How to Promote Generalisation in Evolutionary Robotics: the ProGAb Approach

Tony Pinville
pinville@isir.upmc.fr

Sylvain Koos
koos@isir.upmc.fr

Jean-Baptiste Mouret
mouret@isir.upmc.fr

Stéphane Doncieux
doncieux@isir.upmc.fr

ISIR, Université Pierre et Marie Curie-Paris 6, CNRS UMR 7222
4 place Jussieu, F-75252, Paris Cedex 05, France

ABSTRACT

In Evolutionary Robotics (ER), controllers are assessed in a single or a few environments. As a consequence, good performances in new different contexts are not guaranteed. While a lot of ER works deal with robustness, i.e. the ability to perform well on new contexts close to the ones used for evaluation, no current approach is able to promote broader generalisation abilities without any assumption on the new contexts. In this paper, we introduce the *ProGAb* approach, which is based on the standard three data sets methodology of supervised machine learning, and compare it to state-of-the-art ER methods on two simulated robotic tasks: a navigation task in a T-maze and a more complex ball-collecting task in an arena. In both applications, the *ProGAb* approach: (1) produced controllers with better generalisation abilities than the other methods; (2) needed two to three times fewer evaluations to discover such solutions.

Categories and Subject Descriptors

I.2.9 [**Computing Methodologies**]: Artificial Intelligence—*Robotics*; I.2.6 [**Computing Methodologies**]: Artificial intelligence—*Learning*

General Terms

Algorithms

1. INTRODUCTION

In many robotic applications, the robot has to handle a lot of situations in its daily usage, which are different from the situations used to design its controller: the controller has to show sufficient generalisation abilities (GAb), so that the robot can behave well in all the situations. For instance, if the robot has to perform a task in a room, its performance should not critically depend on the size of this room, nor on its initial position or orientation, and so on. As Evolutionary Robotics (ER) [24, 29] aims at automatically designing controllers for robots, it has to address this need for GAb of controllers: *how can a controller be evaluated in only few instances of a task and also be successful in different contexts?* If a single (or a few) context is used to evaluate the possible solutions, the performance value does not always reward general behaviors compared to specific ones: there is no guarantee that the best solutions are the most general. For all practical purposes, the best evolved solutions often achieve bad GAb and one can argue that the lack of a general methodology to promote the GAb of controllers curbs the application of ER to real-world robotic tasks [13].

The problem of promoting the generalisation ability has been much studied in the framework of supervised machine learning [1]. Most of the works in supervised learning dealing with generalisation are based on a so-called *three data sets methodology* [15], which consists in assessing and validating a model by splitting the learned contexts into three different sets: 1) a training set used to fit the model; 2) a validation set used to measure its generalization capability and to prevent overfitting; 3) a test set used afterhand to assess the GAb of the model on several contexts, which were not used during the learning. This three data sets methodology, also applied in GP [15, 30], appears to be an efficient and simple way to promote and assess the GAb of optimized solutions.

Assessing the performance of each individual on each context of the several sets leads to conduct a lot of experiments with the robot, either in simulation or in reality. Such a methodology is then difficult to adapt to ER, which deals with a population of many solutions to evaluate. Consequently, reducing the number of evaluations is a major issue to take into account, as it concerns the most time-consuming part of an ER experiment. To deal with this constraint, several approaches have been previously proposed: optimizing in a few situations [4], randomly generating a training set from a larger set at each generation [18], adding noise on one specific context to make the optimal controller not dependent on some features [25, 19] or relying on neural networks with dynamic structures, which should automatically adapt to new contexts [20, 14]. However, these works can only be used with a few contexts or if the evaluation context is slightly modified.

The goal of this work is to introduce a generic methodology to Promote the Generalisation Ability, the *ProGAb* approach, which aims at:

- finding more general controllers than state-of-the-art methods in ER;
- minimizing the number of evaluations needed to discover such solutions.

A controller which is optimally general on a given task should behave well in all the possible instances of this task. For practical purposes, this generalisation ability (GAb) is assessed on a large finite set of previously selected instances of the task that have not been used when evaluating the solutions, like the test set in supervised learning. A controller is then optimally general, if it is optimal on both training set and if it achieves similarly high performances on test set.

Promoting the GAb of controllers highlights the following conflict: 1) while the performance value on a small training set is fast to compute, it can quickly lead to over-fitted solutions with low generalisation abilities; 2) the GAb on a larger set is more informative but cannot be computed for each evaluated solution because of the computational cost. In order to keep the number of evaluations on as low as possible, we resort to a methodology based on three major aspects: 1) all the evolved solutions are evaluated on a small training set; 2) we build during the optimization process a surrogate model which approximates the GAb on a larger set with function interpolation techniques; 3) to update this surrogate model, few controllers are assessed on this larger set while optimizing. The solutions are optimized with a Pareto-based Multi-Objective Evolutionary Algorithm (MOEA) in which two goals are defined: maximizing the performance value on the small training set and maximizing the approximated GAb on the larger set. A similar methodology has been successfully applied to the reality gap problem in [21].

The *ProGAb* approach has been tested on two robotic tasks performed in simulation: a navigation task in a T-maze inspired from [38] and [31] and a more complex ball-collecting task in an arena, previously introduced in [12].

