Automatic Generation of an Agent's Basic Behaviors

Olivier Buffet
buffet@loria.fr

Alain Dutech
dutech@loria.fr

François Charpillet
charp@loria.fr

LORIA, BP 239
F-54506 Vandœuvre-lès-Nancy

ABSTRACT

The agent approach, as seen by [9], intends to design "intelligent" behaviors. Yet, Reinforcement Learning (RL) methods often fail when confronted with complex tasks. We are therefore trying to develop a methodology for the automated design of agents (in the framework of Markov Decision Processes) in the case where the global task can be decomposed into simpler -possibly concurrent-sub-tasks. Our main idea is to automatically combine basic behaviors using RL methods. This led us to propose two complementary mechanisms presented in the current paper. The first mechanism builds a global policy using a weighted combination of basic policies (which are reusable), the weights being learned by the agent (using Simulated Annealing in our case). An agent designed this way is highly scalable as, without further refinement of the global behavior, it can automatically combine several instances of the same basic behavior to take into account concurrent occurences of the same subtask. The second mechanism aims at creating **new** basic behaviors for combination. It is based on an incremental learning method that builds on the approximate solution obtained through the combination of older behaviors.

Categories and Subject Descriptors

I.2.11 [**Computing Methodologies**]: Distributed Artificial Intelligence—*Intelligent Agents*; I.2.6 [**Computing Methodologies**]: Learning

General Terms

Algorithms, Design, Experimentation

Keywords

reinforcement learning, scalability, adaptation, complex environments, Markov Decision Processes

1. INTRODUCTION

Our researches aim at automatically designing the behavior of *reactive situated* agents limited to only *local perceptions*. Reinforcement Learning (RL)[11] is a good candidate for this type of problem. Nevertheless, as RL algorithms suffer from combinatorial explosion (as discussed in section 2.1), their use is usually limited to agents facing only very simple tasks.

Our solution is based on an idea used in the Action Selection community (see [12] for example): a complex task can often be solved by a combination of simpler motivations. Figure 1 illustrates such a situation on the tile world problem: an agent has here to manage three simple behaviors (the first one intends to avoid a hole, and the two others to push a tile in the hole).

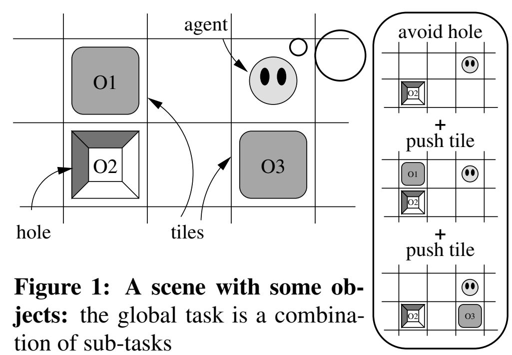

Figure 1: A scene with some objects: the global task is a combination of sub-tasks

As suggested by [6] we tried to solve the action-selection problem using Reinforcement Learning (RL) methods. In previous experiments (detailed in [2]), we provided our agent with basic behaviors, which were then weighted and combined into the agent's global behavior. The main point was that the weights leading to the complex behavior are *learned*. As such, we derived an *adaptive* (through learning relations between behaviors) and *scalable* agent (working with various world sizes).

Although this previous approach gave encouraging results, it also stressed out the importance of using the right basic behaviors. The aim of this present paper is thus to propose a method for automatically generating the right set of basic behaviors to use. The only requirements for the agent is to be able to perceive objects and distinguish between specific reward signals. A notable point is that we use the former combination of behaviors to help learning new basic ones, which can then be re-used as new basic behaviors. This can be seen as a first step towards incremental meta-learning.

The following section develops in more details the context of our work and presents previous works which inspired us. Then, in Section 3, we present the behaviors' combination introduced in [2] along with the main results obtained. This will lead to the incremental learning proposed in Section 4 to improve our method. Section 4.3 is devoted to an experimental validation of our work on the classical tile-world problem. A discussion and a conclusion end this paper.

AAMAS'03, July 14–18, 2003, Melbourne, Australia.

2. FRAMEWORK

This section introduces the use of Reinforcement Learning in our context. Then an analysis of previous works shows their limitations and help us outline the solution we propose.

2.1 Reinforcement Learning and Limitations

Reinforcement Learning (RL) methods are very appealing ways to have agents learn optimal reactive behaviors in uncertain worlds, as only a scalar feedback from the system to the agents is required. But the convergence of RL algorithms (like *Q-Learning* or *TD(λ)*) has only been proven for Markov Decision Processes (MDP).

Definition 1. A MDP is defined as a $\langle \mathcal{S}, \mathcal{A}, T, r \rangle$ tuple where:

- $\mathcal{S}$ is a finite set of states.
- $\mathcal{A}$ is a finite set of actions.
- $T(s, a, s')$ is the transition function and gives the probability of ending in state s' when action a is chosen with the system in state s.
- $r(s, a)$ is then the reward generated by the environment after each transition.

