Influence of Selection Pressure in Online, Distributed Evolutionary Robotics

Iñaki Fernández Pérez, Amine Boumaza, François Charpillet

Influence of Selection Pressure in Online, Distributed Evolutionary Robotics

Iñaki Fernández Pérez Amine Boumaza François Charpillet

Inria, Villers-lès-Nancy, F-54600, France
CNRS, Loria, UMR 7503, Vandœuvre-lès-Nancy, F-54500, France
Université de Lorraine, Loria, UMR 7503, Vandœuvre-lès-Nancy, F-54500, France
firstname.lastname@loria.fr

Résumé

L'effet de la pression à la sélection sur l'évolution dans des algorithmes évolutionnaires (AEs) centralisés est relativement bien compris. La pression à la sélection pousse l'évolution vers des individus plus performants. Cependant, les AEs distribués en robotique évolutionnaire diffèrent du fait que la population est distribuée sur tous les agents et qu'il n'existe pas de vision globale de tous les individus.

Dans cet article, nous analysons l'influence de la pression à la sélection dans un tel cadre distribué. Nous proposons une version de mEDEA qui introduit une pression à la sélection, et nous évaluons son effet sur deux tâches multi-robot : navigation avec évitement d'obstacles et collecte d'objets. Nos expériences montrent que même des légères pressions à la sélection mènent à des bonnes performances, et que la performance augmente avec la pression à la sélection. Ceci s'oppose aux approches centralisées, où une plus faible pression à la sélection est en général préférable afin d'éviter de stagner dans des optima locaux.

Mots Clefs

Evolutionary Robotics, Artificial Neural Networks, Selection pressures, Multi-robot learning.

Abstract

The effect of selection pressure on evolution in centralized evolutionary algorithms (EA's) is relatively well understood. Selection pressure pushes evolution toward better performing individuals. However, distributed EA's in an Evolutionary Robotics (ER) context differ w.r.t. selection in that the population is distributed across the agents, and a global vision of all the individuals is not available.

In this paper, we analyze the influence of selection pressure in such a distributed context. We propose a version of mEDEA that adds a selection pressure, and evaluate its effect on two multi-robot tasks: navigation and obstacle-avoidance, and collective foraging. Experiments show that even small intensities of selection pressure lead to good performances, and that performance increases with selection pressure. This is opposed to the lower selection pressure that is usually preferred in centralized approaches

to avoid stagnating in local optima.

1 Introduction

One of the goals of evolutionary robotics (ER) [Nolfi and Floreano, 2000] is to automatically build robotic agents' controllers using evolutionary algorithms (EA) [Eiben and Smith, 2003]. In ER contexts, EA's are usually seen as a tool to optimize the controller of one or more agents regarding an objective function that measures agent performance (fitness). Once the required behavior is learned and the controller optimized, the agents are deployed and their behavior exploited, while their controllers remain fixed, *i.e.* there is no further evolution. This is known as offline evolution.

On the other hand, in online evolution [Watson et al., 2002] behavior learning takes place at the same time as the execution of the task at hand. As such, optimization is a continuous process, *i.e.* agents are constantly exploring new behaviors and learning to react to new environmental conditions, which is usually referred to as adaptation.

In our work, we focus on on-line distributed evolution of agent controllers. Our research concerns learning agent behaviors in a distributed context where multiple agents adapt their controllers to environmental conditions independently. In this sense, this approach finds many ties with Artificial Life, where the objective is to design autonomous organisms that adapt to their environment. Agents can locally communicate with each other, and no agent has a view of the entire population. Online distributed evolution can be thought as distributing an EA on the team of agents. Standard evolutionary operators (mutation, crossover etc.) are implemented on the agents, and local communication allows for the spread of genomes in the team of agents.

In EA's, selection operators drive evolution toward better performing individuals by regulating the intensity of selection pressure to learn to solve the given task. Selection operators and their influence on evolutionary dynamics have been extensively studied in offline contexts [Eiben and Smith, 2003]. In this work, we analyze their impact in an online distributed setup, where evolutionary dynamics are different from the offline case: selection is local, it acts over partial populations, and fitness values on which selection is performed are not reliable, due to different evaluation conditions. As our experiments show, in this context a strong selection pressure produces the best results. This is opposed to a lower selection pressure that is preferred in offline centralized contexts to maintain diversity in the population and avoid premature convergence. Our results suggest that, in distributed ER algorithms, diversity is implicitly maintained by the fact that there exist disjoint subpopulations across the agents of the team.