2. RELATED WORK

2.1 Formalising the Generalisation Ability

Before discussing previous works dealing with the GAb of controllers, we propose a formal definition of this generalisation ability to be used in our analysis of the literature. Generalisation has been much studied in the field of supervised machine learning. Our definition is inspired from the standard *three data sets methodology* [15] used in supervised learning and is based likewise on three sets of contexts:

- a training set of contexts Ω_{train} is used to optimize the solutions;
- a validation set of contexts Ω_{valid} allows to estimate the GAb of some meaningful solutions;
- at last, a test set [1] of contexts Ω_{test} is brought into play to assess afterhand the GAb of the best found solutions on some not yet encountered contexts.

[1]The literature in machine learning sometimes reverses the meaning of "validation set" and "test set".

In an ER experiment, a context ω is an instance of the task to solve and corresponds to a set of specific values for all the parameters of the task: initial position of the agent, size of the environment, and so on. As the training set Ω_{train} and the validation set Ω_{valid} are both used to evaluate solutions during the optimization, we also define the evaluation set $\Omega_{eval} = \Omega_{train} \cup \Omega_{valid}$, which contains all the contexts possibly used for evaluation.

Let $\mathcal{F}$ be the fitness function defined on the task, let x be an individual of the controller space $\mathfrak{C}$, the generalisation ability G of x on a set of contexts Ω is defined as follows:

$$G(x, \Omega) = \sum_{\omega \in \Omega} \mathcal{F}(x, \omega)$$

An individual x is optimally general if it verifies the two constraints below:

$$\forall y \in \mathfrak{C}, \qquad G(y, \Omega_{eval}) \leq G(x, \Omega_{eval}) \tag{C1}$$

$$\frac{G(x, \Omega_{eval})}{|\Omega_{eval}|} \simeq \frac{G(x, \Omega_{test})}{|\Omega_{test}|} \tag{C2}$$

An optimally general solution has to behave at best on all the evaluation contexts (both from Ω_{train} and Ω_{valid}) (C1) and to behave similarly well on the validation contexts (C2). An additional constraint is that the solutions have to be mainly assessed on the training set Ω_{train} and as infrequently as possible on the validation set Ω_{valid}, although the performance on Ω_{train} and the GAb on Ω_{valid} can be conflicting.

2.2 Promoting robustness

The very first method introduced in ER to deal with the GAb of controllers consists in adding noise to the inputs of the controller [33] in order to evolve solutions which can exhibit robust behaviors to sensor uncertainties. Similar approaches have been used to evolve controllers in simulation able to behave well on the physical robot [25, 19]. The main idea is to hide the specificities of the environment with noise during evaluation, so that controllers cannot exploit them and show better GAb when tested in another context.

Some other methods rely on adaptive controllers, like dynamically-rearranging neural networks [20] or plastic neural networks, which encode learning rules [14]. Such structures should adapt to environmental changes and be able to retrieve good behaviors when the context is slightly modified.

These works deal with robustness, which can be interpreted as a particular case of generalisation. It corresponds to the GAb regarding only small variations of the environment or of the controller: for instance, adding noise to the sensors or testing close initial positions. Contrariwise, generalisation does not make any assumption on the closeness from a context to another: the size of the environment can double or the starting position of the robot can radically change. Consequently, the approaches described above are not necessarily adapted to enhance the GAb of controllers.

2.3 Promoting generalisation

A review of some ER papers (table 1) indicates that, although a lot of works tackle robustness issues, only a few ones deal with the GAb of controllers in a more general sense. On the whole, the most common method consists in evaluating the controllers with a fixed set of arbitrary evaluation contexts, often less than 10 (table 1). For instance in [12], three different initial positions are used to assess the controller for a ball-collecting task. Similarly, in [38], controllers are optimized for a navigation task in six contexts depending on the shape of the maze and the initial position of the robot. In these works, the generalisation abilities of the optimized individuals are not assessed on new contexts.

Some other methods rely on a set of evaluation contexts, which is not fixed during the optimization. For instance, in [18], the individuals are evaluated on 10 contexts, which are randomly picked up for each individual at each generation in a very large set of evaluation contexts. This evaluation set can also be co-evolved with the solutions as a subset of all the possible contexts like in [3]. The best individual found at the end of the optimization is then tested on all the possible contexts to assess its generalisation ability. Nevertheless, each context used for testing has possibly been used for evaluation, although Ω_{eval} and Ω_{test} should be independent to ensure good estimations of the GAb.

In [2], vision-based controllers are evolved to avoid obstacles in an arena. The individuals are evaluated on four contexts depending on the initial position of the robot (Ω_{eval}). The individuals of the last generation are re-evaluated on four different contexts with a different configuration of obstacles (Ω_{test}) and the best one is selected depending on its performances on these new situations. It is more adapted to truly assess the GAb of the controllers, but such an approach cannot be straightforwardly used if Ω_{eval} is large because of high computational costs.

2.4 Reducing the number of evaluations

To deal with the constraint C1, a trivial approach boils down to directly optimise solutions on the whole evaluation set: each solution is evaluated on each context of Ω_{eval}. But, as the evaluation set can be very large, it is often infeasible in a reasonable computational time. The problem of assessing the GAb cannot be addressed in general without considering the need to reduce the number of evaluations.

Several methods have been proposed to lower the number of evaluations in evolutionary algorithms. The racing algorithm [5, 16] consists in sequentially evaluating candidate solutions and discarding the poor ones as soon as statistically sufficient evidence is gathered against them. The early stopping technique [7] is similar, as it ends the evaluation of a solution if further evaluations cannot allow it to escape dominance by another solution. Such approaches require the fitness function to meet some constraints regarding its statistical distribution with the racing method or its monotony with the early stopping algorithm. Besides, it does not allow to quickly discriminate between the performances of two sub-optimal solutions, which is necessary in the state-of-the-art MOEAs.

