Noname manuscript No.
(will be inserted by the editor)

A hierarchical representation of behaviour supporting open ended development and progressive learning for artificial agents

François Suro · Jacques Ferber · Tiberiu Stratulat

the date of receipt and acceptance should be inserted later

Keywords Developmental robotics · Learning agents · Artificial behavior · Modular architecture · Hierarchical architecture · Curriculum learning

Abstract In this paper, we present a supporting control architecture for an artificial agent designed learn behaviours from a curriculum established by a human instructor. A curriculum is a set of independent sub tasks of increasing complexity covering the different aspects of a desired behaviour.

In the optic of open ended or lifelong development, we place a crucial importance on the fact that the current desired final behaviour will be a future starting point for one, or even several, behaviours of greater complexity.

We propose the MIND (Modular Influence Network Design) architecture which encapsulates sub behaviours into modules and combines them into a hierarchy reflecting the modular and hierarchical nature of complex tasks, and allows for the preservation and re-usability of acquired skills.

1 Introduction

Jean Piaget's (Piaget, 1954; Piaget and Duckworth, 1970) theory of cognitive development in humans as a dynamical process of coordination schemes through multiple stages (from sensory-motor schemas to abstract level operations) is a starting point for every research on epigenetic robotics. His main idea is that learning is done progressively through interaction between the child and his environment, more complex tasks being learned on top of simpler tasks. This idea was assimilated in the developmental robotics field, where Oudeyer states that the complexity of knowledge and skills acquired by an artificial agent should increase progressively (Oudeyer, 2012).

According to Piaget, the complexity of learning can evolve along three axes:

1. Environment can progressively be more and more complex, from sand box to real world situations.
2. Motivations can be more and more complex, from grasping an object to trying to build a house.
3. Skills and their structuring into behaviour should develop to handle the increase in complexity of both environment and motivations.

From early Renaissance pioneers identifying areas of the human brain linked to specific functions to more recent works on coordination between these areas (Felleman and Van Essen, 1991), research shows that biological systems use a modular approach. Different areas of the brains are dedicated to specific processes, and organized in modules to accomplish tasks. Works on the primate visual cortex aimed at creating a bridge between biology and computer vision (Kruger et al., 2013) have pointed out the hierarchical structure of the brain and the need for hierarchical design in computer vision.

If computer vision systems can benefit from hierarchical structures to identify objects in a world that is "spatially laid out and structured hierarchically. (Objects can be naturally split into parts and subparts, complex features and simple features)" (Kruger et al., 2013), and since we are inclined to describe processes and tasks in hierarchies of procedures, whether we call it a recipe or an algorithm, couldn't a learning

LIRMM - Laboratoire d'Informatique, Robotique et Microélectronique de Montpellier - Université de Montpellier - CNRS
161 Rue Ada, 34090 Montpellier, France
E-mail: suro@lirmm.fr, ferber@lirmm.fr, stratulat@lirmm.fr

system benefit as well from having each of its sub-skills represented in its own module that coordinates with other modules hierarchically to accomplish complex tasks?

In this article we will build on modular principles to propose an architecture suited to progressive and curriculum learning. It reflects the modular properties of the curriculum into the knowledge representation by creating modules that encapsulate any form of learning structure (neural networks, function approximator, etc.) and provides a way for each module to coordinate with others to accomplish a complex task. This architecture will be able to accumulate new skills endlessly and make use of previous skill coordination to quickly master new tasks of increasing complexity.

We will prove the merit of the MIND architecture in a series of experiments on a simulated agent :

1. We first teach an artificial agent a complex behaviour from scratch, by building a hierarchy of skills of increasing complexity, from reaching a target and simple avoidance to collecting and bringing back objects in an environment with obstacles.
2. We then observe the effects of the curriculum and learning strategies, and the limitations of hand crafted reward functions. We also experiment with retraining and learning in broader context strategies for and established MIND hierarchy.
3. Finally we add new skills to an already trained MIND hierarchy to demonstrate its flexibility and show its merits as an architecture for open ended and cumulative learning.

We will conclude by discussing the possible limits of large hierarchies and show some preliminary work on a real world robot.

2 Related works

Curriculum learning (Bengio et al., 2009) is a progressive learning method investigated to this day in the field of machine learning as a way to accelerate learning and even solve learning problems that are otherwise impossible. Learning requires a feedback, either from the environment or a teaching entity, but when the learning task and environment are too complex, the feedback signal can end up being too complex to allow learning. For instance, consider the case of accumulating different reward sources for different actions, where it could not be possible to tell which action should be rewarded.

Curriculum learning subdivides a learning task into different but complementary sub-tasks (or *source tasks*) to be learned separately, and has been applied successfully to robotics and video games problems (Narvekar et al., 2016). Here the skill is memorised by a single function approximator through the transfer learning of the various source tasks. Curriculum learning has also been applied in a more abstract context to teach neural networks to approximate functions (Gülçehre et al., 2016). In this work the idea is to train the network to match a simplified version of the function and progressively change this target function to match the actual function we want to approximate. While this is an interesting progressive learning method, we could argue about the use of the term curriculum. This learning method just adds more complexity to the same learning task instead of being a collection of complementary learning tasks. Again, in this work the knowledge is represented as a single module, the neural network.

Although these methods allow learning by progressive means, they are limited in scope to the specific task at hand. They do not cover the lifelong, open ended, cumulative learning of a variety of tasks, a challenge at the heart of developmental robotics that is characterised by the acquisition of skills and knowledge progressively (Oudeyer, 2012).

Reflecting the curriculum into separate modules that encapsulate the resulting skill will require a way to coordinate their potentially contradictory outputs.

Previous works in the field of control systems for robots such as the Subsumption architecture (Brooks, 1986) or AuRA (Arkin and Balch, 1997) and Sat-Alt (Simonin and Ferber, 2000), both using a mechanism of vector composition, explore the possibility of having separate skills work in coordination (e.g. either by alternating their actions or by combining them), but on a small scale.

