ldots,n$
$\text{changed} = \{s \mid B^0(s) > \varepsilon\}$
$t = 1$
do
    for all $s \in \text{changed}$
        $J(s,a) = \sum_{k \in \text{rulesof}(s,a)} T(k) U^{t-1}(\text{finalstate}(L(k)))$
        $U^t(s) = \max_a R(s,a) + \gamma J(s,a)$
        $B^t(s) = \left|U^t(s) - U^{t-1}(s)\right|$
    end
    for all $s \in \text{neighbors}(\text{changed}) - \text{changed}$
        $J(s,a) = \sum_{k \in \text{rulesof}(s,a)} T(k) U^{t-1}(\text{finalstate}(L(k)))$
        $U^t(s) = \max_a R(s,a) + \gamma J(s,a)$
        $B^t(s) = \left|U^t(s) - U^{t-1}(s)\right|$
    end
    $\text{changed} = \{s \mid s \in \text{changed} \cup \text{neighbors}(s), B^t(s) > \varepsilon\}$
    $t = t + 1$
while $t < numit$ and $\max_s B^t(s) > \varepsilon$
$\pi(s) = \arg\max_a \{R(s,a) + \gamma J(s,a)\}$ for $s = 1,2,\ldots,n$
return $\pi$
```

Algorithm 4. ARVI3 (synchronous updates of states that change between iterations and improved termination criterion).

```
function $\mathrm{ARVI4}(R, L, T, \gamma, \varepsilon, numit)$
$U(s) = \max_a R(s,a)$ for $s = 1,2,\ldots,n$
$B(s) = U(s)$ for $s = 1,2,\ldots,n$
$\text{changed} = \{s \mid B(s) > \varepsilon\}$
$t = 1$
do
        for all $s \in \text{changed}$
                $J(s,a) = \sum_{k \in \text{rulesof}(s,a)} T(k) U(\text{finalstate}(L(k)))$
                $B(s) = \max_a \{R(s,a) + \gamma J(s,a)\} - U(s)$
                $U(s) = B(s) + U(s)$
        end
        for all $s \in \text{neighbors}(\text{changed}) - \text{changed}$
                $J(s,a) = \sum_{k \in \text{rulesof}(s,a)} T(k) U(\text{finalstate}(L(k)))$
                $B(s) = \max_a \{R(s,a) + \gamma J(s,a)\} - U(s)$
                $U(s) = B(s) + U(s)$
        end
        $\text{changed} = \{s \mid s \in \text{changed} \cup \text{neighbors}(s), B(s) > \varepsilon\}$
        $t = t + 1$
while $t < numit$ and $\max_{s \in \text{changed}} |B(s)| > \varepsilon$
$\pi(s) = \arg\max_a \{R(s,a) + \gamma J(s,a)\}$ for $s = 1,2,\ldots,n$
return $\pi$
```

Algorithm 5. ARVI4 (asynchronous updates of states that change between iterations and improved termination criterion).

```
function  $\mathrm{ARVI5}(R, L, T, \gamma, \varepsilon, numit)$
$\mathrm{staticsort}(R, L, T)$
$U(s) = \max_a R(s,a)$ for $s = 1,2,\ldots,n$
$B(s) = U(s)$ for $s = 1,2,\ldots,n$
$\mathrm{changed} = \{s \mid B(s) > \varepsilon\}$
$t = 1$
do
    for all $s \in \mathrm{changed}$
        $J(s,a) = \sum_{k \in \mathrm{rulesof}(s,a)} T(k) U(\mathrm{finalstate}(L(k)))$
        $B(s) = \max_a \{R(s,a) + \gamma J(s,a)\} - U(s)$
        $U(s) = B(s) + U(s)$
    end
    for all $s \in \mathrm{neighbors}(\mathrm{changed}) - \mathrm{changed}$
        $J(s,a) = \sum_{k \in \mathrm{rulesof}(s,a)} T(k) U(\mathrm{finalstate}(L(k)))$
        $B(s) = \max_a \{R(s,a) + \gamma J(s,a)\} - U(s)$
        $U(s) = B(s) + U(s)$
    end
    $\mathrm{changed} = \{s \mid s \in \mathrm{changed} \cup \mathrm{neighbors}(s), |B(s)| > \varepsilon\}$
    $t = t + 1$
while $t < numit$ and $\max_{s \in \mathrm{changed}} |B(s)| > \varepsilon$
$\pi(s) = \arg\max_a \{R(s,a) + \gamma J(s,a)\}$ for $s = 1,2,\ldots,n$
return $\pi$
```

Algorithm 6. ARVI5 (asynchronous updates of states that change between iterations, static ordering of states and improved termination criterion).