Another method consists in building an approximate model of the computationally costly fitness function, instead of computing the exact fitness value for each solution [32, 17]. Such an approach could be used to approximate the GAb of controllers on Ω_{eval} by a so-called surrogate model, which is updated by computing the exact GAb values of only few individuals during the optimization. A similar methodology has been developed in [21] to solve the reality gap problem. It relies on a surrogate model to approximate the transfer quality from simulation to reality of controllers evolved in simulation. It makes it possible to find well-transferable controllers with few computationally costly experiments on the physical robot. Our methodology is inspired from this last work.

Table 1: summary of ER papers. x indicates which papers deal with robustness and/or GAb and if there is an evaluation during and/or after optimisation process

Papers	Robustness	GAb	$\lvert\Omega_{eval}\rvert$	Ω_{test}?
Jakobi 1995 [19]	x		1	x
Miglino 1995 [25]	x		1	
Jakobi 1997 [18]	x	x	10	x
Kondo 1999 [20]	x		10	x
Di Paolo 2000 [11]	x		1	x
Floreano 2001 [14]	x		1	x
Berlanga 2002 [3]		x	6	x
Ziemke 2002 [38]		x	3	
Barate 2008 [2]	x	x	4	x
Doncieux 2010 [12]		x	3	
Lehman 2010 [22]			1	
Bongard 2011 [6]	x		1	x

3. METHOD

The *ProGAb* approach deals with two goals: 1) evolving controllers with good generalisation abilities regarding several test contexts on a given task; 2) performing as few evaluations as possible on all these contexts.

Our approach is based on the *three data sets methodology* detailed in the section 2.1. As the performance on the training set Ω_{train} and the GAb on the validation set Ω_{valid} can be conflicting, we propose a multi-objective formulation of this methodology in which three objectives are optimized via a Pareto-based MOEA: (1) the performance F on the training set Ω_{train}; (2) the GAb G on the validation set Ω_{valid}; (3) the behavioral diversity objective. The addition of this third objective will be discussed later.

The GAb on the validation set can be very computationally costly to compute on every evolved solution. In order to minimize the number of evaluations on the validation set, we rely on a so-called surrogate model to approximate this second objective during the optimization process.

3.1 Surrogate model of the GAb measure

Surrogate models [37, 17] are used in real engineering problems when evaluating an individual on the target system means very high computational costs or too long experiments. Then, instead of a direct evaluation on the physical system, solutions are optimized via an approximate model of it. In evolutionary algorithms, surrogate models can be used to approximate a computationally costly fitness objective. In order to build a surrogate model of the GAb function during the evolutionary optimization, a few controllers have to be assessed on Ω_{valid} during the run to obtain the corresponding exact GAb values and update the approximate model.

To build the surrogate model, we rely on Inverse Distance Weighting interpolation [35] (IDW). While very simple, this technique has demonstrated to be competitive with more complex methods, like Krieging methods [28].

We assume that a behavioral distance between controllers b_{dist} has already been defined on the task, which compares the individuals based on their behaviors $b(x)$ on Ω_{train}. If some controllers have already been assessed on Ω_{valid} (at least 1) and the corresponding exact GAb values have been computed, a surrogate model $\hat{G}$ of the generalisation ability can be interpolated by Inverse Distance Weighting from these values.

Let $\mathfrak{C}$ be the set of all the possible controllers, let $\mathfrak{C}_{tested}$ be the set of the controllers already assessed on Ω_{valid} and $G^*(c_i)$ the exact GAb value corresponding to each controller $c_i \in \mathfrak{C}_{tested}$. The surrogate model of the GAb $\hat{G}$ is built as follows:

$$\forall c \in \mathfrak{C}, \hat{G}(c) = \frac{\sum\limits_{c_i \in \mathfrak{C}_{tested}} G^*(c_i)\ b_{dist}(c_i, c)^{-2}}{\sum\limits_{c_i \in \mathfrak{C}_{tested}} b_{dist}(c_i, c)^{-2}}.$$

The use of a surrogate model implies the choice of an update heuristic that selects which test experiments have to be conducted from Ω_{valid} in order to pertinently upgrade the model. We choose a relatively simple heuristic that selects, *at each generation*, the best controller on Ω_{train} whose behavioral distance to the controllers of $\mathfrak{C}_{tested}$ is maximal. It should ensure that the surrogate model is not built on a too localized part of the behavior space.

3.2 Evaluation objectives

Each controller is evaluated by three objectives to be maximized:

1. the performance F on Ω_{train}, to find efficient controllers on the training set;
2. the corresponding approximated GAb $\hat{G}$ on Ω_{valid} computed with the surrogate model, to promote the GAb of the solutions;
3. the behavioral diversity objective.

