Self-Growth of Basic Behaviors in an Action Selection Based Agent

Olivier Buffet, Alain Dutech, François Charpillet

To cite this version:

Olivier Buffet, Alain Dutech, François Charpillet. Self-Growth of Basic Behaviors in an Action Selection Based Agent. Eighth International Conference on Simulation of Adaptive Behavior (SAB'04), Jul 2004, Los Angeles, CA, USA, pp.223-232. inria-00000573

HAL Id: inria-00000573

https://hal.inria.fr/inria-00000573

Submitted on 3 Nov 2005

Self-Growth of Basic Behaviors in an Action Selection Based Agent.

Olivier Buffet, Alain Dutech and François Charpillet
Loria - INRIA-Lorraine
Campus Scientifique - BP 239
54506 Vandœuvre-lès-Nancy CEDEX
France
dutech@loria.fr

Abstract

We investigate on designing agents facing multiple objectives simultaneously, that creates difficult situations, even if each objective is of low complexity. The present paper builds on an existing action selection process based on basic behaviors (resulting in a modular architecture) and proposes an algorithm for automatically selecting and learning the required basic behaviors through an incremental Reinforcement Learning approach. This leads to a very autonomous architecture, as the hand-coding is here reduced to its minimum.

1. Introduction

Action selection is the general high-level problem of choosing which action to perform when facing different parallel goals that can be heterogeneous and even conflicting. Although ethologists have studied this problem in depth, their models are not very easy to implement, as argued by (Tyrrell, 1993). A large number of implemented models rely on evaluating the quality of an action thanks to a kind of weighted sum of some quantities in order to evaluate the utility of that action related to the various high-level goals. To prevent the hard and tedious task of hand-coding and hand-tuning huge set of parameters associated with this action weighting and selection process, various *reinforcement learning* schemes have been studied (see (Lin., 1992), (Humphrys, 1996) for example).

Following (Humphrys, 1996) principles, we will use an action selection architecture where an agent knows different low-level behaviors (that we call *basic behaviors*) and must somehow choose an action based on the action's utility given by these basic behaviors. Previous works on this domain have focused on how to select this action, i.e. how to combine various quantities to select an action from a set of actions favored by the related basic behaviors or to suggest a new decision.

While in these works the basic behaviors are defined by hand and given to the agent, our aim is for the agent to define *by itself* the basic behaviors it will need in its action selection process so as to reach a complex satisfactory behavior. To do so, our suggestion is to give the agent the means to learn, from scratch, the basic behaviors it needs. This means, not only learning to achieve a given task, but also to *learn which tasks to learn.*

This kind of learning is made possible in the general framework of reinforcement learning thanks to a new mechanism of action selection (described more thoroughly in (Buffet et al., 2003)) that allows an agent to adaptively combine basic behaviors into a more efficient complex behavior. We show that the same mechanism can be used by the agent to detect if the new, slightly more complex, behavior learned is worth using as a *new basic behavior in future action selection processes.* Thus, starting from scratch, our agent incrementally grows its own *set* of basic behaviors from which, using action selection mechanisms, it can produce a complex behavior answering its needs and goals.

1.1 Motives

The motivations for an agent to learn by itself the basic behaviors needed to solve a complex task by action selection are many, some being highlighted below:

- It can be useful to remind that action selection should allow an agent to cope with complex tasks involving parallel, heterogeneous and sometimes concurrent goals.
- Learning new basic behaviors should bring better results than using intuitively hand-designed basic behaviors.
- Learning makes the design of an agent easier (no more long hand-tuning of parameters).
- Above all, the autonomy of an agent that can learn its own basic behaviors is greatly increased.

1.2 *Our action selection framework*

The basics of our action selection architecture must be clear:

- The agent does not know the model of the dynamics of its environment and can only partially perceive it. The goals of the agents are represented by a scalar reward signal that depends on the action and state of the agent.
- The agent has a set of basic behaviors. Each basic behavior should answer a simple goal/motivation.
- In a given situation (perception + internal drive), the agent uses information (mostly action utility) from its various basic behaviors to decide its global action.
- The action selection process is a "free-flow" as explained by (Tyrrell, 1993): the global action can be different from every action recommended by the basic behaviors. The agent is able to combine behaviors into a *new* action.
- The whole process gives the agent a *global complex behavior* that can cope with parallel, heterogeneous, conflicting needs and goals.

As said before, whereas in most previous works the set of basic behaviors is defined by the human designer[1], we aim for the agent to define and learn this set *by itself*, using an algorithm based on reinforcement learning.

1.3 *Example*

To illustrate our approach and to test our algorithms, we will use the well-known tile-world problem (see (Joslin et al., 1993), (Wooldridge et al., 1995)).

In the tile-world environment, as depicted on Fig. 1, the agent must push tiles into holes while avoiding to fall itself into holes. Intuitively this complex goal can be reached with at least two basic behaviors: pushing tiles in holes and avoiding holes.

Using our learning algorithm, our agent should learn by itself which basic behaviors are really useful for solving the complex task.

1.4 *Articulation of the paper*

To present our algorithm for automatically defining the basic behavior needed by an agent using our action selection architecture, we will first present our conception of basic behaviors (Section 2.). Then Section 3. explains how we combine basic behaviors in new behaviors, these new behaviors being potential new basic behaviors. The actual process of learning a set of basic behaviors is presented in Section 4., experimented and analysed in Section 5.. A discussion about our algorithm and some future works ends this paper.

