TeXDYNA: Hierarchical Reinforcement Learning in Factored MDPs *

Olga Kozlova[1], Olivier Sigaud[1] and Christophe Meyer[2]

[1] Institut des Systèmes Intelligents et de Robotique
Université Pierre et Marie Curie - Paris 6, CNRS UMR 7222
4 place Jussieu, F75252 Paris Cedex 05
Olivier.Sigaud@upmc.fr, Olga.Kozlova@isir.upmc.fr
[2] Thales Security Solutions & Services, ThereSIS Research and Innovation Office
Route départementale 128, F91767 Palaiseau Cedex
Christophe.Meyer@thalesgroup.com

Abstract. Reinforcement learning is one of the main adaptive mechanisms that is both well documented in animal behaviour and giving rise to computational studies in animats and robots. In this paper, we present TeXDYNA, an algorithm designed to solve large reinforcement learning problems with unknown structure by integrating hierarchical abstraction techniques of Hierarchical Reinforcement Learning and factorization techniques of Factored Reinforcement Learning. We validate our approach on the LIGHT BOX problem.

1 Introduction

One central issue in animat research consists in facing the complexity of the real world with adaptive methods. Among such adaptive methods, reinforcement learning (RL) is one of the most studied. However, due to the "curse of dimensionality" inherent to the Markov Decision Process (MDP) framework, standard RL algorithms cannot address large scale real world problems mostly because they must enumerate all states [1]. In order to apply RL to robotics problems, two main lines of research can be followed. One consists in making profit of the structure of the sensory system of the robot using Factored MDPs (FMDPs), using a representation where a state is implicitly described by an assignment of values to some set of state variables. Then, one can exploit the dependencies between variables to get a compact representation [2]. The second line of research consists in making profit of the structure of the decision problem itself, using Semi-MDPs (SMDPs), an extension of MDPs where the number of time steps between one decision and the next is a random variable. The SMDP framework is the basis for Hierarchical Reinforcement Learning (HRL) algorithms as it decomposes the original task into subtasks that are easier to solve individually [3].

The methods to solve FMDPs and SMDPs usually assume that the structure of the problem is given, though that structure is rarely available in practice. As to hierarchical representations, HEXQ [4] and VISA [5] are two algorithms designed to solve this problem. Besides, for FMDPs, [6] have proposed SDYNA to solve Factored Reinforcement

* This work was founded by CIFRE convention - 1032/2006

Learning (FRL) problems. Here, we propose TeXDYNA, an algorithm that combines the benefits of HRL and FRL, building an HRL-augmented version of SDYNA.

The paper is organized as follows. First, in Section 2, we give the minimal background necessary to understand what follows. In Section 3, we present TeXDYNA, introducing the method for discovering options from the internal structure of the problem and showing how to introduce options into FRL algorithms. In Section 4, we present an experimental study of our system based on the LIGHT BOX problem. In Section 5, we discuss the contributions and limitations of our work as well as related work.

2 A quick index to the background

An FMDP is described by a set of state variables $S = \{X_1, \ldots, X_n\}$, where each X_i takes its value in a finite domain $Dom(X_i)$. Structured Dynamic Programming (SDP) algorithms such as SVI [7] make profit of this structure to compute a policy compactly. Structured-DYNA (SDYNA) [6] is a general framework that adapts indirect RL of the DYNA family [8] to the FMDP framework. SPITI is a particular instance of SDYNA based on a decision trees induction process to learn the structure of the problem and on SVI to obtain an efficient policy. While the FMDP representation technique reduces the size of the representation of state-action space of large problems by decomposing states into a set of random variables, the HMDP representation decomposes the overall state-action space into a set of smaller state-action spaces each of which can be factored. SMDPs are an extension to MDPs where the number of time steps between two decisions is a random variable. Among several other frameworks (see [3] for an overview), the options framework [9] is designed to solve SMDPs by building a hierarchy of options. An option is a tuple $\langle \mathcal{I}, \pi, \beta \rangle$, where $\mathcal{I} \subseteq S$ is an initiation set, that is a subset of states in which it is possible to execute o, $\pi : S \times O \rightarrow [0, 1]$ is a policy executed in o, and $\beta : S \rightarrow [0, 1]$ is a termination condition function, that is the probability of terminating the option in each state. Options are a generalization of primitive actions including temporally extended courses of actions. As a result, the algorithms deal only with options and do not have to distinguish options from primitive actions. Previous algorithms combining FMDP representation and HMDP structure learning are HEXQ [4] and VISA [10]. The work most closely related to ours, Incremental-VISA [11, 12] is discussed in Section 5.

