Introducing Weighted Intermediate Recombination in On-line Collective Robotics, the (μ/μ W , 1)-On-line EEA

Amine Boumaza

Introducing Weighted Intermediate Recombination in On-line Collective Robotics, the $(\mu/\mu_{\mathrm{W}}, 1)$-On-line EEA

Amine Boumaza
Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France
amine.boumaza@loria.fr

To appear in the proceding of EvoApp2019

Abstract

Weighted intermediate recombination has been proven very useful in evolution strategies. We propose here to use it in the case of on-line embodied evolutionary algorithms. With this recombination scheme, solutions at the local populations are recombined using a weighted average that favors fitter solutions to produce a new solution. We describe the newly proposed algorithm which we dubbed $(\mu/\mu_{\mathrm{W}}, 1)$-On-line EEA, and assess it performance on two swarm robotics benchmarks while comparing the results to other existing algorithms. The experiments show that the recombination scheme is very beneficial on these problems.

1 Introduction

Embodied evolutionary robotics (EER) [3], aims to learn collective behaviors for a swarms of agents, where evolution is distributed on the agents that adapt on-line to the task [13]. Each agent runs an EA onboard and exchange genetic material with other agent when they meet. Selection and variation are performed locally on the agents and successful genes, whose offspring survive throughout many generations, are those that adapt to the task and also maximize mating opportunities while minimizing the risk for their vehicles.

In this paper we propose the $(\mu/\mu_{\mathrm{W}}, 1)$-On-line EEA which adapts the well known weighted recombination scheme from evolution strategies (ES) to the on-line EER setting. It has been shown that recombination allows to speed up progress of ES and improves robustness against selection errors [2]. These properties are the result of the *genetic repair principal* (GR) which reduces the effect of the harmful components of mutations as a result of the averaging process. Furthermore, [1] extends these conclusions in the case of noisy fitness environments and argues that combined with higher mutation steps, recombination can reduce the signal to noise ratio in the evaluation.

Our main motivation is to investigate if these properties can also benefit EER algorithms. We study the impact of this recombination scheme and show that when correctly designed, it improves significantly the results of the algorithm as the experiments suggests on two different learning tasks.

2 background

Recombination or as commonly know as crossover is not new in on-line EER, different authors have proposed implementations with different form of crossover. We briefly review some of that work in the following.

The Probabilistic Gene Transfer Algorithm (PGTA) [13], is commonly cited as the first implementation of a distributed on-line EER algorithm. As the name suggests, recombination is implemented at a gene level. Agents broadcast randomly sampled genes from their genomes and when they receive genes from other agents, they replace the corresponding gene with a probability. The rate at which the agents broadcast their genes is proportional to their fitness[1] and conversely, the rate at which they accept a received gene is inversely proportional to their energy level. This way, selection pressure is introduced in that fit agents transmit their genes to unfit ones.

In the Embodied Evolutionary Algorithm (EEA) [9], crossover is performed between two genomes. Agents select a genome from their local population (received from potential mates) using a binary tournament. This selected genome is then recombined with the current active genome with a probability proportional to the ratio of its fitness and that of the active genome. The newly created genome is the average of both genomes. In this case, crossover is more probable when the selected genomes comes from a fitter agent.

The above mentioned articles implemented the EA on a fixed topology neuro-controller where the competing conventions problem [10] does not rise. It is also worth mentioning that there exists, although fewer, implementation of EER that proposed evolving topology. These implementations adapt the innovation marking introduced in the Neuro-Evolution of Augmenting Topologies algorithm (NEAT) [12], to the distributed case. In this case, innovations appear locally in the population and are not know to all agents who must order them correctly before performing a crossover [11, 7].

For a more complete review of the existing work, we recommend the recent review article [3].

3 The $(\mu, 1)$-On-line EEA

The main inspiration of the $(\mu, 1)$-On-line EEA is the original version of mEDEA [4] to which with we add a selection operator that we will describe later (Algorithm 1). The algorithm considers a swarm of λ mobile agents a^j with $j = 1, \ldots, \lambda$ each executing a neuro-controller whose parameters are x^j (the active

[1] The authors use a virtual energy level in place of fitness.

genome). Each agent maintains a list L^j, initially empty, in which it stores other genomes that it receives from other agents.