```
// Initialization: dc is an array representing the in-degree of each state. p is a
partition of the state space.
$dc \leftarrow 0$
for all $s \in p$ do
    for all $a \in A$ do
        for all $s' \in p$ do
        if $T(s,a,s') \neq 0$ then increment$(dc[s'])$
    end for
end for

// Main loop: $finalOrder$ is an array representing the final state ordering.
for $i = 0,\ldots,(|p|-1)$ do
    let $s$ be the index of the smallest non-negative value in $dc$
    $dc[s] \leftarrow -1$
    $finalOrder[|p|-1-i] \leftarrow s$
    for all $a \in A$ do
        for all $s' \in p$ do
        if $T(s,a,s') \neq 0$ then decrement$(dc[s'])$
    end for
end for
```

Algorithm 7. Modified topological reordering [23].

6. Experiments

For evaluating our approach, we chose the sailing domain [35]. This is a finite state and action-space stochastic shortest path problem, 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 8 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 4 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 3 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 2 seconds. Finally, if the boat is sailing directly downwind, we say that it is on an away tack (only 1 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 3 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 I 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 1 GB RAM. All algorithms were implemented in the Java language under a robotic planning environment. The initial and 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 $\varepsilon = 10^{-7}$, $\gamma = 1$ and $numit = 1000$. Since $\gamma = 1$, 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 (see equation (5)) no longer holds [36]. Fortunately, the presence of absorbing states (states with null reward and 100% probability of staying in the same state) may allow the algorithm to converge [37].

The lake size was varied from $6´\ 6$ to $200´\ 200$ and the resulting number of states varied from 384 to 940896, respectively. We repeated each run 10 times and then we calculated the average and standard deviation.

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

7.Results

Figure 1 shows the solution time as a function of the number of states for all the algorithms, excepting ARVI6. We can see that ARVI5 (which uses asynchronous updates of states that change between iterations as well as static reordering in decreasing order of maximum reward) is significantly faster than the other algorithms. We can see in Table II that, for 940896 states, ARVI5 was almost 3 times faster than VDP. Even our first algorithm (ARVI, which did not include any acceleration procedure) is approximately 1.2 times faster than VDP. We can see in Figure 2 that all algorithms are clearly faster than classic value iteration.