[1]when speaking of artificial agents

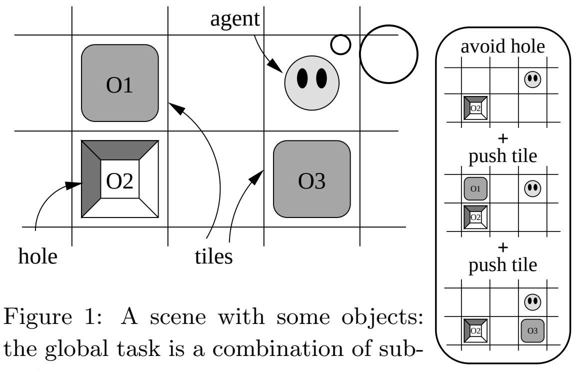


Figure 1: A scene with some objects: the global task is a combination of subtasks

2. Preliminaries

To go further in the presentation of our approach, we first need a precise formulation of what is called here a *basic behavior* (*bb*). As any behavior, it is guided by a *motivation*. This section thus describes what a motivation is, how to learn a behavior, what basic behaviors are, and finally how to use them.

2.1 *Motivation Linked to a Behavior*

A first aspect of a motivation is to know what are the good and bad results that the agent should try to reach or avoid. Having several possible elementary reward signals (two in the case of the tile-world: one for pushing tiles in holes and one for avoiding holes), a subset of these signals should be given.

Yet, the notion of motivation cannot be reduced to such a payoff. We need to also detail the situation through the types of objects taken into account (these types being instantiated once in a given world). If the agent only aims at pushing tiles in holes, it may be of interest for it to consider one hole and two tiles simultaneously, as one of the tiles could otherwise be an unseen obstacle.

In the present paper, a motivation is therefore described by 1- a subset of elementary reward functions and 2- a tuple of objects' types that are taken into account (what will be called a "type of configuration"). This will be represented as shown on Fig. 2.

Such a motivation is just a "reason" for acquiring an adapted behavior. Based on this, we now present a classical approach for learning this behavior.

2.2 *Our Reinforcement Learning framework*

The general Reinforcement Learning (RL) framework is the following (see (Sutton et al., 1998)). An agent per-

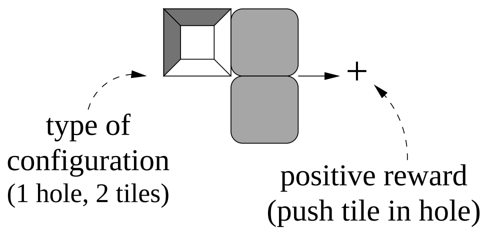


Figure 2: A schematic representation of a motivation for 1 hole, 2 tiles and a positive (+) reward when pushing a tile in a hole.

ceives its environment, decides of an action to alter this environment, and then receives a reinforcement signal. Using only this reinforcement signal (a scalar reward) and no information on the dynamics of the environment, various algorithms (like Q-Learning (Watkins, 1989), TD(λ) (Tesauro, 1992)) allow the agent to automatically learn a behavior (called a policy) to optimize the reward over time.

More formally, the RL framework can be expressed as Markov Decision Processes (MDP) (Puterman, 1994). Optimal solutions are deterministic policies π. Formally, a policy is a function that, for each *state* s of the system (agent + environment) gives an optimal action a. A utility function evaluating the "quality" of an action is often associated with a policy, and denoted $Q(s, a)$.

In our framework, we must face a crucial limitation to the ideal case of MDPs. As our agent has only a partial perception of the environment, it cannot know the full state of the system. The agent must then solve a non-Markovian process. The solution we have adopted is to use stochastic policies (as opposed to deterministic) that give a probability of choosing actions in a given state (see (Singh et al., 1994)). For a given and chosen basic behavior, such a policy can be learned using for example some gradient reinforcement algorithm (Baxter and Bartlett, 2001, Baxter et al., 2001).

2.3 Basic Behaviors

For a given motivation, any RL algorithm suited to partial observations can be used. It only requires: 1- to define the "guiding" reward function as the sum of selected elementary reward functions, and 2- to build the perception based on the objects' observations (the objects corresponding to the type of configuration). The first result that is obtained is a stochastic policy. Nevertheless, other information will need to be stored to make use of this "behavior": the type of configuration will serve to match the appropriate objects in the environment to the behavior, and the utility function Q will help weighting the importance of the behavior.

A **behavior** b is thus defined by a tuple $\langle \mathcal{C}_b^T, P_b, Q_b \rangle$, where:

1. $\mathcal{C}_b^T$ is a type of configuration, that is, a tuple of types of objects involved in the behavior.
2. $P_b(a|o(c))$ is a stochastic decision policy. Given an appropriate configuration, it maps its observations to probability distributions over actions.
3. $Q_b(o(c), a)$ is the Q-table of this policy, giving the expected discounted reward of an observation.

Such a behavior answers to one given motivation. But our objective is to find a set of behaviors which can be combined together to obtain an efficient decision-making system. These selected behaviors are named **basic behaviors** (bb) and constitute a set noted $\mathcal{B}$. Whereas the first step should be the choice of this set, next section first presents the principle of our action selection process, which is based on the combination of basic behaviors.