The problem is then to find the optimal mapping $\pi(s, a)$ between states and actions so as to maximize the reward received over time, usually expressed as a utility function $Q(s, a) = \sum_{t=0}^{\infty} \gamma^t E(r_t | s_0 = s, a_0 = a)$. Such a mapping is called a *policy* and, for a MDP, it is well known that an optimal *deterministic* policy exists [11].

As our agent only has a partial view of its environment, the learning task we are confronted with belongs to the more general class of Partially Observed Markov Decision Processes (POMDP), where the agent has only access to an observation of the current state. Nevertheless, the assumption that the agent faces a Markovian problem when only considering the observations is often made. This is truly a weak approximation and the policies learned this way are clearly sub-optimal as explained by [10]. In fact, it is better in that case to look for *stochastic* policies, using gradient descent algorithms for example (like in [1] or [8]).

Even under the Markovian approximation, the problem of combinatorial explosion remains. The number of an agent's possible observations can still be huge, even though the locality of its perceptions helps reducing it. Our combination algorithm takes advantage of the possibility to decompose the task in subtasks to address this specific problem.

2.2 Previous and Similar Works

A common idea to overcome the curse of dimensionality is to decompose the Markov Decision Process in some way. A taxonomy of MDP decompositions is proposed for example by Wang and Mahadevan in [13], in which our approach stands in the *action decomposition* class. But to our knowledge, there is up to now no satisfying solution to this class of problem in the learning framework.

2.2.1 Why We Need A New Algorithm

If not only considering reinforcement learning, the subject of Action-Selection –defined by [6] as "the problem of choice between conflicting and heterogeneous goals"– fits more appropriately to our approach, since it intends to consider some complex problems as a compromise between simpler ones (eventually at the expense of optimality properties). The agent being given stochastic policies for each of its basic behaviors, the hard point is then to combine these basic policies and derive a global policy. At this point, two major directions can be taken.

Winner-Take-All

A first option, as in the work of Humphrys [6], is to select one particular behavior as the one to be privileged for the immediate perception of the agent. To do this, the actions desired by each behavior and information about the utility of these actions are fed into a controller which determines the "leader" behavior. Many strategies for the controller have been studied by Humphrys, all based on his *W-Learning* framework. It is grounded on behaviors (agents) competing to take the decision, each agent learning for each state the value of being the winner. Even if a goal of this work was to have agents get rid of any global reward function, it requires an adaptation of the local reward functions used (through a Genetic Algorithm in which the global payoff reappears). As such, this adaptation is very specific to the global task.

Free-Flow

Another option is to combine the basic policies into a new policy. This option is more appealing than the preceding one for several reasons. In some cases, as shown on figure 2, the best action to take is not given by any sub-policy but could emerge from a combined policy, especially if the sub-policies are deterministic. Besides, this option is less task-specific in the sense that we can re-use the basic behaviors for different complex tasks as we only need to adjust the weights of the behaviors and not the behaviors themselves.

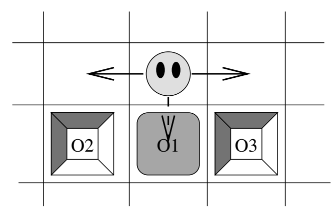

Figure 2: No leading behavior. The action desired by the basic `push` behaviors is either to go west $\{hole\ O_3, tile\ O_1\}$ or east $\{hole\ O_2,\ tile\ O_1\}$, whereas the optimal global action is to go south.

Calculating a new probability distribution over actions this way can be seen as a *free-flow hierarchy* as defined in [12]. One argument for prefering such a free-flow hierarchy to a hierarchical decision structure is that the later winner-take-all algorithm will be subject to its deterministic aspect, not being able to make compromises.

To cite an example of such an approach, the work from Dixon et al [3] briefly describes a method for combining sub-policies where the probability distribution on the actions for each sub-task are linearly combined. A simple weight is associated to each sub-policy, and the problem is then to choose the right weights. As this method only intended to control their exploration, no particular attention was given on how to choose these weights. This is exactly the problem we will address in our algorithm by *learning these weights*, as explained in Section 3.4.3.

2.2.2 Scalability is Desired

We lastly want to point out that existing methods do not offer scalability. When a basic behavior can be applied more than once in a given situation (for example, two holes are to be avoided), this must be specified *a priori* in the algorithms cited before. This specific point is also dealt with in our algorithm. To that end, we learn one weight for each "generic" basic behavior (such as *avoid hole*). Thus, even if many instances of the same generic behavior can be applied in a given situation (several holes to avoid), they will be combined using only one *common* weight. More details are given in Sections 3.2 and 3.4.3.

3. PROPOSITION

A key idea of our approach is that a complex behavior is obtained as an answer to many basic motivations. Moreover, simple basic behaviors can be easily associated to each of these motivations. Given these two points, we make the hypothesis that, in many cases, a good solution of the complete problem can be reconstructed using the basic behaviors. In this section, we will describe how a scene can be analysed through a decomposition in basic behaviors, give more details on these basic behaviors, and finally propose a method to automatically build a complex behavior by learning[1].