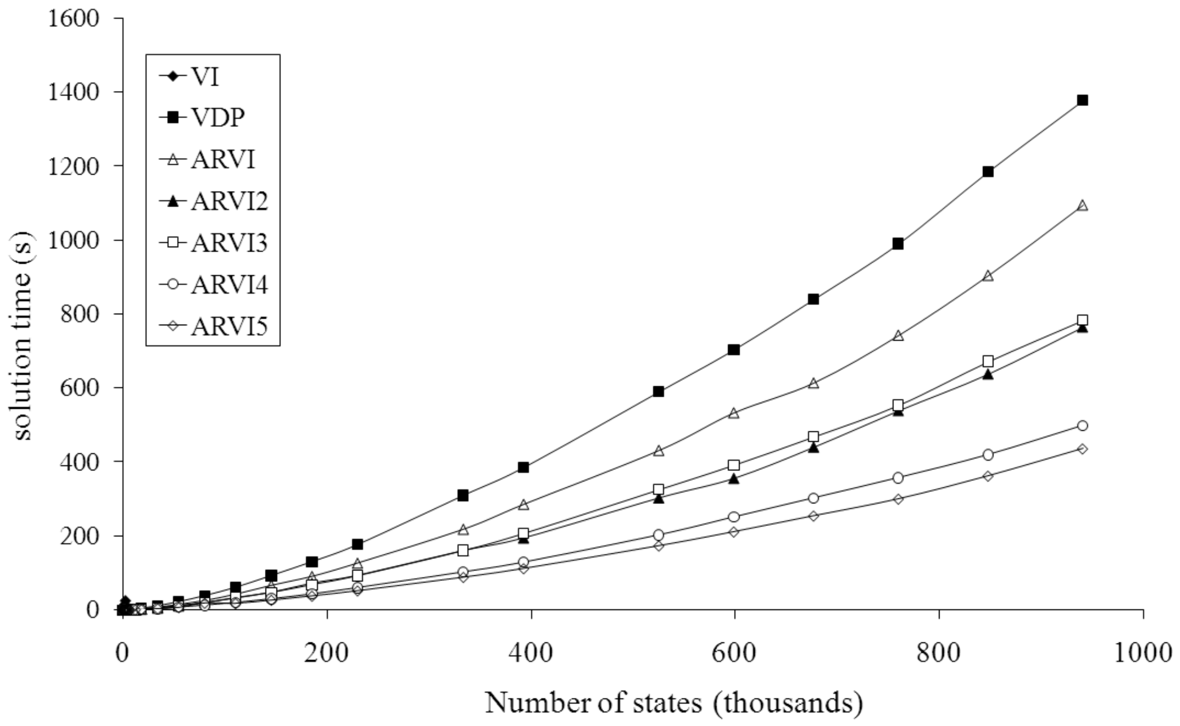

Figure 1. Solution time as a function of the number of states for all the approaches, excepting ARVI6.

Algorithm	Solution time (ms)	Relative solution time
VDP	1376571.9	3.16
ARVI	1095706.3	2.50
ARVI2	766124.9	1.76
ARVI3	782284.5	1.79
ARVI4	499172.0	1.14
ARVI5	436302.9	1.0

Table II. Summary of results in terms of solution time for all the algorithms excepting ARVI6 (the number of states was 940896). Relative solution times are calculated with respect to the solution time of ARVI5.

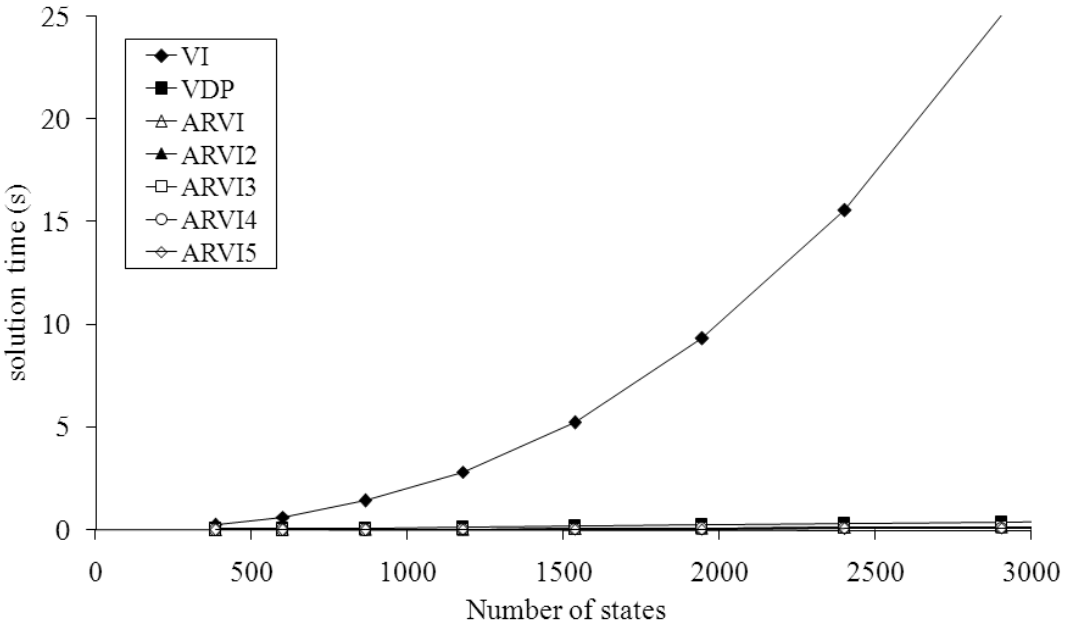

Figure 2. Close-up of the solution time (for less than 3000 states) as a function of the number of states for all the approaches, excepting ARVI6.