3 TeXDYNA

TeXDYNA hierarchically decomposes an FMDP by automatically splitting it into a set of options. Meanwhile, the local policy of each option is incrementally improved by a SDYNA-like approach. The central contribution of this work comes from the fact that the discovery of options and the construction of the model of the FMDP, as well as policy computation, are simultaneous. To achieve simultaneous SMDP structure learning, FMDP structure learning as well as local and global policy computation, TeXDYNA is built on top of SPITI. There are two main advantages to our approach. First, we make profit of the learning method used in SDYNA to learn only a local model for each option. As a result, the models are smaller and, therefore, easier to learn. Second, introducing options in the planning stage results in the possibility to plan over

smaller partitions of the state-action space. In order to decompose hierarchically the overall FMDP into sub-FMDPs represented by options, TeXDYNA builds a global transition function that represents the structure of the problem and uses this function to build a hierarchy of options. For each option, TeXDYNA computes a local transition function and a local policy. Therefore, TeXDYNA can be decomposed into two simultaneous processes: (1) Learning options: learning the transition function of the overall FMDP (updating the FMDP model with (s,a,$s\prime$,r)) and adding or updating options (Algorithm 1); (2) Planning with options: using a modified version of the SPITI algorithm for model learning, planning and acting with options, i.e. updating hierarchical policy $\pi = \langle \pi_{o_0}, \pi_{o_1}, ...\pi_{o_n}, \rangle$(Algorithm 2).

3.1 Defining options

The purpose of our approach is to decompose the overall FMDP into smaller subtasks or mutually independent stand-alone policies. Our representation takes advantage of the FMDP structure using temporal abstraction techniques. We use a specific options representation inspired from the goal-oriented exit options of VISA [10] and HEXQ [4], where options are defined by their exit states that can be seen as subgoals of the task. Our options are noted o: $o = \langle \mathcal{I}, \pi, e \rangle$ where $\mathcal{I} \subseteq S$ is an initiation set, π is a policy executed in o and e is the related exit. We do not use the usual termination function β since it is defined by the exit and computed at each time step during the option execution.

An exit corresponds to changes of values of state variables linked to the reward function, as in HEXQ. However, unlike HEXQ, where exits are state-action pairs, we define exits as a tuple $\langle v, a, v_{ch}, c \rangle$, where v is the variable whose value is changed by this exit, a is the exit action that makes the value of v change at the next state, v_{ch} is a variable change, i.e. a pair of values $\langle x, x' \rangle$ where x is the value before a is executed and $x\prime$ the value after a is executed. In the stochastic case, the variable change is a probability distribution over v_{ch} and the highest probability is kept in the exit definition. Finally, $c = \{x_1, ..., x_n\}$ is the context, that is the set of constraints (i.e. assignment of values to a subset of state variables) that makes this exit available. In this representation, the primitive actions have an empty context. Thus, there is at most one exit per action and per variable change. Thus, each option corresponds to a unique exit.

The initiation set I_o of option o defines the state space where this option can be executed. On the one hand, it determines if the resources necessary for the successful execution of the option are available, and, on the other hand, it specifies the state space from where the exit of the option is reachable. In practice, it is the union of its own exit context and all the exit contexts of its sub-options. If a sub-option is a primitive action, its exit variable is added to I_o with all possible values. Otherwise, the exit variable of the sub-option is added with its value change. By convention, a sub-option with an empty initiation set is admissible everywhere. This is particularly true for primitive actions. Therefore, all the values of the corresponding exit variables of these options are accepted. Thus, I_o contains all the states from which the exit of the option is reachable. This property of direct reachability is ensured by the fact that the exit context copies the constraints of the corresponding branch in the transition tree. In other words, the nodes of the branch represent the variables of the context. When those variables can be

changed by a sub-option, the exit of the parent option becomes reachable from all the states where its sub-options are accessible.