3.1 Some Notations

An agent perceives a scene –the accessible part of its environment– as an **observation**, which is composed of a set of percepts. Such an observation can be broken into **configurations** (i.e: subsets of the observation), and a percept can belong to different configurations. As each percept is characterized by a type (for example a hole, a tile or a door in our case), we also define the very important notion of a **type of configuration** which is described by a set of types of percepts (i.e. $\{hole, tile\}$). With these notations, the next section shows how to analyse a scene.

3.2 Scene Decomposition by the Agent

The starting point of the behaviors' combination is that an agent confronted to a complex situation must decompose it into simpler known configurations in a *scalable* way. To this end, the first step is to look for "familiar" and "useful" configurations in the perceived situation. As any subset of percepts of the agent's current observation can be a configuration associated to a behavior, an agent will only consider configurations which belong to at least one type of configuration associated to one existing basic behavior.

For an observation o, let us call $\mathcal{U}(o)$ the set of these "useful" configurations. Then, for each useful configuration c of $\mathcal{U}(o)$ we will note $\mathcal{B}(c)$ the set of basic behaviors associated to this configuration. Similarly, $\mathcal{C}(b, o)$ will be the set of useful configurations in observation o associated to a given basic behavior b. Scalability derives mainly from the fact that one basic behavior (resp. configuration) can be associated to more than one configuration (resp. basic behavior). This is all the more interesting that, due to the locality of perceptions, the number of useful configurations changes.

To illustrate this on the tile-world scene presented on figure 1, the agent's perceptions concern here objects O_1, O_2 and O_3. With two possible behaviors: $a$voiding the holes (b_a) associated to the type of configuration $\{hole\}$ and $p$ushing blocs in those holes (b_p) associated to $\{hole,\ tile\}$, the agent has to take into account the three following ($behavior$, $configuration$) pairs: $(b_a, \{O_2\})$, $(b_p, \{O_2, O_1\})$ and $(b_p, \{O_2, O_3\})$.

Note that, among the hard points of the method we propose, a combinatorial explosion can be feared as far as the search for useful configurations is concerned. In practice, the number of objects seen remains usually small, as only local perception should be used.

We will now see what knowledge of the basic behaviors is required so that they may be recombined efficiently.

3.3 Basic Behaviors

Definition 2. A **basic behavior** is defined by:

1. a type of configuration,
2. a stochastic decision policy learned by reinforcement and
3. the utility of this policy.

For a basic behavior b, we will note $\mathcal{C}^T(b)$ its type of configuration. This notion of type of configuration is essential for the scalability of our approach, since it allows generic basic behaviors to be instantiated several times in the same observation. Then, the policy P_b of the behavior is a mapping from configurations c of $\mathcal{C}^T(b)$ to probability distributions over actions: $P_b : \mathcal{C}^T(b) \times \mathcal{A} \longrightarrow [0, 1]$. Note that two different configurations can belong to the same type of configuration, so the agent can deal with them by using the same behavior (i.e. the agent can try to avoid two holes at the same time).

The knowledge of this policy for each basic behavior is not sufficient to take efficient decisions when the agent has to deal with concurrent motivations. To weight them in some way -giving a higher priority to danger avoidance or to important reward in sight- we suggest to evaluate a situation by using Q-values, as they will give us the expectation of discounted reward (i.e. the utility) of each configuration-action pair. This Q-values can be learned while also learning the policy of a basic behavior[2].

To sum it up, for each behavior b are calculated two tables –both defined on the same set $\mathcal{C}^T(b) \times \mathcal{A}$– for an upcoming use:

- $P_b(c, a)$: the probability to choose action a while seeing configuration c, and
- $Q_b(c, a)$: the expected discounted reward when choosing action a for the configuration c.

Before considering the way we tried to combine these basic behaviors in a complex one, please note that we have not yet discussed the way to find the right basic behaviors to use.

3.4 Basic Behaviors Combination

3.4.1 General Formula

The next step is to use these useful configurations to choose an action. There are several ways to make this choice (voting, biding, random choice) but we decided to compute a policy $\mathcal{P}(o, a)$ giving a probability distribution over actions a for each possible observation o. As written previously, we chose to define this policy as a recombination of basic behaviors using their P- and Q-tables. To be more precise, we try to define a linear combination of the P policies. The general formula is thus:

$$\mathcal{P}(o, a) = \frac{1}{K} \sum_{c \in \mathcal{U}(o)} \sum_{b \in \mathcal{B}(c)} w(b, c, a).P_b(c, a)$$

where $w(b, c, a)$ are some positive functions of the Q-values called *weights* and K is a normalizing factor ($\sum_a \mathcal{P}(o, a) = 1$). The action is then chosen according to the $\mathcal{P}$ distribution.

How to calculate $\mathcal{P}$ optimally using the Q- and P-tables is subject to discussion. Only the formula that gave us most satisfaction is presented in this paper.

