Autonomous Search and Rescue Rotorcraft Mission Stochastic Planning with Generic DBNs

Florent Teichteil-Königsbuch[1] and Patrick Fabiani[2]

[1] florent.teichteil@cert.fr
[2] patrick.fabiani@cert.fr
ONERA-DCSD
Toulouse, France 31055

1 Introduction

1.1 Motivation

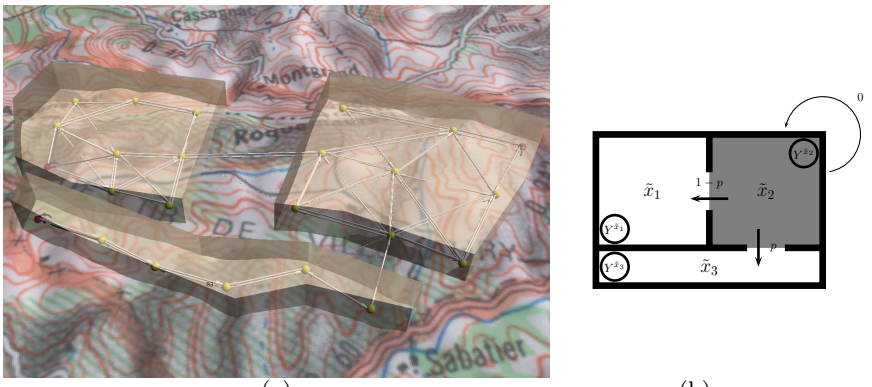
This paper proposes an original generic hierarchical framework in order to facilitate the modeling stage of complex autonomous robotics mission planning problems with action uncertainties. Such stochastic planning problems can be modeled as Markov Decision Processes [5]. This work is motivated by a real application to autonomous search and rescue rotorcraft within the ReSSAC[1] project at ONERA. As shown in Figure 1.a, an autonomous rotorcraft must fly and explore over regions, using waypoints, and in order to find one (roughly localized) person per region (dark small areas). Uncertainties can come from the unpredictability of the environment (wind, visibility) or from a partial knowledge of it: map of obstacles, or elevation map etc. After a short presentation of the framework of structured Markov Decision Processes (MDPs), we present a new original hierarchical MDP model based on generic Dynamic Bayesian Network templates. We illustrate the benefits of our approach on the basis of search and rescue missions of the ReSSAC project.

1.2 Factored Markov Decision Processes

MDPs [5] are a classical model for decision-making under uncertainty. A MDP is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$ where $\mathcal{S}$ is the set of agent's states, $\mathcal{A}$ is the set of its actions, $\mathcal{P}$ and $\mathcal{R}$ respectively are the markovian probability and reward transitions between states for each action. A solution of a MDP is a mapping $\pi : \mathcal{S} \rightarrow \mathcal{A}$ named *policy*, that can be iteratively computed on the basis of the Bellman's equation [5].

Factored Markov Decision Processes (MDPs) [1, 3] are an extension of MDPs where the state space $\mathcal{S}$ is defined as a cartesian product of n subspaces $\mathcal{V}$ corresponding to an equal number of state variables $\mathcal{S} = \otimes_{i=1}^{n} V_i$. State variable transitions are defined using Dynamic Bayesian Networks (DBNs) [1]. For each

[1] http://www.cert.fr/dcsd/RESSAC/



(a) (b)

Fig. 1. (a) Search and rescue autonomous rotorcraft mission: 3 persons must be rescued in the 3 regions of the navigation subspace (software screenshot). (b) Local policy defined in the region $\tau(\pi) = \tilde{x}_2$. Stochastic outcomes are regions $\zeta(\pi) = \{\tilde{x}_1; \tilde{x}_3\}$.

action, a DBN represents the stochastic dependencies between *post-action* state variables $(X'_i)_{i=1}^n$ and *pre-action* state variables $(X_i)_{i=1}^n$ (see Figure 6.a).

For each post-action state variable X_i, a probability tree encodes the stochastic distribution of X'_i values (tree's leafs) knowing the other state variables values (nodes), as shown in Figure 6.b. The reward transitions are encoded as a single decision tree for each action. Classical MDP optimization algorithms are generalized in structured algorithms [1, 3].