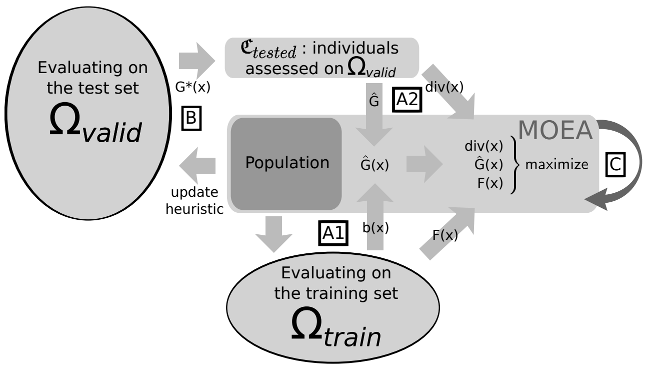
This last objective allows to maintain behavioral diversity among the population, which efficiently enhances exploration of the controller state space [12, 26]. To quantify the diversity of a controller from the controllers already assessed on the validation set, we define a behavioral diversity value as follows. Let $\mathfrak{C}_{tested}$ be the set of the controllers already assessed on Ω_{valid} and b_{dist} the behavioral distance between individuals, the behavioral diversity value $div(x)$ for a given controller x is:

$$div(x) = \min_{x_i \in \mathfrak{C}_{tested}} b_{dist}(x,\ x_i)$$

This diversity value doesn't depend on the genotype nor on the phenotype of the controller being evaluated, as the behavioral distance b_{dist} is only derived from the behavior of the agent on Ω_{train}. It promotes solutions that show the more different behaviors on Ω_{train} from those of the controllers already assessed on Ω_{valid} according to b_{dist}.

3.3 Algorithm

To compute the approximated GAb $\hat{G}$ at the beginning of a run, we assume that a controller x_0 has already been assessed on the validation set Ω_{valid}. AS the corresponding exact GAb $G^*(c_0)$ is known, it permits the initialization of the approximated GAb value for each controller.



Evaluating on the test set
Ωvalid
G*(x)
B
𝕮tested : individuals assessed on Ωvalid
Ĝ
A2
div(x)
MOEA
Population
Ĝ(x)
div(x)
Ĝ(x)
F(x)
maximize
C
update heuristic
A1
b(x)
F(x)
Evaluating on the training set
Ωtrain

Figure 1: Steps of the *ProGAb* approach at each generation – A1. For each controller x of the current population, the behavior $b(x)$ and the performance $F(x)$ are evaluated on Ω_{train}. A2. The simulated behavior of a given controller allows the computation of the corresponding approximated GAb as predicted by the surrogate model $\hat{G}$ along with the diversity value $div(x)$ to the controllers of $\mathfrak{C}_{tested}$. B. If this behavioral diversity value is high enough for some controllers, one among them is randomly picked up depending on the update heuristic to be assessed on Ω_{valid}. The corresponding exact GAb $G^*(x)$ is computed and the assessed controller is added to the set $\mathfrak{C}_{tested}$. C. The evolutionary operators are applied to controllers and the selection step builds the next population.

The algorithm (Figure 1) iterates the following steps:

A. evaluation of the controller x:
   - A1. computation of the behavior $b(x)$ and the performance $F(x)$ of x on Ω_{train};
   - A2. evaluation of the 2 other objectives in relation to the controllers of $\mathfrak{C}_{tested}$ on Ω_{valid} (approximated GAb on Ω_{valid} and diversity objective);

B. depending on the update heuristic, a controller with high diversity is assessed on Ω_{valid};

C. application of evolutionary operators and generation of the next population.

4. TASKS

4.1 Task 1: Navigation in a T-Maze

The first task is an extension of the "roadsign problem" [38, 34]: an agent starts off at the bottom of a T-shaped maze, encounters an instruction stimulus (e.g. a light) while moving along a corridor and, when it reaches the junction, it has to turn left or right, depending on which stimulus has been encountered (Figure 2).

To make this task more cognitive in our experiment, the instruction stimulus is a combination of four stimuli (A, B, X, Y) following the same rule as in the AX-CPT working memory test [8, 31]. This task consists of a context cue (A or B), followed by a probe (X or Y) after some delay. The agent must turn to the left when the stimulus A is

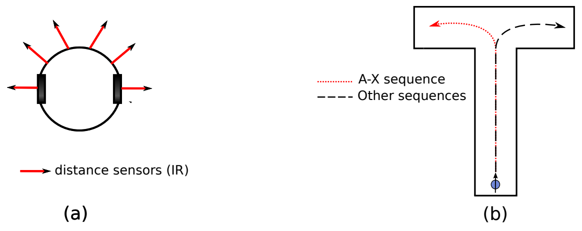


Figure 2: (a) Simulated mobile robot used for the T-maze task. The robot has four additional sensors, one by letter. (b) Map employed for this task.

Table 2: Encoding parameters used in both tasks.

Parameters	Task1	Task2
min./max. nb. of neurons[2]	3 / 20	10 / 30
min/max. nb. of connections[1]	5 / 50	50 / 250
prob. to add/remove a neuron	0.05 / 0.05	0.15 / 0.05
prob. to add/remove a connection	0.05 / 0.05	0.05 / 0.05
prob. to change the weight/bias	0.05	0.15

followed by the stimulus X, and to the right otherwise (for AY, BX, BY). The AX-CPT task into a T-Maze has already been implemented in neuroscience experiments, to test rat's discrimination performance [23].

Here, the agent is a simulated two-wheeled robot receiving sensory inputs from 6 infrared distance sensors and four letter sensors, one sensor for each letter A, B, X, Y, which receives 1 if the letter is presented, 0 otherwise. The robot controls its speed through two output units corresponding to its left and right motors. The agent is evaluated on each letter sequence (A followed by X, AY, BX, BY). The fitness increases by 1 if it turns to the correct side for the sequences AY, BX, BY and by 3 for the sequence AX. The maximal fitness value is 6.

