On-line search for solving Markov Decision Processes via heuristic sampling

Laurent Péret and **Frédérick Garcia**[1]

Abstract. In the past, Markov Decision Processes (MDPs) have become a standard for solving problems of sequential decision under uncertainty. The usual request in this framework is the computation of an optimal policy that defines the optimal action for every state of the system. For complex MDPs, exact computation of optimal policies is often untractable. Several approaches have been developed to compute near optimal policies for complex MDPs by means of function approximation and simulation.

In this paper, we investigate the problem of refining near optimal policies via online search techniques, tackling the local problem of finding an optimal action for a single current state of the system. More precisely we consider an on-line approach based on sampling: at each step, a randomly sampled look-ahead tree is developed to compute the optimal action for the current state. In this work, we propose a search strategy for constructing such trees. Its purpose is to provide good "anytime" profiles : at first, it quickly selects a good action with a high probability and then it smoothly increases the probability of selecting the optimal action.

1 INTRODUCTION

Solving complex Markov Decision Processes (MDPs) usually requires an off-line approximation of the optimal value function. The greedy policy defined by this value function is then exploited on-line. Several approaches have been developed by Artificial Intelligence and Machine Learning communities to efficiently compute near optimal value functions e.g. *structure-based methods* [3], *reinforcement learning* [16, 2].

However, when the state space is huge and when no efficient structured representation is known, the calculation of an accurate value function is often a hard task. In this paper, we investigate the idea of replacing the greedy exploitation of an approximate value function by a search technique : for each current state s_t a search tree is developed over some *receding horizon*. From this tree we will compute an action for the current state. This action is then applied and the system evolves to a new current state (see Figure 1). Deriving a greedy policy from a value function is equivalent to a depth 1 search. We can expect to improve that policy at the expense of a deeper search. This idea has been widely and successfully applied to two-player perfect information games like chess [5].

For complex MDPs, performing a lookahead search to compute the optimal action for a single state might be difficult, in particular when the model of the dynamics is unknown or when the number of possible successors of a state (i.e. the branching factor in terms of tree search) is huge. A common approach in reinforcement learning to overcome these difficulties consists in using a *simulator* of the system. Computations are then performed over sampled states and rewards. Kearns et al [11] have provided a theoretical foundation for combining simulation and on-line search. Their algorithm builds randomly sampled trees defining a stochastic policy. This approach is very appealing since it allows to tackle arbitrarily large MDPs given only a simulator of the system.

The price to pay for this generality is that the number of calls to the simulator required by Kearns et al's basic algorithm is often extremely large. This makes the algorithm impractical in many cases. In this work, we propose a heuristic search strategy for efficiently constructing such sampling based trees. It consists in repeatedly following trajectories from the root state to the leaf states while gradually increasing the receding horizon. Incidentally, we study under which conditions the anticipation is beneficial, bringing to the fore *lookahead pathologies*. A lookahead is pathological if looking deeper ahead *increases* the chances of choosing a suboptimal action. We theoretically and empirically show that pathologies occur when inappropriate search control is used. On the contrary, our control strategies ensure a good "anytime" behavior.

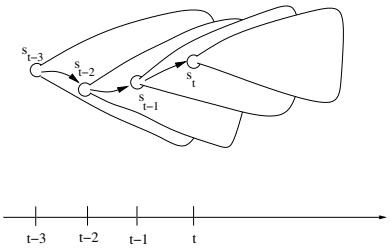

Figure 1. On-line search with receding horizon

The organization of the rest of the paper is the following : in section 2, we introduce the MDP framework and review relevant background for solving MDPs on-line. In section 3, we analyze lookahead pathologies and describe our trajectory-based strategy. Section 4 provides experimental results obtained for a sailing problem.