1.3 A hierarchical approach

Modeling autonomous robotics problems with factored MDPs remains difficult. In the very simple search and rescue mission of Figure 1.a, with 5 actions: `west`, `east`, `north`, `south`, `statio`, and 4 state variables: the rotorcraft's localization and the status of the 3 persons to rescue, the localization variable has 24 possible values (as many as the number of waypoints), that must be enumerated in any decision tree containing a waypoint node. More complicated missions can have hundreds of waypoints, which makes it a burden to model *by hand* the problem because the trees' sizes are polynomial in the arity of state variables.

Our hierarchical model allows to tackle larger state spaces by reducing the size of the decision trees used to model the problem. We use state abstractions in order to decompose the problem with respect to its variables of highest arity: in the search and rescue example of Figure 1, the localization variable (24 positions) is decomposed into a region variable of arity 3.

2 Hierarchical factored MDP

2.1 State subspace splitting

Let X_p be a state variable with a large arity. The state subspace generated by X_p (navigation subspace) is a graph V_p that can be partitioned into smaller weakly coupled abstract subgraphs $\tilde{V}_p$. The partition can be either a mission input, or the result of an automatic partition process [6]. The resulting abstracted states can be considered as the values of a new abstracted state variable $\tilde{X}_p$, which is an abstraction of the original state variable X_p. The abstract state space of the factored MDP becomes $\tilde{\mathcal{V}} = (\otimes_{i \neq p} V_i) \times \tilde{V}_p$. Let us consider the mission of Figure 1: whereas a X_p node would have 24 subtrees, the corresponding $\tilde{X}_p$ node only has 3 subtrees.

2.2 Local policies

Actions need to be abstracted correspondingly into macro-actions. At the region level, abstract actions correspond to local policies defined and applied within the regions of the partition $\tilde{V}_p$. Let π be such a local policy, defined in a region $\tilde{v}_p$. Let Π_p be a set of local policies defined on each region of the partition $\tilde{V}_p$. A minimal set of local policies can be automatically generated [2, 4], in such a way that an optimal policies can be obtained as a combination of such local policies in the regions. Extra local policies can be added by other methods.

Unfortunately, in both cases, the number of local policies can be very large. In theory, the maximum number of local policies is $\sum_{\tilde{v}_p \in \tilde{V}_p} |\mathcal{A}|^{|\tilde{v}_p|}$, each of which should have a corresponding DBN encoding for the dependencies between the pre- and post-action variables.

In order to keep a substantial benefit from the decomposition, it is useful to notice that in most problems, all the local policies DBNs share a common structure. It is indeed possible to define a single DBN structure, where the corresponding local policy, the region where it is applicable, and the reachable regions appear as parameters that can be automatically instantiated when the local policies are computed.

3 Abstract generic Dynamic Bayesian Network

In this section, we present the syntax of our abstract generic DBN for modeling factored stochastic autonomous robotics problems. Our generic DBN is parametrized by a local policy $\pi \in \Pi_p$. Since a local policy is defined for a single region of the reduced variable $\tilde{X}_p$, we can define the mapping $\tau : \Pi_p \rightarrow \tilde{V}_p$ between local policies and the region where they are each one defined.

We illustrate our approach with a small academic instance of a search and rescue autonomous rotorcraft mission (see Figure 1.b). The decomposed subspace matches the localization variable, whose arity is 24, abstracted in 3 regions ($\tilde{X}$). We will consider a local policy π defined in the second region $\tilde{x}_2$,

that consists in going out towards the regions $\tilde{x}_1$ and $\tilde{x}_3$ with respectively the probabilities $1-p$ and p. Last but not least, each region contains a person to rescue: these subgoals are represented by 3 binary state variables $\left(Y^{\tilde{x}_i}\right)_{1 \leqslant i \leqslant 3}$ indicating if each person was already rescued or not.

3.1 Reduced state variable modeling

Let us consider a decision tree (probability tree or reward tree) containing a node of the reduced variable $\tilde{X}_p$. The local policy π is only defined in $\tau(\pi)$ so that the node $\tilde{X}_p$ only has two abstract subtrees: one corresponding to the value $\tau(\pi)$, and one other representing the other values where the policy is not applicable, noted $\overline{\tau(\pi)} = \tilde{V}_p \setminus \{\tau(\pi)\}$.