To fit the experimental setup described in [23], both motors are disabled during the presentation of the letters. The whole task lasts 300 steps and takes place as follows with t the number of elapsed time steps:

- $0 < t < 40$: presentation of the first letter (A/B);
- $40 \leq t < 60$: delay, all the sensors are set to 0;
- $60 \leq t < 100$: presentation of the second letter (X/Y);
- $100 \leq t \leq 300$: the robot can move and must reach the correct side of the T-maze.

4.2 Task 2: Ball-collecting robot

The ball-collecting task (Figure 3) has been introduced by [12]. It requires basic navigation skills—obstacle avoidance, navigation towards several goals—and the ability to change its behavior depending on the context. The goal of this task is to explore an arena, to find balls and to bring them to a basket. A fitness point is granted each time a ball is put into the basket. Before getting a point, the robot must therefore learn to search for a ball, take it and keep it, search for the basket, reach it and then, and only then leave the ball. Furthermore, the task is not completely fulfilled when the first ball is collected, as other balls have to be collected and the robot cannot carry several balls at the same time. Four balls can be collected during each experiment.

[2]Used for the initial random population.

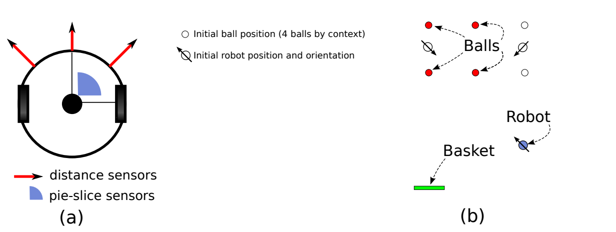

distance sensors
pie-slice sensors
(a)
Initial ball position (4 balls by context)
Initial robot position and orientation
Balls
Robot
Basket
(b)

Figure 3: (a) Overview of the simulated mobile robot for the ball-collecting task. The robot is equipped with two pie-slice sensors that detect the balls and two more that detect the basket. The view range of the right sensors is depicted on the figure, the left ones are symmetric. The robot is moved by two motors, the neural network setting the speed of each of them. (b) Map employed in this set of experiments. Several initial positions and orientations of the robot are depicted with a circle and an arrow.

The agent is a two-wheeled robot with ten sensors: three wall distance sensors, two binary bumpers, two binary ball detection sensors, two binary basket detection sensors and one binary carry ball sensor. The effectors are left and right wheel motors and a "catch ball" actuator, whose value must be greater than 0.5 to pick up or keep a ball and lower than 0.5 to release it.

The fitness function is the number of balls in the basket at the end of the experiment. The robot has 3000 time steps to collect all the balls.

4.3 Control approaches

We compare *ProGAb* with four control approaches:

- *Fit_only*: only one fixed context is used to assess the controllers ($|\Omega_{train}| = 1$).
- *Eval_all*: each individual is evaluated with all the contexts in Ω_{valid} ($\Omega_{train} = \Omega_{valid}$).
- *Fixed-Random-Initial* (FRI): Ω_{train} ($|\Omega_{train}| > 1$) is randomly generated from Ω_{valid} at the beginning of the run and remains fixed. This principle is described in [30].
- *Jakobi*: this method has been developed by [18] to provide robust controllers able to transfer well onto a physical robot. Ω_{train} is generated by randomly picking up a fixed number of contexts in Ω_{valid} at each generation. It is the same principle as in the *Random-Per-Generation* method (RPG) described in [30].

To be fair, all the five methods are tested with the same budget of evaluations. For the four control approaches, all the controllers of the population P are evaluated on Ω_{train} during n_g generations. The number of evaluations E is:

$$E = size(P) * n_g * |\Omega_{train}|$$

In addition, for the *ProGAb* approach, one controller is assessed on Ω_{test} at each generation:

$$E_{ProGAb} = size(P) * n_g * |\Omega_{train}| + n_g * |\Omega_{valid}|$$

All the parameter values used in both tasks are described in Table 3. For *ProGAb*, we use the same Ω_{train} as in

Table 3: Setup parameters for both tasks.

Method	$size(P)$	Task 1: $\lvert\Omega_{valid}\rvert = 180$		Task 2: $\lvert\Omega_{valid}\rvert = 30$	
		n_g	$\lvert\Omega_{train}\rvert$	n_g	$\lvert\Omega_{train}\rvert$
Fit_only	200	19000	1	6000	1
FRI	"	1900	10	1500	4
Jakobi	"	1900	10	1500	4
Eval_all	"	110	180	200	30
ProGAb	"	10000	1	5500	1

Table 4: Tasks parameters

Task 1	$\lvert\Omega_{valid}\rvert = 180$	$\lvert\Omega_{test}\rvert = 180$
map size	500 to 650	500 to 650
init. orientation	-30°to 30°	-30°to 30°
variation on x	-40 to 40	-20 to 20
variation on y	-40 to 40	-20 to 20

Task 2	$\lvert\Omega_{valid}\rvert = 30$	$\lvert\Omega_{test}\rvert = 600$
map size	600	650
init. orientation	1 angle	4 angles
init. position	2 positions	10 positions

Fit_only. A budget of 3,800,000 evaluations is arbitrarily fixed for the T-maze task, which corresponds to 10,000 generations for *ProGAb*. A budget of 1,200,000 evaluations is arbitrarily fixed for the ball-collecting task, i.e. 5,500 generations for *ProGAb*.