The hierarchical structure of the options set is determined by the variables interdependencies expressed in the structure of the transition function. However, for the sake of sound planning, a rank is assigned to each option as the highest rank of its sub-options plus one. As a result, the planning algorithm chooses an option to execute at each abstraction level in the hierarchy by going down from the most abstract options to primitive actions. Transition model learning and options discovery are simultaneous. Thus the options discovered first represent the transitions learned first and consequently have less constraints as they represent the most accessible subgoals. If the problem has a hierarchical structure, this gives a bottom-up direction to the options discovery. First the options with the lowest abstraction level are discovered, then their execution gives access to more constrained options. Defined this way, each option represents a sub-FMDP $\mathcal{M}_o = \langle S_o, O_o, \Psi, T_o, R_o \rangle$ containing a reduced partition of the initial state-action space, where S_o is a set of context variables, O_o is a set of sub-options, Ψ is a set of admissible state-option pairs defined by the initiation set, T_o is the local transition function and R_o is the local reward function. In this respect, when an option is created, we initialize its local FMDP tree structure composed of the transition trees for its context variables, as well as a local planning algorithm. Thus, when learning the internal FMDP structure of the option, the states injected in the learning algorithm are reduced to contain only variables x_i such that $\forall i, x_i \in X_o$.

3.2 Learning: Adding and updating options

To discover options, the overall task transition function is learned in a decision tree form. The FMDP model provides the structure used for the options discovery process, as this structure represents the dependencies between variables and constraints under which those variables change their values.

Algorithm 1 describes the procedure for adding or updating options. An option is introduced each time there is an action that can change one variable value. The sub-options are the options available in the sub-FMDP represented by the corresponding parent option. Sub-options are added in the following way: for each variable in the context of the parent option, if the set of options $\mathcal{E}$ contains an option that modifies the value of this variable, this option is added to the set of sub-options. Note that, when computing the context of an exit, the exit variables are excluded from the context to avoid cross-dependencies between options. This procedure supposes that there is one option per variable change and that each option changes at most one variable value. For instance, if some options change more than one variable, then one variable can be changed by more than one option. As a consequence, these options would be sub-options of one another, creating loops in the hierarchy. Nevertheless, this constraint can be relaxed by reorganizing the hierarchy once the options have been added. The procedure is the following: if a cross-dependency is detected between two options, the hierarchical link between them is removed and both options are attributed the lowest rank of the two. This way, more than one option may have the same exit variable.

The options are defined over the model of the problem while this model is learned. Thus some are incomplete or erroneous, especially in the first stages of the learning

Algorithm 1: Add and update options

init : options set $\mathcal{E} = \emptyset$
input: FMDP $\mathcal{F} = [\forall x_i \in X : Tree(P(x'|x))]$

1 **forall** *transition tree* $Tree(P(x'|x)) \in \mathcal{F}$ **do**
2 **forall** *leaf* l *of the* $Tree(P(x'|x))$ **do**
3 **if** *branch contains an action* a *&* a *modifies the value of the variable* x *in leaf* l **then**
4 **if** *if* $\mathcal{E}$ *does not contain a definition of option* o *with the exit corresponding to variable* x *and action* a **then**
5 introduce new option o defined by exit $e : \langle v, a, v_{ch}, c \rangle$ with :
6 • variable $v \leftarrow x$
7 • action $a \leftarrow$ current action a
8 • variable change $v_{ch} \leftarrow \langle$value in the branch, value in the leaf$\rangle$
9 • context $c \leftarrow$ variables of the branch that leads to the leaf l
10 **else if** $\mathcal{E}$ *contains a partial definition of* o **then**
11 update o with new information

process. To ensure the relevance of discovered options, they are updated every time the model of transitions changes. The algorithm checks if the set of options $\mathcal{E}$ already contains an option defined by the same action and variable, but with a different context definition. If so, it updates it (line 10 in Algorithm 1) using the following procedure:

1. Update the exit context c of the option;
2. Update the transitions trees (add missing ones and discard irrelevant ones);
3. Update the sub-options list (add missing ones and discard irrelevant ones);
4. Re-compute the rank k of the option;
5. Re-compute the Initiation set I_o.