Figure 3 shows the number of iterations required by each algorithm to reach the solution as a function of the number of states, excepting ARVI6. We can observe that or fastest algorithm ARVI5 also required the smallest number of iterations. For example, for 940896 states, VDP took 373 iterations whereas ARVI5 required 338 iterations.

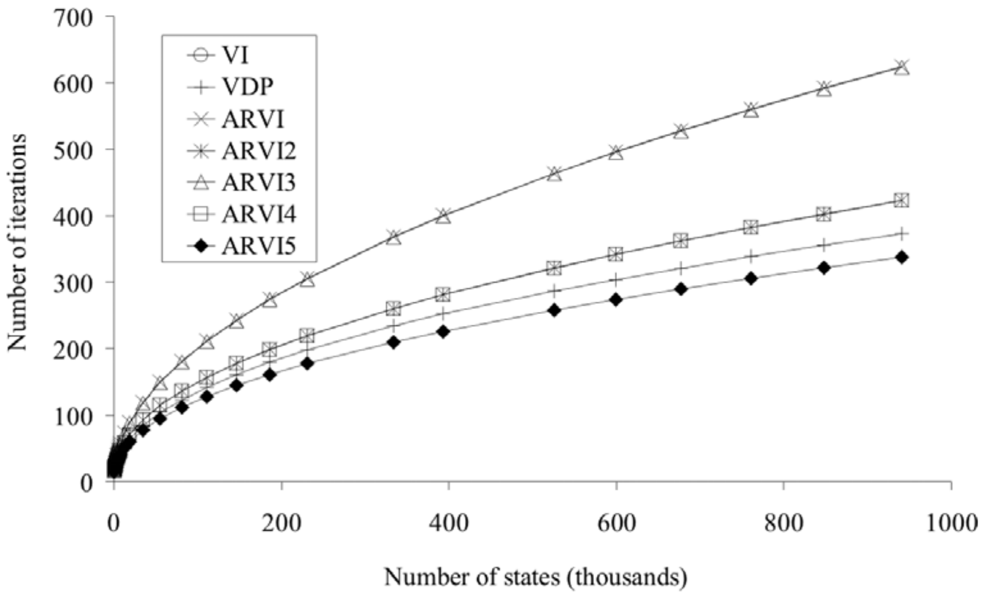

Figure 3. Number of iterations as a function of the number of states for all the algorithms, excepting ARVI6.

Figure 4 shows a comparison between ARVI5 and ARVI6 in terms of solution time as a function of the number of states. Figure 5 shows a comparison between ARVI5 and ARVI6 in terms of the number iterations to reach the solution. We can see that ARVI6 requires slightly less iterations than ARVI5, but ARVI5 is significantly faster than ARVI6, even when they differ only in the way states are sorted. This shows clearly that the modified topological reordering algorithm used by ARVI6 was very slow in comparison with the ordering algorithm used by ARVI5 (in decreasing order of maximum reward by means of quicksort). In this case, the use of topological ordering 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 [38]. Another possibility to find a topological ordering would be to apply a strongly connected components algorithm as in [39]. Anyway, preliminary results obtained in an experiment in which we used a strongly connected component algorithm indicated that our reordering was still faster.