The parameters for each context ω in Ω_{train}, Ω_{valid}, Ω_{test} for both tasks are: 1) the map size; 2) the initial orientation of the robot; 3) the initial position of the robot. We add another parameter in the ball-collecting task: the initial positions of the balls. There are 6 ball positions for 4 balls which means there are 15 different contexts (C_4^6) for a given map size and a given initial position. These parameters are described in Table 4.

All the five approaches have been implemented using the efficient state of-the-art MOEA NSGA-II [10] [3]. Both tasks are simulated in a simple 2D simulator without friction or slippage.

Diversity objective.

To efficiently explore the controller state space, we add a behavioral diversity objective to the fitness function in a Pareto-based multi-objective optimization for all the four control approaches: *Fit_only*, *Eval_all*, *FRI* and *Jakobi*. According to the conclusions of [9], the behavioral diversity objective to maximize $o_{bd}(x)$ is the average behavioral distance to the rest of the population P. Let b_{dist} be the behavioral distance used to compare the individuals and F the fitness function, the controllers are optimized via the following multi-objective scheme:

$$\text{maximize} \begin{cases} F(x) \\ o_{bd}(x) = \frac{1}{size(P)} \sum_{y \in P} b_{dist}(x, y) \end{cases}$$

Neural networks.

Neural networks are evolved structurally and parametrically to control a robot with a simple direct encoding [26], inspired by NEAT [36]. Classic sigmoid neurons are used with an output in $[-5, 5]$ and a bias in $[-5, 5]$.

In this encoding, a neural network is described as a directed graph and five mutation operators are implemented: adding or deleting a connection; adding or deleting a neuron; changing a weight or a bias using polynomial mutation [10]. Cross-over is not employed. All the parameter values used in both tasks are described in Table 2. Further details on the neural encoding can be found in [26].

Behavioral distance.

We use Hamming distance as behavioral distance b_{dist}. This generic distance has been successfully used in [12], where the Hamming-based approaches generated the most efficient controllers. The Hamming distance counts the number of bits that differ between two binary sequences. In both tasks, we use the binarised sequence of all the sensor and effector values.

[3]This work has been implemented within the Sferes$_{v2}$ framework [27]. The source code is available at `http://www.isir.fr/evorob_db`

5. RESULTS

The following results are investigated: 1) the evolution of the GAb on Ω_{valid} for the best controllers obtained with each method (Figures 5 and 7); 2) the GAb on Ω_{valid} and Ω_{test} of the best-of-run individual (Figures 4 and 6). All the comparisons between the methods are made with Wilcoxon rank-sum tests.

5.1 Task 1: navigation in a T-maze

For the *ProGAb* and *Fit_only* approaches, the maximum fitness on Ω_{train} is reached quickly, with respective median values of 55000 and 71000 evaluations (results not pictured). The best controllers obtained with *Fit_only* achieve a median of successful contexts on Ω_{valid} of 3 out of 180, while a median of 170 successful contexts is observed for *ProGAb*. Such results highlight that reaching the maximal performance on only one context is not sufficient to obtain general controllers, as it does not give any information on the GAb on Ω_{valid}.

Performing a lot of evaluations is not necessary relevant: there is no significant difference between the GAb on Ω_{valid} after 1.9 and 3.8 millions of evaluations for all the methods (p-values > 0.35).

Increasing the number of contexts in Ω_{train} improves the GAb: *Jakobi* and *FRI* perform better than *Fit_only* (p-values $< 10^{-3}$). But assessing on a huge number of contexts requires a lot more evaluations: with a budget of 3.8 millions evaluations, *Eval_all* performs clearly worse than *FRI* or *Jakobi* (p-values $< 2 \cdot 10^{-5}$).

Randomly generating Ω_{train} from Ω_{valid} at each generation does not improve significantly the GAb on Ω_{valid}: the p-value obtained by comparing *FRI* and *Jakobi* is 0.15 after 3.8 millions of evaluations, with respective median values of 80 and 153.

Table 5: Successful runs and evaluations before convergence for the T-maze task. There is "convergence" when the best individual of the population achieves at least 150 out of the 180 contexts of Ω_{valid}.

Methods	Number of successful runs (out of 20)	Nb. of evaluations before convergence		
		median (10^6)	mean	sd
Fit_only	0	.	.	.
Eval_all	1	2.59	2.59	0
FRI	6	1.75	1.86	0.95
Jakobi	12	1.21	1.64	1.12
ProGAb	14	0.42	0.88	0.93

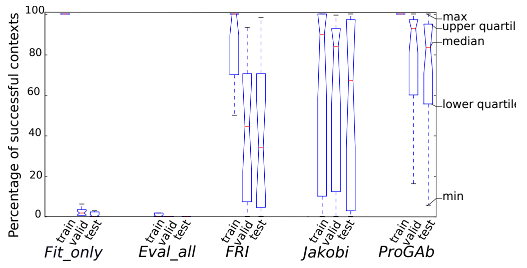


Figure 4: Percentage of contexts in Ω_{train}, Ω_{valid} and Ω_{test} achieved by the best controllers found with each method (20 runs) on the T-maze task.

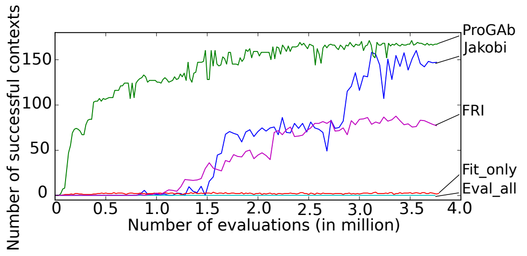


Figure 5: GAb on Ω_{valid} obtained with each method on the T-maze task (median over 20 runs). Every 10,000 evaluations, the best individual is assessed on Ω_{valid}. Ordinate represents the number of successful contexts ($|\Omega_{valid}| = 180$).