At each time step $t < t_{\max}$, an agent executes its active controller and broadcasts its genome within a limited range. In parallel, it listens for genomes originating from other agents, and when a genome is received (a mating event), it is stored in the agent's list L^j (it's local population). This procedure is executed in parallel on all agents during $t_{\max}$ steps, the evaluation period of one generation.

At the end of a generation, the agent selects a genome from its list L^j (the selection step), and replaces its active genome with a mutated copy of the selected one. The list is then emptied and a new generation begins.

In the event where an agent had no mating opportunities and finishes its evaluation period with an empty list $L^j = \emptyset$, it becomes inactive ; a state during which the agent is motionless. During this period, the inactive agent continues to listen for incoming genomes from other agents, and once $L^j \neq \emptyset$ the agent becomes active again at the beginning of the next generation.

The number of genomes the agents collects $\mu^j = |L^j|$ $(0 \leq \mu^j \leq \lambda)$ is conditioned by it's mating encounters. Since the communication range is limited, agents that travel long distances will increase their probability of mating. We should note that mating encounters allow agents to spread their active genome and to collect new genetic material. The algorithm stores only one copy of the same genome[2] if it is received more than once.

The $(\mu, 1)$-On-line EEA can be viewed as λ instances of $(|L^j|, 1)$-EA running independently in parallel, where for each instance, $L^j \subseteq \{x^1, \ldots, x^\lambda\}$, *i.e.* each local population is a subset of the set of all active genomes of the generation. Furthermore, the sizes of the individual local populations μ^j are not constant although bounded by λ and depend on the number of mating events (possibly none) the agent had in the previous generation.

3.1 Implementation details

The original mEDEA [4] was introduced in open-ended environments without specifying a task for the agents to solve. However, when it is applied in a task-driven scenario, we consider an objective or fitness function. In this case selection (line 16 in Algorithm 1) is based on the fitness of the genomes in the individual lists. In this work, the genome is a vector $x \in \mathbb{R}^N$, which represents the weights of the neuro-controller and $f : \mathbb{R}^N \to \mathbb{R}$ is the fitness function. Only the weights undergo evolution (fixed-topology).

In addition to adding a selection operator, there is a significant difference between mEDEA and Algorithm 1 in that we don not consider a listening phase. Agents broadcast if they are active and listen all the time, whereas in mEDEA, agents must be in a specific listening state to record incoming genetic material. This and the added maturation age, described bellow, show that in practice the difference in the results is not significant.

Furthermore, since the EA runs independently on each agent, fitness values are

[2]The term "same" is here used in the sense "originating from the same agent".

Algorithm 1: $(\mu, 1)$-On-line EEA

```
1  for 1 ≤ j ≤ λ in parallel do
2      x^j ← random()
3      a^j is active
4  repeat
5      for 1 ≤ j ≤ λ in parallel do
6          t ← 0, f^j = 0, L^j ← ∅
7          while t < t_max do
8              t ← t + 1
9              if a^j is active then
10                 exec(x^j)
11                 update(f^j)
12                 if t > τ t_max then
13                     broadcast(x^j, f^j)
14             L^j ← L^j ∪ listen()
15         if L^j ≠ ∅ then
16             x^j ← mutate(select(L^j))
17         else a^j is not active
18 until termination condition met
```

assigned to the individual agents based on their performance with regard to the given task. These values are continuously updated during the agent's lifetime (line 11). Each agent stores this value internally and broadcasts it along with its genomes during mating events. Agents on the receiving end store the genome and its fitness values in their lists. Finally, if an agent receives an already seen genome, it updates the genome's fitness as it is the most up to date value. Furthermore, to ensure that genomes are transmitted with accurate fitness values, a *maturation age* is required of the agent before broadcasting [9]. Finally, we should emphasize that fitness evaluation are intrinsically noisy since measurements are performed in varying circumstances.