The work that inspires us the most, and that we hope to improve upon, is the Robot Shaping techniques developed by Dorigo and Colombetti(Dorigo and Colombetti, 1994) (Dorigo and Colombetti, 1998). Here behaviours are learned as a curriculum and represented as individual units, these skills are then combined to achieve higher level behaviours. Multiple architectures are discussed, from monolithic to multi-level hierarchies, learning methods and reward methods tailored towards artificial agents are proposed that are relevant to what we are trying to accomplish. After presenting the MIND architecture (3.3), we will compare it to Robot Shaping and explain how our approch differs.

3 Modular Influence Network Design

Our goal is to mirror the modularity of progressive learning in a modular architecture, but while other solutions focus on improving the learning process of a single task, we designed our architecture using a representation of knowledge that will also improve the learning process of future tasks. We propose MIND (Modular Influence Network Design) a hierarchy of modules able to coordinate separate behaviours, or skills, to accomplish a complex task. This modularity will grant us the following properties:

– Encapsulation: generic behaviours will be encapsulated and combined with others in a number of ways to achieve different goals, rather than specializing a global behaviour to a specific task and losing the ability to branch from the original generic behaviour.
– Identifiable behaviours: MIND will give the ability to identify and modify behaviour locally, working on a single aspect of the behaviour at the time.
– Unifying methods: MIND has few constraints on the decision method used by each module, e.g. neural networks and programming procedures can be used in the same network. MIND only provides a way for the modules to organize with each other in an influence driven network.
– Flexibility of mind: behaviour modules can be replaced, either by a new module or by a hierarchy of modules favouring constant evolution of the system.
– Flexibility of body: MIND gives a generic solution to the organization of sensors and actuators. The method used to coordinate related sensors and actuators into local groups is the same that is used to coordinate all the groups in the system together. This also means we can easily coordinate new sensors and actuators with an already existing system without losing previously acquired behaviours.

3.1 Base skill, complex skill, and influence

In the following we consider an agent whose sensory information and motor commands are represented as a vector of real numbers, normalized between 0 and 1. We can create a module that encapsulates a function $f(x)$ that reads the input vector $V_I = [I_1, I_2, ..., I_n]$ and outputs the vector $V_O = [O_1, O_2, ..., O_m]$. This function can be implemented as a programming procedure, or it can be a function approximator, a neural network, or any other kind of function that associates two vectors of real numbers. We will call such a module a *skill*, and the module whose output vector is used directly as motor commands a *base skill.*

$$V_O = f(V_I) \tag{1}$$

A single base skill that uses all the sensory inputs and all the motor outputs of the agent is sufficient to learn how to perform a complex task, each lesson of the curriculum being memorized in the same unique structure. The skill is therefore the sum of all the different experiences, with no way to differentiate what has been taught.

Instead of performing a complex task by a single skill, we will perform many sub-tasks, some even contradictory, with separate base skills. Each base skill will only associate the inputs and outputs necessary to perform its associated task. To perform the complex task, we will then create a *complex skill* which will coordinate several base skills, that we call its *sub-skills* (Figure 1).

A complex skill, as any skill, encapsulates a function that takes an input vector from the sensors V_I and outputs a vector of real numbers V_O, but the output is directed to its sub-skills. This output vector is called the influence vector $V_{Infl} = [Infl_1, Infl_2, ..., Infl_m]$, and its elements $Infl_x$ are called *influences.*

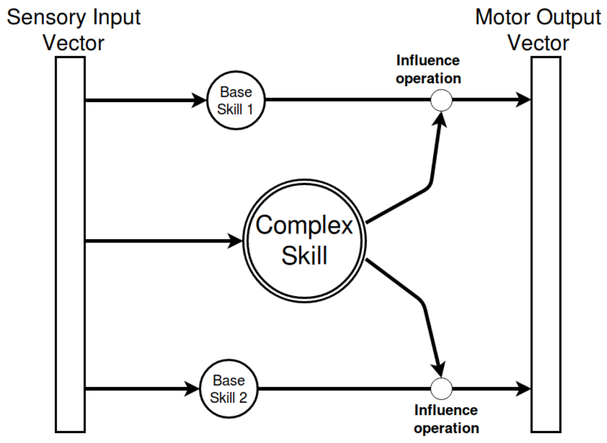

Fig. 1 A *complex skill* influencing two *base skills*

A complex skill can have other complex skills as its sub-skills, creating hierarchies of skills. At the top of the hierarchy is the *master skill* whose only particularity is to receive a constant influence of 1.0.

This hierarchy of skills forms a Directed Acyclic Graph (Figure 2). The influence flows along a vertical axis from the master skill down to the base skills and determines *who* is in charge of the resulting action. The information from the sensors reaches all the skills of the

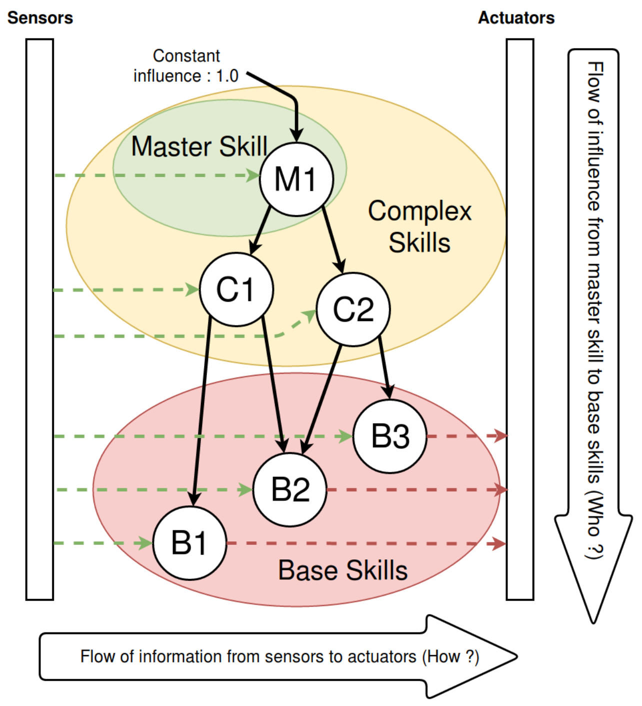
Sensors
Actuators
Constant influence : 1.0
Master Skill
M1
Complex Skills
C1
C2
B3
B2
B1
Base Skills
Flow of influence from master skill to base skills (Who ?)
Flow of information from sensors to actuators (How ?)