2.4 Combining Basic Behaviors

For an agent confronted to a new observation, making a decision while following several basic behaviors is a two-step process. The agent first has to identify which sets of objects trigger some basic behaviors. This leads to several potential stochastic decisions to apply, which must be combined in a single one through a compromise.

We illustrate this process on Figure 1 with only two intuitive bbs: 1- b_p which considers one tile and one hole, and the reward for pushing a tile in a hole, and 2- b_a which considers one hole, and the reward for falling in a hole.

The scene analysis leads here to a decomposition in two reasons for using b_p: the "configurations" (tuples of objects) $cfg_1 = \langle O_1, O_2 \rangle$ and $cfg_2 = \langle O_3, O_2 \rangle$, and one reason for using b_a: the configuration $cfg_3 = \langle O_2 \rangle$. To each pair (bb, cfg) is associated a probability distribution over action and the related Q-values, all this being used to compute a final probability distribution to apply, as shown on Figure 3.

$$\begin{tikzcd}
(b_p, cfg_1) \arrow[r] & P_{b_p}(\cdot|o(cfg_1)), Q_{b_p}\cdots \arrow[r] & \\
(b_p, cfg_2) \arrow[r] & P_{b_p}(\cdot|o(cfg_2)), Q_{b_p}\cdots \arrow[r] & \text{Combination} \arrow[dl] \\
(b_a, cfg_k) \arrow[r] & P_{b_a}(\cdot|o(cfg_k)), Q_{b_a}\cdots \arrow[ur] & \\
 & \text{Immediate policy} &
\end{tikzcd}$$

Figure 3: How to obtain a probability distribution over actions with the three identified (bb, cfg) pairs.

Note: Interested readers will find more details on the combination of basic behaviors in the appendix, Sec. A1.

3. How to detect a new *basic behavior*

In a few words, the principle of building a new basic behavior from an existing set of basic behaviors rely on three steps: learning the best combined behavior for a given motivation, trying to improve this behavior and deciding if it will then be used as a new basic behavior.

3.1 Learn a combination of basic behavior

As described in (Buffet et al., 2003), there is a learning algorithm that, given a set of basic behaviors $\mathcal{B} = \{bb_1, \cdots, bb_n\}$, can produce an efficient combination function of these behaviors. This combination is based on a parameterized function of the policies where the parameters are learned using a simulated annealing algorithm. Let's call the behavior resulting from this learning process a *combined behavior*: $bc = \mathcal{F}(\mathcal{B}, \Theta)$ where $\Theta = (\theta_1, \cdots, \theta_n)$ is a learned vector of parameter of the size of $\mathcal{B}$. Sec. A1 in the appendix gives more details on the combination algorithm.

3.2 Improvement

To improve the combined behavior thus obtained, our idea is to use an online gradient descent reinforcement learning (Baxter et al., 2001). The search done with such an optimization algorithm can help improving a policy, even in a non-Markovian reinforcement learning framework as ours. This search is initiated by using the combined policy of the combined behavior. With the new policy thus obtained, we can derive a new behavior (denoted *nb*) with its associated motivation and quality.

3.3 New basic behavior

The question now is to decide if this new behavior would improve the global behavior of the agent if used as a basic behavior in later action selection mechanism. This decision is based on a heuristic comparison between the quality of the combined behavior and of the new behavior. Our hypothesis is that a newly learned -and significantly better- behavior is likely to bring useful information to other behaviors that will be built on it.

If we call (V_{cb}) the quality of the combined behavior and (V_{nb}) the quality of the new behavior, the new behavior is chosen as a new basic behavior (and added to the set $\mathcal{B}$) if:

- $|V_{cb} - V_{nb}| > \sigma_s$ (to avoid errors due to close estimates) and,
- either $((V_{cb} < 0)$ and $(V_{nb} < \sigma_- * V_{cb}))$
  (little improvement of a negative result)
- or $((V_{cb} \geq 0)$ and $(V_{nb} > \sigma_+ * V_{cb}))$
  (major improvement of a positive result).

This criterion depends on three parameters that have a big influence on the overall algorithm, as detailed in Sec. 4.. In our experiment they have been tuned by hand. One important work would be to test their validity in other tasks.

4. More detailed algorithm

The question raised here is how to automate the choice of basic behaviors for the combination. As explained in Sec. A, it is of major interest since the intuitive selection of "minimalist" basic behaviors is not sufficient and since, moreover, this step is required for the agent to be truly autonomous.

4.1 Principle

Our proposition is to incrementally build a set of basic behaviors. To this end, starting with an empty set of basic behaviors, we progressively add more complex behaviors to this set, using already-selected basic behaviors to design the new ones. This leads to building a set of basic behaviors of increasing complexity.

The definition of a behavior's motivation (see Sec. 2.1) implicitly says that the "complexity" of a behavior comes from two sources: the size of the type of configuration and the combination of elementary reward functions.

To sum up the principle followed, a new type of configuration being chosen, the behaviors corresponding to the possible rewards' combinations are learned and tested to know whether they are of interest or not. Figure 4 illustrates the progression according to types of configuration: behaviors get more complex toward the leaves of the tree.

Using the process described in Sec. 3.3 for selecting a new basic behavior, next section explains the tree growth algorithm.