In the following, we will note $L = \{x_1 \dots x_\mu\}$ the local population on some agent, and we consider a maximization scenario. Depending on the treatment tested, we apply a different selection scheme. On the one hand, for the case were there is no recombination, the next genome is the winner of a tournament selection. On the other hand, when recombination is considered, the next active genome is the result of the weighted recombination on the local population. After either of these steps, the newly generated genome, which we note $\bar{x}$, is mutated. Mutation is Gaussian with a fixed step size $\sigma \in \mathbb{R}$:

$$x := \bar{x} + \sigma^2 \times \mathcal{N}(1, 0) \tag{1}$$

3.1.1 Selection in $(\mu, 1)$-On-line EEA

We use an adapted k-tournament selection scheme which randomly draws $k > 0$ solutions from the local population of μ parent solutions L and returns the best

solution among the parents. The tournament size k, sets the selection pressure of the procedure, and is usually a parameter of the EA. When $k = 1$, the procedure selects a random solution, and when $k = \mu$, it returns the best solution.

Since the size of the parent populations varies from agent to agent and from one generation to another, the size of the tournament cannot be chosen before hand. In practice we fix k as a function of μ. In it's simpler form this function can be a linear projection such as $k = \lfloor \alpha\mu \rfloor + 1$ with $\alpha \in (0, 1)$, but more sophisticated functions could also be considered. In this case, α is the parameter that tunes the selection pressure.

3.1.2 Weighted intermediate recombination in the $(\mu/\mu_{\mathrm{W}}, 1)$-On-line EEA

This recombination scheme is similar to recombination in CMAES [8]. The newly generated genome is defined as:

$$\bar{x} = \sum_{i=1}^{\mu} w_i \, x_{i:\mu}, \qquad \sum_{i=1}^{\mu} w_i = 1, \qquad w_1 \geq w_2 \geq \ldots \geq w_\mu > 0,$$

where $w_{i=1\ldots\mu} \in \mathbb{R}_+$ are the recombination weights and the index $i{:}\mu$ denotes the i-th ranked individual and $f(x_{1:\mu}) \geq f(x_{2:\mu}) \geq \ldots \geq f(x_{\mu:\mu})$. By assigning different values to w_i it can also be considered as a selection scheme since fitter solutions have greater weights in the sum. In this work, the weights $w_i = \frac{w'_i}{\sum_{k=0}^{\mu} w'_k}$ where $w'_i = \log(\mu + 0.5) - \log i$.

We emphasize that, since population sizes may differ between agents, the weights w_i may also be different from agent to agent due to their normalization (Fig. 1 left). This fact suggests that the EA gives a selective advantage to the fittest genomes that belong in the lists of agents that had fewer mating encounters.

4 Experiments

The experiments were performed on the Roborobo simulator [5] an environment that allows researchers to run experiments on large swarms of agents. In this simulator, agents are e-puck like mobile robots with limited range obstacle sensors and two differential drive wheels. The neuro-controller we consider in this work is a simple feed-forward multi-layered perceptron with one hidden layer that maps sensory inputs (values in $(0, 1)$) to motor outputs (values in $(-1, 1)$). All the parameters of the experiment are summarized in Table 1.

We compare $(\mu/\mu_{\mathrm{W}}, 1)$-On-line EEA and $(\mu, 1)$-On-line EEA on two extensively studied collective evolutionary robotic benchmarks: 1) locomotion and obstacle avoidance, 2) item collection. In the first task, agents must travel the largest distance; the fitness function in this case is the accumulated traveled distance since the beginning of the generation. In the second task the agents must collect the most items; the fitness function is simply the number of items collected since the beginning of the generation. The environment of both tasks are shown on Fig. 1.

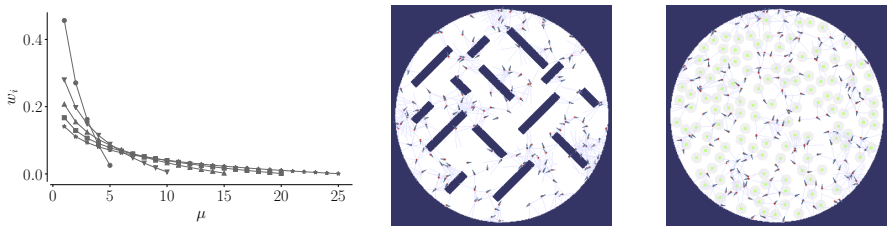

Figure 1: The recombination weights for different population sizes (lef) in $(\mu/\mu_{\mathrm{W}}, 1)$-On-line EEA. An overview of the Roborobo simulator. The locomotion task (middle) and the collection task (right).