Online evolution of agent controllers has been addressed by several authors in different contexts: adaptation to varying conditions [Dinu et al., 2013], automatic parameter configuration [Eiben et al., 2010], light-following and navigation [Karafotias et al., 2011, Silva et al., 2014]. Some of these works are described in the next section. The authors use selection operators that induce different degrees of selection pressure to drive evolution. Here, we evaluate the influence of selection pressure on the performances obtained when learning two multi-agent tasks: navigation and foraging.

2 Selection in Distributed ER

A common characteristic of on-line distributed ER algorithms is that each agent has one controller at a time, that it executes (the active controller), and locally spreads altered

copies of this controller to other agents. In this sense, agents have only a partial view of the population in the swarm (a local repository). Fitness assignment or evaluation of individual genomes is performed by the agents themselves and is thus noisy, as different agents evaluate their active controllers in different conditions. Selection takes place when the active controller is to be replaced by a new one from the repository.

PGTA (Probabilistic Gene Transfer Algorithm), introduced by [Watson et al., 2002], is usually cited as the first algorithm to evolve agent controllers in an online distributed manner. It evolves the weights of neural controllers, and agents locally exchange parts (genes) of their respective genomes when they cross each other. The rate at which an agent spreads its genes is proportional to its performance, and the rate at which an agent accepts received genes is inversely proportional to its performance. In this sense, selection pressure is induced in that fit agents transfer their genes to unfit ones.

In mEDEA (minimal Environment-driven Distributed Evolutionary Algorithm, [Bredeche and Montanier, 2010]), the authors study evolutionary adaptation with an implicit fitness, *i.e.* without a task-driven fitness function. Local selection is performed randomly by a given agent upon the genomes gathered during the execution of the agent's controller. As such, successful genomes in mEDEA are those that spread the most over the agents. The spread of a genome is maximized by increasing mating opportunities and minimizing the risk for the agent carrying it.

The authors show that agents learn to navigate and avoid obstacles, which allows them to better spread their genomes when local selection is random. This work shows that environmental selection pressure alone can maintain a certain level of adaptation in a team of robotic agents. A modified version of this algorithm is used in this work and is detailed in the next section.

[Noskov et al., 2013] proposed MONEE (Multi-Objective aNd open-Ended Evolution), an extension to mEDEA adding a task-driven pressure. The algorithm was designed for dealing with multiple objectives, and implements a mechanism (called market) for balancing the distribution of different tasks among the population of agents, so as not to disregard harder tasks w.r.t. simpler ones. In this sense, a selection pressure toward task diversity is induced. Their experiments show that MONEE is capable of improving mEDEA's performances in a collective foraging task, in which agents have to collect items of several kinds.

The authors show that the agents are able to adapt to the environment (as in mEDEA), and to forage different kinds of items, *i.e.* optimize the task-solving behavior. The algorithm uses an explicit fitness function in order to guide the search toward better performing solutions. In their paper, the agent's next controller is selected using rank-based selection from the agent's list of genomes. The authors argue that when a specific task is to be addressed, a task-driven selection pressure is necessary. This idea is discussed in the remainder of this paper.

In other related algorithms, selection is applied differently. For instance, odNEAT [Silva et al., 2014] (a distributed version of NEAT [Stanley and Miikkulainen, 2002] that evolves both the topology and weights of neural networks) maintains local populations structured in niches w.r.t. topology similarity. The controllers share the fitness of their respective niches, which consists in the average fitness divided by the number of individuals in the niche. By doing this, a diversity of topologies is maintained, since a smaller niche whose individuals have a lower fitness have a chance to survive and potentially improve, given that its shared fitness is divided by a lower factor.

In EDEA (Embodied Distributed Evolutionary Algorithm) [Karafotias et al., 2011], selection pressure is applied by selecting and recombining a received genome, x', with

the current active one, x, with a probability $\frac{f(x')}{s_c \times f(x)}$, where s_c is a factor regulating selection pressure. The higher s_c is, the lower the probability of selecting a received genome.