As to incrementality, [6] proposes to re-initialize the value function each time the reward function changes, mainly because the structure of the value function results from the reward function. In the incremental options learning case, the value function tree is reset each time the reward function changes in order to take into account every modification that changes the structure of the policy. There are exactly as many options as possible variable value changes. Thus there is no need to remove incorrect options given that as soon as their context is correct, they become accurate. Meanwhile, however, inaccurate options can influence planning and exploration by building an erroneous policy (see the next section).

3.3 Planning: FRL over options

The planning stage builds a hierarchical policy over options by incrementally improving and modifying it simultaneously with the learning process. The planning algorithm, built upon the ideas coming from HRL algorithms and FRL methods based on SPITI, is given in Algorithm 2.

Algorithm 2: SPITI with options

input : FMDP $\mathcal{F}$, hierarchy of options $\mathcal{E}$

for each *time step* t

1 **if** *no option is running* **then**
  option $o \leftarrow$ChooseOption(s, $Tree(\pi)$, $\mathcal{E}$)

2 **if** *terminal condition of* o *is satisfied* **then**
2.1 execute exit action a; observe next state $s\prime$ and immediate reward r
2.2.a update local FMDP $\mathcal{F}_o$ with (s,a,$s\prime$,r)
2.2.b update parent FMDP $\mathcal{F}_{parent(o)}$ with (s,o,$s\prime$,r)
2.3.a update local policy π_o with $\mathcal{F}_o$
2.3.b update parent policy $\pi_{parent(o)}$ with $\mathcal{F}_{parent(o)}$

3 **else**
  sub-option $i \leftarrow$ChooseOption(s, $Tree(\pi_o)$, $\mathcal{E}$)
  if *sub-option* i *is primitive action* **then**
3.1 execute i; observe $s\prime$ and r
3.2 update local FMDP $\mathcal{F}_o$ with (s,i,$s\prime$,r)
3.3 update local policy π_o with $\mathcal{F}_o$
3.4 **return**: i
4 **else**
  call SPITI over sub-option i : $i \rightarrow$SPITI ($\mathcal{F}_i$, $\mathcal{E}$)

To operate with options and take into account the hierarchical structure of the policy, we use a modified instance of SPITI for model learning, planning and acting with options. The algorithm executes options recursively by going down the hierarchy of options up to primitive actions that can be executed by the agent in its environment, then performs the updates by going up in the hierarchy. This upward update guarantees that the model of transitions includes the changes that occur during the incremental learning process. Thus, local planning is performed for each option with respect to the current structure of its local model of the transition and reward functions. The inner loop of SPITI is decomposed into three stages:

- *Acting*: choose an action using ϵ-$greedy$ exploration;
- *Learning*: update the model of the transition and reward functions of the FMDP from $\langle X, a, X', R\rangle$ observations using ITI;
- *Planning*: update $Tree(V)$ and $Tree(\pi)$ using one sweep of SVI.

Then, in the learning stage, the ITI algorithm is modified to work with options by replacing primitive actions by options in trees and allowing to remove or add new transition trees while learning. Moreover, ITI uses reduced states representations containing only variables present in the context of the option, so that the local model is updated with observations $\langle S_o, a, {S_o}', R\rangle$ where S_o is a set of local context variables.

Finally, in the planning stage, the adaptation of SDP algorithms like SVI to the options framework is straightforward. Since the action space is restricted by the number of sub-options available on a given hierarchical level, the algorithm has an additional argument that specifies the list of options to iterate through.