4.2 Tree-Growth

The idea is to grow a set of basic behaviors. We build an *exploration* tree of behaviors of increasing complexity in the number of objects. Each node of the tree is linked to a type of configuration and is in fact a short sub-tree of the possible reward combinations. In turns, using a breadth-first traversal of the tree, we compute the best behavior in each case, building upon the combination of earlier *bb*s (as explained in Sec. 4.1). Each new behavior is compared to the corresponding combination (criterion presented in Sec. 3.3) to determine if it must be added to the set of basic behaviors (the tree being then expanded at this node with new behaviors to explore).

Coming back to the example of Fig. 4, this tree's development follows more and more complex types of configurations and, for each of them, there is a short sub-tree of the possible reward combinations ("−" (respectively

"+") stands for the avoid-hole negative reward (resp. the push-tile-in-hole positive one), and "+−" combines both).

4.3 Algorithm

Before testing it, we present here a formal definition of our method in Algorithm 1. Among the data required by the algorithm are:

- d_{max}: As an ever-increasing tree could be possible, this parameter limits the maximum depth of the tree.
- d: Not only the immediate sons of a new basic behavior may be of interest for further exploration. Thus, this second parameter defines the depth of developed nodes. It can be compared to the *fringe depth* used in the U-tree algorithm presented in (McCallum, 1995).
- $Criterion(\cdot,\cdot)$: Already presented in Sec. 3.3, this is the criterion used to evaluate new behaviors.

Algorithm 1 A Growing Tree of Basic Behaviors

```
Require: $Criterion(\cdot,\cdot)$: to evaluate new behaviors.
    $d_{max}$: maximum depth of the tree, and $d$: depth of developed sub-trees.
 1: $T$ empty tree. $\mathcal{B}$ empty set.
 2: Add to $T$ all behaviors with 0 to $d$ types of object.
 3: for all Behavior $b$ still to be visited (up to depth $d_{\max}$) do
 4:   Adapt the combination of current $bb$s from $\mathcal{B}$:
      $V_{cb} \leftarrow$ efficiency of this combined behavior.
 5:   Learn $b$ by a direct policy search (initialized by the combination):
      $V_{nb} \leftarrow$ efficiency of this new behavior.
 6:   if $Criterion(V_{cb}, V_{nb}) = true$ then
 7:     Add to $T$ the sons of $b$ with up to $d$ more types of objects.
 8:     Mark $b$ as [basic]. Add it to $\mathcal{B}$.
 9:   end if
10:   Mark $b$ as [visited].
11: end for
Ensure: A set $\mathcal{B}$ of kept basic behaviors.
```

5. Experiments

5.1 Application to the Tile-World

Description of the 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 (only one is considered) has to push tiles in holes as often as possible, and avoids to go itself in these holes.

To detail 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 (Fig. 1), a simple decomposition of the problem in basic behaviors can be made intuitively: 1- `[avoid hole]` ($b_a, \langle hole\rangle$) and 2- `[push tile in hole]` ($b_p, \langle tile,\ hole\rangle$). They will be used as a reference for our *bb*-tree generation (and noted as the set $\mathcal{B}_{ref}$).

Agent's Perceptions, Actions and Rewards

In these experiments, the agent has always all other objects of the environment in sight. The principle of locality is nevertheless respected through the imprecise perceptions. 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`)) and **direction**(O) (gives the object's direction (`N-NE-E-SE-S-SW-W-NW`)[2]).

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.

Going back to Fig. 1, the agent's perceptions of the 3 objects are here:

O_1:	$near(O_1) = false$,	$direction(O_1) = W$
O_2:	$near(O_2) = false$,	$direction(O_2) = W$
O_3:	$near(O_3) = true$,	$direction(O_3) = S$

Parameters

The parameters appearing in the description of the tree-growth Algorithm 1 (including the criterion from Sec. 3.3) have been tuned by hand, and set as follows:

$$\sigma_s = 500 \quad \sigma_- = 0.9 \quad \sigma_+ = 2 \qquad d_{max} = 4 \quad d = 1$$

Methodology

The experimentations carried out were twofold: 1- applying the tree-growth algorithm to produce a set of basic behaviors $\mathcal{B}_{tree}$ (in a 6×6 grid) and 2- testing its efficiency in various more complex situations (in an 8×8 grid).

[2]The direction is known even for far objects, but brings imprecise information.

The tests conducted in this second phase consisted in using the combination of resulting basic behaviors with various numbers of tiles and holes in the environment (up to five of each). It is compared to the two "intuitive" basic behaviors that serve as reference (as written in Sec.7 5.1). A combination's efficiency is measured through the total payoff received during $100,000$ simulation steps.

Note: due to a lack of space, only the major results are presented below. Nevertheless the discussion enters upon several interesting remarks on the complete experimentations.

5.2 Analysis

Tree-Growth

To first consider the immediate result of the tree-growth algorithm, Fig. 4 shows the developed tree of evaluated behaviors. The signs noted in parenthesis indicate that the behavior linked to this type of reward and the related type of configuration has been retained as a basic behavior.