2 ON-LINE SEARCH FOR SOLVING MARKOV DECISION PROCESSES

2.1 Markov Decision Processes

In few words, an *MDP* [15] is defined by a five-tuple $< S, A, T, P, R >$ where S is the set of every possible *states* of the

[1] Institut National de la Recherche Agronomique, Unité de Biométrie et Intelligence Artificielle, 31326 Castanet Tolosan, France, peret@toulouse.inra.fr

system, A is the set of *actions* that can be applied to it, T is the discrete set of *instants* at which decisions can be made, that is the *global temporal horizon* which can be finite or infinite; P defines the *transition probabilities* between any pair of states in S after executing an action in A; R defines the *local costs* associated with these transitions. Solving an MDP consists in computing an *optimal policy*, that is a function π^* that associates with any state $s \in S$ an *optimal action* $\pi^*(s)$, that is an action that minimizes the *expected global cost* on the remaining temporal horizon. This global cost may be defined as the *sum*, the *discounted sum*, or the *mean value* of the local costs that are associated with the actual transitions.

The MDP theory assigns to every policy π a *value function* V_π, which associates to every state $s \in S$ the expected global cost $V_\pi(s)$, obtained by applying π in s. *Bellman's optimality equations* (see Equation 1) characterize in a compact way the *unique optimal value function* V^*, from which an *optimal policy* π^* can be straightforwardly derived (see Equation 2). In case of a global cost which is defined as the discounted sum of the local costs, these equations are the following:

$$V^*(s) = \min_{a \in A(s)} [r(s,a) + \gamma \cdot \sum_{s' \in S} P(s'|s,a) \cdot V^*(s')], \quad (1)$$

$$\pi^*(s) = \operatorname*{argmin}_{a \in A(s)} [r(s,a) + \gamma \cdot \sum_{s' \in S} P(s'|s,a) \cdot V^*(s')]. \quad (2)$$

where $0 < \gamma \leq 1$ is the discount factor and $A(s) \subset A$ is the set of applicable actions in state s. When there is no discounting ($\gamma = 1$), we assume that there exists a *cost-free termination state* defining a *stochastic shortest path problem* (SSP) - see [2].

The *value iteration* algorithm is a standard numerical algorithm for solving MDPs off-line. It calculates an optimal policy by repeatedly performing *Bellman's updates* over the entire state space :

$$\forall s \in S \quad V(s) \leftarrow \min_{a \in A(s)} [r(s,a) + \gamma \cdot \sum_{s' \in S} P(s'|s,a) \cdot V(s')] \quad (3)$$

If every state is updated infinitely often then the value function V eventually converges to the optimal value function.

2.2 Anytime heuristic search algorithms for MDPs

The main drawback of value iteration is that it performs updates over the entire state space. To focus computations on only relevant states, researchers from the Planning community have generalized real-time heuristic search methods like $LRTA^*$ [12] to non-deterministic problems. For instance, Dean et al's envelope algorithm [8] and LAO^* [9] from Hansen and Zilberstein are based on this principle.

All these algorithms perform a look-ahead search on a subset of states reachable from a given current state. Initially, this subset called envelope in [8] only contains the current state and is progressively expanded. An approximate value function $\tilde{V}$ is generally used to value the fringe states. The general scheme of these algorithms is an alternation of :

(i) fringe expansion: some fringe state s is chosen and its value is updated :

$$V(s) \leftarrow \min_{a \in A(s)} [r(s,a) + \gamma \cdot \sum_{s' \in S} P(s'|s,a) \cdot \tilde{V}(s')]$$

(ii) ancestors update phase: the value of the ancestors of the newly updated states are updated using Bellman's optimality principle 3.

Their output is a *partial policy* that is a policy which is defined only on the envelope. These algorithms offer optimality guarantees when they are not interrupted and ensure a good *anytime behavior* : they produce good quality policies when they are interrupted. Their efficiency can be improved if $\tilde{V}$ satisfies some classical assumptions for heuristic value functions like consistency or admissibility.

These algorithms can solve MDPs very efficiently without evaluating the entire state space. The computation of the optimal or near-optimal partial policies they calculate is usually orders of magnitude faster than the computation of an optimal policy with value iteration.