Since π is only applicable over $\tau(\pi)$, the $\overline{\tau(\pi)}$-subtree of any $\tilde{X}_p$ variable in probability trees is symbolically represented as a nil leaf. Instead of defining these nil leafs inside each probability tree, it is better to define a binary mask tree that indicates where the local policy is applicable. This mask tree should contain at least a node of the state variable $\tilde{X}_p$, as shown in Figure 2.

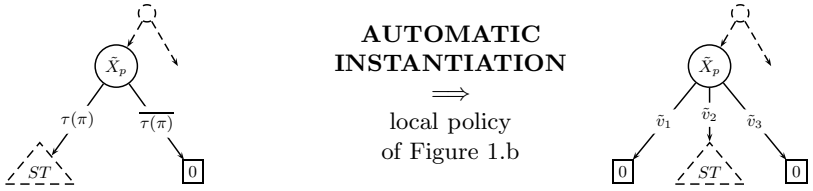


Fig. 2. Generic mask tree example and one of its instantiations

The function that automatically instantiates the subtrees of a $\tilde{X}_p$ node in any decision tree is presented in Algorithm 1. It calls the function `InstantiateTree`, that instantiates the $\tau(\pi)$-subtree of the generic node $\mathcal{T}$ (see Algorithm 6). The $\overline{\tau(\pi)}$-subtrees are nil leafs (nil_leaf).

Algorithm 1: Function `InstantiateXpSubtrees`

Data: $\mathcal{T}$ (generic node), $\mathcal{T}^\pi$ (instantiated node), π, τ, ζ, $[\tilde{v}'_p = -1]$
Result: $\mathcal{T}^\pi$ (instantiated tree)
begin
  $subtree \leftarrow \mathcal{T}.son(\text{`}\tau(\pi)\text{'})$;
  for $\tilde{v}_p \in \tilde{V}_p$ **do**
    if $\tilde{v}_p = \tau(\pi)$ **then** $\mathcal{T}^\pi.sons().push(\texttt{InstantiateTree}(subtree, \pi, \tau, \zeta, \tilde{v}'_p))$;
    else $\mathcal{T}^\pi.sons().push(nil\_leaf)$;
  return $\mathcal{T}^\pi$;
end

The treatment of a $\tilde{X}'_p$ node is slightly different from a $\tilde{X}_p$ node. Let $\zeta : \Pi_p \rightarrow \tilde{V}_p$ be the mapping from a local policy to its reachable regions. It means that π transforms $\tau(\pi)$ into $\zeta(\pi)$. In our small instance depicted in Figure 1.b, only the regions $\tilde{x}_1$ and $\tilde{x}_3$ are reachable with $\pi : \zeta(\pi) = \{\tilde{x}_1; \tilde{x}_3\}$.

A $\tilde{X}'_p$ node can only have 2 abstract subtrees: one for the value $\zeta(\pi)$ and one other for the value $\overline{\zeta(\pi)} = \tilde{V}_p \setminus \zeta(\pi)$. Each subtree must be transformed into as many subtrees as the cardinality of the corresponding abstract value (see Figure 3 and Algorithm 2).

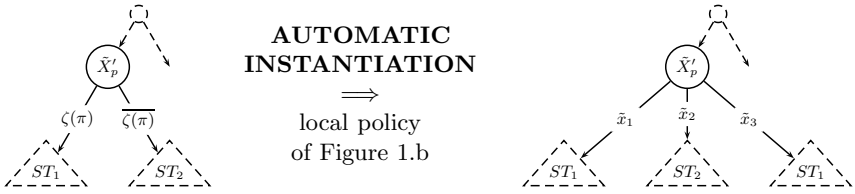


Fig. 3. Example of a decision tree containing a $\tilde{X}'_p$ node and one of its instantiations

