RAIRO Operations Research
RAIRO Oper. Res. **40** (2006) 195–234
DOI: 10.1051/ro:2006019

INFLUENCE OF MODELING STRUCTURE IN PROBABILISTIC SEQUENTIAL DECISION PROBLEMS

Florent Teichteil-Königsbuch[1] and Patrick Fabiani[1]

Abstract. Markov Decision Processes (MDPs) are a classical framework for stochastic sequential decision problems, based on an enumerated state space representation. More compact and structured representations have been proposed: factorization techniques use state variables representations, while decomposition techniques are based on a partition of the state space into sub-regions and take advantage of the resulting structure of the state transition graph. We use a family of probabilistic exploration-like planning problems in order to study the influence of the modeling structure on the MDP solution. We first discuss the advantages and drawbacks of a graph based representation of the state space, then present our comparisons of two decomposition techniques, and propose to use a global approach combining both state space factorization and decomposition techniques. On the exploration problem instance, it is proposed to take advantage of the natural topological structure of the navigation space, which is partitioned into regions. A number of local policies are optimized within each region, that become the macro-actions of the global abstract MDP resulting from the decomposition. The regions are the corresponding macro-states in the abstract MDP. The global abstract MDP is obtained in a factored form, combining all the initial MDP state variables and one macro-state "region" variable standing for the different possible macro-states corresponding to the regions. Further research is presently conducted on efficient solution algorithms implementing the same hybrid approach for tackling large size MDPs.

Keywords. Probabilistic planning, dynamic programming, Markov decision processes, application to autonomous decision making.

Mathematics Subject Classification. 90C40, 68T37, 68T20, 68R05, 11Y05, 68R10

Received December 31, 2005.

[1] ONERA-DCSD, 2 Avenue Édouard-Belin, 31055 Toulouse, France;
`florent.teichteil@onera.fr; patrick.fabiani@onera.fr`

© EDP Sciences 2006

Article published by EDP Sciences and available at http://www.edpsciences.org/ro or http://dx.doi.org/10.1051/ro:2006019

Introduction

Automated sequential decision making under uncertainty is a crucial capability for the feasibility of many possible future applications in the field of autonomous robotics, such as the autonomous unmanned aircraft studied in the *ReSSAC (*`www.cert.fr/dcsd/RESSAC`*)* project at ONERA. The *ReSSAC* project at ONERA addresses the issue of planning under uncertainty for autonomous exploration or "search and rescue" aircraft. An exploration mission is composed of a problem of navigation in a partially known environment on the one hand, and a problem of online information acquisition and re-planning on the other hand. The mission and the flight plan cannot be completely and precisely defined in advance. Explored (or already known) regions of the environment may be traversable or not. Other regions require to be exhaustively or partially explored, mapped or searched for the presence of persons or objects. Such a mission can be partially pre-planned at a higher level of abstraction, as a sequence of mission phases, tasks, or macro-actions: for each mission phase, the system needs to achieve navigation and information acquisition goals, before proceeding with one of the possibly following phases. The intermediate goals to be achieved are pre-conditions of the following tasks as in classical deterministic planning domains. Yet each intermediate goal must be achieved by minimizing risks and costs and thus by optimizing the navigation and the action strategy under uncertainty. This short introduction introduces our motivations for this work, but the techniques presented next are far more general than our intended application and concern all kinds of sequential decision making problems under uncertainty.

Throughout the paper, we use a family of problems of different sizes and complexity, but all based on this kind of navigation-grid exploration-like problems, with weakly coupled regions and additional state variables (see Figs. 2, 3, 20). Small problems (Fig. 3) are used in order to explain the approach and larger ones in order to demonstrate the scalability of the combined approach that we propose in the end.

Markov Decision Processes (MDPs) [33] have become a popular stochastic framework in order to tackle sequential decision problems for autonomous agents which actions have uncertain effects. Due to these uncertainties, a sequential plan of action cannot guarantee to achieve the goals. Instead, a *policy* is optimized as a function giving for each possible state, the action to be performed next. Partially Observable MDPs (POMDPs) have also been widely studied in order to take into account observation uncertainties. Yet, existing algorithms to solve POMDPs [30] are still not of practical use in real world applications [10]. Some works propose to use structured approaches in order to solve POMDPs with larger state spaces [5]. Still for MDPs, classical exact algorithms based on stochastic dynamic programming on an explicitly enumerated state space are not effective enough for realistic applications that often have very large state spaces [8, 41].

This paper gives a discussion about the influence of the possible modeling choices, especially between the choice of a structured, rather than an enumerated, state space representation. After an introduction to related pieces of work in the literature in Section 1, a short presentation of the MDP formalism is first given in Section 2. In Section 3 the utility of using sparse matrices, *i.e.* graph based representations, for MDPs is shown to depend on the problem. The advantages and drawbacks of decomposition techniques for graph based MDPs are discussed in Section 4, where we also provide comparison tests between two classical decomposition algorithms. Alternatively, the state space can also be represented by using a set of state variables: a MDP using such a structured and compact representation is called a *factored MDP* (Sect. 5). Eventually, in Section 6, we propose a global approach that takes advantages of both decomposition and factorization techniques. This global approach is illustrated and discussed on the basis of a simple instance of our probabilistic exploration-like sequential decision problem. We then give a series of results obtained on problems of larger sizes, using more recently developed factored MDP optimization algorithms, in order to show how such an approach scales in larger state spaces. In Section 9, we give some conclusions on the respective advantages of the presented techniques and then some perspectives for further work on structured stochastic sequential decision problems.

1. Overview and related work

Decision-theoretic planning offers a very general framework for stochastic sequential decision problems of all kinds [29]. In Markov Decision Processes (MDPs) [33], the uncertain effects of actions are modeled in a decision-theoretic framework. Partially Observable MDPs (POMDPs) have also been widely studied in order to take into account observation uncertainties. Yet, the existing algorithms for solving POMDPs are still not of tractable use in practical applications, due to the complexity of the problem [10, 30]. It is possible to bypass the difficulty due to partial observability by augmenting the state representation with some memory of selected past events [17]. Classical MDP stochastic dynamic programming algorithms are based on an explicitly enumerated and unstructured state space. The size of the state space is an exponential function of the number of features that intervene in the problem description. The state enumeration itself may rapidly become intractable for realistic problems, or more generally the subsequent optimization computations cannot tackle the size of the problem [8].

A number of techniques have been proposed in recent years, in order to overcome the limitations of the enumerated states MDP framework. Such techniques include learning methods [3, 38], where the framework of stochastic sequential decision making is extended to the family of reinforcement learning problems. The control of both the computing cost and the approximation error are generally a concern, as in other approximating methods and algorithms [22, 37] that have been quite recently developed in order to speed up the optimization time or increase the size of the MDPs. Eventually, approximate linear programming algorithms allow to extend the model to hybrid MDPs, with both continuous and discrete state variables [21, 28]. We plan to integrate continuous variables in our autonomous agent planning problem in future work and to work more specifically on learning techniques.

In this article, we rather focus on approaches that try to exploit the natural structure of the sequential decision problems either by using compact factored representations with (more or less) correlated state variables [6–9], or by decomposing the enumerated state space into sub-regions [12, 24, 31, 36]. Such a decomposition enables a hierarchical approach, which appears to be more effective for problems of large size that have weakly coupled sub-regions. Other hierarchical approaches exist and have been developed for instance for decomposable reinforcement learning problems, such as MAXQ-Q learning in [14], or applied in a POMDP controller of "nursing assistant" robots [32]. The state abstraction approach proposed in [14], result in a problem solution very similar to the techniques described in this paper. Yet, we do not discuss here the possibility of generalized abstraction, nor the perspective of introducing a greater number of hierarchical levels as in [14]. The MAXQ approach seems compatible with our decomposition and factorization approach, and can be seen as a possible generalization. It however seems to require the design "by hand" of the MAXQ hierarchical abstraction graph, as a result from an analysis of the problem by the designer. We rather focus on the automatic exploitation of natural structures such as topological structures, which are known to be automatically computable. Our automatic abstraction process is based on a decomposition that shares some similarities with the approach proposed in [14]. As a matter of fact, the MAXQ decomposition of the value function could be introduced in our approach as a hierarchical generalization of our abstraction process. We could then integrate automatic MAXQ graph building tools. Besides, the use of automatic decomposition techniques allows us to reduce the number of possible values of the navigation state variables, and thus make it possible to efficiently apply factored MDP solution algorithms. Such factored algorithms could be easily combined with or replaced by a Q learning algorithm. The combination of both approaches should be studied in more details in further work. The three family of possible MDP models, or MDP approach, that are mainly addressed in this paper are graph representation, factorization and decomposition.

Graph representation: while classical MDP algorithms use stochastic transition matrices, some approaches such as in [31, 36] suppose that the state space is represented in the form of a graph, whose nodes are the enumerated states of the MDP and whose edges are the stochastic transitions of non-zero probability between the states. The corresponding transition matrices (which store the probabilities and rewards of the problem) are sparse.

Factorization: [8] and [9] put a more general focus on the exploitation of the structure of stochastic planning problems that are described with symbolic features. The use of compact factored representations, instead of enumerated state spaces, remains possible with correlated state variables [6]. The SPUDD library [25] proposes to use decision diagrams, borrowed from the Model Checking community, and to apply it to structured stochastic planning problems. Only the value iteration scheme was implemented with

Decision Diagrams: we had to implement SPUDD-like policy iteration dynamic programming. [7] show how to exploit such a representation by reusing symbolic planning schemes borrowed from recent STRIPS planning work.

Decomposition: [12] or [24] propose a decomposition of the state space in sub-regions, which enables a hierarchical solution. Computations about the interstate connectivity within an enumerated navigation state space as in [36] can provide a topological decomposition at a higher level of abstraction: the MDP is splitted into a set of regions, each region being a local MDP. The work by [31] and [36] show the efficiency of such an approach applied to decomposable navigation grid MDPs. As a matter of fact, the global MDP is then abstracted into a symbolic factored MDP described by a new feature stating for the regions. We exploit this kind of automatic problem abstraction process in our work, contrary to the work by [27], where a partition is used to solve a factored MDP.

Approximate solution algorithms [37] have also been proposed on the basis of factored or decomposed representations. A number of approximation techniques propose the use of parameterized or structured value functions (see [14] for a Q learning approach) or policies: both value functions approximation and policy approximation are combined in [22], using the max-norm projection, which allows to solve large scale factored MDPs. It is noticeable that our global optimal approach is able to solve problems as large as the ones solved in [22], but on MDPs of different natures: our problems combine boolean variables and variables with large ranges of values (navigation variables), whereas the MDPs solved in [22] are based on boolean state variables.

Our algorithms have been developed on the basis of recent work on symbolic heuristic search algorithms for MDPs: the LAO* algorithm generalizing AO* [23] and the LRTDP algorithm [4] generalizing RTDP. Both heuristic schemes lead to bound the explored state space before convergence and apply value iteration within a more focused region. Their respective application in [19] and [1] show that they improve the efficiency of value iteration dynamic programming for structured MDPs. Different implementations of these heuristic search value iteration algorithms were independently compared on navigation grid MDPs in [23] and in [4]: these comparisons tend to show that LRTDP outperforms LAO*. Symbolic versions have been developped for symbolic stochastic planning problems, using factored state space representations: sRTDP is a symbolic version of RTDP value iteration and sLAO* the symbolic counterpart for LAO* value iteration. The results presented in [23] and [20] show that sRTDP has a better on-line behavior than sLAO*, in the sense that it faster gives a good quality solution. On the other hand, sLAO* shows a better off-line convergence efficiency, in the sense that it faster converges towards the optimal solution. Our work in this respect is based on the combination of both decomposition and factorization techniques which allows us to automatically abstract the MDP. This operation is quite powerful for problems combining navigation and task planning. We propose the implementation of both sLAO*-inspired and sRTDP-inspired heuristic search schemes with a policy iteration dynamic programming algorithm. Preliminary experimental results show how these algorithms can scale on problems of larger sizes.

2. Markov decision processes

2.1. Classical model

A MDP [33] is a controlled Markov chain (stochastic automaton) where the choice of an action in a given state at a given time point results in a stochastic transition to an uncertain state, with transition probabilities depending on both the action and the state. The choice of a given action may furthermore lead to a reward or a cost. A control strategy π is a function from the state space to the action space, that associates to each state the choice of an action: it is also called a *policy*. The Markov property means that the probability of arriving in a particular state after an action only depends on the previous state of the chain and not on the entire states history.

Definition 2.1 (Markov decision process)**.** A MDP is a tuple $\langle S, A, T, R\rangle$ where:

- S is the set of states;
- A is the set of actions;
- $T : \begin{array}{lll} S\times A\times S & \longrightarrow & \mathbb{R} \\ (s,a,s') & \longmapsto & T(s,a,s') = P(s'|a,s) \end{array}$
  is the probability transition function;
- $R : \begin{array}{lll} S\times A\times S & \longrightarrow & \mathbb{R} \\ (s,a,s') & \longmapsto & R(s,a,s') = R(s'|a,s) \end{array}$
  is the reward function.

$\mathcal{T}$ and $\mathcal{R}$ values depend on the starting state, the ending state and the chosen action (probabilities and rewards are stored in matrices or tables).