The primitive actions available in the environment are all initialized as options with empty exit context and initiation set. TeXDYNA builds the hierarchical policy $\pi = \langle \pi_{o_0}, \pi_{o_1}, ...\pi_{o_n}, \rangle$ where a higher level policy is based on a set of lower level policies. Each option follows the policy of its FMDP, its sub-options follow their respective policies and so on. Moreover, to propagate the external rewards to the local policies of options, when a high level option is discovered, an additional reward, named "internal reward" r_o (by contrast with the external reward received from the environment) is assigned to its exit action. We set $r_o = \frac{R_o}{2}$, where R_o is the internal reward of the parent option. For the options on the top level of the hierarchy, R_o is the maximal immediate external reward that the agent can get. This heuristics is inspired by the "salient event" heuristics introduced in [13].

As mentioned above, in the first iterations, the hierarchy of options and often options themselves are inaccurate or irrelevant. This is why the procedure to choose options (Algorithm 3) must take inaccuracies into account and use a strategy that favors exploration in the first steps and chooses the options at the right level of the hierarchy.

Algorithm 3: Choose Option

input: current state s, policy $Tree(\pi)$, hierarchy of options $\mathcal{E}$, level k

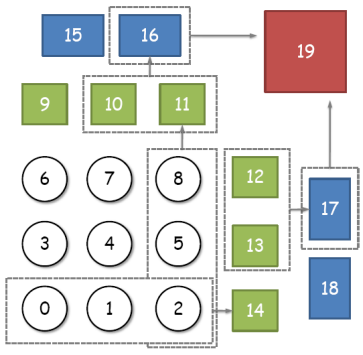
Get option o from to the current policy, $o = max_o[leaf(Tree(\pi)|s)]$
if *o is null* **then**
    choose option o from $\mathcal{E}$ accessible in the current state s with current rank k
while *o is null* **do**
    choose option o from $\mathcal{E}$ accessible in the current state s with rank $k-1$
    (if $k = 0$ choose random primitive action)
return o

The root node of the option hierarchy represents the overall FMDP. The first option o is chosen according to the root policy while its initiation set contains the current state s. The sub-options are selected according to the internal policy π_o of the option o augmented with an *ϵ-greedy* exploration policy. If the policy is incomplete and does not contain any information about options available in a current state, then an option is chosen randomly from the options set $\mathcal{E}$ on the current rank level with respect to its initiation set. If no option is available with a given rank, the algorithm chooses from options with lower rank and so forth. Thus, in the first stages of learning, only the options with rank 0 are available, that is primitive actions chosen randomly and in the end the preference always goes to the options with the higher level of abstraction. This provides a high level of exploration at the beginning and then for unexplored parts of the state space, therefore this speeds up the overall learning process.

In order to avoid an option from being stuck within an erroneous policy that fails to achieve its exit, we introduce the notion of *selection penalty* that forbids one option for a given number of time steps. In practical terms, when an option reaches its exit state but fails to change the corresponding variable value, that means that its structure

is inaccurate. In this case we forbid this option selection for $\frac{(max\ time\ steps\ per\ episode)}{10}$ time steps in order to let the model of transitions of its parent option be improved.

4 Experimental study



(a)

Light	Depends on
9	0 3 6
10	1 4 7
11	2 5 8
12	6 7 8
13	3 4 5
14	0 1 2
15	9 10
16	10 11
17	12 13
18	13 14
19	16 17

(b)

Fig. 1. (a) The LIGHT BOX problem: number and color of "lights" with their dependencies. (b) Internal dependencies of the LIGHT BOX problem.

Here, we have chosen to evaluate TEXDYNA on the LIGHT BOX domain [11], presented in Fig. 1. It consists of a set of twenty "lights", each of which is a binary variable corresponding to ON and OFF, named "0", "1", "2", etc. Each light has a corresponding action that toggles the light ON or OFF. Thus there are twenty actions, $2^{20} \approx 1$ million states, thus approximately 20 million state-action pairs. The 9 white lights are simple toggle lights that can be turned ON or OFF by executing the 9 corresponding actions. The green lights are toggled similarly, but only if certain configurations of white lights are ON, with each green light having a different set of dependencies. Similarly, the blue lights depend on certain configurations of green lights being ON, and the red light depends on configurations of the blue lights. The goal is to turn the red light on, in which case the agent receives a reward of 20.