3.4.2 Chosen formula

We chose w to depend on the Q-values. A first remark is that the absolute value of Q-values will be used in a way to give the same importance to future earnings ($Q > 0$) and immediate danger ($Q < 0$). If these Q-values seem to give good comparisons between state-action pairs of a single behavior, the relative importance of different

[1]These basic behaviors could be usefully collected into a library which would be reusable in other situations.

[2]This subject is discussed further in section 4.2.2

Q-tables can be efficiently corrected by learning a parameter θ_b for each behavior (appearing as a factor e^{θ_b}).

For a given observation-action pair (o, a), the idea is to consider that each behavior tells that its probability $P_b(c, a)$ is the right one with a force of conviction $|e^{\theta_b} * Q_b(c, a)|$ [3]. This leads to compute the mean of the $P_b(c, a)$ probabilities weighted by $|e^{\theta_b} * Q_b(c, a)|$ (for each current behavior):

$$\mathcal{R}(o, a) = \frac{1}{k_{(o,a)}} \sum_{c \in \mathcal{U}(o)} \sum_{b \in \mathcal{B}(c)} e^{\theta_b}.|Q_b(c, a)|.P_b(c, a)$$

$$(k_{(o,a)} = \sum_{c \in \mathcal{U}(o)} \sum_{b \in \mathcal{B}(c)} e^{\theta_b}.|Q_b(c, a)|)$$

After normalizing and putting in common e^{θ_b} for all instances of a type of behavior, the final version is rewritten as:

$$\mathcal{P}(o, a) = \frac{1}{K}.\frac{1}{k_{(o,a)}} \sum_{b \in \mathcal{B}} \underbrace{e^{\theta_b}}_{to\ learn} \left[\underbrace{\sum_{c \in \mathcal{C}(b,o)} |Q_b(c, a)|.P_b(c, a)}_{already\ known} \right]$$

3.4.3 Learning and Scalability

Each set of θ parameters defines a global complex policy for the agent. Tuning the weights of the formula is like learning an optimal parameterized policy in the framework of reinforcement learning. To that end, we have simply used a straightforward simulated annealing algorithm with a geometric decrease of the temperature (only a few parameters have to be learned: one for each behavior).

The scalability of the process derives also from the fact that only one θ_b coefficient needs to be defined for each behavior. Even when many configurations are associated to one behavior, the complex policy can be computed without further learning or refining of the parameters.

In the next section, we present an example showing an application of our methodology on the tile-world problem.

3.5 Experimental results

Here will not be presented completely detailed experiments of the methodology just described. Such information may be found by interested readers in [2]. We will rather insist on some remarkable results that conducted further developments.

3.5.1 The Tile-World

Problem

The tile-world is a grid domain in which a cell may contain a hole, a tile or an agent. In the complete problem, the agent (we consider only one agent) has to push tiles in holes as often as possible, while avoiding to go itself in one of those holes.

To give some details about the simulation, the agent can go freely in a hole (and also go out), but will get a negative reward doing so. Moreover, when a tile is pushed in a hole, both the tile and the hole disappear and reappear anywhere on the grid. Finally, to avoid some blocking situations, holes and tiles cannot be on cells of the grid's border.

In this complete complex problem, many tiles and holes must be handled. As it appears in previous examples, a simple decomposition of the problem in basic behaviors can be made intuitively: `[avoid hole]` $(b_a, \{hole\})$ and `[push tile in hole]` $(b_p, \{hole,\ tile\})$, as shown on figure 1.

[3] With a single non-zero reward r, the Q-table is proportional to r.

Agent's Skills

In these experiments, the agent has always all other objects of the environment in sight. The principle of locality is nevertheless present in the fact that perceptions are unprecise. For any object O in the scene, the agent's perception of O gives:

- `near(`O`)`: tells if object O is in the 9-cells square centered on the agent `(true|false)`,
- direction(O): gives the object's direction `(N-NE-E-SE-S-SW-W-NW)`.

The only actions available for an agent are to move one cell North, South, East or West (it cannot ask to stay on a cell). And to conclude, the reward given is $+1$ when a tile falls in a hole, -3 when the agent goes in a hole, and 0 otherwise.

3.5.2 Performance Evaluation

The best way to appreciate our approach is to compare the average reward obtained in various observation-spaces with the results of specifically learned optimal policies (tabula rasa). In our first attempts, the direct stochastic policy search employed (OLPOMDP described in [1]) stayed most of the time in local optima, only learning how to avoid holes whereas the agent is expected to also push tiles in the holes.

Even if the best accessible references are policies obtained through an adapted Bolzmann Q-learning, experimental results (first four lines of table 1) demonstrate that the basic behaviors' combination can be efficient and that, moreover, the weights learned in simple situations (with two tiles and two holes for example) may be reused in more complex cases without learning them further (last three lines).