2.3 Kearns et al's sparse sampling algorithm

The previous look-ahead approaches are applicable as far as transition probabilities $P(s'|s,a)$ are known and the number of possible successors states for a state/action pair remains low. If it is not the case, a natural way of extending this approach consists in using sampling to generate successor states. The only assumption required for that is the existence of a *simulator* which is able to generate from any pair (s,a) a successor state s' according to probabilities $P(s'|s,a)$. Thus, for any pair (s,a), the probability distribution over the successor states is approximated by generating a sample of these states.

A simple algorithm, which has been proposed by Kearns and al. [11], consists in trying every possible action C times from each state s: C times from the current state s_t and recursively C times too from every state which has been generated from s_t over a given temporal horizon of length H (see Figure 2). An on-line value function is then recursively defined by :

$$V_{h,C}(s) = \begin{cases} \tilde{V}(s) \quad \text{if } h = 0 \\ \min_{a \in A(s)} Q_h(s,a) \quad \text{otherwise} \\ \text{where} \quad Q_h(s,a) = [r(s,a) + \\ \gamma \frac{1}{C} \sum_{s' \in S(s,a,C)} V_{h-1,C}(s')] \end{cases} \quad (4)$$

In this equation, $S(s,a,C)$ is the set of the C states that have been sampled from the pair (s,a). The complexity of the on-line computation of $V_{H,C}$ is $O((|A| \cdot C)^H)$.

After this stochastic tree has been developed, the current action a_t is straightforwardly derived from Equation 4 : $a_t = \operatorname{argmin}_{a \in A(s)} Q_H(s_t, a)$. This algorithm thus defines a stochastic policy. The theoretical result established by Kearns and al. quantifies the performance of this stochastic policy in function of C and H. It stipulates that its expected degradation in terms of global cost with respect to an optimal policy can be as small as required provided that sufficiently large values of C and H are used, *independently of the size of the state space*. In other words, on-line sampling-based search is able to provide near optimal policies for any MDP given only a simulator of the system.

Unfortunately, the values needed for C and H to ensure convergence are extremely large, and make the algorithm often impractical. Recently, Chen et al [6] have proposed an improved two-step algorithm, based on the theory of multi-arm bandit problems [1]. After an initialization phase, additional simulations are distributed taking into account estimated means and variances of states values. However, the complexity of the search is still in $O((|A|.C)^H)$.

3 CONTROLLING THE SEARCH

In practice, we have to use reasonable values for C and H, sacrificing the near optimality guarantees provided by Kearns et al 's result. In

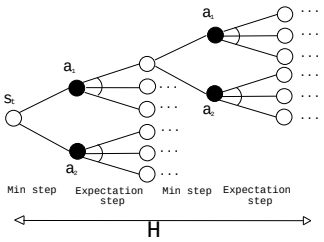

Figure 2. Kearns et al's algorithm with two actions and $C = 3$

this section, we provide an analysis of the performance in function of C and H before introducing our search control algorithm.

3.1 Error bounds and lookahead pathologies

We assume we know an approximate value function $\tilde{V}$, which will be used to value the tree leaf states. From Kearns et al's main result, we can derive a *probably approximately correct bound* on $|V_{H,C}(s_t) - V^*(s_t)|$, which links the probability Δ of making an error to the size of this error: :

$$|V_{H,C}(s_t) - V^*(s_t)| \leq r_{max}\sqrt{\frac{1}{C}.\log\frac{(|A|\cdot C)^H}{\Delta}} + \gamma^H\epsilon \quad (5)$$

with a probability of at least $1-\Delta$, where r_{max} is the maximum local cost and $\epsilon = \max_{s_{t+H}}|\tilde{V}(s_{t+H}) - V^*(s_{t+H})|$ the error propagated from the leaf states s_{t+H}.

For a fixed receding horizon H, if $C \to \infty$, then $|V_{H,C}(s_t) - V^*(s_t)| \to \gamma^H\epsilon$, which is a known result for receding horizon approaches using an exact model of the dynamics [10]. It means that for large values of C, if the receding horizon is long enough, the error will be significantly decayed. Therefore, the probability of selecting an optimal action on the basis of this lookahead will be improved.