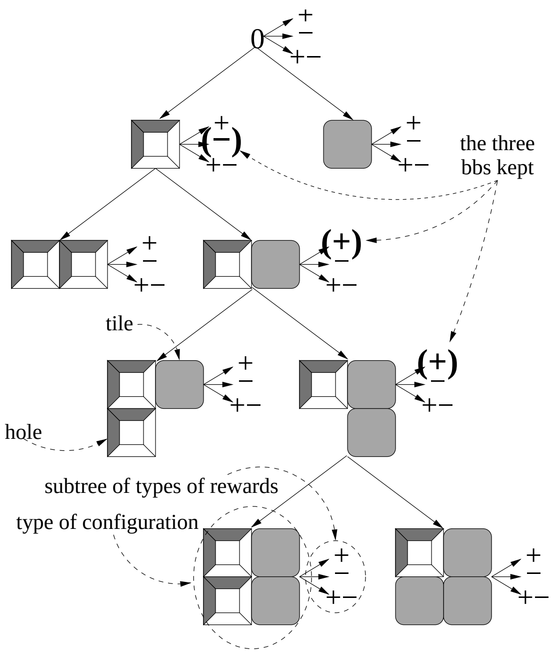
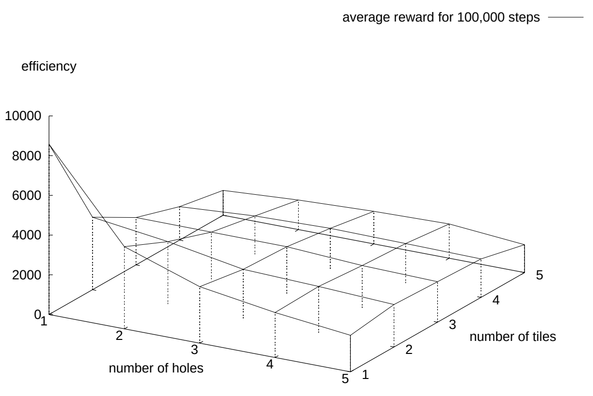


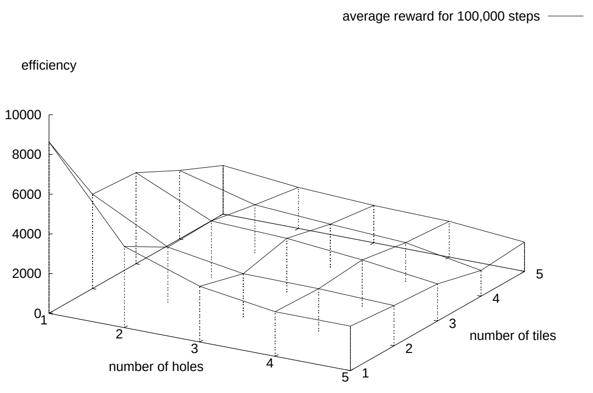
Figure 4: The tree of tested behaviors and (in parenthesis) the ones kept.

The three basic behaviors obtained (set $\mathcal{B}_{tree}$) correspond to the two intuitive ones (from $\mathcal{B}_{ref}$) along with a behavior (1-hole/2-tiles with the "+" reward) "specialized" for solving a blocking[3] (shown on Fig. 8 b)). The generated tree seems pretty satisfying as adding the "specialized" behavior answers a typical blocking situation.

[3]Not considering all objects regularly leads to such blockings.



a) basic behaviors from $\mathcal{B}_{ref}$



b) basic behaviors from $\mathcal{B}_{tree}$

Figure 5: Efficiency of the combination in more complex situations with different sets of basic behaviors.

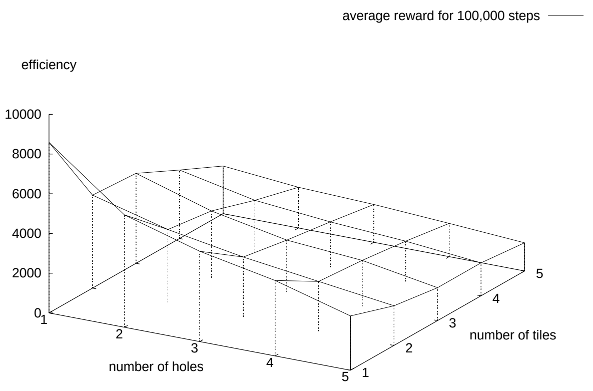


Figure 6: Identical tests as in Fig. 5 with *bb*s from $\mathcal{B}'_{tree}$: $\mathcal{B}_{tree}$ and the basic behavior considering 2 holes and 1 tile with only the "+" reward.

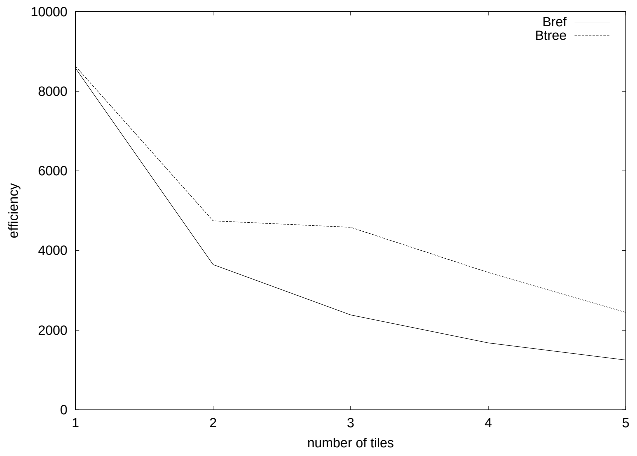


Figure 7: Comparison of $\mathcal{B}_{ref}$ and $\mathcal{B}_{tree}$'s efficiencies with 1 hole and several tiles.