Table 1: Simulation parameters

$t_{\max}$	300	Arena diam.	400	λ	160
Sensors	24	Sensor range	32	N	135
Effector	2	Com. range	32	$g_{\max}$	300
Topology	$24 \times 5 \times 2$	Nb. items	$\frac{3}{4}\lambda$	σ	0.5
Bias (output)	$+1$	Agent rad.	3	α	0 or 1
Max tr. vel	2 / tic	Item rad.	10	mat. age	$0.8 \times t_{\max}$
Max rot. vel	30 deg/tic	Nb runs	30		

In the case of $(\mu, 1)$, we fix the selection pressure to the highest ($\alpha = 1$) to maximize the performance [6]. Furthermore we also compare each algorithm with its "naive" instance as a control experiment. In the case of recombination, this is an instance where all weights are equal ($w_i = \frac{1}{\mu}$, $i = 1 \dots \mu$). Here there is no selection bias towards the fittest solutions. The second "naive" instance performs a tournament with $\alpha = 0$ (selects the next genome randomly from the list). We use the notations $(\mu/\mu_{\mathrm{I}}, 1)$ and $(\mu_{\mathrm{RND}}, 1)$ to refer to these "naive" algorithms.

5 Results and discussion

The results of the experiment are presented in Fig. 2. To compare both algorithms, we are interested in two measures, the fitness and the number of mates the agents inseminated. Both these measures are summarized on the curves as the median (of 30 independent runs) of :

$$f(t) = \frac{1}{\lambda} \sum_{j=1}^{\lambda} f^j(t),$$

where $f^j(t)$ is the fitness of agent j at generation t. We use a similar formulation for the second measure.

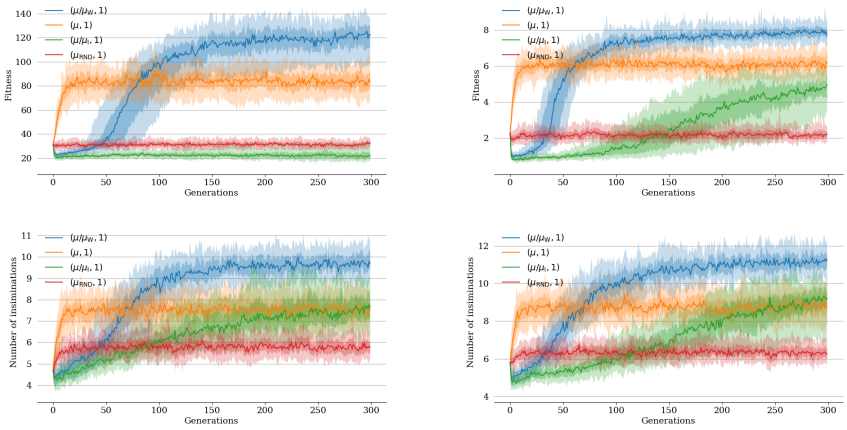

Figure 2: Fitness and number of mates for the locomotion task (left) and the collection task (right). Curves represent median (solid line), the range between the 25^{th} and the 75^{th} percentile (darker area) and the between the 5^{th} and the 95^{th} percentile (lighter area) of 30 independent runs.

On both tasks, $(\mu/\mu_{\text{W}}, 1)$ outperforms $(\mu, 1)$ and the difference is statistically significant. On locomotion for instance, in 95% of the runs, the average traveled distance by each agent reaches the range $(100, 140)$ compared to $(60, 100)$. On the collection task we have the same trend ; agent collect roughly 8 items per generation, two more than in $(\mu, 1)$. The fact the $(\mu/\mu_{\text{W}}, 1)$ performs better than a purely elitist EEA is very interesting, and shows that it is not sufficient to take the best solution and evolve it. There is still room for improvement and better solution can be found by combining traits from multiple solution.