The *infinite horizon discounted* optimization criterion is most frequently used. It consists in maximizing the value function $V^\pi = E\left(\sum_{t=0}^{\infty} \beta^t\, r_t^\pi\right)$, *i.e.* the infinite sum of expected rewards r_t^π obtained while applying the policy π, discounted by a factor $0 < \beta < 1$ at each time step. β guarantees the sum's convergence, but can also be interpreted in the sense that $1-\beta$ is an uncontrolled stopping probability between two time points. It has been shown (see [33]) that the optimal policy is a *deterministic Markovian policy* $\pi \in \Pi^{DM}$ (instead of a stochastic history-dependent policy) so that solving a MDP amounts to solving the problem $\mathcal{P}$:

$$\max_{\pi\in\Pi^{DM}} \quad \sum_{t=0}^{\infty} \beta^t\, r_t^\pi \ ,\ 0<\beta<1 \qquad (\mathcal{P}).$$

The *discount factor* β guarantees that $\mathcal{P}$ has a solution. Depending on the applications, the discount factor can have a particular interpretation. For instance in robotics, $1-\beta$ can represent the probability of a complete system failure, terminating the mission, between two time points. More generally, β^t can be interpreted as a unit of reward gained at time t.

2.2. Solution algorithms

The solution of the problem $\mathcal{P}$ is obtained by stochastic dynamic programming [2]. Putting apart the direct application of linear programming techniques, which rapidly cannot face the complexity burden, there exist two main classes of iterative algorithms: *value iteration* and *policy iteration*. All these algorithms iteratively improve a policy π or a value function V defined on the state space S, $V_\pi(s)$ being for instance the discounted infinite sum of expected rewards applying the policy π an infinite number of times starting from the initial state $s \in S$.

2.2.1. *Value iteration*

Different value iteration algorithms exist, that are all based on Bellman's optimality equations, whose fixed point is the optimal value function V^*:

$$\forall\, s \in S\ ,\ V^*(s) = \max_{a\in A} \sum_{s'\in S} T(s,a,s') \cdot (R(s,a,s') + \beta V^*(s'))\,. \tag{1}$$

The optimal policy π^* corresponds in the previous equation to the actions that maximize the value function. Value iteration algorithms iteratively compute the fixed point V^* as the limit of the sequence of value functions defined by:

$$\begin{cases} V_0 = 0, \text{ and } \forall\, s \in S\ , \\ V_{n+1}(s) = \max_{a\in A} \sum_{s'\in S} T(s,a,s') \cdot (R(s,a,s') + \beta V_n(s'))\ ,\ n \geq 0. \end{cases}$$

Algorithm 2.2 (value iteration)**.**

```
***Require:*** $\epsilon > 0$ *and* $0 < \beta < 1$
  $V_0 \leftarrow 0$
  $n \leftarrow 0$
  ***repeat***
    ***for*** $s \in S$ ***do***
      $V_{n+1}(s) = \max_{a\in A} \sum_{s'\in S} T(s,a,s') \cdot (R(s,a,s') + \beta V_n(s'))$
    ***end for***
    $n \leftarrow n+1$
  ***until*** $\|V_{n+1} - V_n\| < \epsilon$
  ***for*** $s \in S$ ***do***
    $\pi(s) \leftarrow \arg\max_{a\in A} \sum_{s'\in S} T(s,a,s') \cdot (R(s,a,s') + \beta V_n(s'))$
  ***end for***
***Ensure:*** $\|V_n - V^*\| < \epsilon$
```

The most frequently used value iteration algorithm is the *Gauss-Seidel* algorithm [33] for which $V_{n+1}(s)$ replaces $V_n(s)$ as soon as the new value is computed for s. *Asynchronous dynamic programming* algorithms [38] rather update the value of randomly chosen states at each iteration, or along a simulated trajectory. Eventually, the actions and the strategies that are dominated, can be eliminated during the value iteration steps since they cannot participate in any optimal strategy [8].

2.2.2. *Policy iteration*

In the policy iteration scheme, the policy is globally assessed on all states and on the infinite horizon, and it is improved locally, state by state, at each iteration. The value of a policy π is the unique fixed point solution of Bellman's optimality equations:

$$\forall\, s \in S\;,\; V^{\pi}(s) = \sum_{s' \in S} T(s, \pi(s), s') \cdot (R(s, \pi(s), s') + \beta V^{\pi}(s'))\,. \tag{2}$$

Algorithm 2.3 (policy iteration)**.**

```
Require: $\epsilon > 0$ and $0 < \beta < 1$
  Initialize $\pi \in \Pi^{DM}$
  $n \leftarrow 0$
  repeat
    Solve: $\forall\, s \in S\;,\; V^{\pi}(s) = \sum_{s' \in S} T(s, \pi(s), s') \cdot (R(s, \pi(s), s') + \beta V^{\pi}(s'))$
    for $s \in S$ do
      $\pi_{n+1}(s) = \arg\max_{a \in A} \left\{ \sum_{s' \in S} T(s, a, s') \cdot (R(s, a, s') + \beta V_n(s')) \right\}$
    end for
    $n \leftarrow n + 1$
  until $\pi_n = \pi_{n+1}$
Ensure: $\pi_n = \pi^*$
```

A policy iteration algorithm converges in fewer iterations than a value iteration algorithm, because the optimal policy (made of discrete choices) stabilizes sooner than the corresponding optimal value function, which has an asymptotic convergence to the fixed point in the *value iteration* schemes. The rate of asymptotic convergence of V may also depend on the value of the discount factor β.

On the other hand, in the policy iteration scheme, each iteration is much more computationally costly than in the value iteration algorithm, due to the linear equation system solution. Exactly solving this linear system is not always feasible in a reasonable time.

In the *modified policy iteration* algorithm [33] the value of the current policy, at each iteration, is approximated by applying a number of steps of value iteration without changing the policy. In general, this is the most effective algorithm in order to solve classical MDPs of "reasonable" size [8].

2.3. Limits of the classical model

The classical MDP framework does not take into account the particular structure of the state space. First, all transitions between states are stored even if most of these have a zero probability. In the classical instance of the parking

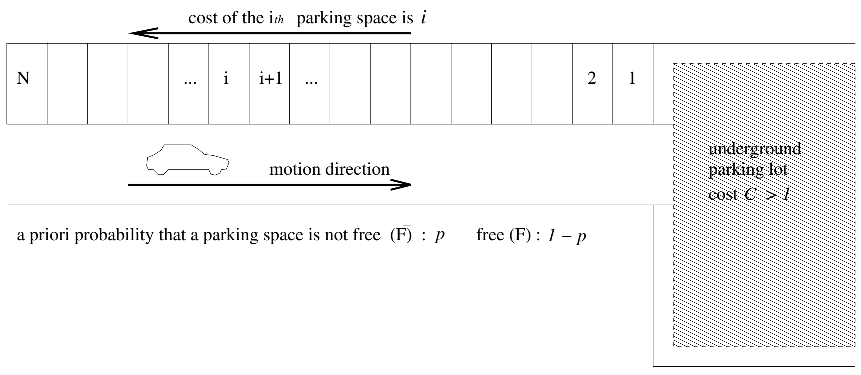

cost of the ith parking space is i
N
...
i
i+1
...
2
1
motion direction
underground parking lot cost C > 1
a priori probability that a parking space is not free ($\bar{F}$) : p free (F) : 1 − p

FIGURE 1. Parking lot problem [33]: the driver would like to park as close as possible (lowest possible cost) to a shop being situated above an underground parking lot of cost C, with $1 < C \leq N$. Nevertheless, the closer he gets to the shop's entry, the more risk he takes not to find a free parking space outside the underground parking lot.

lot problem shown in Figure 1, the driver facing the parking space i can only go forward to the parking space $i+1$ (in one move). The transitions from a state $(i,\cdot)$ to states $(j,\cdot)$ where $j \neq i+1$ are impossible. Therefore, updating the value function with Bellman's equations requires to sum over a number of terms that are predominantly equal to zero, which uselessly increases the computation burden of the solution algorithms.

Moreover, classical MDP algorithms 2.2 and 2.3 perform their value or policy iteration steps by updating all the states in plain listing order, without any consideration of the connection structures between states. For instance, in Figure 2 the value function is updated in the second region before it has converged in the first region whereas its value in the second region strongly depends on its value in the first region. On the contrary, grouping the states into weakly coupled regions can, for instance, allow to update quite independently each set of densely connected states (within each region) before propagating the result to the other regions. Such an approach takes advantage of the "topology" of the state space and seems particularly well-fitted to navigation-grid problems, whenever it is made up of weakly coupled regions where the MDP can be locally solved.

Eventually, the state space is often *factored*, in the sense that it is described by a set of *state variables* whose possible combinations of values correspond to the possible states. The classical MDP model is not well-adapted to this structure of the state space since the number of enumerated states grows exponentially with the number of state variables.

Let us consider an instance of an exploration problem, that is to say: a navigation problem with intermediate and final goals located in different regions of the environment. Some regions have to be exhaustively or partially explored, mapped

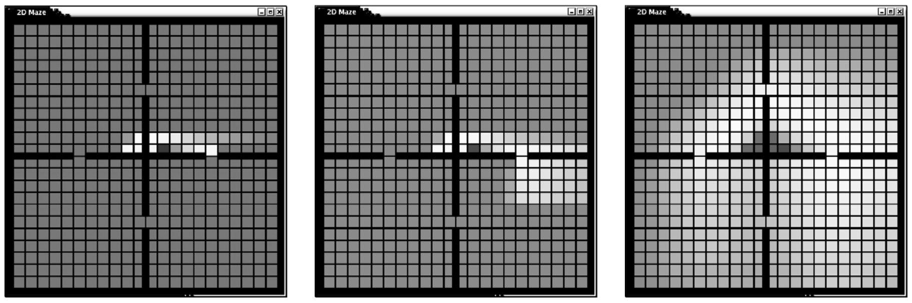


FIGURE 2. Propagation of the value updates through three successive iterations of the value iteration algorithm over a grid made up of 4 weakly coupled regions.

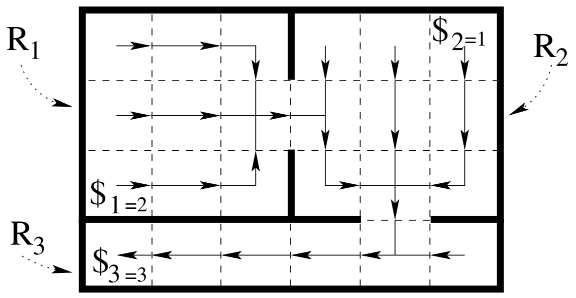


FIGURE 3. Navigation MDP: 3 regions and 3 possible rewards.

or searched for the presence of persons or objects before trying to achieve other goals. Information acquisition tasks may be required for the achievement of some intermediate goals of the mission, and may also impact on the subsequent tasks or navigation actions. Such missions can be modelled at a higher level of abstraction, as a sequence of mission phases, tasks, or macro-actions: for each mission phase, the system needs to achieve navigation and information acquisition goals, before proceeding with one of the possibly following phases. The order in which the intermediate sub-goals or tasks must be achieved can be constrained or not.

These intermediate goals, or sub-goals, can be seen as pre-conditions of other tasks as in classical deterministic planning domains. Yet each intermediate goal must be achieved by minimizing risks and costs and thus by optimizing the navigation and the action strategy under uncertainty. Figure 3 shows a exploration-like MDP. There clearly are three regions R_1, R_2, R_3, each one associated with a subgoal, *i.e.* each region contains a reward: $\$_1 = 2$ in R_1, $\$_2 = 1$ in R_2 and $\$_3 = 3$ in R_3. Taken as a classical 4×6 navigation grid MDP, the optimal strategy is: go to the highest reward $\$_3 = 3$ in R_3 as shown in Figure 3. Contrary to the navigation grid MDP in Figure 3, our exploration planning problem contains a number of preliminary tasks, associated with rewards, each of which can be obtained once in turn, no matter the order, before reaching the goals, and final rewards. To model this, we need to introduce binary variables O_1, O_2 and O_3 equal to 1 or 0 depending on whether the rewards $\$_1$, $\$_2$ and $\$_3$ have already been obtained or not. Our exploration planning problem is thus factored, and difficult to represent graphically. It still corresponds to the same navigation grid, but the size of the state space is multiplied by 2^3, as it is exponential in the number of state variables. The level of energy autonomy is another (possibly continuous) state variable A: the size of the state space of our problem is multiplied by n_A the number of possible values of variable A. Flight is thus limited by energy autonomy, and the aircraft can *Abort* its mission, if it is running short of fuel, or if the goals are reached. On the other hand, we have no transition costs: rotorcraft always have to consume energy to fly (other models are possible, *e.g.* putting a higher cost on hovering than on translation flight). *Abort* corresponds to a "return to base", or to a security or emergency landing of the aircraft.

Enumerating the possible states of such a factored MDP is possible with this simplistic instance, but it seems a dull piece of work that not only duplicates computing efforts but also annihilates the possible benefits that could be driven from the structure of the problem.

This example points out how difficult it can be to model and solve more realistic problems with the classical MDP model. In the next sections, we present alternative models for probabilistic sequential decision problems that make it possible to take advantage of the state space structure. For instance, the state space of the above *exploration-like sequential decision problem* derived from the MDP in Figure 3 presents two interesting features: on the one hand, the state space is naturally factored and on the other hand, the navigation-grid subspace (possible positions of the agent) can be decomposed in three weakly coupled navigation regions. This leads us to propose a new hybrid model, that would apply in problems for which the state space can be decomposed into different state components that each have a particular structure: *e.g.* here a topological structure for navigatio, and a factored structure for the higher level mission description (like in the taxi example in [14]).

3. Graph representations

3.1. Sparse matrices

In the classical MDP framework, the problem model (transition probabilities, rewards) is stored in matrices or tables, so that the computational efficiency of the algorithms depends on the type of matrices that are used. In general, the data have no particular structure, which means that the distribution of non-zero elements in the matrices presents no specificity.