However, the [2-holes/1-tile, "+" reward] behavior that solves another blocking (Fig. 8 a) in the appendix) is not added to the basic behaviors set. In fact, the increase in performance it brought does not fit the criterion ($V_{nb} \simeq 1.5 * V_{cb} < 2 * V_{cb}$) and, as a consequence, is left over. When we use a "weaker" criterion ($\sigma_+ = \mathbf{1.3}$), both specialized *bb*s are included in $\mathcal{B}$, along with 4 more complex ones. This brings difficulties, as the combination learning process is made more complex and more prone to local optima, even though individually, each of these behaviors answers an interesting specific situation. This problem comes rather from the combination algorithm itself than from the *bb* learning algorithm.

Tree's Efficiency

Figures 5 a) and b) present the efficiency of both sets of basic behaviors: $\mathcal{B}_{ref}$ and $\mathcal{B}_{tree}$, so as to compare the later (automatically generated through our algorithm) with the former (designed by hand). The x and y axes of the horizontal plane indicate the number of holes and tiles in the evaluated situations (the numerous objects to handle obliged to extend our grid to a size of 8×8). On the z axis is measured the efficiency in each situation (note: Table 1 also presents numerical values for some of these points).

At first glance, both surfaces are quite similar, with a common peak in the simple 1-tile/1-hole situation (to which a basic "push" behavior is dedicated). An important increase ($*1.3$) can be observed in the 2-tiles/1-hole case. It was in fact expected, as $\mathcal{B}_{ref}$ and $\mathcal{B}_{tree}$ only differ in the behavior managing the corresponding "push" motivation. Nevertheless, this increase appears to be persistent with a growing number of tiles to push (results are multiplied by 2) along the tiles-axis, as also shown on Fig. 7.

Considering a growing number of holes, the results from Fig. 5 b) are even closer to the reference surface. The remaining variations are partly due to the estimate of the efficiency, and we saw that a problem linked to the criterion prevented improvements in these cases with two holes and more.

Observation of a Weakest Criterion

Figure 6 shows the results obtained with the "weakest" criterion we already mentioned ($V_{nb} > \mathbf{1.3} * V_{cb}$) and a maximum tree-depth $d_{max} = 3$. Here, the behavior of the 2-holes/1-tile node (with the "+" reward) is now retained in set $\mathcal{B}$. It produces on the holes-axis a similar increase as the one observed previously on the tiles-axis. With this last set $\mathcal{B}'_{tree}$, the agent may even benefit from both improvements simultaneously, as shown on Table 1. The same table illustrates the fact that having useless *bb*s leads to a slight decrease in efficiency (see $\mathcal{B}_{tree}$ in cases 2/1 or 2/2, or $\mathcal{B}'_{tree}$ in case 1/2).

$\#_{holes}/\#_{tiles}$	$\mathcal{B}_{ref}$	$\mathcal{B}_{tree}$	$\mathcal{B}'_{tree}$
1 / 2	3648.3	**4747.5**	4680.5
2 / 1	4137.8	4089.6	**5660.8**
2 / 2	3142.3	2807.5	**3673.4**
2 / 3	2369.6	2873.9	**3356.4**
3 / 2	2468.3	2190.3	**3007.7**

Table 1: Comparison of the three sets' efficiencies. With more than five objects, the improvement gets harder to appreciate.

6. Discussion, similar work

If we come back to the motivations of our work, the main goal of action selection (i.e. being able to cope with multi-tasks problem) has been reached. Let us recall that classical Reinforcement Learning tools would not be able to manage more than one tile and one hole simultaneously (experiments are shown in (Buffet et al., 2002)). An agent using classical RL only finds out how to avoid falling in holes. Measuring their efficiency as done in Fig. 5 would produce a similar peak for one tile and one hole, the remaining surface lying with a null payoff. With our method, the agent is still able to behave correctly even with 5 holes and 5 tiles. Of course, there is a compromise to make between a reduced number of basic behaviors (fastest learning) and highest performances (in term of quality).

The outcome of our algorithm, and thus the global behavior of the agent, still depends on parameters and, more importantly, on a heuristic criterion. The parameters are linked to the exploration and growth of the tree of behaviors. As presented in Sec. 5., the influence of the criterion can be critical. If we keep with the same type of criterion, a more intelligent process would be to adapt the σ parameters to have a loose criterion at the start of the process and a more constraining criterion as the number of basic behaviors increases. That way, further behaviors would be added if they really bring something in term of performance. Of course, a more "intelligent" criterion and a way to automatically tune the parameters obviously need to be looked for.

It would be interesting to test and compare our work more thoroughly with similar projects. Testing are currently done on the "house environment" designed and used in (Humphrys, 1996), which is both easier and more difficult. It is easier as there is no need for several instances of the same basic behavior, but it is more difficult because the branching factor (due to many types of rewards and objects) in the exploration tree is high (8 for objects, 14 for rewards). Primary results show that the first basic behaviors learned are different from the basic behaviors hand-coded by Humphrys, but we have not the complete results to compare the global performances of the two approaches.

Closer to our work, (Digney, 1998) proposes a fully automated hierarchical Q-Learning algorithm that learns to create and structure "sub-modules" to solve a multi-task problem. One major difference with our work is that his agent has a perfect perception of its environment. The Q-modules learned using Digney's algorithm seem more strongly linked to the type of task at hand, and it is not clear if they could be re-used in very different environments. Furthermore, his approach to action selection is not scalable in the sense used in this paper: if many sub-tasks are similar, Digney must learn as many Q-modules whereas our algorithm would only need one basic behavior. On the other hand, Digney's work seems more automated (less dependent to parameters tuning) but still restricted to simpler tasks.