The results presented below are averaged over 20 runs of 150 episodes where each episode is limited to 300 steps. The curves are smoothed by computing the moving average weighted over ten neighboring values. The algorithms use the following parameters: $N = 300$ in DYNA-Q and $\epsilon = 0.1$ in ϵ-$greedy$. The ITI algorithm uses $\chi^2 = 30$ in stochastic problems and $\chi^2 = 0$ in deterministic ones. The algorithms are coded in C# and run on Intel Core2Duo 1.80GHz processor with 2Go RAM.

An example of hierarchy of options obtained on the LIGHT BOX problem is given in Fig. 2. The complete state representation in the LIGHT BOX problem is given by 20 variables corresponding to 20 lights, but the states used to update the internal structure of the FMDP corresponding to the option $toggle16$ contains only 2 variables "10" and

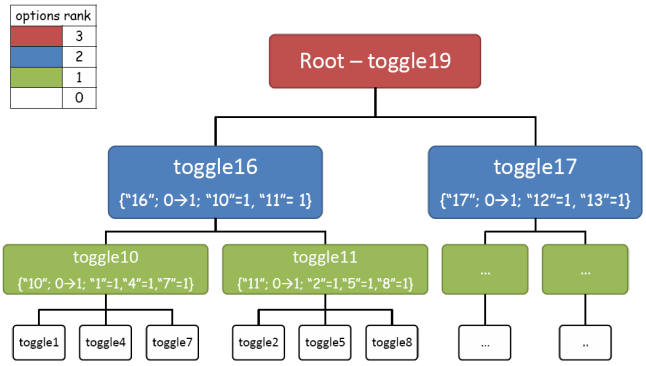


Fig. 2. Example of options discovered in the LIGHT BOX problem.

"11". As a consequence, the sub-FMDP has 2 variables $\times$ 2 sub-options instead of 20 variables $\times$ 20 actions.

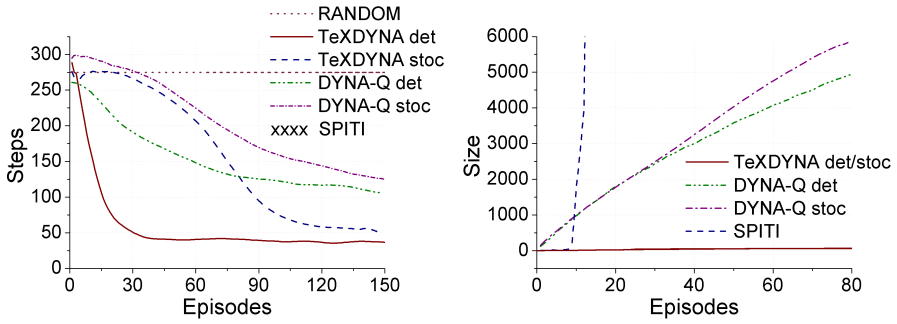


Fig. 3. (a) Convergence over episodes on the LIGHT BOX problem. (b) Policy size on the LIGHT BOX problem.

Fig. 3 shows the performance in number of time steps required to complete one episode of the LIGHT BOX problem within three experimental contexts: random policy, TEXDYNA and DYNA-Q applied to both the deterministic version of the LIGHT BOX problem and a stochastic version where the agent performs random actions with a 10% probability. We could not obtain the convergence curve for SPITI because SPITI attempts to build the complete value tree representing all possible combinations that is 1 million leaves. This assertion is explained by the results presented in Table 1 that recaps the average time in seconds per step and the size of the functions in the stochastic LIGHT BOX problem. TEXDYNA requires less time to perform one step since it works on a smaller representation. While SPITI is struggling to check if the red light is dependent on the white light, the policy built by TEXDYNA goes straight to the goal state by achieving subgoals one by one at each level.

	TEXDYNA	SPITI	DYNA-Q
Transition function size	780 ± 14	790 ± 25	–
Value function size	240 ± 20	> 15000	> 10000
Policy function size	180 ± 8	> 15000	> 10000
Time/step(sec)	0.04	> 100	> 2

Table 1. Performance on the stochastic LIGHT BOX problem (Policy and value function size in total number of nodes in decision trees).