For SSPs, when $\gamma = 1$, it is necessary that $\epsilon = \epsilon_H$ decreases when H grows to make the lookahead beneficial. In other words, the error must be smaller for the leaf states than for the current state. Like for two-player games, this happens if some "improved visibility" property holds for the approximate value function: the accuracy of $\tilde{V}$ improves as we approach a termination state.

Lookahead pathologies

Surprisingly, increasing the horizon H with a fixed width C eventually *increases* the error since the term due to finite sampling approximation grows with H. This means that a simple "iterative deepening" approach will eventually provides a bad choice for action a_t. This pathology phenomenon has been widely studied in game theory: to deepen search does not automatically improve the quality of the decision. For example, Pearl [14] considers a 2-player game with an evaluation function which predicts if a position leads to win or to loose. This function is imperfect and has some error rate. A minmax search is performed to improve the quality of the decision. Tree leaves are evaluated by using this imperfect function and information is propagated from leaves up to the root. Under this simple model, Pearl shows that, without any special assumption, the deeper the search, the worse the decision. More recently, Bulitko et al [4] have provided a similar analysis for single agent search. However, pathologies have never been observed to our knowledge in classical games like chess or any problem of practical interest. We will show in section 4 for a common SSP that pathology can occur when we perform on-line sampling-based search.

In practice, with a limited amount of simulations, we have to look for a good trade-off between the horizon H and the width C. Figure 3 plots the error bound as a function of H for different computational budgets. By computational budget, we mean the number of simulated transitions used on-line for the development of the tree. Notice that the optimal horizon depends on this budget. If we maintain a good horizon/width trade-off as we expand the tree, we can hope to observe a good anytime behavior i.e. a continuous improvement of the policy derived from the tree.

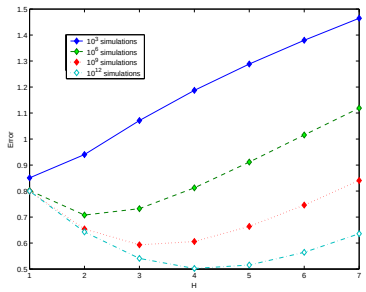

Figure 3. Error bound on $|V_{H,C}(s_t) - V^*(s_t)|$ as a function of the horizon H for different computational budgets. $\Delta = 0.1$, $\epsilon = 1.0$, $r_{max} = 0.5$, $|A| = 3$, $\gamma = 0.8$

3.2 A trajectory-based strategy

The control strategy we propose consists in repeatedly following some trajectories of length H from the current state s_t to a leaf state s_{t+H}. The global width C is thus no more specified and depends on the state/action pair (s,a) considered.

In order to maintain a good global horizon/width trade-off, we introduce a dynamic horizon control based on the estimation of the global sampling error on the tree. This estimation is a heuristic which indicates whether some state/actions pairs need additional sampling. The idea is to increase the horizon when this estimated error is stabilized.

3.2.1 Estimating a global sampling error

We have seen that pathological behavior can be caused by the amplification of the error due to finite sampling. We would like to have an estimation of this error in order to monitor the increase of H. As the width of the different expanded state/action pairs are not uniform in our algorithm, we have to use a heuristic decentralized approach to estimate this error. Such an approach which propagates errors through a Markovian graph was developed in [13] to design *exploration policies* for reinforcement learning (see next paragraph).