Algorithm 2: Function `InstantiateXppSubtrees`

Data: $\mathcal{T}$ (generic node), $\mathcal{T}^\pi$ (instantiated node), π, τ, ζ
Result: $\mathcal{T}^\pi$ (instantiated tree)
begin
  $st_1 \leftarrow \mathcal{T}.son(\text{`}\zeta(\pi)\text{'})$;
  $st_2 \leftarrow \mathcal{T}.son(\text{`}\overline{\zeta(\pi)}\text{'})$;
  for $\tilde{v}'_p \in \tilde{V}_p$ **do**
    if $\tilde{v}'_p \in \zeta(\pi)$ **then** $\mathcal{T}^\pi.sons().push(\texttt{InstantiateTree}(st_1, \pi, \tau, \zeta, \tilde{v}'_p))$;
    else $\mathcal{T}^\pi.sons().push(\texttt{InstantiateTree}(st_2, \pi, \tau, \zeta, \tilde{v}'_p))$;
  return $\mathcal{T}^\pi$;
end

3.2 State variables depending on the reduced state variable

We can take advantage of our abstract model to introduce state variables that are defined for each value of the reduced state variable. In the case of our small exploration mission (Figure 1), let us consider a person to rescue in each region of the navigation subspace. Each value $\tilde{v}_p$ of the abstract navigation state variable corresponds to a subgoal to achieve, represented by a binary state variable $Y^{\tilde{v}_p}$ (see Figure 1.b).

Only can be achieved the subgoal corresponding to the region where the unknown local policy of our generic DBN is defined. The other subgoals can not be realized with this local policy, since it is not applicable inside the regions where

they are enclosed. Therefore, for each set $\left(Y^{\tilde{v}_p}\right)_{\tilde{v}_p \in \tilde{V}_p}$ of variables depending on the reduced variable, the generic DBN defines 2 abstract variables: the variable $Y^{\tau(\pi)}$ defined for the abstract value $\tau(\pi)$, and the variable $Y^{\overline{\tau(\pi)}}$ representing all the variables defined in the regions $\overline{\tau(\pi)}$.

Figure 4 depicts the decision trees of $Y^{\tau(\pi)}$ and $Y^{\overline{\tau(\pi)}}$ and an instance of their automatic instantiation for a given local policy. A decision tree containing a $Y^{\tau(\pi)}$ node is illustrated too.

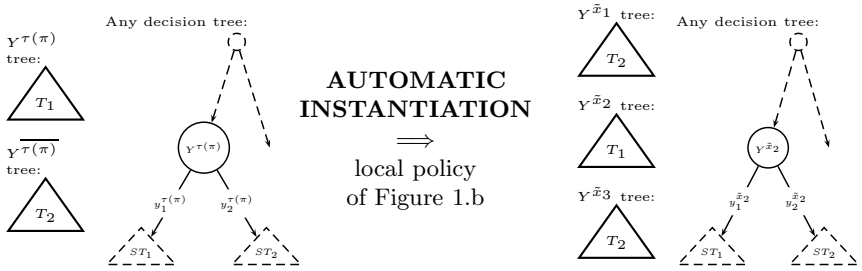


Fig. 4. Example of the decision trees of $Y^{\tau(\pi)}$ and $Y^{\overline{\tau(\pi)}}$, and of a decision tree containing a $Y^{\tau(\pi)}$ node. An automatic instantiation is presented.

Algorithm 3 details the automatic instantiation of the two abstract probability trees $\mathcal{T}_{Y^{\tau(\pi)}}$ and $\mathcal{T}_{Y^{\overline{\tau(\pi)}}}$. Since a node of any decision tree can be an abstract $Y^{\tau(\pi)}$ node (primed or not), it must be analyzed before being instantiated, as done in Algorithm 4.