The "naive" instances $(\mu/\mu_{\text{I}}, 1)$ and $(\mu_{\text{RND}}, 1)$ perform very poorly in comparison. Although we note, that in the case of the former, the swarm starts to collect items at around generation 150 almost reaching $(\mu, 1)$ at the end. When we consider the number of insemination, the qualitative results are similar to the exception of $(\mu/\mu_{\text{I}}, 1)$ which seems to evolve agent that learn to successfully spread their genomes.

In the case of $(\mu/\mu_{\text{I}}, 1)$ and $(\mu_{\text{RND}}, 1)$, even though there is no selection pressure induced by the task, the environment exerts a pressure that can reduce the chances of mating for agents, an environmental pressure [4]. The improvements in the fitness values, are due to this. When evolution finds solutions that allow their vehicles to travel longer distances, these solutions spread to other agents and increase their chance of survival. This may explain why $(\mu/\mu_{\text{I}}, 1)$ is able to improve overtime. This is further supported when we see that the improvement happens only in the collection task. The presence of obstacles in locomotion increase the environmental pressure.

6 Conclusions

Weighted intermediate recombination is a crucial component of evolution strategies and there is a large body of work that attest of it importance both theoretically and practically. In this paper we asked the simple question : could this be the case in online EER? We proposed $(\mu/\mu_{\mathrm{W}}, 1)$-On-line EEA and designed a experiment to measure the effect of recombination on simple well studied benchmarks. The results suggests, at least on these tasks, that recombination outperforms a purely mutative strategy. Among the remaining question we would like to pursue, the most natural one is to design a truncation mechanism, akin to what exists in the evolution strategies. Such a procedure takes only the better performing parents into account in the recombination and disregard the rest. The added selection pressure improves convergence in ES which may also be the case in EER. Furthermore, it would be also interesting to compare the algorithms in terms of diversity (genetic and behavioral). Does recombination reduce the diversity? Finally, it is also important to extend the tasks and test the algorithm in more challenging environments.

References

[1] Dirk V. Arnold. *Noisy Optimization With Evolution Strategies*. Springer, 2002.

[2] Hans-Georg Beyer. Toward a theory of evolution strategies: On the benefits of sex - the $(\mu/\mu, \lambda)$ theory. *Evolutionary Computation*, 3(1):81–111, 1995.

[3] Nicolas Bredeche, Evert Haasdijk, and Abraham Prieto. Embodied evolution in collective robotics: A review. *Front, in Robo. and AI*, 5:12, 2018.

[4] Nicolas Bredeche and Jean-Marc Montanier. Environment-driven Embodied Evolution in a Population of Autonomous Agents. In *Proc. PPSN 2010*, pages 290–299, Krakow, 2010.

[5] Nicolas Bredeche, Jean-Marc Montanier, Berend Weel, and Evert Haasdijk. Roborobo! a fast robot simulator for swarm and collective robotics. *CoRR*, abs/1304.2888, 2013.

[6] Iñaki Fernández Pèrez, Amine Boumaza, and François Charpillet. Comparison of selection methods in on-line distributed evolutionary robotics. In *Proc. of Alife'14*, pages 282–289, New York, 2014. MIT Press.

[7] Iñaki Fernández Pèrez, Amine Boumaza, and François Charpillet. Decentralized innovation marking for neural controllers in embodied evolution. In *Proc. of GECCO '15*, pages 161–168, Madrid, 2015. ACM.

[8] Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evol. Comput.*, 9(2):159–195, 2001.

[9] Giorgos Karafotias, Evert Haasdijk, and Agoston E. Eiben. An algorithm for distributed on-line, on-board evolutionary robotics. In *Proc. of GECCO '11*, pages 171–178. ACM, 2011.

[10] David J. Schaffer, , Darrell Whitley, and Larry J. Eshelman. Combinations of genetic algorithms and neural networks: a survey of the state of the art. In *Proc. of COGANN '92*, pages 1–37, 1992.

[11] Fernando Silva, Paulo Urbano, Sancho Oliveira, and Anders Lyhne Christensen. odneat: an algorithm for distributed online, onboard evolution of robot behaviours. In *Artificial Life*, volume 13, pages 251–258. MIT Press, 2012.

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

[13] Richard Watson, Sevan Ficici, and Jordan Pollack. Embodied evolution: Distributing an evolutionary algorithm in a population of robots. *Rob. and Auto. Sys.*, 39:1–18, 2002.