Table 1: Comparative table between policies obtained tabula rasa and by recombination

objects #tiles+holes	reward (for 10.000 steps) tabula rasa	**c(1)**	**c(2)**
1 + 1	**1380**	1017	842
2 + 1	300	302	**493**
1 + 2	~ 0	**405**	259
2 + 2	200	283	**411**
3 + 2	-	166	134
2 + 3	-	262	247
3 + 3	-	179	129

- c(1): our combination of policies
- c(2): the same combination with noise added in the decisions of the basic behaviors

Note: the difference of efficiency between combination c(1) and results in table 2 comes from basic behaviors whose learning has been stopped early in this first experiments (and is thus less accurate).

3.5.3 A Particular Problem

To go further in analysing these experiments, the complex behaviors obtained have simply been observed while running.

As expected, the agent appears to take pretty good decisions most of the time, and predictably hesitates when having a symetric choice between two tiles to push in a hole. So, if we stopped at that point, the agent should have near-optimal performances, which is not the case.

In fact, some particular situations emphasize a critical weakness of our approach. Figures 2 and 3 illustrate this point. In the later figure, none of three current $(behavior, configuration)$ pairs suggest the right action to choose: both "push" behaviors are for going west and the "avoid hole" one has no influence, whereas a move to the north would be appropriate. This blocking is only resolved thanks to the stochastic aspect of the policies, which sometimes brings by chance to choose unfavoured actions.

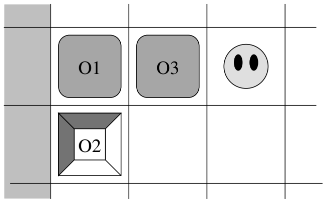

Figure 3: Two behaviors making a wrong choice. Both basic `push` behaviors lead the combination method to go west, whereas the optimal global action is to go north (considering that there is a wall on the west side).

These particular problems –where a linear combination of policies is far from being the solution– are responsible for the low agent's efficiency, since most of the time is spent facing such blockings, whereas they only represent a tiny part of the possible observations. Next section presents a possible solution to overcome this difficulty and to potentially have a much more automatic agent's designing system.

4. INCREMENTAL APPROACH

The study of the behaviors' combination which has just been presented confirmed that the resulting policy was close to a good policy, even if suffering from various drawbacks. This induced us in the idea that such a policy would be a good initialization for a complete direct policy search, and consequently for computing a new behavior. This section will point out in more depth the reasons of this new developement, show the principle it is based on, and finally comment its practical efficiency on the tile-world testbed.

4.1 Motivation

The main reason motivating a new algorithm is to overcome "non-linear" situations. As explained in section 3.5.3 –and as illustrated on figure 3– some configuration cannot be solved by a linear combination of basic behavior. These configurations require taking into account more element of the perception to be handled. Two solutions were studied:

- **Learning specific rules to correct just the critic situations** as in [5]. But in fact, for one situation, many rules dealing with probable preceding situations must be learned to avoid oscillations. And it is not clear how rules could be combined with other behaviors. So we opted for a second solution:

- **Learning new complete basic behaviors with more complexe types of configuration** (involving two tiles and one hole in the case of figure 3). Unfortunately, the experiments quickly described in section 3.5.2 showed that learning a complete behavior from scratch becomes hard even with an apparently reasonable number of observations. This would precisely happen if the agent tried to directly learn a "push" behavior with two tiles and one hole as required for figure 3.

Our idea is then to ease the learning of such new behaviors by starting from the combination policy learned and increasing it through gradient learning. This is a kind of bootstrapping process.

4.2 Description

We present here the general scheme of our incremental approach that aims at designing new basic behaviors.

4.2.1 Principle

To define the new behavior that is to be learned, we must explicit its type of configuration $\mathcal{C}^T$ and a global reward $\mathcal{R}$ acting as a motivation. Then the following three consecutive phases compose the algorithm, as also illustrated on figure 4 (the second phase does not appear explicitly):

1. A set of basic behaviors $BB = \{bb_1, bb_2 \dots bb_n\}$ being chosen and a new "goal" being considered (given by $\mathcal{R}$), the algorithm presented in section 3 is executed to adapt a combination behavior by learning optimal weight parameters.

2. The combined behavior obtained by the optimal combination is converted into a starting behavior $\langle \mathcal{C}^T, \pi, Q \rangle$. π is described as a parametrized stochastic policy suitable for gradient learning and Q is initially set to 0.

3. π serves as a starting policy to learn a complete behavior b' (with an optimized policy π') by a gradient descent (here, the OLPOMDP algorithm from [1]), and simultaneously estimating the related Q-values on the observation-action space.

$$
\begin{array}{c}
bb_1 \quad bb_2 \quad \dots \quad bb_n \\
\text{(combination)} \downarrow \\
\pi \\
\text{(RL)} \downarrow \\
b' \ (\pi')
\end{array}
$$

Figure 4: Principle of the incremental method. The proposed algorithm combines basic behaviors $bb_1,\ bb_2 \dots bb_n$ (learns the weighting parameters), obtains a policy π, and uses it as a root of a new learning phase bringing to a new behavior b' (with improved policy π').