In these works, authors used standard selection operators from evolutionary computation in distributed ER contexts. Nevertheless, it is unclear if similar evolutionary dynamics can be expected as when they are used in an offline centralized ER setup, given that online distributed contexts have different properties: selection is performed locally at the agent level, and over the genomes of the other agents it had the opportunity to meet. Furthermore, as it is common in many ER setups, in online distributed evolution, fitness evaluation is intrinsically noisy, since agents evaluate their controllers in different conditions, and this may strongly influence their performance and resulting behaviors. In this sense, a question we study here is: does it still make sense to use selection in distributed contexts? And if yes, what intensity of selection pressure is adequate?

In this paper, we study the influence of four different selection methods inducing different intensities of selection pressure. We apply these methods in a version of mEDEA adding task-driven selection, and evaluate their impact on two different multi-robot tasks.

3 Algorithm and selection operators

In this section, we describe the algorithm used in our experiments (Alg. 1), a variant of mEDEA. The algorithm is independently executed by all the agents. Each agent has a single active controller, which is initialized randomly.

The main difference with the original mEDEA is that, in our variant an agent alternates between two phases, as proposed in [Noskov et al., 2013]: an *evaluation phase* lasting T_{e}, in which the agent runs, evaluates and locally broadcasts its controller to listening agents, and a *listening phase* lasting T_{l}, in which the agent stops and listens to incoming genomes sent by nearby agents. For different agents, phases are desynchronized, so agents in the evaluation phase are able to spread their genomes to other agents that are in the listening phase.

The agent's controller is executed and evaluated during the evaluation phase. At each time-step the agent reads its sensors, passes them as input to its controller and computes the motors' outputs. It also updates the fitness value of the controller depending on the result of its actions, and broadcasts the genome corresponding to its controller and its current fitness value to listening robots in the vicinity.

At the end of the T_{e} evaluation steps, the listening phase begins. At this point, the agent stops for T_{l} time-steps and listens for genomes from nearby passing agents (agents that are evaluating their controllers). Since the genomes are broadcast along with their respective fitness values, at the end of this phase, an agent has a local list of genomes and fitnesses, or local population. In our variant, unlike original mEDEA, an agent's current genome is added to its local population, to ensure that all agents always have at least one genome in their respective populations. This can occur if an agent is isolated during its listening phase and thus does not receive any genome. In the original mEDEA, agents stay inactive until they receive a new genome.

Once the listening phase is finished, the agent loads a new controller for the next evaluation phase. This is done by selecting a genome from the local population based on its fitness, and using one of the selection operators presented below. The selected genome is mutated and becomes the agent's active controller. In our experiments,

Algorithm 1 mEDEA variant

$g_a := random()$
while true **do**
  $\mathbf{l} := \emptyset$
  // Evaluation phase
  for $t = 1$ to T_e **do**
    $exec(g_a)$
    $broadcast(g_a)$
  end for
  // Listening phase
  for $t = 1$ to T_l **do**
    $\mathbf{l} := \mathbf{l} \bigcup listen()$
  end for
  $\mathbf{l} := \mathbf{l} \bigcup \{g_a\}$
  $selected := select(\mathbf{l})$
  $g_a := mutate(selected)$
end while

mutation is performed by adding a normal random variable with mean 0 and variance σ^2 to each connection weight.

At this point, before the new controller's evaluation phase begins, the local population is emptied. As such, selection is performed on a list of genomes gathered by the agent during the previous listening phase. One complete iteration of the algorithm (evaluation and listening phase) is referred to as one generation.

Here, we study four selection operators, each inducing a different intensity of task-driven selection pressure, from the strongest to the lowest: *Best*, *Rank-based*, *Binary Tournament* and *Random* selection. *Best* always selects the genome with the highest fitness, while *Rank-based* assigns probabilities of selection proportional to the rank of each genome in the population once sorted w.r.t. fitness. *Binary Tournament* selection consists in drawing two genomes at random and selecting the best between them. Finally, *Random* selection picks a random genome from the local population, thus completely disregarding fitness values, as in mEDEA. The choice of these selection methods aims at giving a large span of intensities of selection pressure.

4 Experiments on selection pressure

Here, we compare the four presented selection methods on a set of experiments in simulation for two tasks, navigation with obstacle avoidance and collective foraging, which are two well-studied benchmark tasks in multi-robot setups. Our experiments were performed on the RoboRobo simulator [Bredeche et al., 2013].