Algorithm 3: Function `InstantiateYpTrees`

Data: $\mathcal{T}_{Y^{`\tau(\pi)'}}$, $\mathcal{T}_{Y^{`\overline{\tau(\pi)}'}}$, π, τ, ζ
Result: $\left(\mathcal{T}^{\pi}_{Y^{\tilde{v}_p}}\right)_{\tilde{v}_p \in \tilde{V}_p}$
$\mathcal{T}^{\pi}_{Y^{\tau(\pi)}} \leftarrow$ `InstantiateTree`$(\mathcal{T}_{Y^{`\tau(\pi)'}}, \pi, \tau, \zeta)$;
for $\tilde{v}_p \in \overline{\tau(\pi)}$ **do** $\mathcal{T}^{\pi}_{Y^{\tilde{v}_p}} \leftarrow$ `InstantiateTree`$(\mathcal{T}_{Y^{`\overline{\tau(\pi)}'}}, \pi, \tau, \zeta)$;

3.3 Abstract leafs of the generic probability trees

Due to action uncertainties, the outcome of a local policy is not deterministic. Let us consider for instance the local policy depicted in Figure 1.b: starting from region $\tilde{x}_2$, the local policy can lead to regions $\tilde{x}_1$ and $\tilde{x}_3$ with respectively probabilities $1-p$ and p. Let $\tilde{\mathcal{P}}^{\pi}$ be the abstract probability transition distribution over the partitioned subspace $\tilde{V}_p$ for the local policy π: this distribution is the stationary probability distribution of the markov chain resulting from application of the local policy π inside $\tau(\pi)$ [2].

Algorithm 4: Function `InstantiateNode`

Data: n (generic node), π, τ
Result: n^{π} (instantiated node)
begin
  if $n = Y^{`\tau(\pi)'[']}$ **then** $n^{\pi} \leftarrow Y^{\tau(\pi)[']}$;
  else $n^{\pi} \leftarrow n$;
  return n^{π};
end

The probabilities of obtaining the different values of any state variable may depend on the local policy probability distribution. These state variable probabilities are stored in the leafs of their probability trees. We suppose that they can be expressed as functions of 2 abstract local policy probabilities:

- $p^{\tau(\pi)}$: probability of staying in the region $\tau(\pi)$
- $p^{\zeta(\pi)}$: if the reduced post-action state variable ($\tilde{X}'_p$) is a parent node, probability of going to the value of the parent reduced state variable

An example of abstract probability leaf and one of its possible instantiations are shown in Figure 5. The abstract leaf is a formal algebraic expression of $p^{\tau(\pi)}$ and $p^{\zeta(\pi)}$. Given the abstract probability transition distribution $\tilde{\mathcal{P}}^{\pi}$ over the partitioned subspace $\tilde{V}_p$ for the local policy π, Algorithm 5 computes the probability of an instantiated leaf. It calls the function `Evaluate` from the computer algebra library to assess the leaf. If $\tilde{v}'_p \neq -1$, it means that $\tilde{X}'_p$ is a parent node of the leaf l, and l belongs to the $\tilde{v}'_p$-subtree of the $\tilde{X}'_p$ parent node.

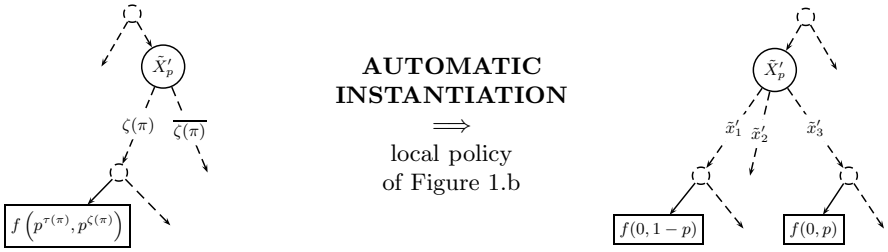


Fig. 5. Generic probability leaf example and one of its instantiations

3.4 Abstract leafs of the generic reward tree

Local policy transition probabilities are associated with local policy transition rewards. Let $\tilde{\mathcal{R}}^{\pi}$ be the transition rewards defined for the local policy π over the reduced state variable subspace. These reward transitions can be computed on the basis of the local policy transition probabilities just defined.