To complete this general description of the algorithm, we develop some aspects below.

4.2.2 Complements

Q-table Estimation

The objective also consists in learning a Q-table estimation. With the choice of the classical discounted reward (among various possible definitions of the Q-values), and taking into account the non-markovianness of our framework, the updating formula used simply becomes:

$$Q(o,a) \leftarrow (1-\alpha) * Q(o,a) + \alpha * \left(r + \gamma \sum_{b \in \mathcal{A}} [\pi(b) * Q(o',b)] \right)$$

This subject does not require here more details. But readers more largely interested in "learning without state-estimation in POMDPs" are encouraged to refer to [10].

What About Rewards ?

Another aspect to discuss is how to manage rewards for new behaviors. The tile-world problem considered in our experiments does not satisfy the hypothesis made in section 4.2.1 of a single accessible payoff. The two basic behaviors given as examples correspond to two different causes of rewards: a negative one when falling in a hole, and a positive one if a tile is pushed in a hole.

When designing an agent, being able to distinguish sources of reinforcement seems a reasonable assumption. It of course depends on the point of view adopted in its conception. But in suitable cases, we suggest to introduce them on purpose as *types of reward* (noted $\mathcal{R}^T$). Technically, it allows the selection of the agent's goals when learning a new behavior.

This ability to separate basic behaviors according to types of reward has a twofold interest:

- On the one hand it permits to sharpen the “classification” of these basic behaviors, eventually using them in various ways whether they act as inhibiting or incitating motivations (for example).
- On the other hand, reinforcement learning algorithms are prone to fall in local optima. This could sometimes be avoided by only considering one goal out of many.

The former argument will not be applied in this paper, as it mainly concerns perspectives on better combinations. On the contrary, the later argument will bring direct improvements to the methodology, as shown by experiments on the tile-world.

As written above, this idea may not fit to all situations, but should be taken into account if possible. Let us finally mention that dealing with type of reward was of not useful when only combining basic behaviors , since policies and Q-tables sufficed. The question arose when dealing with the initialization of new basic behaviors.

4.3 Conducted Experiments

To assess the method described in this paper, we conducted experiments on the tile-world problem. After precising the methodology applied, we exhibit and comment the results of our simulations.

4.3.1 Methodology

In this paper, three ways of obtaining policies dedicated to a given problem have been tested:

1. *Tabula rasa*: learns this policy from scratch through a classical direct policy search.
2. *By combination*: learns the parameters of a basic behaviors' combination as described in section 3.
3. *Incrementally*: uses the incremental approach.

The first manner is the reference, even if it may not be successful (because of difficulties to converge, often due to local optima). The two others relate to the two main steps of our algorithm, and are both of interest in our framework. On the table 2 that lists the results of the experiments, the last three columns are dedicated to these three different methods.

Table 2 presents the simulation conditions we used in its first two columns. In one column appears the number of tiles and holes of the simulations performed, and in the other the type of reward $\mathcal{R}^T$ involved. A positive (respectively negative) reward –when a tile is pushed in a box (resp. when going in a hole)– is symbolized by a '+' (resp. '-'). When both rewards are combined, the '+-' symbol is employed.

Lastly, when trying to obtain a new policy through a combination, the set of basic behaviors to use has to be defined. Here, the tests have been conducted in the order given by the table and, begin with an empty set of BBs, new behaviors considered as basic (in bold font in the table) have been progressively added.

4.3.2 Results and Analysis

Two types of information can be found in the table 2 that sums up our tests:

- the average reward for 10.000 simulation steps, and
- the approximate number of series of 10.000 simulation steps before reaching the best policy (in parenthesis when available).

But before considering the table, do not forget that the first zeros concern an agent that only expect to avoid holes, and thus does not get any positive payoff when pushing a tile in a hole.

Table 2: Comparative table between policies obtained tabula rasa, by recombination and through the incremental approach

situation		reward (for 10.000 steps)				
#tiles+holes	$\mathcal{R}^T$	tabula rasa		comb.	inc.	
0 + 1	-	**0**		-	-	
0 + 2	-	0		0	0	
1 + 1	+	1380	(20)	70	**1380**	(20)
1 + 1	+-	150	(300)	1250	1380	(20)
1 + 2	+	1650		950	1660	(20)
1 + 2	+-	0		380	0	
2 + 1	+	1230	(250)	30	1280	(40)
2 + 1	+-	0		30	1250	(40)
2 + 2	+	1250	(200)	250	1260	(80)
2 + 2	+-	0		150	0	

- comb.: combination of policies
- inc.: incremental approach

Falling in local optima

If only considering the data of the “tabula rasa” column, the phenomenon of classical learning falling in local optima when both rewards are in competition ('+-') distinctly appears. On any of these '+-' lines, the behaviors' combination always gives better results and allows pushing a tile in a hole more or less often. This returns no impressive average rewards, but produces as expected a good starting policy for our incremental learning.