The algorithm finds the same structure in the stochastic case as in the deterministic one. One can notice that it takes more time to the algorithm to learn a stochastic transition function, but in the end it discovers the same hierarchy of options and the same policy and value functions. Similarly, stochastic DYNA-Q needs more episodes than its deterministic counterpart to converge. Furthermore, as shown in Table 1, TEXDYNA builds much smaller representations than its competitors in the stochastic case.

In order to explain the lesser performance of SPITI compared to TEXDYNA, we record the policy function size for SPITI and TEXDYNA, and the number of state-action pairs in DYNA, at the end of each episode. Fig. 3(b) shows the evolution of the corresponding functions size over episodes. TEXDYNA quickly reaches a plateau on its optimal policy size, while the policy discovered by DYNA continues to grow up until representing all possible states. Therefore, DYNA results given in Fig. 3(a) are biased by the fact that even if it discovered a kind of sub-optimal policy in a reasonable time, the algorithm is unable to perform the complete policy computation. As to SPITI, the system fails to achieve convergence because of a too strong memory requirement. Indeed, as soon as one tree size exceeds 15000 nodes, the system runs out of memory. Further code optimization might reduce this limitation, but the main point is that exponential task structure simplification allows TEXDYNA to scales much better. Indeed, local models combined with state reduction to the set of context variables ignoring anything else provides a significant state-action space reduction within the structure of each option.

5 Discussion

TEXDYNA ideas are first inspired by Sutton's DYNA architecture [8], enriched and adapted to FMDPs by [6]. Second, as to the exit oriented options representation, some ideas come from the HEXQ [4] and VISA [10, 5] frameworks, where the exit definition proposed in HEXQ is extended to include variable change and context in order to address more complex structures. TEXDYNA builds a hierarchy of options online and directly from the transition trees taking advantage of their structure, whereas VISA builds a variable influence graph from the given DBNs and then builds transition graphs and reachability trees to determine the initiation sets of the options. Furthermore, it only discovers options linked to the variables directly connected to reward, while TEXDYNA backpropagates the reward among subgoals.

Finally, Incremental-VISA [11, 12] is adapting VISA to the case where the model of the problem is not known in advance. Like TEXDYNA, it attempts to simultaneously

learn the hierarchical (options discovery) and factored (DBN learning) structure of the MDP. The approach learns incrementally and autonomously both the causal structure of the environment and useful skills that exploit this structure. It uses DBN structure learning techniques to learn the environment structure and SDP algorithms like SVI to build hierarchical policies online. The authors propose an active learning scheme to improve the efficiency with which this structure is acquired that bootstraps on existing structural and procedural knowledge. As new structure is discovered, more complex skills are learned, which in turn allow the agent to discover more structure, and so on. The major differences between TeXDYNA and Incremental-VISA is, for the moment, that the latter is limited to the deterministic case, whereas TeXDYNA is adapted to stochastic problems. The second difference lies in the way of introducing options into planning: Incremental-VISA waits the option to be "mature enough" before introducing it in the hierarchy by using a measure of entropy on the transition functions whereas TeXDYNA inserts options directly in the hierarchy in order to accelerate its completion. In the same way as VISA, Incremental-VISA uses the DBNs to discover dependencies between state variables. Therefore, it needs to build intermediate graphs and trees to catch the internal hierarchical structure, while TeXDYNA operates on decision trees and discovers the structural links directly. Unfortunately, we were not able to compare the performance mainly because of the absence of a common metrics. Indeed, the experimental results published in [12, 11] are based on the number of value changes and the time to compute the policy without giving the corresponding metrics criteria.

At this point in time, the algorithm presents some limitations. Firstly, instead of using "internal reward" to propagate the external reward to local policies so that all options have fixed interest, the options discovery algorithm could be combined with task-specific knowledge to identify useful, salient or challenging subroutines. Secondly, we consider that there is at most one option per variable value change. This assumption simplifies computations within the algorithms, but can be relaxed. More importantly, as to the problem representation, our option-specific state abstraction is strongly goal-oriented, that is reaching a unique exit context. This can result in the creation of excessive number of options in problems where an action can change more than one variable at the same time. Finally, the hierarchy of options is strictly ordered, that means that we cannot address problems where the FMDP structure includes synchronic arcs or post-action variable dependencies, because it would introduce cross-dependencies between options and cycles in the hierarchy of options.