Fig. 2 A skill hierarchy, a *master skill* influences *complex skills* which in turn influence the *base skills*

hierarchy and the motor commands are output from the base skills to the actuators forming an horizontal information flow. Its purpose is to determine *how* the resulting action is going to be executed.

3.2 Using influence to determine motor commands

Starting from the master skill, each complex skill computes its output vector V_O and multiplies each element by the sum of the influences it received, forming the influence vector V_{Infl}. The skill then sends each element $Infl$ of the influence vector to the corresponding sub-skill (figure 3).

$$V_{Infl} = V_O * \sum_{x=1}^{n} Infl_x \tag{2}$$

With V_{Infl} the influence vector to the sub-skills, $V_O = f(V_I)$ the output vector of the internal function of the skill, $\sum_{x=1}^{n} Infl_x$ the sum of all influences the skill received (also noted $\Sigma Infl$).

The base skill, like a complex skill, computes its output vector and multiplies each element by the sum of the influences it received, similarly to equation (2), forming the motor command vector $V_{Com} = [Com_1, Com_2, ..., Com_m]$. The base skill then sends each element Com_x of the motor command vector to

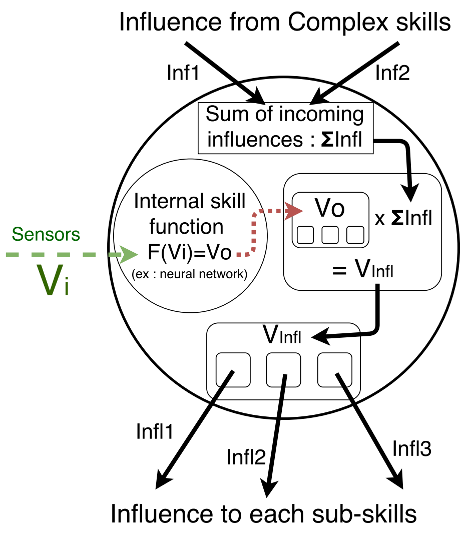
Influence from Complex skills
Inf1
Inf2
Sum of incoming influences : ΣInfl
Internal skill function F(Vi)=Vo
(ex : neural network)
Sensors
Vi
Vo
x ΣInfl
= VInfl
VInfl
Infl1
Infl2
Infl3
Influence to each sub-skills

Fig. 3 Internal architecture of a skill

the corresponding motor module along with the sum of the influences the base skill received $\Sigma Infl$.

Each motor module then computes the corresponding motor command for its actuator as a normalised weighted sum:

$$M = \frac{\sum_{x=1}^{n} Com_x}{\sum_{x=1}^{n} \Sigma Infl_x} \tag{3}$$

with M the resulting final motor command, x the index of the base skill that is sending a motor command, Com_x the weighted motor command for this motor module from the base skill x, $\Sigma Infl_x$ the sum of influences from the base skill x.

3.3 A brief comparison with Robot Shaping

Robot Shaping (Dorigo and Colombetti, 1994) (Dorigo and Colombetti, 1998) Hierarchies (RSH) being the closest related architecture to MIND, we offer here a few points on where they differ and, we hope, MIND improves.

All skills have access to sensor data. In RSH the lower level skills send request to the higher level skills (coordinators). Based solely on these requests the higher level skill chooses which and how sub-skills should be coordinated. In MIND, the higher level skills base his coordination decision on the environment and context by directly accessing sensor data, including data from sensors not involved with the sub-skills. We feel this is an important concept because in dealing with how to solve a problem, information as to why, when and if is often discarded, leading to the same response when subtle details in the context would call for an entirely different approach.

The output of all skills in MIND are a vector of real numbers between 0 and 1 instead of RSH's binary strings. While it has a higher resource cost, it allows some finesse in the output commands and, with the influence method, also function as the coordinator's combination operators, eliminating the need to specify these operators.

This use of a single type of output for all commands (influence, sensor information or motor commands) allows any combination between the elements of the architecture. RSH makes a clear distinction between the base skills and the coordinators. In MIND, we also differentiate between master skill, complex skill and base skill, but this distinction is only conceptual. There is no strict limitation that would prevent a complex skill to influence several base skills and also send motor commands to several actuators.

The use of this standard communication method between all modules opens a number of possibilities, such as using the previous state of an actuator the same way that we use sensor data, or creating a number of memory modules that store skill output in the same manner an actuator receives a motor command, and can provide it to other skills like a sensor module would. This will be covered in an following article.

4 Experiments

To prove the merits of the proposed architecture, we experimented in a simulator with a reactive agent, a (simulated) robot, using a genetic algorithm and neural networks. The first experiment was to learn the task of fetching an object and bringing it back to a specific area. In order to test the ability to accumulate new skills while preserving the previously acquired ones, we then modified the task to include energy consumption and the need to go to a specific area to recharge the batteries of the robot.

4.1 The Robot

The robot we simulated is composed of two motors, one for each wheel, and a claw to grab the target object. It also has 18 sensors, from obstacle detectors to target orientation and energy levels. The robot's software, through what we call a Sensor Module, is able to compute the derivative of any sensor input and use it as a virtual sensor. It can also generate a sinusoidal wave that can be used in the same way (useful to solve deadlock situations).