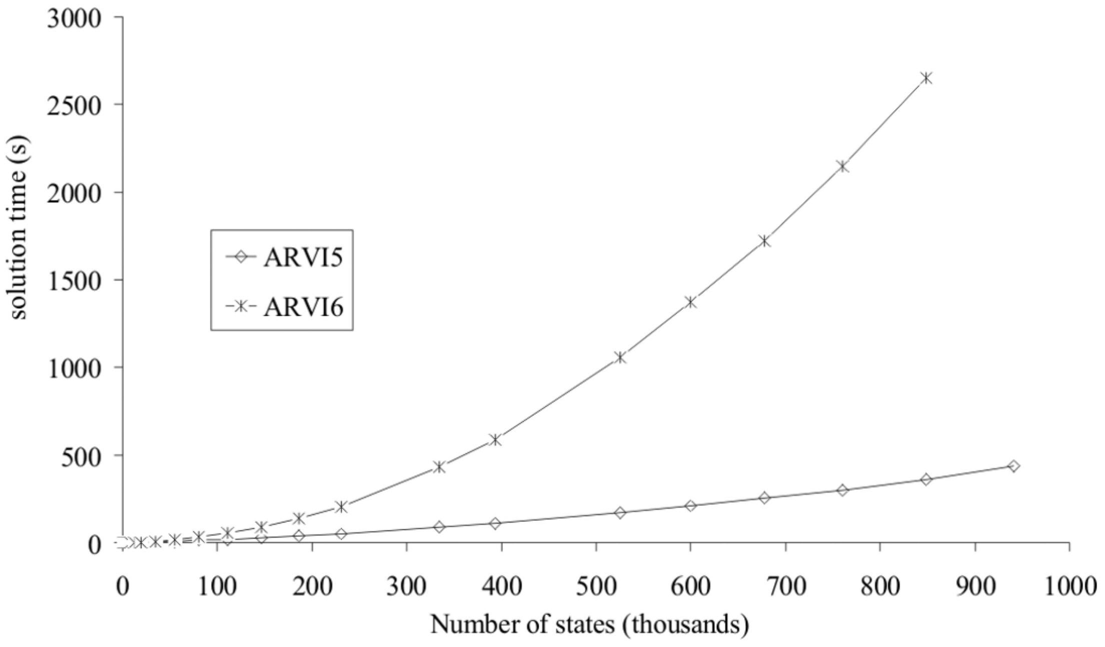

Figure 4. Comparison in terms of solution time for two algorithms using different reordering methods. ARVI5 uses a reordering method based on maximum reward and ARVI6 uses a modified topological ordering algorithm.

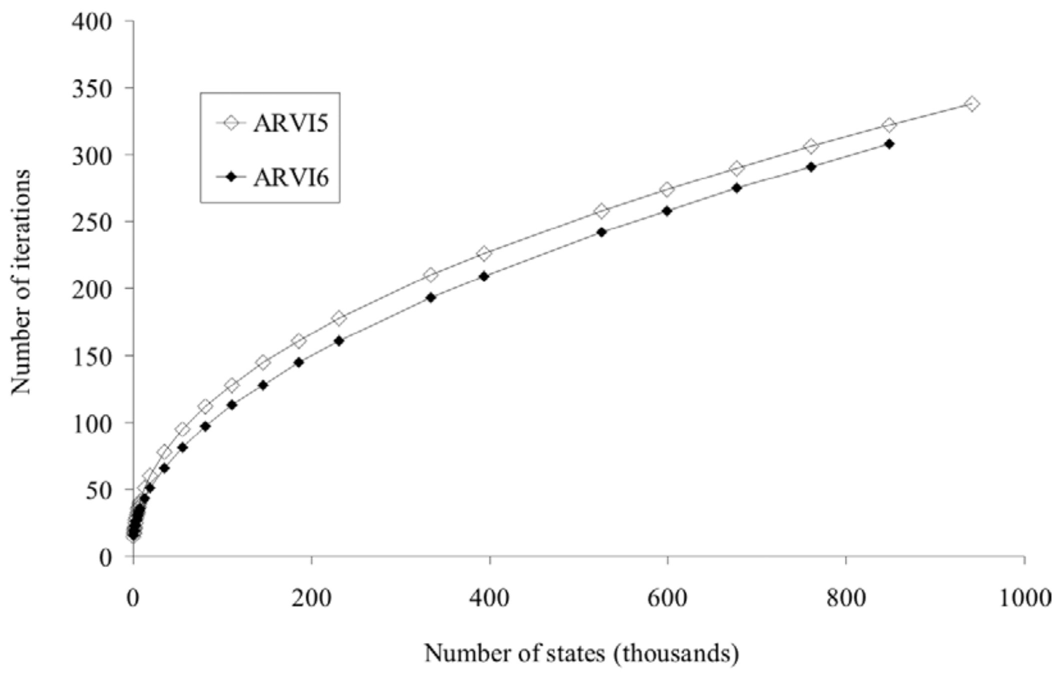

Figure 5. Comparison in terms of number of iterations for two algorithms using different reordering methods. ARVI5 uses a reordering method based on maximum reward and ARVI6 uses a modified topological ordering algorithm.