There exist two different ways for storing these matrices, depending on the density of the non-zero elements. When the model data has few zero elements (*dense*) it is reasonable to use *dense* matrices, where each null term takes as much

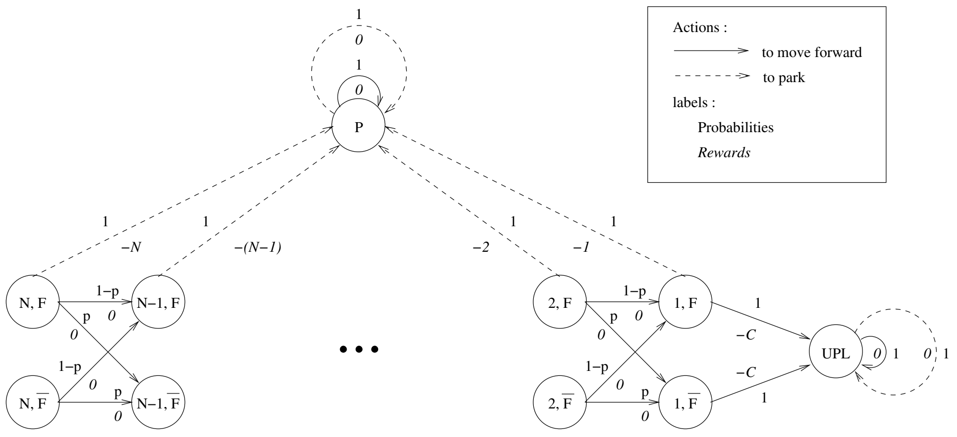


FIGURE 4. Graph-like MDP definition of the parking lot problem depicted in Figure 1.

memory space as any non-null term. On the contrary, when the model data comprises a significant number of zero elements, it more efficient to use *sparse* matrices [15] where only non-zero elements are stored in memory.

In the MDP community, sparse matrices are known to be generally more efficient. MDPs (*e.g.* navigation-grid MDPs) are often based on a graph where the nodes are the states and the edges correspond to transitions with non-zero probability. MDP transition probabilities can be equivalently represented by graphs or by sparse matrices [35]. For instance, the graph-like MDP definition of the classical parking lot problem (see [33] and Fig. 1) is depicted in Figure 4. Transitions are labeled by the corresponding probabilities and rewards. In this problem, the definition in the form of a graph is much more compact than the definition based on dense matrices: $5N + 6$ transitions are stored in the graph against $8(N + 1)^2$ transitions in the dense matrices (2 actions and $2(N + 1)$ states). The same property holds for navigation grid MDPs (Fig. 3) since, from any initial position, there are at most 8 possible transitions with non-zero probabilities.

For each *sparse* matrix, three arrays are necessary: the first array contains the non-zero terms and the two others respectively, the rows and columns indices corresponding to these non-zero terms. There exist direct [16] or iterative [35] algorithms for linear algebra operations on sparse matrices. The computation time required for a sparse matrix operation is generally proportional to the number of arithmetic operations per non-zero term times the total number of non-zero terms. It is rather proportional to the product of matrices dimensions for the same operation on a dense matrix. When the MDP transition structure is *dense*, the handling of the data structures specific to the sparse matrices may increase both the memory space and the computing time spent for each operation, compared with the case of the same operation performed on dense matrices.

In other words, the complexity of linear algebra algorithms can theoretically be worse with sparse matrices than with dense matrices when there are many non-zero elements in the transition probabilities. Thus, the use of sparse matrices should be limited to the case when the number of non-zero terms is significantly smaller than the product of matrices dimensions.

As far as our exploration-like MDP models are concerned, some of their state components are naturally factored. Factored MPDs are often more likely to present dense transition probabilities matrices. For instance, the state spaces of the "SysAdmin" factored MDP instances solved in [22] are generated by the cartesian product of the possible values of a number m of boolean state variables that can change their values simultaneously (and not independently) at each time step. The matrix of transitions probabilities between these factored states can clearly be dense in many instances.

3.2. Can sparse matrices be a bad choice?

Surprisingly enough, we have not found in the literature any comparative study on the question of whether to use sparse matrices or not, depending on the MDP transition probabilities structure. In order to assess the possible loss of efficiency using sparse matrices on dense transition structures, we have performed two different kinds of tests. Our tests consist in comparing the computation time with respectively dense and sparse matrices for an operation of multiplication.

In the first test, the percentage of non-zero terms is fixed, and the dimensions of the matrices vary, which is representative of navigation-grid MDPs. A low density of non-zero elements easily confirms the advantage of using sparse matrices in that case, as shown in Figure 5a (on the left):

$$\forall\, 0 \le i < n \;,\; \begin{cases} M_1(i,0) = M_1(i,n-1) = \alpha \\ M_2(0,i) = M_2(n-1,i) = \alpha, \end{cases} \quad \alpha \neq 0.$$

Let ν_1 and ν_2 denote the respective density of non-zero terms of $M_1^{m\,n}$ and $M_2^{n\,l}$, the computation time C of the operation of multiplication is as follows:

- for dense structures: time $C \sim mln^2$;
- for sparse structures: time $C \sim \nu_1\nu_2 mln^2$.

In our test, the density of non-zero terms for each matrix is $\nu_1 = \nu_2 = \frac{2n}{n^2} = \frac{2}{n}$ and the matrices are square ($m = l = n$) so that the gain in computation time is:

$$\frac{n^4 - \frac{2}{n}\frac{2}{n}}{n^4} = 1 - \frac{4}{n^2}\cdot$$

In the second test, the matrices are of fixed size (10 000 elements) and the densities in non-zero elements varies:

$$\forall\, 0 < \nu \le 1 \;,\; M_1(i,j) = M_2(i,j) = \begin{cases} \alpha & \text{if } i\nu \in \mathbb{N} \text{ and } j\nu \in \mathbb{N} \\ 0 & \text{else,} \end{cases} \quad \alpha \neq 0.$$

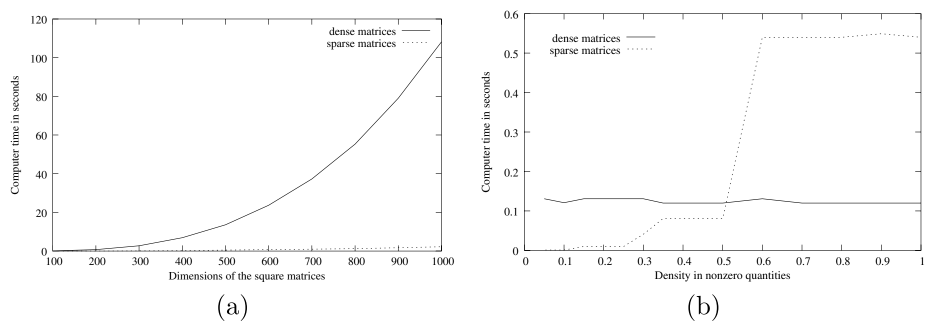


FIGURE 5. Multiplication on dense and sparse matrices: (a) with a fixed low density of non-zero terms and (b) with a varying density of non-zero terms.

Figure 5 shows that using sparse matrices can become counterproductive when the density of non-zero elements increases. Using sparse matrices is not necessarily a good choice, especially when the density of non-zero elements is not known, and most probably for factored MDPs .

A sparse graph-based representation of the MDP state space is worthwhile when the number of non-zero probability transitions is negligible compared to $|A| \cdot |S|^2$.

3.3. Graph-like representation for navigation MDPs

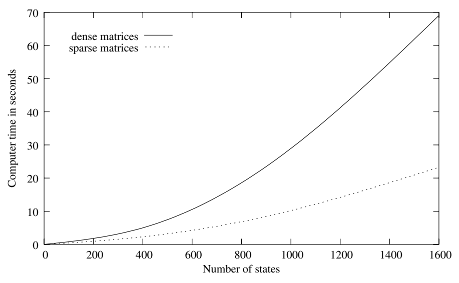
The above property holds for navigation-grid MDPs since the density of non-zero probability transitions is, in this case:

$$\frac{4 \times 8 \times lw}{4 \times (lw)^2} = \frac{8}{lw},$$

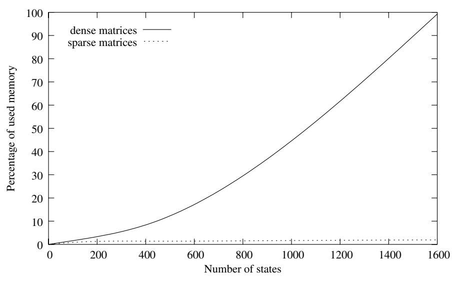
where l and w are the dimensions of the grid.

Figure 6 shows a comparison of the computation time and the memory space spent for the solution of MDPs of varying size with respectively dense and sparse matrices. The gain in terms of memory is more significant than the gain in computation time but the solution of large realistic problems is more limited by memory constraints more than by time constraints. On the other hand, both memory space and time constraints are important for embedded applications.

As a preliminary conclusion, it is true that sparse representation can prove very efficient, but they cannot be as efficient for all the state space components. We therefore have to combine them with something else. Although a graph-based representation of MDPs can be particularly well-adapted when the density of non-zero probability transitions is small, some large size problems have lead to develop new approaches, especially since the simple enumeration of the possible states in the state space is not tractable in some realistic problems.



(a)



(b)

Figure 6. Comparison of (a) the cpu time and (b) the memory space spent for the solution of a MDP navigation problems based on a square grid with various dimensions.

4. State space decomposition

The decomposition of MDPs consists in grouping the states of an entirely enumerated state space into weakly coupled regions (macro-states). This operation can be performed automatically using *e.g.* graph decomposition techniques [36]. If the state space is built on the basis of a map given as a numerical terrain model, then it can be partitioned with using computational geometric techniques. The partition should minimize the number of transitions between different regions. For each region, the states from which it is possible to reach another region are called as *exit states* and those where it is possible to arrive from another region are *entrance states*. Thus, the partition minimizes the transitions between *exit* and *entrance states*. An abstract MDP is built, whose (macro-)states are the regions stemming from the decomposition, whose (macro-)actions are local policies defined in each region and whose (macro-)transitions are computed from these local policies.

Generally speaking, the MDP decomposition makes it possible to optimize the global solution on the basis of local solutions obtained for smaller size weakly coupled sub-problems. This is proved to be more efficient on large size problems that can be splitted into such weakly coupled sub-regions, but not an all problems. The state space decomposition globally adds to the complexity of the dynamic programming solution algorithms. As a result, classical iteration algorithms that work on enumerated states can still be quicker in the end on some counter-example highly coupled problems. Factored MDPs are more likely to present a more important coupling.

As far as the on-line behavior of the approach is concerned, a change of the environment model in one local region can easily be dealt with locally in a decomposition-based approach, whereas it requires to start again with a complete re-optimization of the global policy for the whole model with classical algorithms.

Our MDPs decomposition framework takes advantages from two different state space decomposition models from [24] and [12]. We have tested two different algorithms [24, 31] to generate local policies whose results are next compared.

4.1. Decomposing the state space into regions

The decomposition of a MDP is based on a partition of its state space into non empty, distinct and complementary regions.

Definition 4.1 (state space decomposition in regions)**.** A decomposition Π in regions of a MDP $\langle S, A, T, R\rangle$ is a partitioning of the state space such that Π is the set of regions:

$$\exists\, n \in \mathbb{N}\ ,\ \Pi = (R_1, \ldots, R_n) \text{ with } \begin{cases} S = \bigcup\limits_{i=1}^{n} R_i \\ \forall\, i \neq j\ ,\ R_i \bigcap R_j = \emptyset. \end{cases}$$

For each region $\mathcal{R} = R_i$, we can identify the entrance and exit states of the region that play an important role in the different models and algorithms of MDP decomposition.

Definition 4.2 (peripheral states of a region $\mathcal{R} = R_i$)**.** Let R_i be a specific region of a decomposition of a MDP $\langle S, A, T, R\rangle$. The periphery $Per(R_i)$ of the region R_i is the union of the entrance periphery $EPer(R_i)$ and exit periphery $XPer(R_i)$:

- $EPer(R_i) = \{s \in R_i \;/\; \exists\, s' \in S - R_i\ ,\ \exists\, a \in A\ ,\ T(s', a, s) \neq 0\}$;
- $XPer(R_i) = \{s \in S - R_i \;/\; \exists\, s' \in R_i\ ,\ \exists\, a \in A\ ,\ T(s', a, s) \neq 0\}$;
- $Per(R_i) = EPer(R_i) \bigcup XPer(R_i)$.

Since these regions form a partition of the state space, the entrance periphery of a region matches the exit periphery of other regions and vice versa. The set of entrance peripheral states and the set of exit peripheral states are identical and only depend on the partitioning of the state space.

$$\bigcup_i EPer(R_i) \;=\; \bigcup_i XPer(R_i).$$

4.2. Abstract MDP

Macro-states of a new abstract MDP are then defined that can either be defined as the peripheral states of each region [24], that is to say $\bigcup_{\mathcal{R}\in\Pi} Per(\mathcal{R})$, or be defined as aggregate states, grouping together the states of each region [12] as shown in Figure 7.

We do not use the first model (whose computed policies may be sub-optimal) mainly because it does not account for local behaviors consisting in staying in a region so as to collect an inner positive reward before leaving: an absorbing state per region is required to account for such behaviors. From that point of view, our decomposition algorithm is based on the techniques proposed in [24], but some of its principles, borrowed from [12] and [31], remain closer to classical MDP solution algorithms.