We can easily estimate the local error due to finite sampling using a basic result of statistics for Monte-Carlo simulation about *confidence intervals*. For any state-action pair (s,a) with local width C we define the local error as: $e(s,a) = \sigma\frac{t_{\theta\backslash 2}^{C-1}}{\sqrt{C}}$ where σ is the sample standard deviation of C observations of $\gamma V_{h-1}(s')$ and $t_{\theta\backslash 2}^{C-1}$ is Student's t-function with $C-1$ degrees of freedom at user specified confidence level $\frac{\theta}{2}$ (for instance $\theta = 0.05$). Notice that this expression only quantifies the local error due to finite sampling and neither the

local error due to successors states nor the error due to $\tilde{V}$. Assuming that $V_{h-1}(s')$ is a stationary normally distributed random variable of mean v then $Q_h(s,a) \in [r(s,a) + \gamma v - e(s,a), r(s,a) + \gamma v + e(s,a)]$ with probability $1-\theta$. As this assumption does not usually hold because V_{h-1} is itself an estimate, we don't have this property but still use $e(s,a)$ for estimating the local error.

We *propagate* this error through the tree, in the same way as actions and states values were propagated in Equation 4. Thus the global sampling error $E_h(s)$ for a state is defined as :

$$E_h(s) = \begin{cases} \sigma_{init} \quad \text{if } h = 0 \\ \min_{a \in A(s)} M_h(s,a) \quad \text{otherwise} \\ \text{where} \quad M_h(s,a) = e(s,a) + \\ \gamma \frac{1}{C} \sum_{s' \in S(s,a,C)} E_{h-1}(s') \end{cases} \quad (6)$$

Finally, we have at our disposal for each state of the tree an estimation of the global sampling error, and in particular for the root state s_t.

3.2.2 *Organizing the search along successive trajectories*

Like heuristic search algorithms for MDPs described in section 2, our algorithm alternates expansion phases of some fringe states and update phases. An important difference is that *every state in the tree is a potential fringe state* : as we use a finite sampling approximation, a state is never perfectly expanded. Indeed, this would mean that every applicable action for this state has been simulated an infinite number of times. Consequently, a large number of control strategies can be considered.

A sensible one consists in following successive trajectories of length H from the root. These trajectories are guided by some exploration policy. Exploration policies were designed by the reinforcement learning community for the purpose of efficiently converging towards optimal policies. The general problem they face is the classical exploration/exploitation dilemma. Several algorithms have been proposed (see [16]) and are relevant for our tree exploration problem as they are able to rapidly focus on promising state/action pairs. Here are the different policies we have experimented :

Uniform exploration: a random action is selected.

Boltzmann's exploration: an action a is selected with probability

$$\frac{e^{-Q(s,a)\backslash\tau}}{\sum_{b \in A(s)} e^{-Q(s,b)\backslash\tau}}$$

τ is a positive parameter called the temperature parameter. High temperatures cause (nearly) uniform exploration and low temperatures cause (nearly) greedy selection.

Meuleau's IEDP+ algorithm [13]: the selected action a maximizes $Q(s,a) + B(s,a)$. $B(s,a)$ is an exploration bonus which is very similar to $M(s,a)$. It favors actions whose global sampling error is high.

Our trajectory-based algorithm is described in Algorithm 1. The routine GenerateTrajectories(H,π,N) generates N trajectories of length H following policy π from s_t. These trajectories are stored in the tree. The routine UpDateStatesValues() updates values and errors of states newly expanded and of their ancestors.

Input : *exploration policy π, approximate value function $\tilde{V}$, current state s_t, tolerance δ, size of the batch of trajectories N*
Output : *action a_t*
$H \leftarrow 1$
$OldE \leftarrow +\infty$
while *true* **do**
  while $|E_H(s_t) - OldE| > \delta$ **do**
    GenerateTrajectories(π,H,N)
    $OldE \leftarrow E_h(s_t)$
    UpdateStatesValues()
  end
  $H \leftarrow H + 1$
end

Algorithm 1: Trajectory-based algorithm for on- line heuristic sampling

4 EXPERIMENTS

4.1 Optimal sailing domain

To validate our propositions, we chose a sailing problem that is modeled as an SSP (8400 states and 8 actions). We wish to find the quickest path for a sailboat from initial state s_0 to final state s_f taking into account random wind fluctuations (see [17] for a complete description and a web interface of the simulator).