4.1 Robots and tasks description

In all experiments, a team of 50 robotic agents is deployed in a square bounded environment containing static obstacles. Other agents are perceived as mobile obstacles.

All the agents in the team have the same physical properties, sensors and motors, and the only difference between them lays in the weights of their neural controllers. Agents have 8 obstacle proximity sensors, evenly spaced around the agent. 8 item

Table 1: Experimental settings.

Number of food items	150
Number of runs	30
Evolution length	~ 250 generations
T_e	$2000 - rand(0, 500)$ sim. steps
T_l	200 sim. steps
Mutation step-size	$\sigma = 0.5$

sensors are added for the foraging task. An item sensor measures the distance to the closest item in the direction and range of the sensor.

The agents' neural controllers are recurrent neural networks, where the inputs of the network are the activation values of all sensors, and the 2 outputs correspond to the translational and rotational velocities of the agent. The activation function of the output neurons is a hyperbolic tangent, taking values in $[-1, +1]$. Two bias connections (one for each output neuron) and 4 recurrent connections (previous speed and rotation to both outputs) are added. In total, 22 connection weights need to be optimized for the navigation task, and 38 for the foraging task. The genome of the controller is the vector of these weights. Table 1 summarizes the parameters we used in the experiments.

The navigation task consists in learning to move rapidly and minimizing turns, while avoiding static and mobile obstacles. Foraging requires agents to gather food items in the environment. Items are gathered when agents pass over them, and when an item is collected, it is replaced by another one at a random position.

The fitness function for navigation is defined after the one introduced in [Nolfi and Floreano, 2000]. An agent r computes its fitness at generation g as:

$$f_r^g = \sum_{t=1}^{T_e} v_t(t) \cdot (1 - |v_r(t)|) \cdot min(a_s(t)) \tag{1}$$

where $v_t(t)$, $v_r(t)$ and $a_s(t)$ are respectively the translational and rotational velocities at time t, and the vector of activations of the obstacle sensors of the agent at time-step t of its evaluation phase. As for the foraging task, the fitness is computed as the number of items collected by the agent during its evaluation phase.

Since in our work we want to study the performance of the entire team of agents, we define swarm fitness as the sum of the fitness of all agents at the end of each generation:

$$F_s(g) = \sum_{r \in swarm} f_r^g \tag{2}$$

4.2 Performance measures in online ER

Online evolving agents learn in an open-ended manner at the same time as they are performing the actual task. Consequently they are always exploring new solutions, and the best fitness ever reached may not be a reliable estimator of the quality of the algorithm, since a high best fitness only reflects a good performance at one point of evolution. Additionally, online fitness evaluation is noisy in essence, given the different conditions in which agents evaluate their controllers. Taking in account these issues, in [naki Fernández Pérez et al., 2014] we introduced four measures, that we use to analyze the influence of the intensity of selection pressure in our experiments. These

measures integrate swarm fitness information spanning over several generations, and are computed after evolution has finished in order to compare the selection operators. A pictorial description of these four measures is shown in Figure 1.

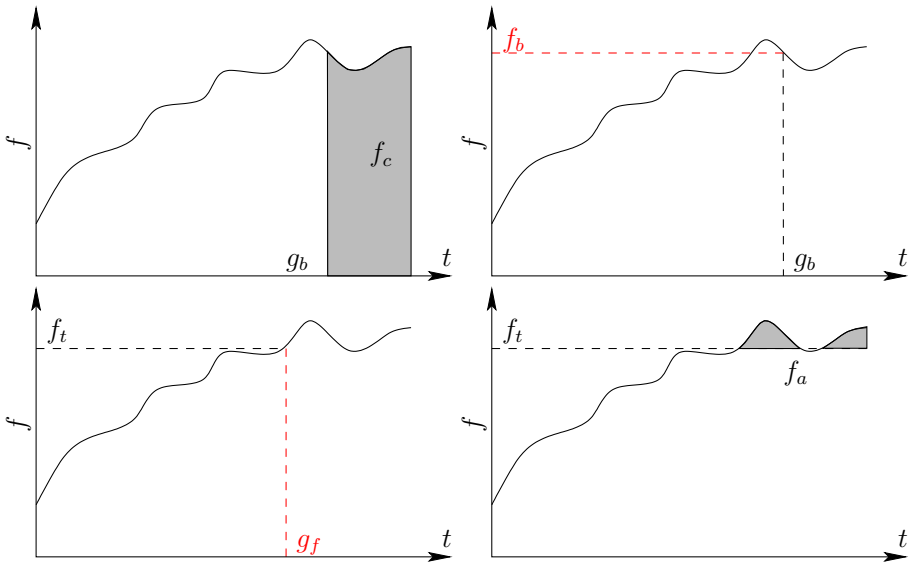

Figure 1: From left to right, top to bottom: f_c, f_b, g_f, f_a.