Let us suppose that we have obtained a number of local policies for each region. These local policies correspond to stochastic transitions between the aggregate

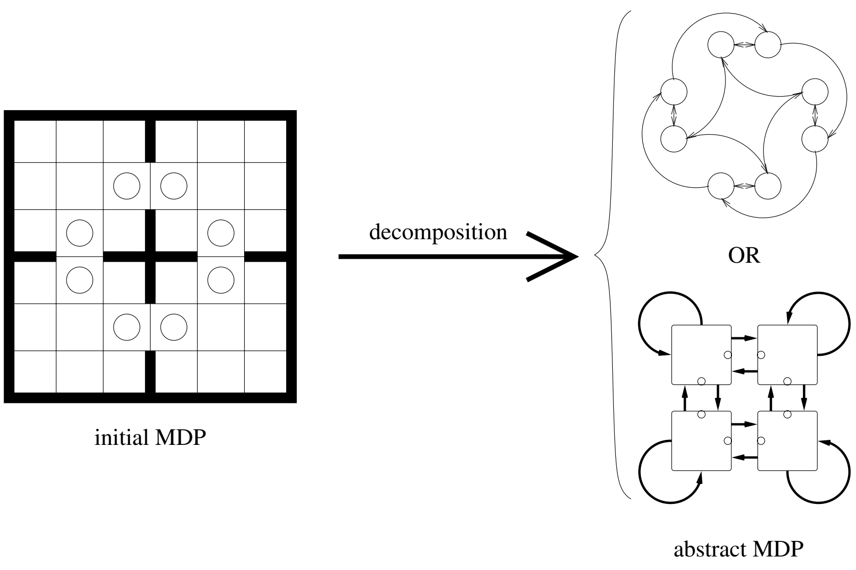


FIGURE 7. A 4 regions MDP decomposed and "abstracted" in two different ways.

states (regions) of the abstract MDP and can therefore be identified to *macro-actions* in this abstract MDP.

Definition 4.3 (abstract MDP [12])**.** Let Π be a decomposition of a MDP $\langle S, A, T, R\rangle$ and $\bigcup_{\mathcal{R}\in\Pi}\{\pi_1^{\mathcal{R}},\dots,\pi_{m_{\mathcal{R}}}^{\mathcal{R}}\}$ some local policies generated for each region of the decomposition. The abstract MDP $\langle S', A', T', R'\rangle$ is defined by:

- $S' = \bigcup\limits_{\mathcal{R}\in\Pi}\{\mathcal{R}\}$;
- $A' = \bigcup\limits_{\mathcal{R}\in\Pi}\{\pi_1^{\mathcal{R}},\dots,\pi_{m_{\mathcal{R}}}^{\mathcal{R}}\}$;
- $\forall \mathcal{R}, \mathcal{R}' \in S', \forall \pi_j^{\mathcal{R}} \in A'$, $T'(\mathcal{R}, \pi_j^{\mathcal{R}}, \mathcal{R}')$ and $R'(\mathcal{R}, \pi_j^{\mathcal{R}}, \mathcal{R}')$ depend on the discounted macro-transition model.

Building the abstract MDP also requires, for each region $\mathcal{R}$ and each local policy $\pi_j^{\mathcal{R}}$ the calculation of the macro-probabilities $T'(\mathcal{R}, \pi_j^{\mathcal{R}}, \mathcal{R}')$ and the macro-rewards $R'(\mathcal{R}, \pi_j^{\mathcal{R}}, \mathcal{R}')$. Our discounted macro-transition model is inspired from [12] for the macro-probability model and from [24] for the macro-reward model. Both are described next.

4.3. Discounted macro-transition model

The macro-probabilities of transition are the limit of the probabilities of moving (after an infinite number of time periods) from somewhere in a region $\mathcal{R}$ to the entrance states of a region $\mathcal{R}'$. These macro-probabilities are computed for each starting region while supposing that the possible starting states in this region all have the same *a priori* probability of occurrence as a result of past actions.

- $T'(\mathcal{R}, \pi_j^{\mathcal{R}}, \mathcal{R}') = \dfrac{1}{|\mathcal{R}|} \sum_{s' \in Xper(\mathcal{R}) \cap \mathcal{R}'} \sum_{s \in \mathcal{R}} T(s, \pi_j^{\mathcal{R}}, s')$;
- $R'(\mathcal{R}, \pi_j^{\mathcal{R}}, \mathcal{R}') = \dfrac{1}{|\mathcal{R}|} \sum_{s' \in Xper(\mathcal{R}) \cap \mathcal{R}'} \sum_{s \in \mathcal{R}} T(s, \pi_j^{\mathcal{R}}, s') \cdot R(s, \pi_j^{\mathcal{R}}, s')$.

$T'(s, \pi_j^{\mathcal{R}}, s')$ and $R'(s, \pi_j^{\mathcal{R}}, s')$ can be iteratively computed using Gauss-Seidel value iteration algorithmic steps [33] since they are solutions of equations that are very similar to Bellman's optimality equations:

- $T(s, \pi_j^{\mathcal{R}}, s') = \sum_{s'' \in \mathcal{R} \cup Xper(\mathcal{R})} T(s, \pi_j^{\mathcal{R}}(s), s'') \cdot A_{s'', \pi_j^{\mathcal{R}}, s'}$

$$\text{with } A_{s'', \pi_j^{\mathcal{R}}, s'} = \begin{cases} T(s'', \pi_j^{\mathcal{R}}, s') & \text{if } s'' \in \mathcal{R} \\ 0 & \text{if } s'' \in Xper(\mathcal{R}) - \{s'\} \\ 1 & \text{if } s'' = s'; \end{cases}$$

- $R(s, \pi_j^{\mathcal{R}}, s') = \sum_{\substack{s'' \in \mathcal{R} \cup \{s'\} \\ T(s'', \pi_j^{\mathcal{R}}, s') \neq 0}} T(s, \pi_j^{\mathcal{R}}(s), s'') \cdot B_{s, s'', \pi_j^{\mathcal{R}}, s'}$

$$\text{with } B_{s, s'', \pi_j^{\mathcal{R}}, s'} = \begin{cases} R(s, \pi_j^{\mathcal{R}}(s), s'') + \beta R(s'', \pi_j^{\mathcal{R}}, s') & \text{if } s'' \in \mathcal{R} \\ R(s, \pi_j^{\mathcal{R}}(s), s') & \text{if not.} \end{cases}$$

4.4. Generating local policies with local MDPs

Local policies can be generated at least in two different ways according to the advantage that we want to draw from the MDP decomposition.

On the one hand, the decomposition could aim at reducing the complexity for globally solving MDPs with large state spaces. In this case, the computation of a set of local behaviors is not a goal in itself. Each local optimal policy in each region can be optimized:

- either through local iterations interleaved with the global iterations on the abstract MDP [12];
- or by locally solving the initial MDP for a set of possible peripheral values before using these solutions to obtain the solution of the global abstract MDP [24].

On the other hand, it could be interesting to deliberately generate a large set of local policies for each region so as to constitute a database of macro-actions, *i.e. local behaviors* corresponding to possible *actions* at the level of the resulting abstract MDP. Such a set of local behaviors can be computed in order to be later used within an extended abstract MDP model, whose state space would for instance include different other state variables in addition to the regions of the decomposition. In that case, the optimality of the global solution should obviously be examined carefully. According to [24], local policies can be computed by solving local MDPs that are parameterized by the peripheral values $(\lambda(s))_{s \in Xper(\mathcal{R})}$ on peripheral states, whose combination affect the optimal local policy (Fig. 8).

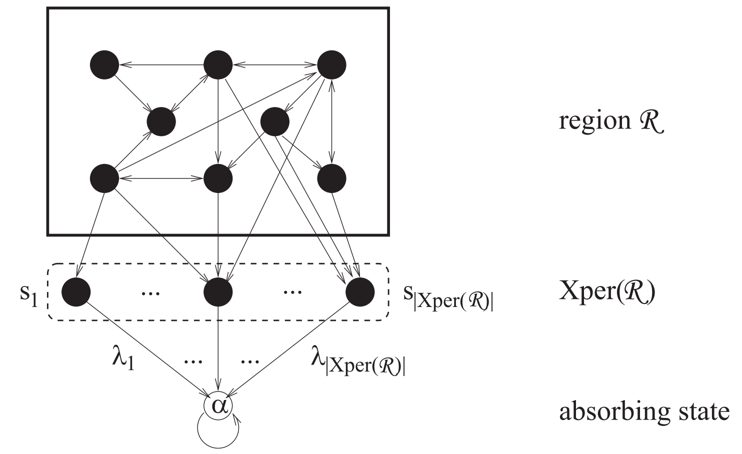


FIGURE 8. Local MDP of a region $\mathcal{R}$.

Each combination of peripheral values, taken on a set of possible exit states, corresponds to a local policy that forces the agent to leave the region *via* the very same combination of exit states.

Definition 4.4 (Local MDP)**.** Let $\mathcal{R}$ be a region in a decomposition Π of a MDP $\langle S, A, T, R\rangle$. The local MDP $\langle S', A', T', R'\rangle^{(\lambda(s))_{s\in Xper(\mathcal{R})}}$ of the region $\mathcal{R}$ is defined by:

- $S' = \mathcal{R} \cup Xper(\mathcal{R}) \cup \{\alpha\}$, where α is an absorbing state;
- $A' = A$;
- $T(s,a,s') = \begin{cases} T(s,a,s') & \text{if } (s,s') \in \mathcal{R} \times (\mathcal{R} \cup Xper(\mathcal{R})) \\ 1 & \text{if } (s,s') \in (Xper(\mathcal{R}) \cup \{\alpha\}) \times \{\alpha\}; \end{cases}$
- $R(s,a,s') = \begin{cases} R(s,a,s') & \text{if } (s,s') \in \mathcal{R} \times (\mathcal{R} \cup Xper(\mathcal{R})) \\ \lambda(s) & \text{if } (s,s') \in Xper(\mathcal{R}) \times \{\alpha\} \\ 0 & \text{if } (s,s') \in \{\alpha\}^2. \end{cases}$

For each region, it is possible to generate a limited set of local policies corresponding to chosen behaviors, such as "leaving the region" *via* some chosen exit states: this can be obtained by solving the local MDP for a combination of attractive peripheral values on the corresponding exit states. For instance, it could be sufficient to generate as many local policies as they are ways of leaving the region, that is to say: $\sum_{\mathcal{R}\in\Pi} 2^{|\mathcal{R}|}$ local policies. This gives no guaranty that such local policies can participate in any globally optimal policy, because the peripheral values that are chosen in order to obtain these local behaviors may not correspond to any optimal global value function at all.

According to the following theorem, in order to generate the local optimal policy for each region, it is necessary to generate a set of local policies using a set of peripheral values that must be *exhaustive* in the sense that at least one local policy per region will match the optimal policy in this region. However, considerations on the computational cost will rather lead to minimize the size of the set of peripheral values.

Theorem 4.5 [24]**.** *Let Π be a decomposition of a MDP $\langle S, A, T, R\rangle$ and let V be the optimal value function of this MDP. Let $A' = \bigcup_{\mathcal{R}\in\Pi}\left\{\pi_1^{\mathcal{R}}, \dots, \pi_{m_\mathcal{R}}^{\mathcal{R}}\right\}$ be a set of macro-actions generated by the solution of the local MDPs for some combinations of peripheral values, some of these verifying $|\lambda(s) - V(s)| \leq \epsilon$ for each region $\mathcal{R}$ and for all $s \in Per(\mathcal{R})$. If V' is the optimal value function of the abstract MDP induced by Π using macro-actions set A', then:*

$$|V'(s) - V(s)| \leq \frac{2\epsilon\beta}{1-\beta}.$$

Since we do not know the optimal value function of the global MDP for the peripheral states of the regions but only its bounds, we must generate an exhaustive range of peripheral values within the lower and upper bounds given by:

$$V^*_{min} = \frac{\min\limits_{s\in S}\max\limits_{a\in A}\sum\limits_{s'\in S} T(s,a,s')\cdot R(s,a,s')}{1-\beta} \text{ and } V^*_{max} = \frac{\max\limits_{s\in S}\max\limits_{a\in A}\sum\limits_{s'\in S} T(s,a,s')\cdot R(s,a,s')}{1-\beta}.$$

Two different methods have been compared for the generation of such a "sufficient" or "optimal" range of peripheral values, *i.e.* such that at least one peripheral value corresponds to the global optimal value function on the corresponding peripheral states of the region at stake.

4.5. Comparison of two methods for generating macro-actions

4.5.1. *Coverage technique (CT)* [24]

CT consists in generating a mesh of peripheral values covering $[V^*_{min}, V^*_{max}]$ with a sampling step δ that is small enough so that there exist at least one peripheral value in the mesh, and therefore a corresponding local policy, which approximates the optimal policy value on that peripheral state with an error smaller than $\frac{\delta}{2}$. This method is rather inefficient because it requires to generate at least $\prod_{s\in Xper(\mathcal{R})}\frac{V^*_{max}(s)-V^*_{min}(s)}{\delta}$ grid points and to solve as many local MDPs for this region. Many generated local policies are redundant because many different policies can be obtained for a same value. Redundant policies that have been unnecessarily computed need to be eliminated.

As a consequence, it would be preferable to generate only one policy per combination of values on the exit states. This is precisely what linear programming optimization algorithms can achieve while furthermore eliminating the dominated policies: these methods are inspired by the solution algorithms developped for Partially Observable MDPs [10, 30].