In all cases our ARVI5 yielded the smallest solution time but slightly more iterations than ARVI6. This implicates that, at least in the sailing strategies problem, the combination of asynchronous updates, prioritization with static ordering of the states in decreasing order of their maximum reward, results in the fastest algorithm. The use of prioritization and partitioning (excepting static ordering in decreasing value of maximum reward) in the sailing strategies problem resulted in excessive overhead. This may be due in part to the cyclic nature of the resulting MDPs as shown in [23] for the SysAdmin problem.

8. Conclusions

In this paper we have successfully tested a new approach for the estimation and solution of MDPs based on association rules (obtained by using efficient methods such as Apriori and other powerful association rule mining techniques) and the application of state-of-the-art acceleration procedures such as asynchronous updates and prioritization with static reordering of the states. In addition, we tested a new criterion for reordering states and compared it with the modified topological reordering algorithm proposed in [23].

We compared our approach with other methods such as classic value iteration [18] and dynamic programming [40, 41]. At least in the sailing domain, our approach combined with asynchronous updates and prioritization using a static reordering of states in decreasing order of maximum reward yielded the lowest solution time with the lowest number of iterations. In general, the use of prioritization and partitioning excepting static ordering by 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 [23]). Since the use of the modified topological reordering algorithm resulted in less iterations than the state reordering algorithm based on maximum reward, further work will focus other criteria for finding an optimal state ordering.

References

[1] Boutilier, C., Dean, T., 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] Bonet, B., Geffner, H., Learning depth-first search: A unified approach to heuristic search in deterministic and non-deterministic settings and its application to MDP, International Conference on Automated Planning and Scheduling, ICAPS, 2006, Cumbria, UK.

[5] Darwiche, A., Goldszmidt M., Action networks: A framework for reasoning about actions and change under understanding, 10th Conference on Uncertainty in Artificial Intelligence, UAI, 1994, pp 136-144, Seattle, Washington, USA.

[6] Van Otterlo, M., A Survey of Reinforcement Learning in Relational Domains, Technical Report Series CTIT-05-31, ISSN 1381-3625, July 2005.

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

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

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

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

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

[12] Bonet, B., 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.

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

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

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

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

[17] Bellman, R. E., Dynamic Programming, Princeton United Press, Princeton, USA, 1957.

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

[22] 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.

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

[24] Dai, P., Hansen, E. A., Prioritizing Bellman Backups Without a Priority Queue, Association for the Advancement of Artificial Intelligence, 17th International Conference on Automated Planning and Scheduling, ICAPS, 2007.

[25] Agrawal, R., Imielinski, T., Swami, A., Mining Association Rules between Sets of Items in Large Databases, ACM SIGMOD International Conference on Management of Data, May 1993, Washington DC, USA.

[26] Hahsler, M., Hornik, K., Reutterer, T., Implications of Probabilistic Data Modeling for Mining Association Rules, Studies in Classification Data Analysis and Knowledge Organization, Springer Verlag, 2005.