The *ProGAb* approach finds better general controllers than the other methods with a median of successful contexts on Ω_{valid} of 170 out of 180 (p-values $< 4{\cdot}10^{-2}$). Moreover, these controllers are found quicker than with the other methods. If we look at the runs where the individual's GAb is greater than 170 out of 180 contexts (Table 5), we see that *ProGAb* needs about two to three times fewer evaluations than *Jakobi* to converge, with respective median values of 0.43 and 1.20 millions of evaluations (p-value $= 3 \cdot 10^{-2}$). At last, there is no significant difference between the GAb obtained with the *ProGAb* approach on Ω_{valid} and on Ω_{test} (p-value $=$ 0.45), with respective median values of 170 and 150 successful contexts out of 180. It emphasizes the effective GAb of the controllers obtained with *ProGAb*.

5.2 Task 2: ball-collecting task

The Figure 6 shows the average number of balls collected by the best controllers on 20 runs obtained with each method on the three sets. The *Fit_only* approach does not always reach the maximal performance on Ω_{train} (the four balls are collected in only 13 runs out of 20). It is consistent with previous experiments on this setup [12] and reflects the complexity of the ball-collecting task.

The *ProGAb* approach finds controllers with significantly better GAb than the four other approaches, on Ω_{valid} (p-value $< 10^{-2}$) as well as on Ω_{test} (p-value $< 10^{-2}$). Moreover, according to Figure 6, the GAb on Ω_{valid} (median $=$ 2.63 balls) and on Ω_{test} (median $=$ 2.23 balls) are not significantly different (p-value $=$ 0.11) for this approach. Contrariwise, the GAb significantly decreases for *Jakobi*'s method, as the median drops from 2.13 on Ω_{valid} to 0.74 on Ω_{test} (p-value $< 5 \cdot 10^{-3}$).

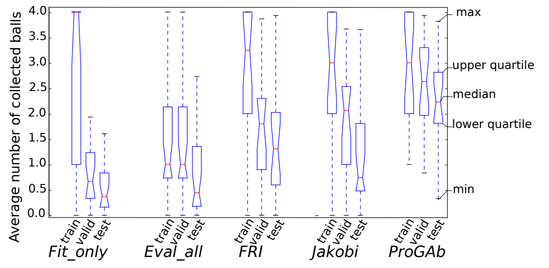


Figure 6: Average number of balls collected on Ω_{train}, Ω_{valid}, Ω_{test} by the best controllers found with each method on the ball-collecting task (20 runs).

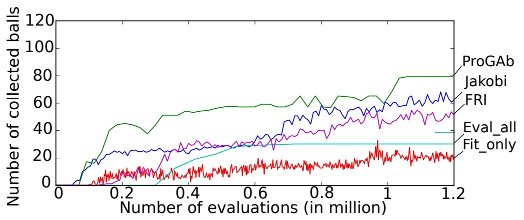


Figure 7: GAb on Ω_{valid} obtained with each method on the ball-collecting task (median over 20 runs). Every 10,000 evaluations, the best individual is assessed on Ω_{valid}. Ordinate represents the number of collected balls ($max = 4 \times |\Omega_{valid}| = 120$).

Several conclusions can be drawn from the results obtained on both tasks:

- evaluating with a single context or a few fixed contexts does not lead to general controllers;
- changing the training set from a generation to another can lead to controllers with a good GAb if the number of evaluations is sufficently high;
- the *ProGAb* approach obtains significantly better general controllers, while requiring fewer evaluations.

6. CONCLUSION

This paper proposes the *ProGAb* approach wich promotes the generalisation abilities (GAb) of controllers. It is based on the three data sets methodology used in supervised machine learning and relies on a surrogate model for approximating the GAb of the controllers in order to cope with computational cost issues.

In the first experiment where a robot must turn to the correct side of a T-maze according to a previously encountered stimulus, *ProGAb* converges quicker than the other methods. With the more complex ball-collecting task, where a robot must explore an arena to find balls and take them to a basket, *ProGAb* is both quicker to converge and provides controllers with better generalisation abilities.

Based on these successful results, our approach appears to be a simple and relevant method to efficiently evolve controllers with good generalisation abilities.

7. ACKNOWLEDGEMENT

This project was funded by the ANR-09-EMER-005-01 project.

8. REFERENCES

[1] E. Alpaydin. *Introduction to machine learning.* MIT Press, 2004.

[2] R. Barate and A. Manzanera. Generalization performance of vision based controllers for mobile robots evolved with genetic programming. *Proc. of GECCO*, 2008.

[3] A. Berlanga, A. Sanchis, P. Isasi, and J. Molina. Neural Network Controller against Environment: A Coevolutive approach to Generalize Robot Navigation Behavior. *Journal of Intelligent and Robotic Systems*, pages 139–166, 2002.

[4] T. Bersano-Begey and J. Daida. A discussion on generality and robustness and a framework for fitness set construction in genetic programming to promote robustness. In *Proc. of GP*, 1997.

[5] M. Birattari, T. Stutzle, L. Paquete, and K. Varrentrapp. A racing algorithm for configuring metaheuristics. In *Proc. of GECCO*, 2002.