Algorithm 5: Function `InstantiateLeaf`

Data: l (generic leaf), π, τ, ζ, $[\tilde{v}'_p = -1]$
Result: l^π (instantiated leaf)
$p^{\tau(\pi)} \leftarrow \tilde{\mathcal{P}}^\pi(\tau(\pi), \tau(\pi))$;
if $\tilde{v}'_p \neq -1$ **then** $p^{\zeta(\pi)} \leftarrow \tilde{\mathcal{P}}^\pi(\tau(\pi), \tilde{v}'_p)$;
$l^\pi \leftarrow$ `Evaluate`$(l, `p^{\tau(\pi)}` = p^{\tau(\pi)}, [`p^{\zeta(\pi)}` = p^{\zeta(\pi)}])$;

As for the local policy transition probabilities, we suppose that the local policy transition rewards are formal algebraic expressions of:

- $r^{\tau(\pi)}$: average reward obtained if staying in $\tau(\pi)$
- $r^{\zeta(\pi)}$: if the reduced post-action state variable ($\tilde{X}'_p$) is a parent node, average reward if going to the value of the parent reduced state variable

Figure 5 still is a good example of a generic reward tree and its instantiation for the local policy of Figure 1.b, with the proviso of replacing $p^{\cdot}$ by $r^{\cdot}$. In the same way, Algorithm 5 presents the automatic reward leaf instantiation algorithm, with the proviso of replacing $p^{\cdot}$ by $r^{\cdot}$ and $\tilde{\mathcal{P}}$ by $\tilde{\mathcal{R}}$.

3.5 Main automatic DBN instantiation algorithm

Algorithm 6 automatically instantiates a decision tree for a given local policy. The version of our algorithm presented in this paper is recursive. It is called from functions `InstantiateXpSubtrees` and `InstantiateXppSubtrees`, when instantiating the subtrees of the nodes $\tilde{X}_p$ and $\tilde{X}'_p$ (see Algorithms 1 and 2). Notice that the optional argument $\tilde{v}'_p$ is not an input of `InstantiateXppSubtrees`: otherwise, it would mean that $\tilde{X}'_p$ is a parent node of itself, what is impossible.

4 Application to a search and rescue mission

We applied our generic MDP model to search and rescue missions described in section 1.1. We tested our generic model with 4 state variables (see Figure 6):

- R : regions of the environment (stands for $\tilde{X}_p$)
- $O.$: person to rescue in the region where the unknown local policy is defined (stands for $Y^{`\tau(\pi)'}$)
- $O_{\_}$: persons to rescue in the other regions (stands for $Y^{`\overline{\tau(\pi)}'}$)
- A : rotorcraft's autonomy (binary variable, `full` or `empty`)

In '$O.$' probability tree leafs, $Lp.$ stands for '$p^{\tau(\pi)}$' and $Lp_{\_} = 1 - Lp. = p^{\overline{\tau(\pi)}}$.

Table 1 shows the elapsed time comparison between automatic instantiation and optimization stages, when increasing the sizes of both the state and action spaces. Note that **the same generic DBN** was used to model all of the tested

Algorithm 6: Function `InstantiateTree` (recursive)

Data: $\mathcal{T}$ (generic decision tree), π, τ, ζ, $[\tilde{v}'_p = -1]$
Result: $\mathcal{T}^\pi$ (instantiated decision tree)
begin
  if $\mathcal{T}.root().type() = leaf$ **then** $\mathcal{T}^\pi \leftarrow$ `InstantiateLeaf`$(\mathcal{T}.root(), \pi, \tau, \zeta, \tilde{v}'_p)$;
  else
    $\mathcal{T}^\pi \leftarrow$ `InstantiateNode`$(\mathcal{T}.root(), \pi, \tau)$;
    switch $\mathcal{T}.root()$ **do**
      case $\tilde{X}_p$: $\mathcal{T}^\pi \leftarrow$ `InstantiateXpSubtrees`$(\mathcal{T}.root(), \mathcal{T}^\pi, \pi, \tau, \zeta, \tilde{v}'_p)$;
      case $\tilde{X}'_p$: $\mathcal{T}^\pi \leftarrow$ `InstantiateXppSubtrees`$(\mathcal{T}.root(), \mathcal{T}^\pi, \pi, \tau, \zeta)$;
      otherwise
        for $subtree \in \mathcal{T}.root().sons()$ **do**
          $\mathcal{T}^\pi.sons().push($`InstantiateTree`$(subtree, \pi, \tau, \zeta, \tilde{v}'_p))$;
  return $\mathcal{T}^\pi$;
end