6 Conclusion

We have presented TeXDYNA, a powerful framework that combines factored and hierarchical reinforcement learning. This framework is built on three main ideas:

- The use of the transition function structure represented as decision trees to discover options results in efficient learning and planning capabilities that are integrated into the (factored) model-based RL framework.
- The localization of the models results in an exponential reduction of the state-action space of each option. Instead of taking transition trees from the global structure,

the models of transitions are learned locally for each option. This accelerates the solution process.
- The immediate use of the just discovered options in the planning process speeds up the learning of its internal structure and of its parent option structure.

We have evaluated on the LIGHT BOX problem the capability of TEXDYNA to efficiently generate hierarchical policies and shown that it performs better than its non factored and non hierarchical ancestors. The main issue for future work consists in replacing the basic ϵ-$greedy$ exploration strategy by a more sophisticated policy, either along the "optimism in the face of uncertainty" line [14] or based on adding internal motivations such as an artificial curiosity process [15, 13] into the framework. Then the application of TEXDYNA to a robotics problem will be the matter of a more experimental work.

References

1. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press, Cambridge, MA (1998)
2. Boutilier, C., Dearden, R., Goldszmidt, M.: Exploiting structure in policy construction. In: Proceedings of the 14th International Joint Conference on Artificial Intelligence. (1995) 1104–1111
3. Barto, A., Mahadevan, S.: Recent advances in hierarchical reinforcement learning. Discrete Event Systems Journal **13** (2003) 41–77 Special Issue on Reinforcement Learning.
4. Hengst, B.: Discovering hierarchy in reinforcement learning with HEXQ. In: Proceedings of the 19th International Conference on Machine Learning. (2002) 243–250
5. Jonsson, A., Barto, A.: Causal graph based decomposition of factored MDPs. Journal of Machine Learning Research **7** (2006) 2259–2301
6. Degris, T., Sigaud, O., Wuillemin, P.H.: Learning the structure of factored markov decision processes in reinforcement learning problems. In: Proceedings of the 23rd International Conference on Machine Learning, Pittsburgh, Pennsylvania, ACM (2006) 257–264
7. Boutilier, C., Dearden, R., Goldszmidt, M.: Stochastic dynamic programming with factored representations. Artificial Intelligence **121**(1-2) (2000) 49–10
8. Sutton, R.S.: DYNA, an integrated architecture for learning, planning and reacting. In: Working Notes of the AAAI Spring Symposium on Integrated Intelligent Architectures. (1991)
9. Sutton, R., Precup, D., Singh, S.: Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. Artificial Intelligence **112** (1999) 181–211
10. Jonsson, A.: A causal approach to hierarchical decomposition in reinforcement learning. PhD thesis, University of Massachusetts Amherst (2006)
11. Vigorito, C.M., Barto, A.G.: Autonomous Hierarchical Skill Acquisition in Factored MDPs. In: Yale Workshop on Adaptive and Learning Systems, New Haven, Connecticut (2008)
12. Vigorito, C., Barto, A.: Hierarchical Representations of Behavior for Efficient Creative Search. In: AAAI Spring Symposium on Creative Intelligent Systems, Palo Alto, CA. (2008)
13. Singh, S., Barto, A., Chentanez, N.: Intrinsically motivated reinforcement learning. Advances in Neural Information Processing Systems **18** (2004) 1281–1288
14. Szita, I., Lörincz, A.: The many faces of optimism: a unifying approach. In: Proceedings of the 25th International Conference on Machine Learning, New York, NY, USA, ACM (2008) 1048–1055
15. Oudeyer, P.Y., Kaplan, F., Hafner, V.: Intrinsic motivation systems for autonomous mental development. IEEE Transactions on Evolutionary Computation **11** (2007)