The motor command, issued by the Motor Module to its actuator as a real number between 0 and 1, is interpreted as a percentage of the actuator's capability (either power, speed, angular position, etc.). In the case of the wheels, 1 corresponds to full speed forward, 0 to full speed backwards and 0.5 to stopped. In the case of the claw, the number corresponds to its position (treated as binary): above 0.5 is closed, under 0.5 is opened.

4.2 Skill internal algorithm and the learning algorithm

The skills we used implement a multi-layered perceptron as their internal function, for which the input layer corresponds to the input vector of the skill and the output layer to its output vector. Depending on the skill, its neural network will use 2 or 3 hidden layers of N_{HIDDEN} neurons, with:

$$N_{HIDDEN} = Max(N_{Vi}, N_{Vo}) + 2 \tag{4}$$

N_{Vi} the number of neurons on the input layer
N_{Vo} the number of neurons on the output layer

The transfer function of the layers of the neural network is sigmoid, except for the last layer which uses a linear transfer function that is clamped between 0 and 1.

We acknowledge that this configuration has a high convergence time for the genetic algorithm we use but it is generic enough to cover most kinds of skills.

Unlike the traditional non-hierarchical approach which use a unique skill with a single neural network that connects all sensors and actuators,in the hierarchical approach, each skill has its own neural network using only the appropriate actuators , sensors or subskills.

To learn the tasks, we used a simple genetic algorithm. The weights of the neural network's connections are arranged in a vector, which corresponds to the genome of the individual from the genetic algorithm's point of view. The genome will be bred and mutated in accordance with the fitness function the environment provides as feedback. The relation between the skills and the genetic algorithm is illustrated in Figure 4.

The genetic algorithm works as follows. An initial population is generated randomly (Alg.1 line 1). Each individual is evaluated in an environment related to the task to learn and is given a score by the fitness function (or reward function) of the environment (Alg.1 line 6). The individuals with the best scores are selected (Alg.1 line 7 and 8) and their genomes mixed and mutated to

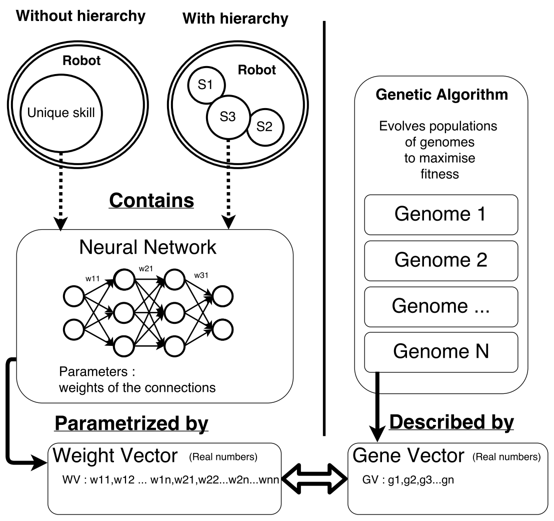

Fig. 4 Relation between the skills and the genetic algorithm

Algorithm 1: Genetic algorithm

```
   Data:
   current population: $Pop_c$
   selected population: $Pop_s$
   new population: $Pop_n$
   population size: $PopSize$
   iteration limit: $LIM$
   Result:
1  $Pop_c \leftarrow generateRandomGenomes()$
2  while $LIM$ not reached do
3      $Pop_s \leftarrow \emptyset$
4      $Pop_n \leftarrow \emptyset$
5      foreach genome $G$ in $Pop_c$ do
6          $evaluateFitness(G)$
       end
7      $sortByHighestFitness(Pop_c)$
8      $Pop_s \leftarrow top30\%(Pop_c)$
9      while $Pop_n \leq PopSize$ do
10         $Pop_n \leftarrow Pop_n + crossbreed(Pop_s)$
       end
11     $Pop_c \leftarrow Pop_n$
   end
```

generate a new population to be evaluated (Alg.1 line 10).

We choose to use a genetic algorithm for its simplicity, exploratory properties and good performance with delayed rewards. By nature, there is no need to provide it with a description of how to achieve a goal (unlike methods which use Backpropagation) but only with a way to measure if the performance of an individual is better or worse than the performance of other individuals. Unlike other reinforcement algorithms where the reward signal modifies the behaviour, the quality of the behaviour is judged at the end of the life cycle of the individual. This allows the individual to get negative rewards if it will lead to a superior positive reward later on, thus circumventing the issue of delayed reward.

Algorithm 2: CrossBreeding algorithm

```
   Data:
   input population: $Pop_{in}$
   parent genome A: $G_A$
   parent genome B: $G_B$
   new genome: $G_{New}$
   genome size: $GSize$
   Result:
1  $G_A \leftarrow pickRandomGenome(Pop_{in})$
2  $G_B \leftarrow pickRandomGenome(Pop_{in})$
3  $G_{New} \leftarrow \emptyset$
4  $pivot \leftarrow random(1, GSize)$
5  while $iterator \leq GSize$ do
6      if $iterator \leq pivot$ then
7          $G_{New} \leftarrow G_{New} + getGene(iterator, G_A)$
       else
8          $G_{New} \leftarrow G_{New} + getGene(iterator, G_B)$
       end
   end
```

One of the major drawbacks of the genetic algorithm is its high cost in resources, specifically the evaluation of the fitness function which means running the individual, i.e. an agent, in a simulated environment for a sufficient period of time. However, as the evaluation of each agent is completely independent, it can be run in parallel. This allows for the use of High Performance Computing solutions.

4.3 The tasks

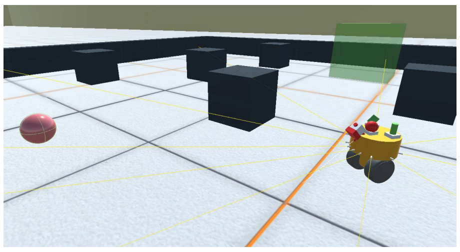

Fig. 5 The agent collecting the object

4.3.1 Scenario 1: Collect

In this scenario the goal is to teach the robot a collection task which consists of picking up an object and bringing it back to a drop zone without colliding with obstacles.