In fact, in our work, learning both basic behaviors and how to combine basic behaviors, did indeed increase the autonomous aspect of the agent. Of course, human interference is still needed, especially to tell about different kinds of objects and rewards in the environment. This is also the case in (Digney, 1998), and might seem inevitable from (Urzelai et al., 1998) point of view. In their work, (Urzelai et al., 1998) specifically aim at finding the right balance between human design and machine learning in efficiently designing autonomous agents; and they argue that human is more efficient for designing basic behaviors, even if they are crude and refined later on by the agent. It seems that our work, along with (Digney, 1998), and to some extends (Nehmzow et al., 1993), gives even less work to the human designer: he only has to provide rewards and perceptions types. Such approaches relate to the domain of shaping (Randløv and Alstrøm, 1998) and could be worth a comparison to cognitive development (Piaget, 1967, Weng, 2002). And we argue that it would be still interesting to go even further, but that is still one of the greatest challenge of AI: agents that auto-organize and represent their environment by themselves.

To keep on considering practical aspects, our approach, as presented here, requires working in a simulated environment, so as to be able to conduct controled experiments. In an application on a real-world problem, this would imply designing such a simulation, for example by learning a model of the environment. This leads to another important machine learning problem, but may appear more realistic than a crude trial-and-error approach in a real world.

7. Conclusion

7.1 Our Contribution

We have presented our work on designing complex behaviors for autonomous agents. This work builds on an algorithm set in the Action Selection framework where a complex behavior is a combination of basic ones. Our main contribution deals with automatically designing the set of basic behaviors that will be used by the combination algorithm. This task is crucial for building complex behaviors and is also a first step towards meta-learning: in some way, the agent is able to select the skills it needs.

Only reward signals and types of objects present in the environment are needed by the agent to incrementally define more and more complex behaviors. The added value of a new behavior is used to determine if it will be used to augment the set of basic behaviors of the agent, thus being used to further build more complex behaviors. This iterative process ends when no more complex behaviors can be added.

7.2 Validation

Our algorithm has been tested on the tile-world problem. It gave good results as it was able to propose a set of behaviors that is more complete than an intuitive one designed by hand. Furthermore, note that the complex problems it was confronted to could not be solved directly by classical RL.

7.3 Future Works

Still, several improvements and open problems are left. It is very important to work on a better criterion for selecting "pertinent" behaviors, as the current criterion is a crude heuristic. Testing our algorithm on other environments and problems is also necessary to assess more thoroughly its capacities and to better understand the influence of the few parameters of the algorithm that still need to be hand-tuned.

As the efficiency of the combination algorithm can be affected by the size of the basic behaviors' set, we think that the combination process could be studied and probably improved further.

Another complex domain we are interested in is the field of multi-agents systems. If we are confident in the capacity of our algorithm to deal with the added complexity in terms of number of objects, we need to investigate further on the influence of the local perception of the world and the dynamic induced by other agents.

A very interesting perspective would be for the agent to build itself *objects abstractions*. Currently, the types of objects present in the environment are given to the agent by a human designer, but we think that side effects of behavior selection could be used to automatically categorize objects in classes (like for example “obstacles”, “goal”, “to be moved”, etc). This would be another step towards true meta-learning and greater autonomy in agents.

References

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

Baxter, J., Bartlett, P., and Weaver, L. (2001). Experiments with infinite-horizon, policy-gradient estimation. *Journal of Artificial Intelligence Research*, 15:351–381.

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

Buffet, O., Dutech, A., and Charpillet, F. (2003). Automatic generation of an agent's basic behaviors. In *Proceedings of the 2nd International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS'03)*.

Digney, B. (1998). Learning hierarchical control structure for multiple tasks and changing environments. In *Proceedings of the Fifth Conference on the Simulation of Adaptive Behavior (SAB'98)*.

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

Joslin, D., Nunes, A., and Pollack, M. E. (1993). Tileworld users' manual. Technical Report TR 93-12.

Lin., L.-J. (1992). Self-improving reactive agent based on reinforcement learning, planing and teaching. *Machine Learning*, 8:293–321.

McCallum, R. A. (1995). *Reinforcement Learning with Selective Perception and Hidden State*. PhD thesis, University of Rochester.

Nehmzow, U., Smithers, T., and McGonigle, B. (1993). Increasing behavioural repertoire in a mobile robot. In *From Animals to Animats: Proceedings of the Second Conference on the Simulation of Adaptive Behavior (SAB'93)*.

Piaget, J. (1967). *La Psychologie de l'Intelligence*. Armand Colin.

Puterman, M. L. (1994). *Markov Decision Processes– Discrete Stochastic Dynamic Programming*. John Wiley and Sons, Inc., New York, USA.

Randløv, J. and Alstrøm, P. (1998). Learning to drive a bicycle using reinforcement learning and shaping. In *Proceedings of the 15th International Conference on Machine Learning, (ICML-98)*, pages 463–471.

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

Sutton, R., Precup, D., and Singh, S. (1998). 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.

Tesauro, G. (1992). Practical issues in temporal difference learning. *Machine Learning*, (8):257–277.