The aforementioned measures are the following. The *average accumulated swarm fitness* (f_c) is the average swarm fitness in the last generations (here, the last 8% generations). The *fixed budget swarm fitness* (f_b) is the swarm fitness reached at a certain generation (computational budget, here 92% of the evolution, *i.e.* the first generation considered in f_c). The *time to reach target* (g_f) is the first generation at which a predefined target fitness is reached, or the last generation if this level is never reached. Here, we fixed the target at 80% of the maximum fitness reached over all runs and all selection operators. Finally, the *accumulated fitness above target* (f_a) is the sum of all swarm fitness values above the same target value as for g_f.

These measures need to be considered in combination when comparing two experiments. For instance, f_c and f_b provide information of the level and stability of the performance reached by the agent team at the end of evolution. If they are close, the performance is stable. Also, g_f and f_a combined reflect how rapidly a target fitness value is attained, and by how much that level is exceeded.

4.3 Results and discussion

We launched 30 independent runs and measured F_s at each generation for each variant of the experiment (each selection operator in both tasks). The median F_s per generation over all runs is presented in Figures 2 (navigation) and 3 (foraging). The four performance measures are shown in Figure 4 (navigation), and in Figure 5 (foraging). For both tasks, we performed 99% confidence Mann-Whitney tests on the four measures, for all pairwise combinations between the four selection operators.

When observing Fig. 2 and Fig. 3, we notice that, in both tasks, a high fitness level is rapidly reached whenever there is a task-driven selection pressure, *i.e.* with *Best*, *Rank-*

based, or *Binary tournament* selection. Furthermore, the algorithm reaches similar levels of swarm fitness (median values). An exception can be noted for *Best* selection in the foraging task, which outperforms all other selection operators. However, if no selection pressure is induced at the agent level (*i.e. Random* selection), learning is much slower, and thus reaches lower levels in the allotted time.

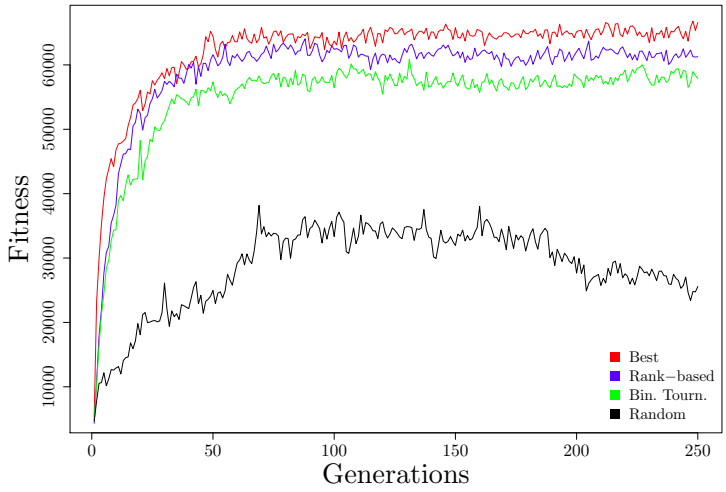

Figure 2: Median swarm fitness for the navigation task.