[6] J. Bongard. Morphological change in machines accelerates the evolution of robust behavior. *PNAS*, 108(4):1234–1239, 2011.

[7] J. Bongard and G. Hornby. Guarding against premature convergence while accelerating evolutionary search. *Proc. of GECCO*, 2010.

[8] T. S. Braver, J. D. Cohen, and D. Servan-Schreiber. A computational model of prefrontal cortex function. *NIPS*, pages 141–148, 1995.

[9] L. T. Bui, H. Abbass, and J. Branke. Multiobjective optimization for dynamic environments. *Proc. of IEEE CEC*, 2005.

[10] K. Deb. *Multi-objective optimization using evolutionary algorithms.* John Wiley and Sons, 2001.

[11] E. Di Paolo. Homeostatic adaptation to inversion of the visual field and other sensorimotor disruptions. In *From Animals to Animats 6: Proc. of SAB*, 2000.

[12] S. Doncieux and J.-B. Mouret. Behavioral diversity measures for Evolutionary Robotics. *Proc. of IEEE CEC*, 2010.

[13] S. Doncieux, J.-B. Mouret, N. Bredeche, and V. Padois. *Evolutionary Robotics: Exploring New Horizons*, pages 3–25. Springer, 2011.

[14] D. Floreano and J. Urzelai. Evolution of plastic control networks. *Autonomous Robots*, 11(3):311–317, 2001.

[15] C. Gagné and M. Schoenauer. Genetic programming, validation sets, and parsimony pressure. *Genetic Programming*, pages 109–120, 2006.

[16] V. Heidrich-Meisner and C. Igel. Hoeffding and Bernstein races for selecting policies in evolutionary direct policy search. *Proc. of ICML*, 2009.

[17] T. Hemker, H. Sakamoto, M. Stelzer, and O. von Stryk. Hardware-in-the-loop optimization of the walking speed of a humanoid robot. In *Proc. of CLAWAR*, 2006.

[18] N. Jakobi. Evolutionary robotics and the radical envelope-of-noise hypothesis. *Adaptive Behavior*, 6(2):325–368, 1997.

[19] N. Jakobi, P. Husbands, and I. Harvey. Noise and the reality gap: The use of simulation in evolutionary robotics. *Advances in Artificial Life*, pages 704–720, 1995.

[20] T. Kondo, A. Ishiguro, S. Tokura, Y. Uchikawa, and P. Eggenberger. Realization of robust controllers in evolutionary robotics: a dynamically-rearranging neural network approach. *Proc. of IEEE CEC*, 1999.

[21] S. Koos, J.-B. Mouret, and S. Doncieux. Crossing the reality gap in evolutionary robotics by promoting transferable controllers. In *Proc. of GECCO*. ACM, 2010.

[22] J. Lehman and K. O. Stanley. Abandoning Objectives: Evolution through the Search for Novelty Alone. *Evolutionary Computation*, pages 1–34, 2010.

[23] J. H. R. Maes, B. M. Bouwman, and J. M. H. Vossen. Effects of d-amphetamine on the performance of rats in an animal analogue of the AX-CPT. *Journal of Psychopharmacology*, 15(1):23–28, 2001.

[24] J. Meyer, P. Husbands, and I. Harvey. Evolutionary robotics: A survey of applications and problems. *Evolutionary Robotics*, pages 1–22, 1998.

[25] O. Miglino, H. H. Lund, and S. Nolfi. Evolving mobile robots in simulated and real environments. *Artificial Life*, 2(4):417–434, 1995.

[26] J.-B. Mouret and S. Doncieux. Using behavioral exploration objectives to solve deceptive problems in neuro-evolution. *Proc. of GECCO*, 2009.

[27] J.-B. Mouret and S. Doncieux. Sferes_v2: Evolvin' in the multi-core world. In *Proc. of IEEE CEC*, 2010.

[28] T. G. Mueller, N. B. Pusuluri, K. K. Mathias, P. L. Cornelius, R. I. Barnhisel, and S. A. Shearer. Map quality for ordinary kriging and inverse distance weighted interpolation. *SSSAJ*, 68(6):2042–2047, 2004.

[29] S. Nolfi and D. Floreano. Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines. *MIT Press*, 26(3), 2000.

[30] L. Panait. Methods for evolving robust programs. *Proc. of GECCO*, 2003.

[31] T. Pinville and S. Doncieux. Automatic Synthesis of Working memory neural networks with neuroevolution methods. In *Neurocomp 2010*, 2010.

[32] A. Ratle. Accelerating the convergence of evolutionary algorithms by fitness landscape approximation. In *Proc. of PPSN*, 1998.

[33] C. Reynolds. Evolution of corridor following behavior in a noisy world. In *From Animals to Animats 3: Proc. of SAB*, 1994.

[34] R. M. Rylatt and C. A. Czarnecki. Embedding Connectionist Autonomous Agents in Time: The 'Road Sign Problem'. *Neural Processing Letters*, 12(2):145–158, 2000.

[35] D. Shepard. A two-dimensional interpolation function for irregularly-spaced data. In *Proc. of the ACM national conference*, 1968.

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

[37] A. Voutchkov and I. Keane. Multi-objective optimization using surrogates. *Computational Intelligence in Optimization*, pages 155–175, 2010.

[38] T. Ziemke and M. Thieme. Neuromodulation of Reactive Sensorimotor Mappings as a Short-Term Memory Mechanism in Delayed Response Tasks. *Adaptive Behavior*, 10(3-4):185–199, 2002.