To mimic an imperfect value function, we generated $\tilde{V} = V^*.(1+\epsilon)$ where ϵ is a uniform random variable drawn in $[-0.1; 0.1]$. $V^*(s_0) = 119.0$ (minutes to cross the lake).

We evaluate the performances of the stochastic policy derived from the lookahead search for various control schemes and computational budgets. The performance of these various stochastic policies are estimated over 500 trajectories.

4.2 Implementation issues

- To save memory, we collapse identical states at the same level; the tree is therefore actually a graph.
- If after execution of a_t, the resulting state s_{t+1} belongs to the set of sampled successors of s_t, we then reuse the corresponding subtree.
- Parameters values : tolerance for horizon control $\delta = 0.75$; initial error $\sigma_{init} = 10.0$; confidence level $\theta = 0.10$; size of the batch of trajectories : $N = 100$.

4.3 Pathologies and optimal horizon

The basic Kearns et al's algorithm exhibits pathological behavior : for $C < 10$ we can observe a degradation of the policy's performance provoked by deeper search (see Figure 4). The line labeled with $C = +\infty$ is a (time-consuming) lookahead search using the exact model. All other experiments only use the simulator.

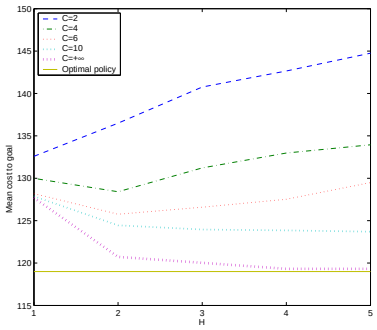

Figure 4. Lookahead pathology : performance of the policy derived from the tree as a function of the horizon H for a fixed width.

To demonstrate the influence of the horizon's choice, we have run our trajectory-based algorithm with different *static* horizons for a given computational budget. With 100 transitions, the best static horizon is 2 for Boltzmann and IEDP+. With 1000 transitions, the best static horizon is 3 with Boltzmann exploration and 6 with IEDP+ exploration (see Figure 5).

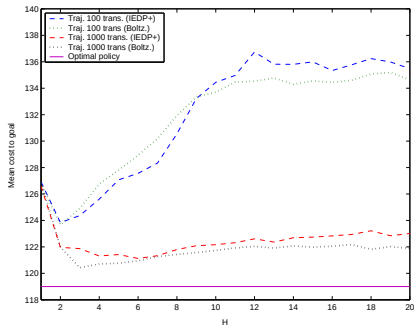

Figure 5. Optimal static horizon: performance of the policy derived from the tree as a function of the horizon H for a fixed computational budget.

4.4 Anytime behaviors

Figure 6 plots the anytime profiles of diverse control strategies, namely : Kearns et al's basic algorithm with simple iterative deepening, our algorithm 1 guided by uniform, Boltzmann and IEDP+ exploration. We have tested too an improved version of Kearns'algorithm which increases C and H simultaneously to obtain a better trade-off.

Our algorithm performs better than Kearns'algorithm when Boltzmann or IEDP+ exploration is used : it exhibits good anytime profiles, with fast convergence towards near-optimal policies. The dynamic horizon management ensures a good trade-off between C and H. Indeed, for different computational budgets, the gap between the performance of the policy and the performance of the policy obtained with the best static horizon is small : when Boltzmann's exploration is used, this gap is less than 2 % gap. .

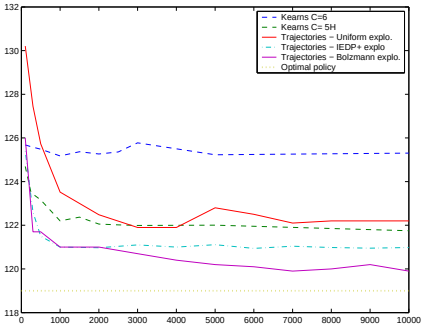

Figure 6. Anytime profiles of different exploration strategies: performance of the policy derived from the tree as a function of the computational budget.