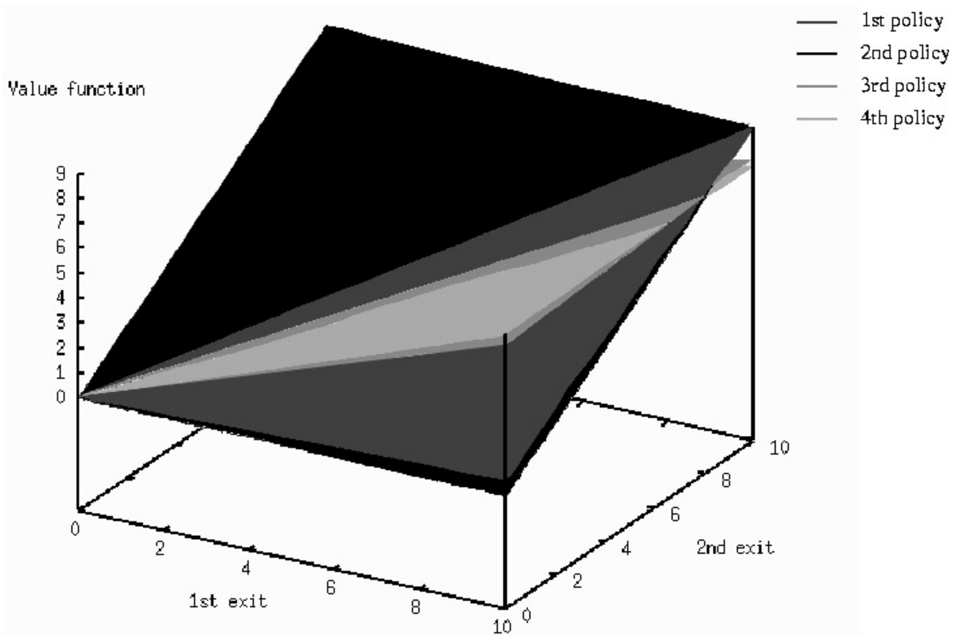


FIGURE 9. Value of the dominating policy at an internal state of a region with 2 exit states.

4.5.2. *Linear programming (LP)* [31]

LP is based on the following theorem that is illustrated in Figure 9:

Theorem 4.6 [31]. *The value of internal states for a given policy and a given region is a linear function of the exit states of the region. It follows that the dominating policies at any state form a piecewise-linear convex function of the peripheral values.*

For a given policy π, the optimal linear value $V_s^\pi\left(\lambda_1,\dots,\lambda_{|XPer(\mathcal{R})|}\right)$ for each inner state of a region $\mathcal{R}$ is a solution of the following linear system:

$$V_s^\pi\left(\lambda_1,\dots,\lambda_{|XPer(\mathcal{R})|}\right) = K_s^\pi + \sum_{i=1}^{|XPer(\mathcal{R})|} a_{s\,i}^\pi \lambda_i \text{ with } \begin{cases} K_s^\pi = V_s^\pi(0,\dots,0) \\ a_{s\,1}^\pi = V_s^\pi(1,0,\dots,0) - K_s^\pi \\ \vdots \\ a_{s\,|XPer(\mathcal{R})|}^\pi = V_s^\pi(0,\dots,0,1) - K_s^\pi. \end{cases}$$

This system is not solved for each state s but for each distribution of values $\{(0,\dots,0),(1,0,\dots,0),\dots,(0,\dots,0,1)\}$. Since applying a number of value iteration steps on the local MDP with a chosen policy π is an efficient way of computing the corresponding value on every state, the calculation of the linear value function $V_s^\pi\left(\lambda_1,\dots,\lambda_{|XPer(\mathcal{R})|}\right)$ only requires to run the value iteration algorithm a number $|XPer(\mathcal{R})|+1$ of times. Once the value function is computed, it is possible to compute the necessary and sufficient number of local policies, so

TABLE 1. Comparison of local policies generation in seconds between CT and LP for $\beta = 0.7$ while decomposing the 4-regions MDP of the Figure 7.

	Number of states per region	Decomposition time	Generated policies			
			R_1	R_2	R_3	R_4
CT	9	19.88	4	15	9	12
LP		0.23	5	5	5	5
CT	25	58.89	24	15	14	15
LP		2.9	17	5	5	5
CT	49	92.78	63	38	32	30
LP		17.72	30	5	5	5

that at least one of these local policies matches the global optimal policy on the considered sub-region. From an initial optimal policy for the peripheral values $\Lambda^0 = (V^*_{min}, \ldots, V^*_{min})$, a cache of policies $\Pi = (\pi_1, \ldots, \pi_q)$ and a cache of linear value functions $\left((V_s^{\pi_1})_{s\in S_i}, \ldots, (V_s^{\pi_q})_{s\in S_i}\right)$ are built, that both grow as long as there can be found at least one distribution of peripheral values for which there is no optimal policy in the cache. This amounts to finding, for each policy of the cache, that is to say for each linear component of the value function, a point in the Λ space where the policy can be improved.

The new linear components of the value function are solutions of the following linear problem, for each state s of the region $\mathcal{R}$, for each state t of the entrance periphery $EPer(\mathcal{R})$, for each action a and for each policy π of the current cache Π:

$$\begin{aligned}
&\text{Maximize: } \sum_{s'\in S_i} T(s,a,s')\cdot\left(R(s,a,s') + \beta V^{\pi}_{s'}\left(\lambda_1,\ldots,\lambda_{|XPer(\mathcal{R})|}\right)\right) - V^{\pi}_s\left(\lambda_1,\ldots,\lambda_{|XPer(\mathcal{R})|}\right)\\
&\text{Subject to: } \forall\, \pi' \in \Pi\ ,\ V^{\pi'}_t\left(\lambda_1,\ldots,\lambda_{|XPer(\mathcal{R})|}\right) \le V^{\pi}_t\left(\lambda_1,\ldots,\lambda_{|XPer(\mathcal{R})|}\right)\\
&\qquad\qquad\ \forall\, 1 \le i \le |XPer(\mathcal{R})|\ ,\ \lambda_i \le V^*_{max}.
\end{aligned}$$

A new linear component is generated from the point $(\lambda_i)_{1\le i\le |XPer(\mathcal{R})|}$ as long as the maximum Bellman error over all states of the region, all actions and all policies is not equal to zero (with a small ϵ margin).

4.5.3. *Comparative test*

Since the author in [31] does not provide any numerical results, we have compared these two methods on the very classical problem of Figure 7, as shown in Table 1. The LP method appears to be the most efficient one from many viewpoints. Firstly, the complexity of LP is polynomial in the number of exit states whereas that of CT is exponential. Secondly, CT generates a number of useless dominated policies, contrary to LP whose policies are all non-dominated. Thirdly, more than 99% of the policies generated by CT are superfluous and must be eliminated because they correspond to equivalent peripheral values. Eventually, CT sometimes generates fewer policies than LP because some optimal policies for some peripheral values on exit states lie exactly between two points of the mesh. For

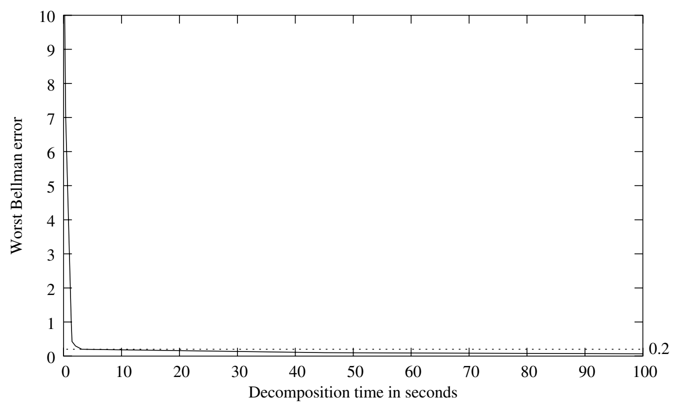


FIGURE 10. Worst Bellman error for the decomposition of the MDP depicted in Figure 7 with the linear programming method but with twice more gates between the regions.

these reasons, we decided to implement the LP technique in our planner in order to exploit the decomposition of the navigation-grid state space into sub-regions and in order to compute the corresponding navigation macro-actions.

When the number of gates between the regions is doubled on the instance shown in Figure 7, and if the optimality requirements are lowered down to accept, for instance 0.2-optimal policies instead of 0.01-optimal ones (Fig. 10), then CT decomposes the MDP in more than 4 h, compared to a few seconds with LP.

5. STATE SPACE FACTORIZATION

In this section, we give an introduction to factored MDPs, but the reader can refer to [9] for a more complete overview.

In factored MDPs, the state space is defined as the cartesian product of the possible values of a number of state variables. For each action, the probability of transition into a given state is no longer given as a function of the initial state but now depends conditionally on the different state variables. Therefore, state transitions can be represented either as Dynamic Bayesian Networks [11] or with probabilistic STRIPS operators [13]. The iteration scheme on which are based the solution algorithms for factored MDPs, is called *Decision-Theoretic Regression.* It consists in avoiding the explicit enumeration of all states at each iteration, by manipulating implicitly sets of states.

5.1. DYNAMIC BAYESIAN NETWORKS (DBNs)

A factored MDP can be represented by a set of action networks represented as DBNs, as shown in the left part of Figure 14. These networks describe the effects of actions through conditional dependencies between the state variables before and after the actions have been performed. They include *diachronic arcs*, directed from pre-action variables to post-action variables. They may also include *synchronic arcs* standing for the dependences between post-action variables only. Given a factored state space $S = \otimes_{i=1}^{n} x_i$ and an action a, the probability of reaching a state s^{t+1} from a state $s^t = \wedge_{i=1}^{n} x_i^t$ is:

$$T(s^t, a, s^{t+1}) = \prod_{i=1}^{n} P\left(x_i^{t+1} | a, x_i^t \wedge \left(\wedge_{j \neq i} \left(x_j^t \wedge x_j^{t+1}\right)\right)\right).$$

Synchronic seriously complicate the solution algorithms, but they are required in most realistic applications, for instance in our model of stochastic exploration-like problem as we will see in Section 5. Algorithms are proposed in [6] to solve factored MDPs whose action networks contains synchronic arcs.

5.2. Decision trees

In DBNs, the transition probabilities are represented by *Conditional Probability Tables*, *i.e.* large matrices, one for each post-action variables. However, these tables are sparse in most problems. [9] propose to represent the conditional probabilities as probability "decision trees" that are particularly compact because each leaf does not necessarily correspond to a single state. For each post-action variable state, each leaf of its probability tree stores a list containing the probabilities to obtain each possible value x_i^{t+1} of this variable, knowing the values of the other variables x_i^t, x_j^t x_j^{t+1}, which correspond to the path $x_i^t \wedge \left(\wedge_{j \neq i} \left(x_j^t \wedge x_j^{t+1}\right)\right)$ from the root of the tree to the considered leaf (see Fig. 16). In the same way, the transition rewards are stored in decision trees as shown in Figure 15. These trees are often very compact, consisting most of the time in, on the one hand a single value for a particular set of variables, which depicts some special conditional effect of the action network, and on the other hand a constant value for the other variables. Like probability trees, the nodes of reward trees can be post-action variables that can be removed and replaced by pre-action variables with the same techniques proposed in [6] for uncorrelated probability trees.

5.3. Operations on decision trees

Both the value function and the policy are represented as decision trees to take advantage of such a compact description of the state space all along the different iteration steps of the optimization process. The handling of all these decision trees at each iteration is based on two basic operations (Fig. 11):

- *the merging of two trees*: successively adding a tree to all the leaves of the current growing tree while labeling the new leaves with both the labels of the previous tree leaves and those of the leaves of the added tree;
- *the simplification of a tree*: removing redundant interior nodes in a tree and fusing nodes whose subtrees are identical.

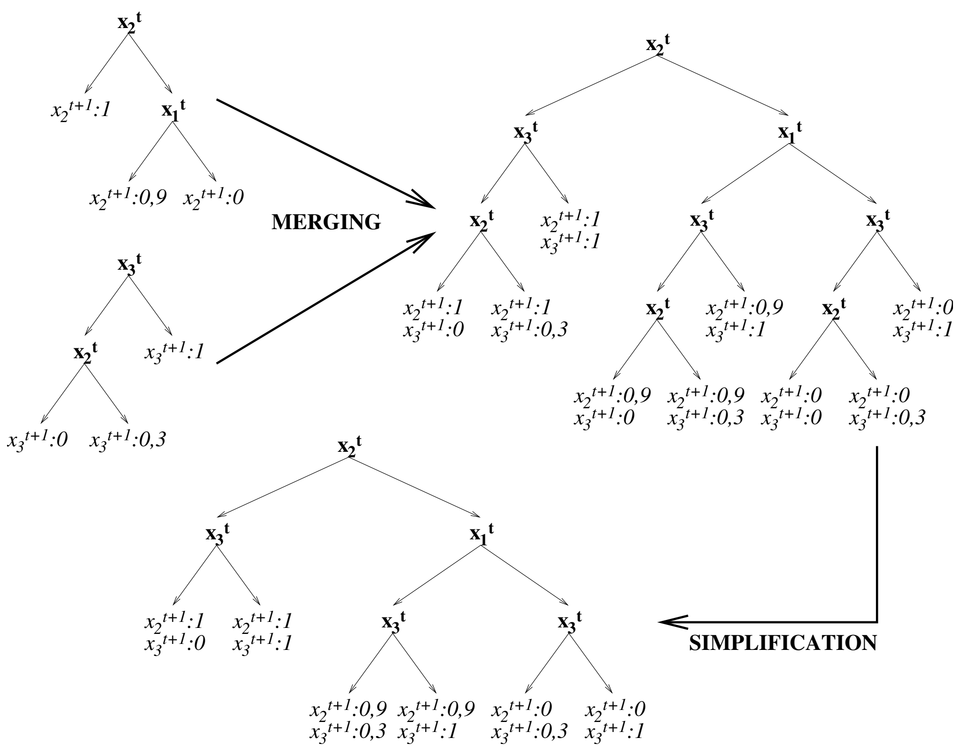


FIGURE 11. Merging of two probability tree and simplification of the resulted tree.