Tyrrell, T. (1993). *Computational Mechanisms for Action Selection*. PhD thesis, University of Edinburgh.

Urzelai, J., Floreano, D., Dorigo, M., and Colombetti, M. (1998). Incremental robot shaping. *Connection Science Journal*, 10.

Watkins, C. (1989). *Learning from delayed rewards*. PhD thesis, King's College of Cambridge, UK.

Weng, J. (2002). A theory of mentally developing robots. In *Proceedings of the 2nd International Conference on Development and Learning (ICDL'02), MIT, Cambridge, MA*.

Wooldridge, M., Müller, J.-P., and Tambe, M. (1995). Agent theories, architectures and languages: A bibliography. In *Intelligent Agents II, IJCAI'95 Workshop*, pages 408–431.

A Appendix

We briefly present the action selection mechanism (learning and behavior combination) used in our framework. A detailed presentation of this process can be found in (Buffet et al., 2002).

A1 Scene Decomposition

The goal of the decomposition is to find which relevant basic behaviors are applicable for a given situation.

The environment of the agent is made of typed objects (such as holes and tiles in the example of Fig. 1). The agent's observation is a set of perceived objects of the environment. Each basic behavior is only concerned with some types of objects. Thus, they are "instantiated" by associating them with subsets of observable objects with appropriate types (such a set is called a **configuration** (cfg)). This leads to a decomposition of the scene in a set $\mathcal{BC}(o)$ of (*basic behavior*, *configuration*) pairs. Coming back to Fig. 1, three such pairs may here be extracted: $(b_a, \langle O_2\rangle)$, $(b_p, \langle O_1, O_2\rangle)$ and $(b_p, \langle O_3, O_2\rangle)$.

We have seen how each scene is decomposed in (bb, cfg) pairs. There is a stochastic decision policy for each such pair ($P_{bb}()$). And, as each configuration is associated with an observation $o(cfg)$, each pair implies a given probability distribution over actions $P_{bb}(a|o(cfg))$.

A1.1 Weighted Combination of basic behaviors

Each basic behavior bb of $\mathcal{BC}(o)$ identified in the scene decomposition step is associated with a policy of action $P_{bb}()$ and a quality $Q_{bb}()$. Based on this, the task of the combination step is to compute a new distribution of probabilities over the actions, thus determining the agent's immediate and global policy.

The combination can take various forms (see (Buffet et al., 2002)). The one we have used depends on a set $\Theta = (\theta_1, \cdots, \theta_n)$ of parameters and is expressed by the following formula (where $\mathcal{C}(b,o)$ is the set of observable configurations linked to a behavior b):

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

$$(\text{ with } k_{(o,a)} = \sum_{(b,c)\in\mathcal{BC}(o)} w_b(o,c,a) = \sum_{(b,c)\in\mathcal{BC}(o)} e^{\theta_b}.|Q_b(o,c,a)|\ )$$

A1.2 Learning the Weights

The global combined policy depends only on the set Θ of parameters and there are few of them as there is only **one parameter for each basic behavior** (i.e. for each different type of configuration).

As tuning these weights is an optimization problem (maximizing the expected average reward), these parameters can be learned by reinforcement using simulated annealing. The algorithm simply has to make a good estimation of the expected average reward for each set of parameters.

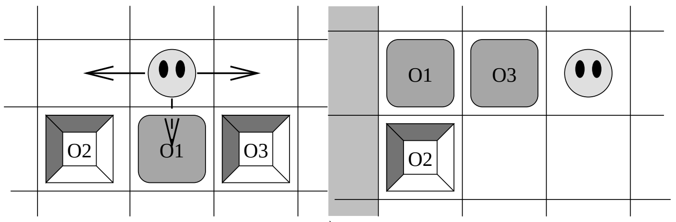
It is interesting to note that, to some extend, the whole action selection process is scalable as several instances of the same basic behavior can be used without having to learn new parameters.

A2 Open Problems

The algorithm that we briefly presented above was tested on the tile-world problem (described in Sec. 5.) by (Buffet et al., 2002) using only two "intuitive" hand-designed basic behaviors (`[push tile in hole]` and `[avoid hole]`). Despite "good results", optimality was not reached and several limitations were mentioned. Among them we want to mention the following ones:

- **Local optima**. Learning the parameters is difficult because of the many local optima simulated annealing can converge to. In fact, as the algorithm presented is an approximation scheme to solve a non-Markovian process, it is more than likely that no globally optimal solution exists. Nevertheless, a side effect of our method (see Sec. 4.1) could help finding an improved policy by refining the raw outcome of the behaviors' combination.

- **Choice of basic behaviors**. The set of available basic behaviors greatly influences the performances of the combination algorithm. Very general basic behaviors are required for learning and scalability, but for a particular task it would be great to adapt or propose new "ad hoc" basic behaviors more specialized. This would apply for example in the case of the two blocking situations on Fig. 8. This is exactly the purpose of the present article.



a) Blocking with 1 tile and 2 holes: None of the basic behaviors suggests to go south.

b) Blocking with 2 tiles and 1 hole (and an east wall): Both "push" basic behaviors suggest to go west, whereas it is not the right option. Going north would be better.

Figure 8: In both situations a) and b), none of the "intuitive" basic behaviors employed suggest an appropriate decision, whence an agent blocked for a long time.