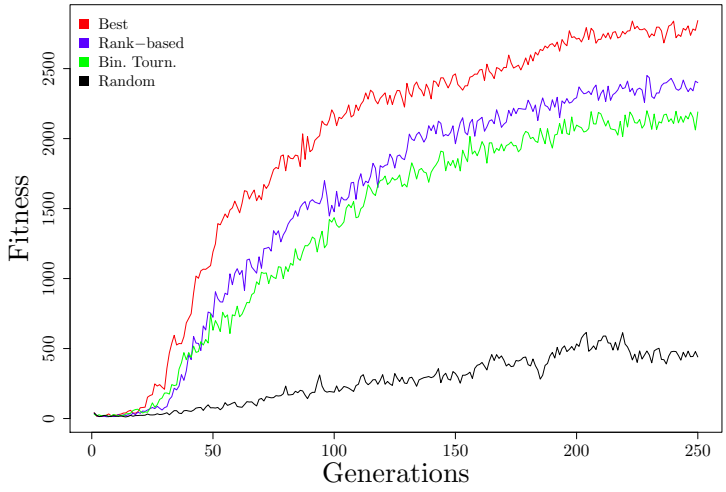

Figure 3: Same as in Figure 2 for the foraging task.

Even if the results achieved by *Random* are worse, agents still improve their performances in both tasks. This can be observed in the increasing trend in Fig. 2 and Fig. 3. This is expected for the navigation task, since environmental pressure leads to behaviors that increase mating opportunities by exploring the environment, thus improving the swarm fitness, as in mEDEA [Bredeche and Montanier, 2010].

As for the foraging task, when *Random* selection is applied the improvement is slower but still present. The explanation is that collecting items can be a byproduct of exploring the environment. Items are gathered by chance in this case, when agents move while trying to mate. Upon inspection of the evolved behaviors in the simulator, we noticed that, when selection pressure is present, agents move toward the food items, which means that evolution drove the agents to exploit the item sensors. However, without selection pressure (*Random*), there can not be a similar drive, which we confirmed by inspecting the simulator: agents were not attracted by food items for *Random* selection.

When analyzing the comparison measures we introduced above, the same trends are observed. Figure 4 (respectively Figure 5) presents the box and whiskers plots of the four measures for each selection method over the 30 runs for the navigation task (respectively the foraging task).

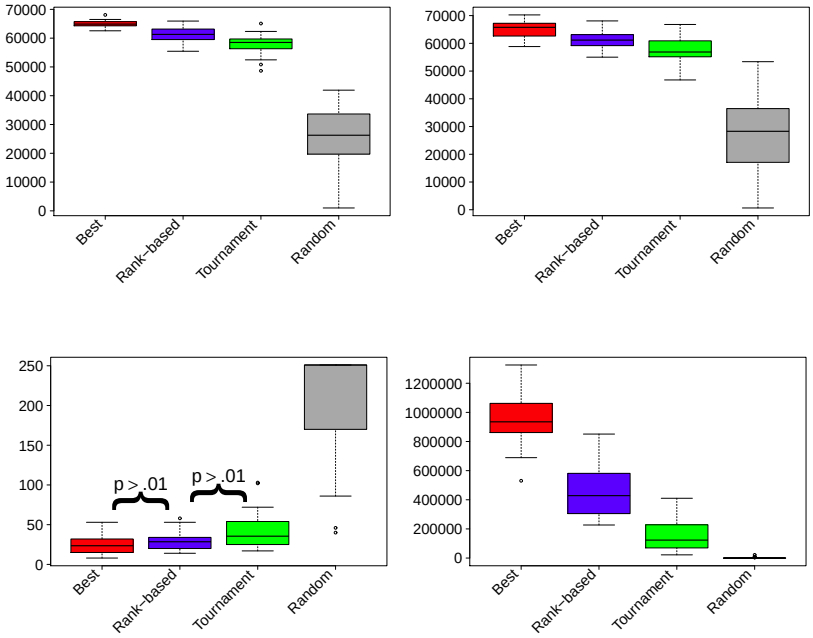

Figure 4: Performance measures on the navigation task. From left to right and top to bottom: f_c, f_b, g_f and f_a. The label $p > 0.01$ indicates no statistical difference.

For the navigation task, pairwise comparisons of the four measures result in a significant difference between all selection operators, except for the time to reach target (g_f) between *Best* and *Rank-based* ($p{-}value = 0.07$) and between *Rank-based* and *Binary tournament* ($p{-}value = 0.012$). We observe that *Best* reaches a higher swarm fitness than the other selection operators, and this level is maintained at the end of evolution, as indicated by f_c and f_b (upper left and right in the figure). The target fitness level is rapidly attained for the three operators with selection pressure, and there is not significant difference between *Best* and *Rank-based*, nor between *Rank-based* and *Binary Tournament* regarding g_f (lower left). Moreover, in the case of *Best*, the target level is not only reached but surpassed during the entire evolution, yielding much higher values of f_a than the rest of selection operators (lower right). However, this

is not the case for *Random* selection that leads to lower f_b and f_c (top), and does not reach the target fitness level on more than half of the runs (bottom).