A systematic "simplification while merging" process enables to limit the memory space spent to store the trees and insures a quicker parsing of the trees. In particular, we are sure that the depth of the value function and policy trees does not exceed the number of state variables.

The first stage of the solution is the value tree *regression*: *i.e.* the building, for each action network, of a regressed probability tree. In order to propagate Bellman's equations, this stage is performed by first merging, for the considered action network, the probability trees of the state variables involved in the previous value tree, in order to compute the probabilities of reaching, with this action, all the states having a same value (same leaf) in the previous value tree. This stage is depicted in Figure 12.

The second stage depends on the chosen solution algorithm. In the value iteration algorithm, the new value tree is obtained by merging the regressed value trees with the previous value tree and the reward trees for all the actions, so as to keep the maximum value at each leaf of the new tree. After the last iteration, the policy tree is built by replacing the maximum value of each leaf by the action corresponding to this value. In the policy iteration algorithm, the new value tree

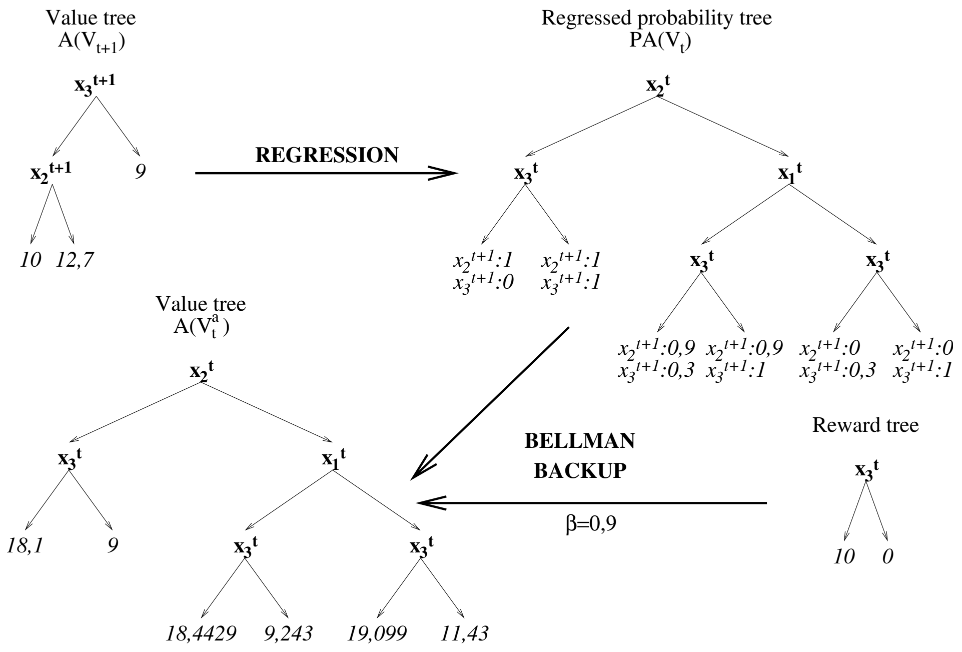


FIGURE 12. Building of the value tree of an action network knowing the value tree at the next time of the process, the reward tree and the probability trees of Figure 11 for this action.

is built by merging at each leaf of the current policy tree, the regressed value tree of the action network which corresponds to the action that labels this leaf. A full description of these algorithms with many examples is presented in [6, 9].

5.4. Reachability analysis

On nice feature of factored MDPs, is that they allow to combine MDP solution algorithms with techniques recently used in the field of symbolic Artificial Intelligence Planning, based on reachability analysis or heuristic search. If we know the initial state of the agent, we can dismiss the combinations of values of the state variables that correspond to states that will not be reached. For instance, the algorithm ReachableK proposed in [7] enables to remove the nodes of states that are not reachable in K iterations.

Moreover, the solution algorithms require a systematic variable replacement operation in order to keep only pre-action variables at each decision tree nodes. If this was to be performed online, it would occur each time a post-action variable is encountered in the trees. In most cases, it saves time to do it once and for all at the creation of the MDP, even if some decision tree are uselessly processed. Action networks are thus initialized by examining all the decision trees, except the reward trees, and by replacing all post-action variables by the corresponding pre-action variables.

5.5. State variables with large range of values

State variables with large range of values considerably slows down the solution algorithms because the trees to be parsed have a very large width. Particularly for exploration problems, we can not define the robot's position as a single variable whose range is the number of atomic navigation states. Even if the position is factorized in the three classical space variables, the range of these variables is very rapidly not tractable.

Therefore, we propose in the next section to aggregate states that present a large range of possible values and cannot be efficiently factored, like navigation sub-states, into fewer macro-states, which will become the possible values of one state variable named *region*, by applying the state space decomposition techniques presented above [12, 24].

6. A hybrid model of structured MDP

In this section, we propose a hybrid model of MDPs that takes advantage of both the MDP factorization and the MDP decomposition, contrary to [27] where a partition is used to solve a factored MDP. Our proposal shares some similarities with what is proposed in [14]. The MAXQ approach also leads to a factored MDP resulting from a decomposition of the problem consisting in a MAXQ hierarchical abstraction graph designed by hand. Our approach rather proposes to automatically generate the factored MDP, on the basis of a first topological decomposition of the navigation sub-space, that can also be automated [36]. A state variable R can stand for the navigation state variable corresponding to the occupied region in the factored MDP.

Therefore, our planner includes and optimizes the solution at two hierarchical levels: a local, lower abstraction, navigation solution level and a higher abstraction factored solution level.

Although we illustrate our hybrid model on the little example of Figure 3, the following methods and algorithms can of course be applied to any factored MDPs for which some state variables have a big range of possible values and sparse transition probabilities.

6.1. Local MDPs and abstract factored MDP

A preprocessor initially decomposes the subspace of navigation states with the LP technique so that we obtain the abstract MDP represented in Figure 13. A factored MDP is then built, whose atomic actions are the local policies stemming from the decomposition. The regions are the values of a same state variable R whose range is 3 in our example. We can add other state variables describing the problem to solve, like the binary variables O_1, O_2 and O_3 standing for the

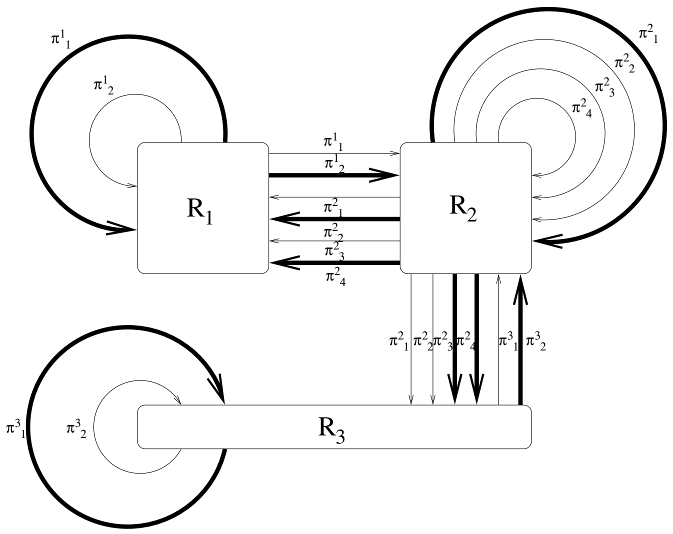


Figure 13. Example of abstract MDP decomposed from the MDP of Figure 3. The thick strokes correspond to wished transitions.

agent's sub-goals status (achieved or not) or the agent energy autonomy level A. Note that our planner assumes that state variables are not continuous so that we suppose that the energy autonomy is a binary variable too: which correspond to a test function returning "1 = enough autonomy" or "0 = insufficient autonomy". Furthemore, the transition model from "1" to "0" is not history-dependent. This is certainly not sufficient as a model of "energy autonomy level" but we can use different models in future work. Moreover, we intend to study the extension of our model to continuous variables.

6.2. Transitions model of the abstract MDP

In our approach, all the decision trees of the factored abstract MDP are automatically built (see Fig. 14). These decision trees are the DBNs representing all the local policies generated as the macro-actions of the obtained factored abstract MDP. Even the partition of the navigation-space can be automated in the future.

The probability tree of the state variable R comes directly from the macro-transitions of the decomposed MDP that provides the macro-probabilities and macro-rewards of the transitions between the regions. Dependences toward other variables can be added to the variable R depending on the problem to solve. For instance, our exploration problem requires that the probability to reach some given region from a another given region depends on the energy autonomy level of the agent, as shown in Figure 16.

It is assumed that the agent only has a chance to achieve a goal, if and only if it is already located in the region that contains the goal position, if the goal has not

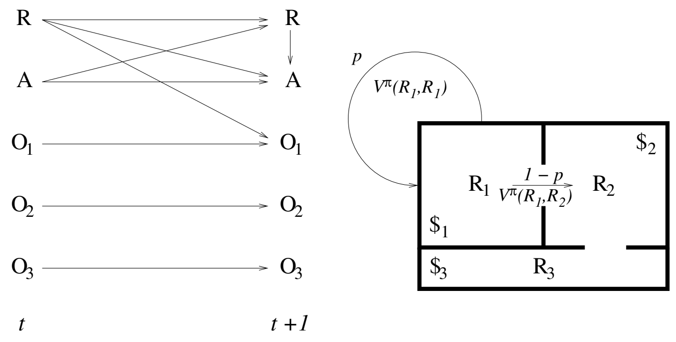


FIGURE 14. Action networks of the abstract factored MDP and transitions of the decomposed MDP corresponding to the local policies of the region R_1.

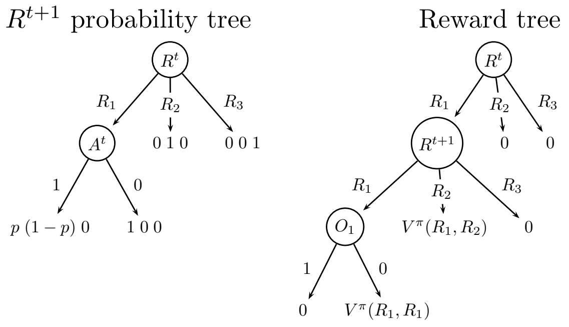


FIGURE 15. The R^{t+1} probability tree and reward tree of the action networks corresponding to the local policies of the region R_1.

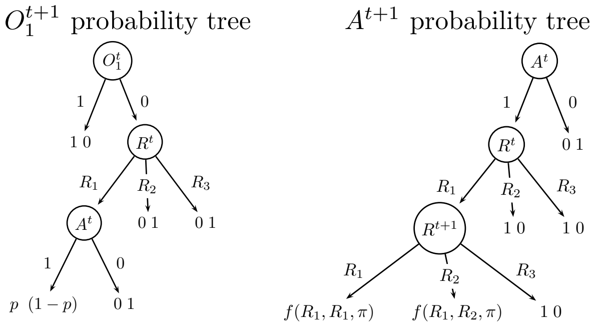


FIGURE 16. The O_1^{t+1} and A^{t+1} probability trees, two action networks corresponding to a local policy π applicable in the region R_1.

been already achieved and if the agent has a sufficient energy autonomy level to do so. As a consequence, for each action network corresponding to a local policy applicable in region R, the probability tree of each "goal" state variable outside R is like a *NO-OP* operator (no value change for the "goal" variable). A non-trivial probability tree defines the transition probabilities for the variables corresponding to the goals in the region R. The transition is possible provided that the goal has not been achieved yet and that the autonomy level is sufficient to achieve it (see Figs. 15 and 16).

The probability trees involving the energy autonomy level variable, as depicted on the right in Figure 16 for instance, are parameterized by a function of the achieved macro-transition $f(R, R', \pi)$: the energy consumption depends on the region transition and on the applied local policies.

The macro-rewards of the decomposed navigation MDP are in the leaves of the reward tree corresponding to the macro-actions. In Figure 15, the reward tree corresponds to a macro-action applicable in region R_1 and the tree shows that the agent receives a macro-reward either when the goal O_1 is achieved or when the region R_2 is reached.

6.3. Global solution scheme

The solution algorithm of our hybrid model of MDP is presented in Figure 17. Our decomposition technique is used in a preprocessing phase. It creates a number of macro-states using a partition of the sub-space of states that correspond to the state variables with relatively large ranges of values (navigation states). The decomposition also generates the corresponding macro-actions, that is to say the local policies, defined in each macro-state. Then, the solver builds the factored MDP with the macro-states as values of a single *"region"* state variable and with the macro-actions as atomic actions. The solution of the hybrid MDP is global policy pointing, depending on the region, towards a set of local policies indicating the microscopic behavior of the agent inside the regions.

6.4. Solution algorithms for the factored MDP

The solution algorithms corresponding to factored MDPs, named *Decision-Theoretic Regression*, consist in avoiding the explicit enumeration of all states at each iteration. They apply factored versions of the Dynamic Programming scheme, either on decision trees or on ADDs, like *SPUDD* [25]. There are some technicalities though: in DBNs, *diachronic arcs* are directed from pre-action variables to post-action variables, and *synchronic arcs* stand for the dependences between post-action variables at the same time point, as in our instance represented in Figure 14. Synchronic arcs make the algorithm, still feasible [6] but more complicate. The most efficient solution to take synchronic correlations into account remains to compute a *total action diagram* for each action network as proposed in [26], so that all correlations between pre- or post-action variables are implicitly computed.