We choose to divide the complex task into a curriculum of five sub-tasks, of which three are base tasks: going to the object in an empty environment, going to the drop zone in an empty environment, and avoiding collision while moving in an environment with obstacles. On these base tasks we build two higher level complex tasks: going to the object while avoiding collision in an environment with obstacles, and going to the drop zone while avoiding collision in an environment with obstacles.

Learning the largest complex task is an interesting challenge. We can see in its sub-tasks that going to the object and going to the drop zone will both use motor functions in a completely opposite way and will have to be used sequentially and exclusively. Conversely, the Avoid task would be best used as a composition of vectors, by altering the set trajectory to smoothly avoid an obstacle.

4.3.2 Scenario 2: learning with sub-optimal subskills, retraining and learning in broader context

After the Scenario 1 resulted in a trained and functional hierarchy we observed the Avoid skill was causing a bottleneck for the performance of the hierarchy. Drawing inspiration from the following quote we experimented with an other learning strategy.

> Hierarchical architectures are particularly sensitive to the shaping policy; indeed, it seems reasonable that the coordination modules be shaped after the lower modules have learned to produce the simple behaviours. [...] experiments [...] show that in fact good results are obtained by: shaping the lower CSs, then "freezing" them and shaping the coordinators, and finally letting all components free to go on learning together.

(Dorigo and Colombetti, 1994)

In this scenario we use the already trained hierarchy of Scenario 1 an try to improve its performance by retraining the Avoid skill causing a bottleneck. We allocate more resources to the original training of the Avoid skill and measure the performance improvement. We also use an alternative training method of learning in a broader context. In this method we will retrain the Avoid skill from scratch while the agent is driven by the Collect skill and it's hierarchy, using the final Collect task as the environment and reward function.

4.3.3 Scenario 3: Collect with power management and the modularity of MIND

In this scenario we add to the initial collection skill the energy consumption and the necessity to recharge the robot's battery. A sensor which indicates the battery level and sensors which give information about the direction of the power source are added to the robot. This scenario adds to the previous one the base task of going to the power source to recharge and the complex task of going to the power source without colliding with obstacles, re-using the previously learned avoid task.

4.4 The hierarchies

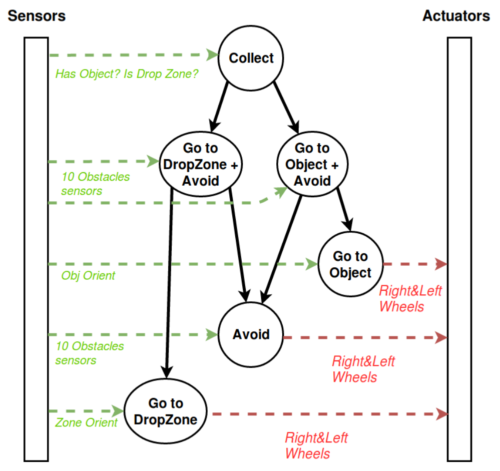

Fig. 6 The complete hierarchy for the initial collection task, with sensor and motor information shown

To create an initial hierarchy there are many possible ways to organize the different sub-skills, all of which are valid, as long as they are able to learn from the curriculum.

When adding a new skill to an already existing skill in the hierarchy there are two possible ways to go about it:

– modify the existing complex skill so that it integrates the new one, and retrain it to fit (Retrained variant as in Figure 7),
– create a new complex skill that coordinates the new skill with the existing complex skill. The new complex skill will be the one that undergoes the training and not the pre existing one (Encapsulated variant as in Figure 8).

5 Results

A total of seven complete behaviours have been taught using High Performance Computing. six of them , the initial Collect hierarchy, have been trained ten times separately to minimize the impact of the stochastic nature of genetic algorithms on the results.

Videos of the results are available at :
`https://www.lirmm.fr/~suro/Works.html`

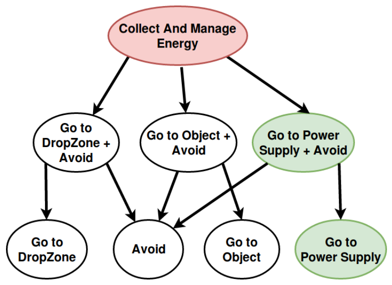

Fig. 7 The hierarchy for the collection task with energy consumption, Retrained variant : the old master skill is replaced

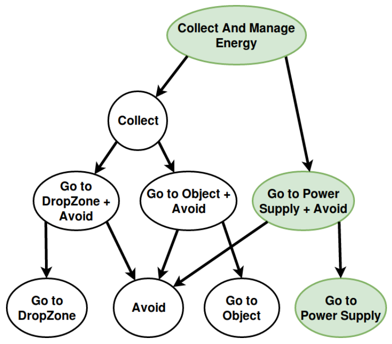

Fig. 8 The hierarchy for the collection task with energy consumption, Encapsulated variant : the old master skill becomes a sub-skill of the new master skill

5.1 Scenario 1: Collect

Below we show the results obtained on the Collect scenario.

Here we aim to demonstrate that the MIND architecture is able to organize simple skills into complex behaviours, and is able to do it reliably even when using a simple genetic learning process.

The complexity of tasks can be difficult to assess, but it is easy to understand that the neural network assigned to the GoToDropZone skill, which uses two inputs, will have far less connections, and thus less weights to adjust, than the neural network assigned to the Avoid skill, which uses more than 10 inputs. Figures 9 and 10 shows how this difference in complexity impacts learning. Most attempts of the GoToDropZone and GoToObject skills reach their maximum value in a hundred generations, whereas most attempts of the Avoid skill continue to evolve past 500 generations.