5 CONCLUSIONS AND FUTURE WORK

Our algorithm shares characteristics with classical search algorithms - receding horizon, tree memory structure - and with reinforcement learning algorithms - exploration by following exploration policies, dealing with uncertainty.

Future works include the design of exploration policies for trajectories-based approaches. Indeed, the exploration/exploitation dilemma to tackle is slightly different from the classical one : we do not want to minimize costs received during the exploration phase as we are only interested in quickly selecting an optimal action for the current state. An appropriate framework for designing exploration policies is provided by ordinal optimization [7].

We believe that on-line sampling is an efficient approach for solving complex MDPs when standard greedy search fails to generate a good policy. As on-line sampling requires quite a lot of computation, its main applications are more likely to be within the domains of simulation of complex industrial systems rather than within real-time control of embedded systems. We are currently developing this on-line sampling approach for tackling a real world problem of satellite management constellation which was proposed by the French Space Agency and modeled in [18]. Significant improvement of off-line optimized policies confirms the relevance of this on-line approach.

REFERENCES

[1] D. A. Berry and B. Fristedt, *Bandit problems: Sequential allocation of experiments*, Chapman and Hall, London, England, 1985.

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

[3] C. Boutilier, T. Dean, and S. Hanks, 'Decision theoretic planning: Structural assumptions and computational leverage', *Journal of Artificial Intelligence Research*, **11**, 1–94, (1999).

[4] V. Bulitko, L. Li, R. Greiner, and I. Levner, 'Lookahead pathologies for single agent search', in *IJCAI' 03*, (2003).

[5] M. Campbell, A.J. Hoane Jr., and F. Hsu, 'Deep blue', *Artificial Intelligence*, **134**(1-2), 57–83, (2002).

[6] H. S. Chang, M. C. Fu, and S. I. Marcus, 'An adaptive sampling algorithm for solving Markov Decision Processes', *Operations Research, to appear*, (2004).

[7] H. C. Chen, C. H. Chen, and E. Yucesan, 'Computing efforts allocation for ordinal optimization and discrete event simulation', *Journal of IEEE Transactions on Automatic Control*, **45**(5), 960–964, (2000).

[8] T. Dean, L. P. Kaelbling, J. Kirman, and A. Nicholson, 'Planning under time constraints in stochastic domains', *Artificial Intelligence*, **76**(1–2), 35–74, (1995).

[9] E. A. Hansen and S. Zilberstein, 'LAO* : A heuristic search algorithm that finds solutions with loops', *Artificial Intelligence*, **129**, 35–62, (2001).

[10] O. Hernandez and J.B. Lasserre, 'Error bounds for rolling horizon policies in discrete-time Markov control processes', *IEEE Transactions on Automatic Control*, **35**(10), 1118–1124, (1990).

[11] M. J. Kearns, Y. Mansour, and A. Y. Ng, 'A sparse sampling algorithm for near-optimal planning in large Markov decision processes', *Machine Learning*, **49**, 193–208, (2002).

[12] R. E. Korf, 'Real-time heuristic search', *Artificial Intelligence*, **42**, 189–211, (1990).

[13] N. Meuleau and P. Bourgine, 'Exploration of multi-state environments: Local measures and back-propagation of uncertainty', *Machine Learning*, **35**(2), 117–154, (1999).

[14] J. Pearl, 'On the nature of pathology in game searching', *Artificial Intelligence*, **20**, 427–453, (1983).

[15] M. L. Puterman, *Markov decision processes: discrete stochastic dynamic programming*, Wiley-Interscience, New York, 1994.

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

[17] R. Vanderbei, 'Optimal sailing strategies, statistics and operations research program, University of Princeton - http://www.sor.princeton.edu/ rvdb/sail/sail.html', (1996).

[18] G. Verfaillie, F. Garcia, and L. Péret, 'Deployment and maintenance of a constellation of satellites: a benchmark', in *Proc. ICAPS 03 Workshop on Planning under Uncertainty and Incomplete Information*, (2003).