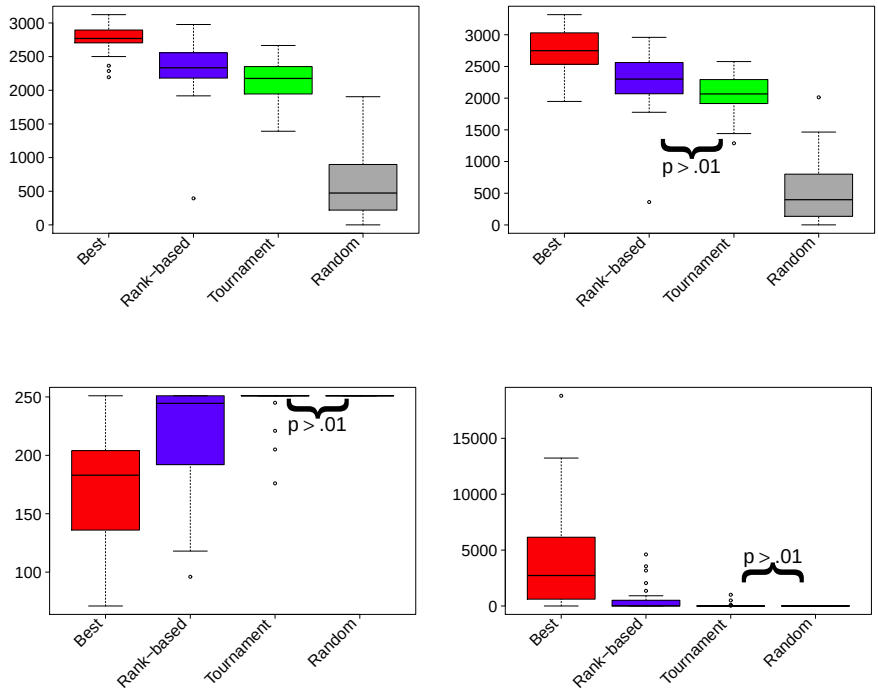

Figure 5: Same as in Figure 4 for the foraging task.

Concerning foraging, differences are significant for all pairwise comparisons, except between *Binary Tournament* and *Random* for the time to reach target, g_f, and the accumulated fitness above target, f_a ($p{-}value = 0.042$ in both cases). The reason behind this is that very few runs reached the required level, and thus g_f is the last generation and f_a is almost zero, for both selection operators. There is also no statistical difference between *Rank-based* and *Binary Tournament* on the fixed budget swarm fitness, f_b ($p{-}value = 0.011$). This means that *Binary Tournament* reaches a fitness at the given budget that is comparable to the one of *Rank-based*, but it is not able to maintain the level so effectively, since the difference on f_c between these two operators is significant.

On the foraging task, *Best* also leads to better results: a high swarm fitness is reached and maintained (f_b and f_c, upper left and right). It surpasses the required fitness level in almost all runs much faster and to a larger extent than *Rank-based*, that also manages to reach the target level for most runs (g_f, lower left), although by a lesser extent (f_a, lower right). The picture is different regarding *Tournament* and *Random*, which do not achieve the target fitness level for most runs (lower left and right).

To summarize, we can confirm that all task-driven selection pressures lead to much better results on both tasks compared to *Random* selection. Consequently, we may conclude that selection pressure has a positive impact on performances when solving a given task, *i.e.* when the goal is not only to achieve environmental adaptation as it was the original motivation of mEDEA. Moreover, statistical tests show a direct correla-

tion between selection pressure and the performances achieved on the two considered tasks. In other words, the stronger the selection pressure is, the better the performances reached by the team of agents.