We see that in both GoToObject and GoToDropZone skills, one attempt in ten failed to reach the optimal result within the number of generation given. It is interesting to note that both failed attempts are still improving with each generations, which is positive in the context of our work aimed at supporting open ended and lifelong development of artificial agents. These failed attempts also led to the questions of whether it is possible to continue the learning process, training the complex skills, with sub-optimal subskills and how would the system react to improving and retraining subskills after the whole hierarchy has been trained. We will answer these questions in Scenario 2.

Based on the best base skills, we learn the complex skills GoToObject+Avoid and GoToDropZone+Avoid. The GoToObject+Avoid seems to be slightly more difficult to learn than GoToDropZone+Avoid, which can be explained by the fact that the object is smaller than the dropZone and requires more precision (the "claw" must be aligned with the object to grab it). Both skills still succeed on each of their respective 10 attempts within the given number of generations.

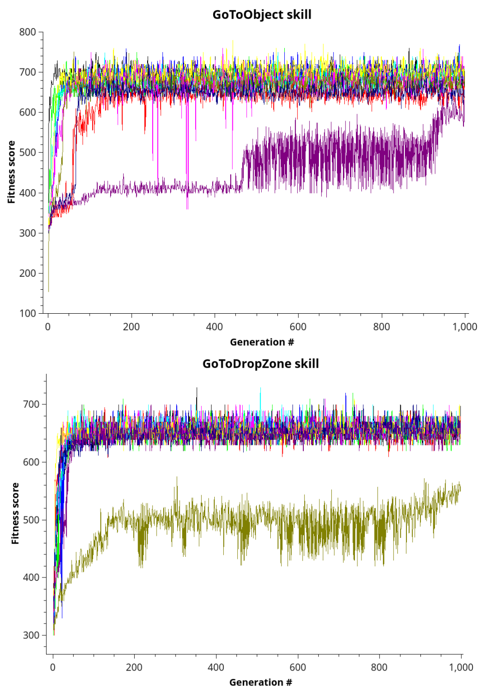

Fig. 9 GoToObject and GoToDropZone base skills : best individual score for each generation, 10 separate attempts over 1000 generations.

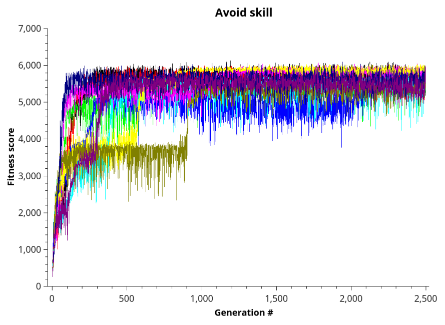

Fig. 10 Avoid base skill : best individual score for each generation, 10 separate attempts over 2500 generations.

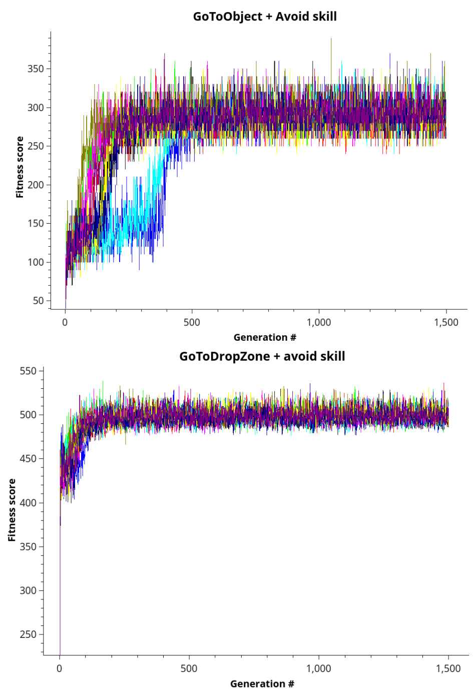

Fig. 11 GoToObject + Avoid and GoToDropZone + Avoid complex skills : best individual score for each generation, 10 separate attempts over 1500 generations.

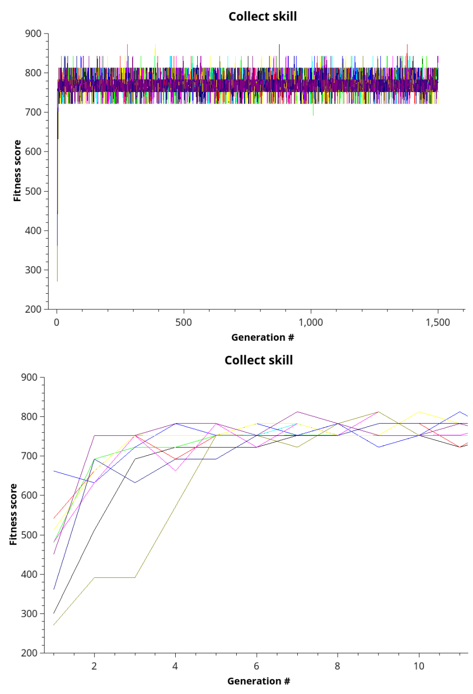

Fig. 12 Collect master skill : best individual score for each generation, 10 separate attempts over 1500 generations. (bottom : showing only the fist 10 generations scores)

Finally the master skill Collect, having a very simple network to train, succeeds on each of it's 10 attempts in under 10 generations.

5.2 Scenario 2: learning with sub-optimal subskills, retraining and learning in broader context

During the many experiments we made and attempts to find the right number of generation needed for the evaluation of MIND presented in Scenario 1, we were confronted by the question of what constitutes an adequate skill, when to stop learning, and what happens when we build a hierarchy based on sub optimal subskills.

We initially set up the learning process of each skill with 1000 generations. The resulting Collect skill did work, but still regularly failed due to collisions. It shouldn't be a surprise that the bigger neural network of the avoid skill, coordinating more than 10 sensors, would need more resources to train appropriately than other smaller networks.

We retrained Avoid from scratch with 2500 generations and retrained all the higher level skills. With 2500 generation, the final fitness score of the Avoid skill increased 10% compared the Avoid skill trained with 1000 generation. The Collect skill based on the retrained Avoid skill saw its final benchmark score increase 14% , strongly indicating that the Avoid skill is the bottleneck in the overall performance of the hierarchies.