Only the situation considering two holes and one tile leads to a failure, probably because of a learning parameter that does not give enough space to exploration. The fact that the agent goes generally less than ten times in a hole when just learning to avoid it also suggests this hypothesis, which would require an in depth analysis.

Computation time

One annoying point about the behaviors' combination is that its optimization –here through simulated annealing– requires making good estimations of the average reward gained for a given set of parameters. Nevertheless, in our situation, the cost of this optimization phase was negligible compared to a complete policy learning by a gradient descent. This is all the more important that the observation space in this later algorithm is here approximately multiplied by 16 for each new object considered (we in fact reduce this size thanks to symetries of the problem).

In all experiments, the adapting combination phase always lasted $400 * 10.000$ simulation steps (even if the optimum was reached early), i.e. less than a minute on our computer. The number of simulation steps to learn complete policies –either tabula rasa or incrementally– was also fixed, but to $300 * 10.000$ only (these two duration have not been tuned), such a phase took less than a minute for 1 object, about 5 minutes for 2 objects and about 1 hour for 3.

Considering that optimizing a combination is far from being time-expansive, it is of real importance to notice that learning incrementally is much faster (in term of simulation steps, and thus in computation time) than from scratch.

Random behavior

As explained in 3.5.3, the so called "blocking situations" induce most of the loss in efficiency in the basic behaviors' combination. Nevertheless, it is well known that adding some noise in the decision mechanism –a random action being picked from time to time– is a good way to get rid of these blockings. It brought the idea to add some chance in our algorithm.

This has been done through an artificial *random* basic behavior, which has 1- no observation, 2- a uniform probability distribution over actions and 3- the same positive Q-value for all actions (a null value would make the behavior useless). With this, and as the complementary noise only works up to a point, the weighted combination modulates this noise by itself.

With the exception of the new random basic behavior, other experiments have been conducted in the same conditions as in table 2. As expected, they confirmed an increase in efficiency when using two tiles and one hole (and at least the positive reward). This is in aggreement with the former experiments in [2] on the combination, where the used basic behaviors were a little less deterministic.

To conclude

To only speak about the incremental learning, these experiments succesfully demonstrated the benefits of this approach. The main controversial point to retain is that the "low computation time" argument holds only if the simulation of the environment itself is not a time-consuming algorithm (if considering a simulation as in our case).

5. DISCUSSION

We propose to first discourse about the combination before insisting on specific aspects and then on possibilities of the incremental learning.

5.1 Combination of Basic Behaviors

Optimality ?

The combination method is clearly not a sure way of reaching optimality. One of the reasons is that only a subclass of the possible policies can be explored. Furthermore, the simulated annealing which learns the weights of the global behavior converges only to *local* optima. The lack of optimality is the price to pay for a tractable algorithm.

Improved Combination

The really reduced number of parameters to learn may suggest the idea of refining the level of combination. This could be achieved thanks to weights depending not only on the behaviors, but also on the actions, observations... The drawbacks of this approach lies in a bigger number of parameters to learn. It could even bring toward learning a completely new policy –as done in the incremental step– whereas scalability and genericity issues are our main concern in this part of the work.

Another possibility for a finer control of the global behavior would be to make use of the sign of the utility of a basic policy. This way, a clear distinction could be made between basic "negative" behaviors (forbidden actions) and "positive" behaviors that could just promote actions. This would require altering the policy combination function but without bringing too much task-specific knowledge into the process.

For a Greater Scalability

Considering the sign of a policy's utility is a first step toward taking into account the specificity of the basic behaviors, and eventually relationships between them. The "additive" nature of our present combination of behaviors is for example well adapted to additive environments (where the global utility is the sum of sub-tasks utilities). Nevertheless, when it is not the case, too many instances of the same basic behavior (many tiles) could abusively overweight other behaviors (avoid hole).

Other experiments that we conducted encourage not considering only additive computations as the linear combination presented in the current paper (a simple multiplicative approach gave in fact a little better results on the tile-world). One advanced solution could be the use of a "critic" module trying to predict the global utility so as to adapt the way the behaviors are handled (additive, concurrent, preemptive...). This perspective is yet another argument in favor of further work on behaviors' combination, even though other ideas have been explored first.

5.2 Incremental Learning

We know that, because of its limited perception, an agent is confronted to a non-Markovian task. Consequently, the Markovian approximation we use cannot give an optimal result. At any rate, optimality does not seem a reasonable objective as growing observation-spaces are faced.

Looking for improving methods that mimic a progressive development of the agent's behavior appears to be more satisfying in that it gives the agent the ability to be adaptive –and scalable in the present paper–. Nonetheless, such approaches always require an exploration phase and, thus, the definition of an arbitrary stopping criterion. It is not a critical aspect, since most optimisation –and therefore learning– algorithms needs some parameters to be set.

5.3 Tree-Growth ?

The present work would be complete with an automated choice of basic behaviors, in the sense that an agent would really require a minimal human intervention. A first problem would be to decide whether a new behavior has to go in the basic behaviors' set (the "base") or not. But the discussion about a good criterion remains open.