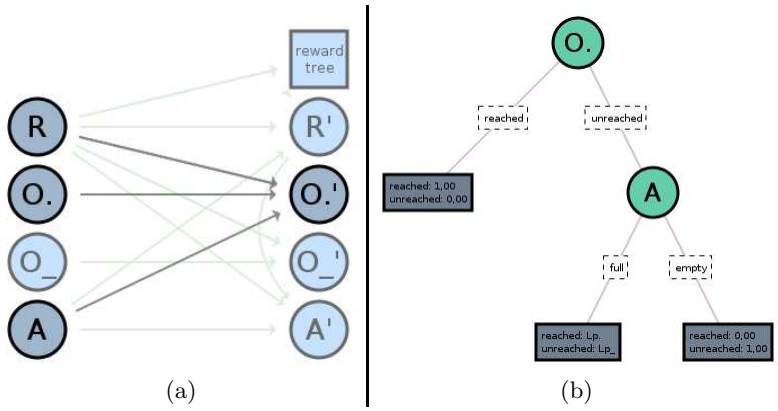


Fig. 6. (a) Generic DBN and (b) $O.$ generic probability tree (software screenshot)

Nb of enumerated states	Nb of regions (states per region)	Nb of generated local policies	**DBNs instantiation time**	MDP optimization time
82944	9 (9)	21	**0.01**	0.12
746496	9 (81)	61	**0.01**	16.77
58982400	17 (9)	69	**0.03**	1621.62
530841600	17 (81)	117	**0.06**	$>$ 1 hour

Table 1. Elapsed time comparison between instantiation and optimization stages, for growing size search and rescue missions (in seconds, with a P4-2.8GHz processor)

instances. First, it appears that the number of states exponentially grows with number of regions, so that unstructured enumerated models of MDP would have been very tedious and quite impossible to model. Second, the number of generated local policies (automatic generation algorithm of [4]) is round 100, what means that usual factored MDP models would have required to manually input a hundred or even more DBNs, in order to define our real search and rescue missions. On the contrary, our generic hierarchical DBN model enables to define **only one DBN** for the whole mission. Third, the automatic DBNs intanciation time is insignificant compared to the optimization time ($< 1\%$): it confirms the modeling and effectiveness benefits of our approach.

5 Conclusion

In this paper, we proposed an original generic hierarchical framework for modeling large factored Markov Decision Processes. Our approach is based on a decomposition into regions of the state subspaces engendered by the state variables with large arity. The regions are macro-states of the thus abstracted MDP. Local policies can then be computed (or defined by other means) in each region of the decomposition and taken as macro-action of the abstract MDP. The factored MDP model is then defined at the abstract level. A generic DBN template can be defined, symbolically parametrized by the local policies. We illustrated and showed the significance of our method on real instances of search and rescue aerial robotics missions (within the ReSSAC project) where the navigation subspace can easily be decomposed into regions: the use of classical unstructured MDP models would have been very tedious and perhaps impossible for the kind of real planning missions we tackle.

References

1. Craig Boutilier, Richard Dearden, and Moises Goldszmidt. Stochastic dynamic programming with factored representations. *Artificial Intelligence*, 121(1-2):49–107, 2000.
2. Milos Hauskrecht, Nicolas Meuleau, Leslie Pack Kaelbling, Thomas L. Dean, and Craig Boutilier. Hierarchical solution of markov decision processes using macro-actions. In *Proceedings 14th UAI*, pages 220–229, San Francisco, CA, 1998.
3. Jesse Hoey, Robert St-Aubin, Alan Hu, and Craig Boutilier. Optimal and approximate stochastic planning using decision diagrams. Technical Report TR-2000-05, University of British Columbia, 10 2000.
4. Ron Parr. Flexible decomposition algorithms for weakly coupled markov decision problems. In *Proceedings 14th UAI*, pages 422–430, San Francisco, CA, 1998.
5. Martin L. Puterman. *Markov Decision Processes.* John Wiley & Sons, INC, 1994.
6. R. Sabbadin. Graph partitioning techniques for markov decision processes decomposition. In *Proceedings 15th ECAI*, pages 670–674, Lyon, France, July 2002.