It has been argued that in general, elitist approaches are not desirable in traditional EA's, and this also applies to traditional ER. The reason behind this is that elitist strategies can result in a premature convergence at local optima. This has been extensively studied, especially in non-convex optimization, where it is preferable to explicitly force a certain level of diversity in the population to allow evolution to escape local optima and deal with the exploration versus exploitation dilemma. As our experiments show, this requirement is perhaps not as strong in distributed ER. Since selection is performed at the agent level and over a fraction of the population, we might argue that these algorithms already maintain a certain diversity, given that subpopulations are distributed on the different agents. Investigating possible ties with other approaches in which separated subpopulations are evolved, *e.g.* spatially structured EA's [Tomassini, 2005] and island models [Alba and Tomassini, 2002], could provide more information on the dynamics of distributed evolution.

5 Conclusions and future work

This paper has studied the influence of different degrees of selection pressures in an online distributed context for agent behavior evolution. In distributed ER, the impact of selection pressures on evolution is unclear, given that selection is applied over partial populations and fitness values are noisy. Four selection operators inducing different degrees of selection pressure were compared on two tasks: navigation with obstacle avoidance and collective foraging. In our experiments, we show that even a small degree of selection pressure largely improves performances, and that the intensity of the selection operator positively correlates with the performances of the team of agents.

Foraging and navigation are arguably relatively simple tasks, and we deem interesting to study selection pressures on more complex and challenging ones, involving deceptive fitness landscapes. This could further clarify the impact of selection on evolution dynamics in distributed approaches to the evolution of agent behavior.

References

[Alba and Tomassini, 2002] Alba, E. and Tomassini, M. (2002). Parallelism and evolutionary algorithms. *Evolutionary Computation, IEEE Transactions on*, 6(5):443–462.

[Bredeche and Montanier, 2010] Bredeche, N. and Montanier, J.-M. (2010). Environment-driven Embodied Evolution in a Population of Autonomous Agents. In *PPSN 2010*, pages 290–299, Krakow, Poland.

[Bredeche et al., 2013] Bredeche, N., Montanier, J.-M., Weel, B., and Haasdijk, E. (2013). Roborobo! a fast robot simulator for swarm and collective robotics. *CoRR*, abs/1304.2888.

[Dinu et al., 2013] Dinu, C. M., Dimitrov, P., Weel, B., and Eiben, A. (2013). Self-adapting fitness evaluation times for on-line evolution of simulated robots. In *Proc. of GECCO'13*, pages 191–198. ACM.

[Eiben et al., 2010] Eiben, A., Karafotias, G., and Haasdijk, E. (2010). Self-adaptive mutation in on-line, on-board evolutionary robotics. In *Fourth IEEE Int. Conf. on Self-Adaptive and Self-Organizing Systems Workshop (SASOW), 2010*, pages 147–152. IEEE.

[Eiben and Smith, 2003] Eiben, A. E. and Smith, J. E. (2003). *Introduction to Evolutionary Computing*. Springer.

[Karafotias et al., 2011] Karafotias, G., Haasdijk, E., and Eiben, A. E. (2011). An algorithm for distributed on-line, on-board evolutionary robotics. In *Proc. of GECCO '11*, pages 171–178, New York. ACM.

[naki Fernández Pérez et al., 2014] naki Fernández Pérez, I., Boumaza, A., and Charpillet, F. (2014). Comparison of selection methods in on-line distributed evolutionary robotics. In *Proceedings of the Int. Conf. on the Synthesis and Simulation of Living Systems (Alife'14)*, pages 282–289, New York. MIT Press.

[Nolfi and Floreano, 2000] Nolfi, S. and Floreano, D. (2000). *Evolutionary Robotics*. MIT Press.

[Noskov et al., 2013] Noskov, N., Haasdijk, E., Weel, B., and Eiben, A. (2013). Monee: Using parental investment to combine open-ended and task-driven evolution. In Esparcia-Alcázar, editor, *App. of Evol. Comput.*, volume 7835 of *LNCS*. Springer Berlin.

[Silva et al., 2014] Silva, F., Urbano, P., Correia, L., and Christensen, A. L. (2014). odneat: An algorithm for decentralised online evolution of robotic controllers. *Evol. Comput.* MIT Press. Available online.

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

[Tomassini, 2005] Tomassini, M. (2005). *Spatially structured evolutionary algorithms*. Springer Berlin.

[Watson et al., 2002] Watson, R. A., Ficici, S. G., and Pollack, J. B. (2002). Embodied evolution: Distributing an evolutionary algorithm in a population of robots. *Robotics and Autonomous Syst.* Elsevier.