Preprocessor

```
Partition the state space in regions
For each region
   Build the local MDP
   For each combination of peripheral values
       Solve the local MDP
       Compute the macro-transitions
```

Solver

```
Initiate an empty factored MDP
Add variables 'regions','goals','energy autonomy'
Add actions 'local policies of regions'
Generate decision trees
Solve Factored MDP
```

Solution

```
For each factored state of the abstract MDP
   Identify region and optimal local policy
   For each navigation state of the region
       Apply action of the local policy
```

FIGURE 17. Global solution scheme.

We first tested a *modified policy iteration* algorithm for solving factored MDP, which simply was an adaptation of the classical *policy iteration* algorithm to decision trees. In particular, the solution of our simple instance shown in Figure 16 was obtained with this *modified policy iteration* algorithm. Most of the basic operations on decision trees have been described in Section 5. Decision tree based algorithms do in fact loose a lot of cpu time in the treatment of correlations, especially when the size of the problem grows.

We eventually found it more efficient to adopt ADDs and binary variable encodings in our problem [39], *e.g.* with a region variable R splitted into as many boolean variables as needed. Again, for readability reasons, we did not make it appear as a boolean variable in the ADD instance in Figure 19, just as we present the decision tree versions of our probability and reward ADDs in Figures 16 and 15. In ADD based algorithms, both the value function and the policy are represented as ADDs as in the example presented in Figure 19, which correspond to an ADD strictly equivalent to the decision tree shown in Figure 18.

The first stage of a dynamic programming iteration is the *value ADD regression*: for each action network, the probability of reaching each leaf of the current value ADD has to be computed. Then, for the value iteration algorithm, the new value ADD is obtained by comparing for all the actions, the regressed value ADDs merged with both the previous value ADD and the corresponding reward ADDs. The maximum value at each leaf of the new ADD is kept. After the last iteration, the policy ADD is built by replacing the maximum value of each leaf by the action corresponding to this value. For the policy iteration algorithm, the new value ADD is built by merging at each leaf of the current policy ADD the regressed value ADD of the action network corresponding to the action that labels this leaf.

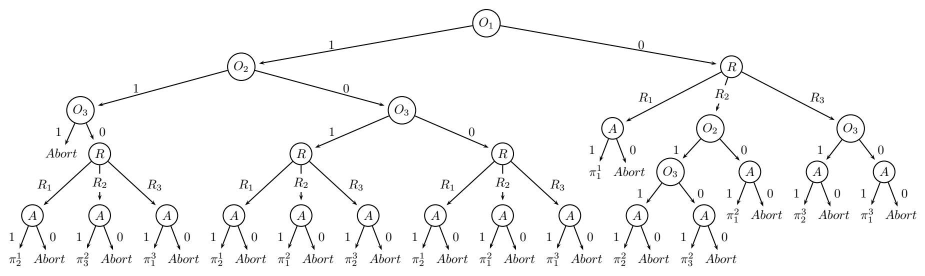


FIGURE 18. Optimal policy tree of the simple abstract factored MDP.

A full description of the equivalent algorithms based on decision trees is presented with many examples in [6, 9, 26]. More technicalities can also be found in [39].

The global solution algorithm of our hybrid model of MDP is the one presented in Figure 17. The factored MDP solution was first performed using SPUDD/VI for value iteration. We implemented a symbolic policy iteration algorithm using ADDs of our own: SPUDD/PI. We have tested our method with the example of Figure 3 for which we have just presented the hybrid model of MDP. This algorithm was applied to our instance of exploration-like MDP. The factored MDP was built and solved using our modified policy iteration algorithm using ADDs. The optimal policy tree is depicted in Figure 19 (see Tab. 2 for the optimal macro-actions stored in the leafs of the optimal policy tree for region R_1).

7. Our simple instance solved

We first tested our approach with the example of Figure 3 for which we have presented in details the hybrid model of MDP using decision trees. We have used an infinite horizon discounted criterion with $\beta = 0.9$. For simplicity, we assumed in this instance that the consumption of energy is constant, and thus independent of the regions and of the followed local policies, which is not very realistic. The function f giving the probability of "loosing the minimal energy autonomy level" at the leaves of the right tree in Figure 16 is chosen to be $f(R_i, R_j, \pi) = [\ 0.65\ 0.35\ ]$ for all i,j and π: it only accounts for the fact that after some flight time, chances are that the aircraft will run out of petrol.

The navigation MDP was decomposed with the linear programming decomposition technique and produced 7, 6 and 2 local policies respectively in regions R_1, R_2 and R_3. The computed macro-transitions for each generated local policy are shown in Table 2. Although we do not specify the micro-actions of local policies, whose three of them are depicted in Figure 3, the probabilities of the macro-transitions give a good idea of the local behaviors.

The factored MDP was solved after 28 iterations using the modified policy iteration algorithm based on decision trees. The optimal policy tree is depicted in Figure 18 with its equivalent ADD shown in Figure 19. Refer to Table 2 for an

TABLE 2. Local policies and corresponding macro-transitions stemming from the decomposition of the navigation subspace.

Origin region	Local policies	End region	Probability	Value
R_1	π_1^1	R_1	1	17.6327
	π_2^1	R_1	0.111317	2.21295
		R_2	0.888683	0
	π_3^1	R_1	0.211306	4.00467
		R_2	0.788694	0
	π_4^1	R_1	0.433556	8.18566
		R_2	0.566444	0
	π_5^1	R_1	0.666778	11.5589
		R_2	0.333222	0
	π_6^1	R_1	0.988889	17.1386
		R_2	0.0111111	0
	π_7^1	R_1	0.446756	8.21195
		R_2	0.553244	0
R_2	π_1^2	R_2	1	8.77549
	π_2^2	R_1	1	0
	π_3^2	R_3	1	0
	π_4^2	R_1	0.977632	0
		R_3	0.0222222	0
	π_5^2	R_1	0.0111111	0
		R_2	0.0135267	0
		R_3	0.975362	0
	π_6^2	R_2	0.988889	8.52971
		R_3	0.0111111	0
R_3	π_1^3	R_2	0.0333333	0
		R_3	0.966667	24.0827
	π_2^3	R_2	0.5	0
		R_3	0.5	14.4589

overview of the optimal macro-actions stored in the leaves of the optimal policy tree.

Note the action *Abort*, whose network only contains no-ops trees and is added at the end of the action networks list, so that it is chosen by the solution algorithm when no action can be achieved instead. *Abort* logically labels the tree leaves corresponding to the states where there is not a sufficient energy autonomy level for the agent to pursue its mission (security crash for an aircraft) or those states where all goals have been reached and the mission is ended.

The compactness of the solution is noticeable since, on the one hand only 31 sets of states, or tree branches, are enumerated in the solution diagram over a total of

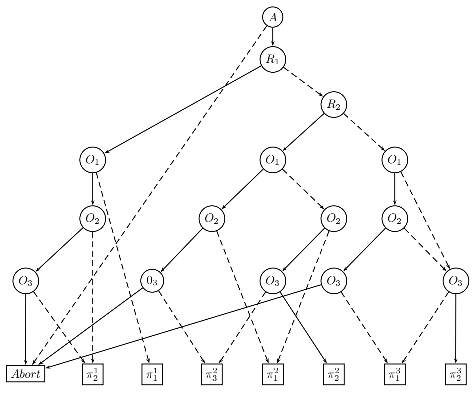


FIGURE 19. Optimal policy ADD of the simple abstract factored MDP.

$6 \times 4 \times 2^4 = 384$ possible states, and on the other hand only 8 leaves appear in the optimal policy ADD (see Fig. 19), which makes it easier to interpret. This policy (except for *Abort*) corresponds to the thick transitions arrows in Figure 13.

8. Efficient solution algorithms for factored MDPs

In this last section, we only give a sketch view on the scalability and efficiency of our hybrid approach combining factorization and decomposition techniques, in order to tackle larger size MDPs. We cannot describe in more details our on-going research on efficient solution algorithms for factored MDPs: the results obtained so far can be found in [18, 39, 40].

The general algorithm presented in Figure 17 has been developped into various versions that only differs in the solution stage: "Solve Factored MDP".

8.1. Symbolic Heuristic dynamic programming

All the algorithms developped and compared in the following preliminary results, follow the general "Symbolic Heuristic Dynamic Programming" scheme presented in Algorithm 8.1.

We assume in the following the case where a (set of) initial state(s) I and a (set of) goal state(s) G are given prior to the solution. If no initial set of states is given, then the reachability stage does not reduce the size of the state space, but the rest of the scheme still applies. An initial state is generally given in our stochastic exploration planning problems. Besides, it is noticeable that the use of ADDs naturally makes no difference between a unique initial or goal state, or a set of initial or goal states, since ADDs represent states as sets of states. As proposed in [7], we apply an initial reachability analysis that allows us to provide our dynamic programming iterative algorithm with an initial policy or an initial value and an initial sub-state space. Contrary to the function *ReachableK* used in [7], we propose to make full use of the information available.

We call $\mathit{Reachable}(I, \Pi_A, \mathit{Stop})$ a function that takes as inputs the sets of initial and goal states I and G, uses the set of applicable actions $\Pi_A \subset \mathcal{A}$ (Π_A can be a policy or $\mathcal{A}$ itself) and computes the set R_0 of all the states that are reachable from I with successive applications of deterministic actions in $\mathcal{A}$ in an iterative loop that stops as soon as the *Stop* condition is reached: *e.g.* *Stop* can be $G \subset R_0$ or 1 *step lookahead.* The actions are made deterministic by setting the maximum transition probability to 1 and the other one to 0, which enables us to convert the ADDs into BDDs (Binary Decision Diagrams) that are more efficient [34].

We call $\mathit{ShortestStochasticPath}(R_0 \to G)$ a function that takes R_0 and G as inputs and computes a shortest stochastic path from each state in R_0, using the stochastic actions in $\mathcal{A}$ without considering their rewards. Better simplification schemes should certainly be studied, but this heuristic function seems to be efficient in many problems, such as in navigation grid MDPs in [23] and in [4].

We call $\mathit{FilterStates}(R_0, P(s) < \epsilon \cdot P(I))$ a filtering function that filters the states that have a very low reachability in terms of probability when the non-deterministic actions are applied along the shortest path trajectories. Low probability of reachability is assessed compared to the probability of the initial states. All three functions, included in the scheme shown in Algorithm 8.1, constitute our common algorithmic basis for testing and comparing different heuristic dynamic programming schemes, with value iteration or more originally with policy iteration.

Algorithm 8.1 (Solve Factored MDP with Symbolic Heuristic Dynamic Programming Iteration)**.**

```
Init
$R_0 \leftarrow \mathit{Reachable}(I, \mathcal{A}, G)$
$\Pi_0 \leftarrow \mathit{ShortestStochasticPath}(R_0 \to G)$
$S_0 \leftarrow \mathit{FilterStates}(R_0, P(s) < \epsilon \cdot P(I))$
      $\Longrightarrow (\Pi_0, V_0, S_0)$

$k \leftarrow 0$
repeat
   $S_{k+1} \leftarrow \mathit{Reachable}(I, S_k, \Pi_k, G)$
   $\mathit{DynamicProgramming}(\Pi_k, V_k, S_{k+1})$
   $k \leftarrow k + 1$
until convergence over $S_k$
```

Six factored MDP solution algorithms using ADDs and inspired from SPUDD, sRTDP and sLAO were implemented for the Solve Factored MDP stage, to which we gave the following numbers: **1.SPUDD/VI – 2.SPUDD/PI – 3.sLAO/VI – 4.sLAO/PI – 5.sRTDP/VI – 6.sRTDP/PI.**

8.2. Experimentations

We run all the algorithms on exploration-like MDP instances of different sizes: see Figures 20a and b. The "Concentric problem" shown in Figures 20a has

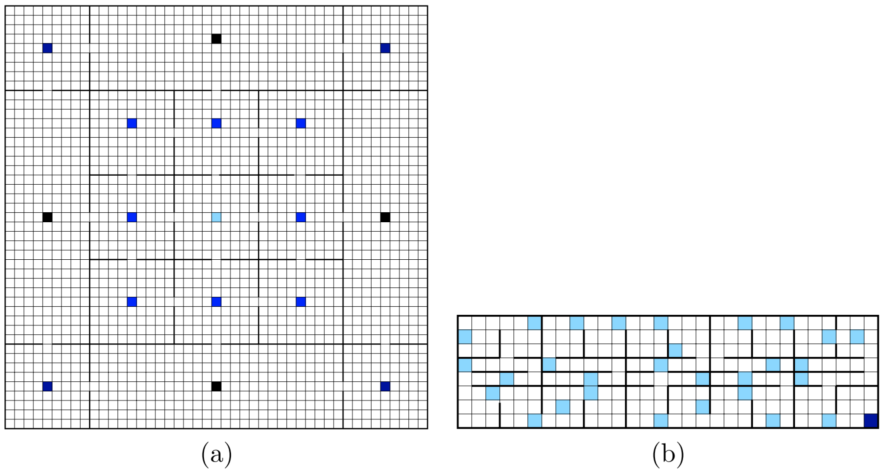


FIGURE 20. Concentric problems (a) and Linear problems (b).

17 goals, 1 binary energy autonomy variable, a 45×45 navigation-grid, a total number of 530 841 600 enumerated states and leads to solve an abstract MDP with 23 variables and 17 local MDPs. The "Linear problem" shown in Figures 20b has 30 goals, 1 binary energy autonomy variable, a 8×30 navigation-grid, a total number of 515 396 075 520 enumerated states and leads to solve an abstract MDP with 36 state variables and 30 local MDPs. We first performed the comparison between the factored MDP and enumerated MDP approaches whose results are in Figure 21, which shows that state space modeling is really a crucial issue. Columns are for problems type (either "linear" or "concentric"), state space $size$, number $rg.$ of regions, model of the MDP (F for factored MDP and E for enumerated MDP), time $B.$ for building the MDP, time $D.$ for decomposition, number π_r of macro-actions (local policies computed), total solution time T. The time required for the building of the state transition data structures for the enumerated MDP illustrates handicap of the enumerated approach. Problems of larger size could not be solved in comparable time: this is why they do not appear in the table. The complexity burden is apparently higher for "concentric problems" than for "linear problems" (compare the complexity step between the "linear problems" and the "concentric problems").