Even by allocating more resources to the Avoid skill (more than doubling it), we were not able to improve significantly the overall quality. This leads us to question the quality of the Avoid learning task in the curriculum.

The choices made in the elements of the reward function are certainly to blame, and what we find reasonable to guide the development of a behaviour, it is the nature of evolution to exploit it to achieve the best score, regardless of the spirit of the law. For instance, in very early experiments the only reward for the Avoid skill was a negative one which the individual received upon colliding with an obstacle, and so the fittest individual simply did not move. Each subsequent laws added was in turn exploited until we found a set of laws which gave us a behaviour close to what we wanted. This illustrate the problem of defining the reward function, which can be as difficult as simply programming the behaviour we want.

We needed an Avoid skill to build the hierarchy, but now that each skill has it's own defined role in the hierarchy, instead of trying to improve the quality of Avoid on its own, we will train it in the final context of the Collect skill, as an element of the hierarchy.

To that end, we retrained the Avoid skill from scratch by using the already trained hierarchy of scenario 1 and by putting the robot in the final Collect environment. The master skill was the Collect skill from scenario 1 and reward signal came from accomplishing the Collect task,instead of our hand crafted Avoid reward function, but it was the Avoid skill that underwent training. By only using 1500 generations on the Avoid skill with this method, the final benchmark score of the collect skill improved by 30%.

This result suggests that there could be many more learning strategies to consider when teaching to a hierarchy beyond a simple curriculum from basic to complex tasks, and that going back to further improve the sub-skills in the final context, once the whole curriculum has been roughly taught, can yield better results than trying to maximize each sub-skill before moving on to the next.

The initial progressive training of the hierarchy is still of great importance, even if each skill shouldn't be trained to perfection, this first run through the curriculum is where each skill will specialize in its role within the hierarchy. During previous experiments not detailed in this paper, we attempted to setup the collect hierarchy and make all skills learn at the same time, as a result most branches of the hierarchy were ignored and a single skill attempted to learn the complete task.

5.3 Scenario 3: Collect with power management and the modularity of MIND

In this scenario we plan to demonstrate that the MIND architecture is suited to open ended development by adding new skills to expand the abilities of an already trained hierarchy.

The benchmark for this scenario is the same as that in scenario 1 with the addition of power management. The battery of the robot will discharge unless the robot stands in the power source area, in which case the battery will quickly recharge. The benchmark is run for a given time frame or until the robot fails (a collision occurs or the power level of the robot reaches zero). Each time the robot brings back the object to the drop zone, it scores one point and a new object is placed at random in the environment.

No indication of a what constitutes a low battery level was given to the robot, simply the current battery level. The robot determined when to abandon his current collect task and head for the power source based on its own experience with his battery discharge speed and the size and complexity of the environment.

Both variants of the hierarchy (Retrained (fig: 7) and Encapsulated (fig: 8)) obtain comparable scores showing both are valid. However we want to point out that the encapsulated variant preserves the original collect skill, making it available for future combinations, which is an important part of what we are trying to accomplish with the MIND architecture.

6 Limits

Our experiments have shown the ability of MIND to handle small hierarchies, however there could be some concern with large hierarchies, specifically deep hierarchies where the successive transfer of influence could result in a form of noisy motor output caused by several unrelated subskill receiving a very small amount of influence. In our present context, the motor tasks do not require extreme precision and we suspect the hierarchy isn't deep enough to generate this noisy output. The reason this problem could arise is mainly due to the use of genetically trained artificial neural networks as the skills internal functions giving a "good enough answer" (for a given input that should result in a left turn, if the output is 0.99 for left turn and 0.01 for right turn, the impact of right turn isn't noticeable). Theoretically if an influence command is transmitted from master skill to base skill is 1.0 with all other influences remaining at 0.0, then no matter how many complex skills are involved the output will be the exact command of the base skill. What it means in practice is that if noisy outputs become noticeable, it is due to imprecision in the neural networks that should be finely retrained (as the depth of the hierarchy increases, the acceptable imprecision of the internal functions decreases). Future works will tell us if this concern is justified, should noisy output become a problem and finer training of the internal function be costly and impractical, we would investigate ways to reduce noise in large networks such as filters, threshold layers, or logit functions (invert sigmoid).

7 Using MIND on a real world robot

Given that the MIND hierarchy learned in simulation is entirely reactive, we did not anticipate any problem using it on a real world application. We designed a simple task using the two base skills GoToObject and Avoid, and the complex skill (in this case also the master skill) GoToObject + Avoid.

The only difficulty we encountered was the delay of sensor acquisition and motor response, most probably because of the hobby grade hardware being used. Nonetheless, simply by setting proper sensor dead zones and maximum range, our crude robot demonstrated the expected behavior using the hierarchy and skills learned in a simulation that wasn't calibrated in any way to fit the robot or real world environment.

Our robot used a Raspberry PI3 running linux and our java implementation of MIND. Sensori-motor equipement was made from plug and play hobby kit elements (Grove Pi) and included 10 ultrasonic range finders, a dual motor control board, a 9dof sensor, a basic switch, a servo control card to control a twin servo pan and tilt, a basic webcam and two servos to control the claw.

The pan and tilt arm and webcam were used, with the OpenCV computer vision library, to find and track a luminous ball. The position of the pan arm was converted to a sensor information replicating the orientation sensor used in the simulation.

Finally the robot was driven by two shopping trolley wheels, actuated by a friction motor roller, the chassis was balanced by two free rotating wheels.