Nevertheless, knowing how to explore behaviors to augment the base stays unanswered. The incremental algorithm suggests to begin with simple behaviors taking only few objects into account, and then use the basic ones to develop others. This description naturally induces the growing of a "tree" as exemplified on figure 5.

This principle can be compared to the exhaustive observables of [4] or the U-trees of [7]. And, in a similar way, difficulties should be met to stop exploring new leaves at a given depth.

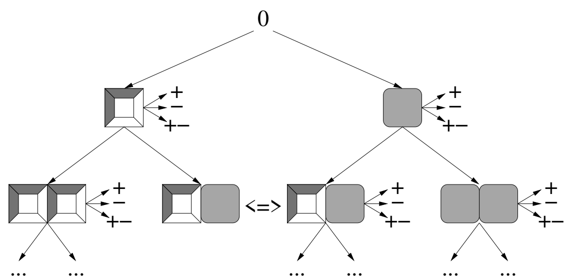

Figure 5: Behaviors' exploration tree. There is no precision here on which behaviors are basic or not. But notice that various combinations of types of reward must be tested.

6. CONCLUSION

The general framework of this paper is the automated design of agents facing a complex task through Reinforcement Learning. Our work is motivated by the fact that complex tasks are often the composition of simpler tasks. In this context, we presented a two step process that, starting from a set of generic low level behaviors and a given task, learned to define a complex behavior adapted to the task. Scalability and genericity are two important aspects of our method.

The first step of our method builds a complex behavior by learning how to combine basic behavior. This complex behavior is a linear combination of the basic behavior and the weights are learned by simulated annealing. Scalability derives from the fact that several instances of the same basic behavior can be combined "on line" without learning new weights, thanks to what we called "types of configurations". Furthermore, as the basic behaviors are not modified in the process, our method allows for genericity and reusability. The second step of our method uses a gradient learning algorithm to refine the policies obtained through combined behaviors. It is possible to build new behaviors that are more efficient. As shown on examples, such efficiency could not be obtained while learning from scratch.

Several experimentations have been made to validate and justify our approach. The next step, which indeed guided our work, is to derive an incremental algorithm that, starting from a basic set of behavior, iteratively builds new behaviors and adds them to its set of basic behaviors to use. This would be a first step toward a kind of meta-learning, which would serve both complex task solving and knowledge transfer. But some hard points, like the detection of useful "types of configuration" and motivations are still to be more thoroughly studied.

7. REFERENCES

[1] J. Baxter and P. Bartlett. Infinite-horizon policy-gradient estimation. *Journal of Artificial Intelligence Research*, 15:319–350, 2001.

[2] O. Buffet, A. Dutech, and F. Charpillet. Adaptive combination of behaviors in an agent. In *Proceedings of the 15th European Conference on Artificial Intelligence (ECAI'02)*, 2002.

[3] K. Dixon, R. Malak, and P. Khosla. Incorporating prior knowledge and previously learned information into reinforcement learning agents. Technical report, Carnegie Mellon University, Institute for Complex Engineered Systems, 2000.

[4] A. Dutech. Solving pomdp using selected past-events. In *Proceedings of the 14th European Conference on Artificial Intelligence (ECAI'00)*, 2000.

[5] S. Gadanho and L. Custodio. Asynchronous learning by emotions and cognition. In *Proceedings of the Seventh International Conference on the Simulation of Adaptive Behavior (SAB2002)*, 2002.

[6] M. Humphrys. Action selection methods using reinforcement learning. In *From Animals to Animats 4: 4th International Conference on Simulation of Adaptive Behavior (SAB-96)*, September 1996.

[7] R. A. McCallum. Instance-based utile distinctions for reinforcement learning with hidden state. In *Proceedings of the 12fth International Machine Learning Conference (ML'95)*, 1995.

[8] L. Peshkin, K. Kim, N. Meuleau, and L. Kaelbling. Learning to cooperate via policy search. In *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence (UAI'00)*, 2000.

[9] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Englewood Cliffs, NJ: prentice Hall, 1995.

[10] S. Singh, T. Jaakkola, and M. Jordan. Learning without state estimation in partially observable markovian decision processes. In *Proceedings of the 11th International Conference on Machine Learning (ICML'94)*, 1994.

[11] R. Sutton, D. Precup, and S. Singh. Between MDPs and Semi-MDPs: Learning, planning, and representing knowledge at multiple temporal scales. Technical report, University of Massachusetts, Department of Computer and Information Sciences, Amherst, MA, 1998.

[12] T. Tyrrell. *Computational Mechanisms for Action Selection*. PhD thesis, University of Edinburgh, 1993.

[13] G. Wang and S. Mahadevan. Hierarchical optimization of policy-coupled semi-markov decision processes. In *Proceedings of the 16th International Conference on Machine Learning (ICML'99), Seattle*. Morgan Kaufmann, 1999.