[27] Brijs, T., Swinnen, G., Van Hoof, K., Wets, G., Building an association rules framework to improve product assortment decisions, Data Mining and Knowledge Discovery, 8 (1), 2004, pp 7-23.

[28] Lawrence, R. D., Almasi, G. S., Kotlyar, V., Viveros, M. S., Duri, S., Personalization of supermarket product recommendations, Data Mining and Knowledge Discovery, 5 (1/2), 2001, pp 11-32.

[29] Van den Poel, D., Schamphelaere, J., Wets, G., Direct and indirect effects of retail promotions on sales and profits in the do-it-yourself market, Expert Systems with Applications, 27 (1), 2004, pp 53-62.

[30] Agrawal, R., Srikant, R., Fast Algorithms for Mining Association Rules, 20th VLDB Conference, IBM Almaden Research Center, 1994.

[31] Sutton, R. S., Barto, A. G., Introduction to Reinforcement Learning, MIT Press, USA 1998.

[32] Scherrer, B., Mannor, S., Error Reducing Sampling in Reinforcement Learning, Institut National de Recherche en Informatique et Automatique, INRIA, 98352, Vol.1, September 2006.

[33] Gupta, G. K., Introduction to Data Mining with Case Studies, Prentice-Hall of India, Pvt. Ltd, 2006, pp 76-82.

[34] Ceglar, A., Roddick, J. F., Association Mining, ACM Computing Surveys, Vol. 38, No.2, Article 5, July 2006.

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

[36] Blackwell, D., Discounted dynamic programming, Annals of Mathematical Statistics, Vol. 36, 1965, pp 226-235.

[37] Hinderer, K., 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.

[38] Garey, M. R., 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.

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

[40] Reyes, A., Ibarguengoytia, P., Sucar, L. E., 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.

[41] Vanderbei, Robert J., Linear Programming: Foundations and Extensions, Springer Verlag, 3rd Edition, January 2008.

Authors´ Biography

Ma.de Guadalupe GARCIA-HERNANDEZ

She is a researcher and full-time professor at the Universidad de Guanajuato, at the Engineering Division of the Campus Irapuato-Salamanca, in Mexico. She holds a recognition and artificial intelligence doctorate degree from the Universidad Politécnica de Valencia, Spain. She obtained the mechanical engineer master's degree in 1992 from the Universidad de Guanajuato, Mexico. She graduated in chemical engineer in 1985 from the same university. For twenty years, she has been a full-time professor in the aforementioned institution. She has presented eighteen papers in international scientific events and eleven papers in national scientific events; she has six collaborations in indexed journals and four collaborations in national journals. She is member of the Spanish Association for Artificial Intelligence since January, 2006.

J. Gabriel AVIÑA-CERVANTES

He received the engineering degree in electronics and communications from the Universidad de Guanajuato in 1998, the M.S. degree in electrical engineering from the same university in 1999, the Ph.D. degree 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 at the Universidad de Guanajuato, at the Engineering Division of the Campus Irapuato-Salamanca.

Sergio LEDESMA

He got his M.S. degree from the Universidad de Guanajuato while working on the setup of Internet in Mexico. In 2001, he got his Ph.D. degree from the 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 the Universidad de Guanajuato in Mexico. His areas of interests are Artificial Intelligence and software engineering.

Eva ONAINDIA

She is an assistant professor of computer science at the Universidad Técnica de Valencia in Spain. She received her Ph.D. degree in computer science from the same university in 1997. She currently leads the Reasoning on Planning and Scheduling Group where she conducts research in temporal and classical planning, development of integrated techniques of planning and scheduling and re-planning. She is currently collaborating with 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 sittings 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.

Alberto REYES-BALLESTEROS

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

José RUIZ-PINALES

He received the BA degree in electronics and communications engineering and the MSE degree in electrical engineering from Universidad de Guanajuato in 1993 and 1996. He received the PhD degree in computer science from the Ecole Nationale Supérieure des Télécommunications de Paris in 2001. He joined the Department of Electronics Engineering, 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.