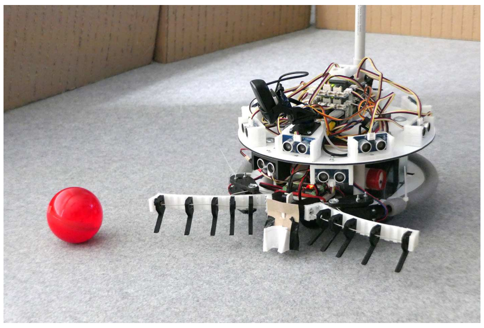
Fig. 13 Our crude robot

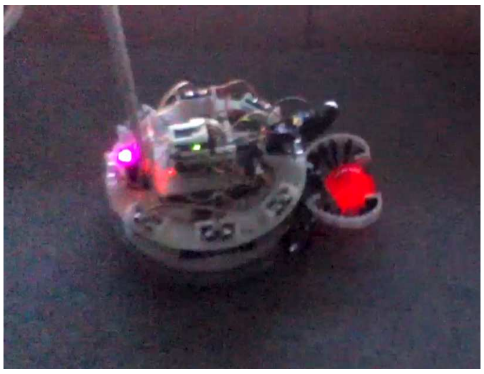
Fig. 14 Basic object detection using a luminous ball, a webcam, OpenCV and low lights

Videos of the results are available at :
`https://www.lirmm.fr/~suro/Works.html`

8 Conclusions

The positive results obtained on the collect task is encouraging, the MIND architecture was able to transform the curriculum into a hierarchy of skills with both coordination and mutual exclusion of skills. It was able to handle both low level sensori-motor tasks and complex skill influence operations, without providing any information on the nature of the task to the internal skill function (Exclusive tasks are a classification problem while coordination requires a function approximation approach).

Even if we only discussed skills with multi-layer perceptrons as internal functions because of our focus on the learning problem, the encapsulation of the internal function into a skill can give any function the ability to integrate into a MIND hierarchy. As long as a wrapper can convert inputs and outputs vectors of real numbers, any kind of functions can be used, including non learning ones. For instance, the function controlling the "claw" of the robot in our experiment is a trivial Java procedure that has been warped into a skill.

Other authors (Narvekar et al., 2016), (Gülçehre et al., 2016) already established the advantages of progressive learning, by guiding the learning entity through increasing the complexity of the task or by learning a curriculum of complementary tasks aimed towards the completion of a complex task. The positive results we presented in this paper by retraining sub-skills of the hierarchy seem to indicate the advantage of going back and forth from simple to more complex lessons rather than trying to attain perfect results on the basic tasks before moving on to more complex ones.

At the time of writing, several improvements have been made to the implementation of the MIND architecture, most notably using the NEAT algorithm (Stanley and Miikkulainen, 2002) to replace our basic genetic algorithm. this greatly improves performance but also completely removes the need to define the neural network topology of the skill's internal function.

As we suggested before, we added a number of simple memory modules conforming to the MIND system of influence in order give the ability to the agent to evolve beyond simple reactive behaviour. We are currently investigating a way to teach a knowledge representation with the constraint of not breaking the barrier of the interiority of the agent.

Our next step towards our goal of creating and artificial agent capable of open ended and lifelong development through curriculum learning under human supervision is the automatic creation of new skill modules. For a new lesson given by the instructor, what are the relevant sensor inputs? what actuator outputs or sub skills should the agent use? what language and strategies in formulating a lesson would help the agent understand what's expected while at the same time be natural and simple to define for the human instructor?

The MIND architecture seems promising, but further studies need to be made on its ability to handle very large hierarchies of skills if it is to be applied in a lifelong learning and development context.

Acknowledgements We thank Eric Bourreau (LIRMM) and Marianne Huchard (LIRMM) and the members of the SMILE team (LIRMM) for their comments that helped improve the manuscript.

This work has been realized with the support of the High Performance Computing Platform HPC@LR, financed by the Occitanie / Pyrénées-Méditerranée Region, Montpellier Mediterranean Metropole and the University of Montpellier, France.

References

Arkin, R. C. and Balch, T. (1997). Aura: Principles and practice in review. *Journal of Experimental & Theoretical Artificial Intelligence*, 9(2-3):175–189.

Bengio, Y., Louradour, J., Collobert, R., and Weston, J. (2009). Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 41–48. ACM.

Brooks, R. A. (1986). A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1):14–23.

Dorigo, M. and Colombetti, M. (1994). Robot shaping: Developing autonomous agents through learning. *Artificial intelligence*, 71(2):321–370.

Dorigo, M. and Colombetti, M. (1998). *Robot shaping: an experiment in behavior engineering.* MIT press.

Felleman, D. J. and Van Essen, D. C. (1991). Distributed hierarchical processing in the primate cerebral cortex. *Cerebral cortex*, 1(1):1–47.

Grove Pi. `https://www.dexterindustries.com/grovepi/`.

Gülçehre, Ç., Moczulski, M., Visin, F., and Bengio, Y. (2016). Mollifying networks. *CoRR*, abs/1608.04980.

Kruger, N., Janssen, P., Kalkan, S., Lappe, M., Leonardis, A., Piater, J., Rodriguez-Sanchez, A. J., and Wiskott, L. (2013). Deep hierarchies in the primate visual cortex: What can we learn for computer vision? *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1847–1871.

Narvekar, S., Sinapov, J., Leonetti, M., and Stone, P. (2016). Source task creation for curriculum learning. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, pages 566–574. International Foundation for Autonomous Agents and Multiagent Systems.

OpenCV computer vision library. `https://opencv.org/`.

Oudeyer, P.-Y. (2012). Developmental robotics. In *Encyclopedia of the Sciences of Learning*, pages 969–972. Springer.

Piaget, J. (1954). *The construction of reality in the child.* Basic Books, New York.

Piaget, J. and Duckworth, E. (1970). Genetic epistemology. *American Behavioral Scientist*, 13(3):459–480.

Raspberry PI3. `https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/`.

Simonin, O. and Ferber, J. (2000). Modeling self satisfaction and altruism to handle action selection and reactive cooperation. In *Proceedings of the 6th International Conference on the Simulation of Adaptive Behavior*, volume 2, pages 314–323.

Stanley, K. O. and Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127.