Figure 22 presents the comparison between the six factored algorithms presented above, with the cpu time given in seconds. The respective columns stand for problems state space $sizes$, the number $rg.$ of regions, the number π_r of local policies computed, the algorithm number $A. \in \{1, 6\}$, the cpu time $D.$ for decomposition, the total solution time $T.$, the region number of the initial state $I.$, the size of the explored search space in percentage $\%s$ of the state space, and the evolution of it in $\%r$ percentage between the starting and the end of the search.

It is noticeable that large scale problems with up to 5×10^{11} states are solved. It is interesting to notice that our global optimal approach matches the state

type	size	rg.	model	$B.$	$D.$	π_r	$T.$
linear	384	3	F	< 0.01	0.08	10	0.02
			E	0.03	–	–	0.01
	$6 \cdot 10^3$	6	F	0.01	0.38	33	1.51
			E	4.21	–	–	0.38
	$7 \cdot 10^4$	9	F	0.03	0.56	47	26.8
			E	587.62	–	–	5.03
concentric	$8 \cdot 10^5$	9 (9 s./r.)	F	0.02	0.13	21	0.12
			E	746.98	–	–	2.25
	$7 \cdot 10^6$	9 (81 s./r.)	F	0.02	40.61	61	16.77
			E	> 1hr	–	–	–

FIGURE 21. Comparison between the factored MDP and enumerated MDP approaches with the elapsed time given in seconds.

space sizes solved in [22] with approximate solution algorithms. Furthermore, we can claim that our results were obtained on quite different MDPs: our problems combine boolean variables and variables with large ranges of values (navigation variables). Eventually, the results obtained on "Linear problems" show that the size of the state space is not always a good criteria to assess the problem difficulty. In our experimentations, we show that, depending on the respective positions ("proximity") of the initial and goal states in the state space, problems of larger state space can be simpler than much smaller problems (see Fig. 22).

9. Conclusion

This paper has presented a range of possible modeling structures for state space representations in MDPs and discussed the influence of the possible modeling choices with respect to both the tractability and the efficiency of the corresponding approach to the MDP solution.

It is proposed to develop a hybrid approach combining both MDP decomposition and factorization techniques. Many algorithmic perspectives are now open on this basis and preliminary results show interesting behaviors in termes of efficiency and scalability: symbolic heuristic search dynamic programming algorithms are presently studied. This proposal is not completely new [14], yet the abstraction process and the resulting approach to the MDP solution using symbolic dynamic programming is original.

In any case, the approach proposed in [14] provides another perspective of combination, towards a generalized automatic abstraction process, for large size MDPs. Possible original future developments with respect to the MDP solution algorithms in this perspective could be in a research on either heuristic, approximated, or learning based methods.

Models combining factored state variables and enumerated state sub-spaces with large ranges of possible values are susceptible to apply in robotics problems

size	rg.	π_r	*A.*	*D.*	*T.*	*I.*	%s	%r
384	3	10	1	0.08	0.02	–	–	–
			2	0.07	0.03	–	–	–
			3	0.06	0.03	1	67	78
			4	0.06	0.03	1	67	78
			5	0.06	0.03	1	42	11
			6	0.06	0.03	1	42	11
6.10^3	6	33	1	0.38	1.51	–	–	–
			2	0.36	1.73	–	–	–
			3	0.37	1.89	3	60	129
			4	0.37	0.88	3	63	115
			5	0.37	1.28	3	56	114
			6	0.38	0.71	3	56	91
7.10^4	9	47	1	0.56	26.8	–	–	–
			2	0.56	45.86	–	–	–
			3	0.55	16.89	4	54	112
			4	0.54	15.23	4	56	114
			5	0.56	5.38	4	43	70
			6	0.56	4.35	4	43	66
7.10^6	12	59	1	0.63	192.1	–	–	–
			2	0.65	391.95	–	–	–
			3	0.65	0.12	6	46	-66
			4	0.63	0.11	6	46	-66
			5	0.64	0.85	6	19	-86
			6	0.62	0.57	6	19	-86
7.10^7	15	73	1	0.76	$>$ 1hr	–	–	–
			2	0.76	$>$1hr	–	–	–
			3	0.79	0.29	7	50	-53
			4	0.76	25.97	7	26	74
			5	0.77	0.97	7	17	-84
			6	0.75	18.86	7	13	-11
5.10^{11}	30	156	3	2.19	4.77	25	18	-83
			4	2.17	8.25	25	14	119
			5	2.2	1.56	25	5	-95
			6	2.19	6.51	25	8	34

size	Nreg	Nspr	π_r	*A.*	*D.*	*T.*	%s	%r
8.10^5	9	9	21	1	0.13	0.12	–	–
				2	0.13	0.15	–	–
				3	0.15	0.04	8	-40
				4	0.13	0.03	11	-20
				5	0.14	0.04	14	0
				6	0.13	0.04	14	0
7.10^6	9	81	61	1	40.61	16.77	–	–
				2	40.44	21.42	–	–
				3	40.59	0.16	8	-40
				4	40.45	0.14	11	-20
				5	40.94	0.2	14	0
				6	40.64	0.15	14	0
6.10^8	17	9	69	1	0.58	$1.6.10^3$	–	–
				2	0.55	$>$1hr	–	–
				3	1.01	2.34	0.19	-0.09
				4	0.97	2.22	0.19	-0.09
				5	1.0	1.65	0.1	-0.5
				6	0.98	1.45	0.1	-0.5
5.10^9	17	81	117	3	93.83	36.88	44	48
				4	93.88	96.95	44	49
				5	93.83	1.32	10	-68
				6	93.88	8.87	10	-68

FIGURE 22. Linear (top) and concentric (bottom) MDPs.

including navigation and information acquisition tasks. Further work will focus on the on-line behavior of factored MDP solution algorithms and should also attempt to deal with continuous variables.

References

[1] D. Aberdeen, S. Thiébaux and L. Zhang, Decision-theoretic military operations planning, in *Proceedings of 14th Conf. ICAPS 2004*, *Whistler, Canada* (2004) 402–412.

[2] R. Bellman, *Dynamic Programming.* Princeton University Press, Princeton, NJ (1957).

[3] D. Bertsekas and J. Tsitsiklis, Neuro-dynamic programming: an overview (1995).

[4] B. Bonet and H. Geffner, Labeled rtdp: Improving the convergence of real-time dynamic programming, in *Proceedings of 13th Conf. ICAPS 2003*, *Trento, Italy* (2003) 12–21.

[5] C. Boutilier and D. Poole, Computing optimal policies for partially observable decision processes using compact representations. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, *Portland, Oregon, USA*, AAAI Press / The MIT Press (1996) 1168–1175.

[6] C. Boutilier, Correlated action effects in decision theoretic regression, in *Uncertainty in Artificial Intelligence* (1997) 30–37.

[7] C. Boutilier, R.I. Brafman and C. Geib, Structured reachability analysis for Markov decision processes, in *Uncertainty in Artificial Intelligence* (1998) 24–32.

[8] C. Boutilier, T. Dean and S. Hanks, Decision-theoretic planning: Structural assumptions and computational leverage. *J. Artificial Intell. Res.* **11** (1999) 1–94.

[9] C. Boutilier, R. Dearden and M. Goldszmidt, Stochastic dynamic programming with factored representations. *Artificial Intell.* **121** (2000) 49–107.

[10] A.R. Cassandra, *Exact and Approximate Algorithms for Partially Observable Markov Decision Processes.* Computer science, U. of Illinois, Providence R.I. (1998).

[11] T. Dean and K. Kanazawa, A model for reasoning about persistence and causation. *Computational Intelligence* **5** (1989) 142–150.

[12] T. Dean and S.-H. Lin, Decomposition techniques for planning in stochastic domains, in *Proceedings of the 14th IJCAI 1995*, San Francisco, CA (1995) 1121–1129.

[13] R. Dearden and C. Boutilier, Abstraction and approximate decision-theoretic planning. *Artificial Intell.* **89** (1997) 219–283.

[14] T.G. Dietterich, Hierarchical reinforcement learning using the maxq value function decomposition. *J. Artificial Intell. Res.* **13** (2000) 227–303.

[15] I.S. Duff, A survey of sparse matrix research, in *Proceedings of the IEEE*, Prentice Hall, New York **65** (1977) 500–535.

[16] I.S. Duff, A.M. Erisman and J.K. Reid, *Direct Methods for Sparse Matrices.* Clarendon Press, Oxford (1986).

[17] A. Dutech, Solving pomdp's using selected past events, in *Proceedings of the 14th ECAI 2000*, *Berlin, Germany* (July 2000) 281–285.

[18] P. Fabiani and F. Teichteil-Königsbuch, Symbolic heuristic policy iteration algorithms for structured decision-theoretic exploration problems, in *Workshop on Reasoning under Uncertainty in Robotics - RUR'2005*, Edinburgh, Scotland (2005).

[19] Z. Feng and E. Hansen, Symbolic heuristic search for factored markov decision processes, in *Proceedings of 18th Conf. AAAI 2002*, *Edmonton, Alberta, Canada* (2002) 455–460.

[20] Z. Feng, E.A. Hansen and S. Zilberstein, Symbolic generalization for on-line planning, in *Proceedings of 19th Conf. UAI 2003*, *Acapulco, Mexico* (2003) 209–216.

[21] C. Guestrin, M. Hauskrecht and B. Kveton, Solving factored mdps with continuous and discrete variables, in *Proceedings of 20th Conf. UAI 2004*, Banff, Canada (2004).

[22] C. Guestrin, D. Koller, R. Parr and S. Venkataraman, Efficient solution algorithms for factored mdps. *J. Artificial Intell. Res.* **19** (2003) 399–468.

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

[24] M. Hauskrecht, N. Meuleau, L.P. Kaelbling, T.L. Dean and C. Boutilier, Hierarchical solution of markov decision processes using macro-actions, in *Proceedings of 14th Conf. UAI 1998*, San Francisco, CA (1998) 220–229.

[25] J. Hoey, R. St-Aubin, A. Hu and C. Boutilier, Spudd: Stochastic planning using decision diagrams, in *Proceedings of 15th Conf. UAI 1999*, San Francisco, CA (1999) 279–288.

[26] J. Hoey, R. St-Aubin, A. Hu and C. Boutilier, *Optimal and approximate stochastic planning using decision diagrams.* Technical Report TR-2000-05, University of British Columbia, 10 (2000).

[27] K.-E. Kim and T. Dean, Solving factored mdps using non-homogeneous partitions. *Artificial Intell.* **147** (2003) 225–251.

[28] B. Kveton and M. Hauskrecht, Heuristic refinements of approximate linear programming for factored continuous-state markov decision processes, in *Proceedings of 14th Conf. ICAPS 2004, Whistler, Canada* (2004) 306–314.

[29] S.M. Lavalle, *A Game-Theoretic Framework for Robot Motion Planning.* Electrical engineering, University of Illinois, Urbana-Champaign (1995).

[30] W.S. Lovejoy, *A survey of algorithmic methods for partially observed markov decision processes.* Technical Report 28, Annals of Operation Research (1991).

[31] R. Parr, Flexible decomposition algorithms for weakly coupled markov decision problems, in *Proceedings of 14th Conf. UAI 1998, San Francisco, CA* (1998) 422–430.

[32] J. Pineau, G. Gordon and S. Thrun, Policy-contingent abstraction for robust robot control, in *Conference on Uncertainty in Artificical Intelligence (UAI)* (2003) 477–484.

[33] M.L. Puterman, *Markov Decision Processes.* John Wiley & Sons, INC (1994).

[34] R.I. Bahar, E.A. Frohm, C.M. Gaona, G.D. Hachtel, E. Macii, A. Pardo and F. Somenzi, Algebraic Decision Diagrams and Their Applications, in *IEEE /ACM International Conference on CAD, Santa Clara, California, 1993.* IEEE Computer Society Press 188–191.

[35] Y. Saad, *Iterative Methods for Sparse Linear Systems.* Society of Industrial and Applied Mathematics, second edition (2003).

[36] R. Sabbadin, Graph partitioning techniques for markov decision processes decomposition, in *Proceedings of the 15th ECAI 2002, Lyon, France* (July 2002) 670–674.

[37] R. St-Aubin, J. Hoey and C. Boutilier, APRICODD: Approximate policy construction using decision diagrams, in *NIPS* (2000) 1089–1095.

[38] R.S. Sutton and A.G. Barto. *Reinforcement Learning: An Introduction.* MIT Press, Cambridge, MA (1998).

[39] F. Teichteil-Königsbuch and P. Fabiani, Processus Décisionnel de Markov décomposé et factorisé pour l'optimisation d'une stratégie d'exploration. *Revue d'Intelligence Artificielle* **20** (2006) 133-179.

[40] F. Teichteil-Königsbuch and P. Fabiani, Symbolic heuristic policy iteration algorithms for structured decision-theoretic exploration problems, in *Workshop on Planning under Uncertainty for Autonomous Systems ICAPS'2005*, Monterey, CA, USA (2005).

[41] G. Verfaillie, F. Garcia and L. Péret, Deployment and Maintenance of a Constellation of Satellites: a Benchmark, in *Proceedings of ICAPS'03 Workshop on Planning under Uncertainty and Incomplete Information*, Trento, Italy (June 2003).