Decentralized Innovation Marking for Neural Controllers in Embodied Evolution

Iñaki Fernández Pérez, Amine Boumaza, François Charpillet

Decentralized Innovation Marking for Neural Controllers in Embodied Evolution

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

Abstract

We propose a novel innovation marking method for Neuro-Evolution of Augmenting Topologies in Embodied Evolutionary Robotics. This method does not rely on a centralized clock, which makes it well suited for the decentralized nature of EE where no central evolutionary process governs the adaptation of a team of robots exchanging messages locally.

This method is inspired from event dating algorithms, based on logical clocks, that are used in distributed systems, where clock synchronization is not possible. We compare our method to odNEAT, an algorithm in which agents use local time clocks as innovation numbers, on two multi-robot learning tasks: navigation and item collection. Our experiments showed that the proposed method performs as well as odNEAT, with the added benefit that it does not rely on synchronization of clocks and is not affected by time drifts.

1 Introduction

Evolutionary robotics (ER) deals with automatic robot behavior learning with techniques inspired from natural evolution [Nolfi and Floreano, 2000]. Embodied ER [Watson et al., 2002] explores the idea of several agents simultaneously learning behaviors using evolutionary algorithms that the robots run onboard. It has been applied on different learning tasks, usually in the context of online multi-robot adaptation. Similarly to mainstream ER, agents' controllers are usually represented as neural networks (neuroevolution) that map sensor inputs to motor outputs, and the learning algorithm adapts the neural architecture and parameters such that the agents execute the correct behavior.

When evolving neural controllers, two different approaches are usually followed: on the one hand, adaptation can take place at the synaptic level, *i.e.* the evolutionary algorithm optimizes the weights of a fixed topology neural network. On the other hand, adaptation can also take place at the structural level of the controller, in which case the topology and the synaptic weights of the neural network are optimized simultaneously. Searching in this richer space of topologies and weight configurations allows for evolution of neural controllers with topologies adapted to the task and removes the need of hand-tuning the topology.

There exists a large body of work on the evolution of topology in neural networks [Gruau, 1993, Stanley and Miikkulainen, 2002, Kowaliw et al., 2014]. However, many algorithms do not consider cross-over operators, since such operators, if not designed properly or if applied blindly, often tend to destruct functional building blocks. Among the algorithms that use cross-over, NEAT (NeuroEvolution of Augmenting Topologies) [Stanley and Miikkulainen, 2002] introduced a simple gene marking scheme (called innovation numbers), where each element of the neural network (neurons and synapses) is uniquely identified and the chronological order of insertion of such elements is tracked, which allows to correctly recombine building blocks.

In our work, we are interested in Embodied Evolution (EE) to deal with a team of coexisting robots performing open-ended behavior learning, where robots are always adapting to the task at hand [Bredeche et al., 2010]. Each robot has an active genome that is locally broadcasted when robots cross each other, and an internal population, which corresponds to a local view of the population that is distributed across all robots. No robot has global knowledge of all genomes.

In the last years, several works have dealt with Embodied Evolution of neural robot controllers in general, usually limiting the evolution to the synaptic weights of fixed-topology neural networks. Recently, odNEAT (Online Distributed NEAT) [Silva et al., 2014] proposed a distributed version of NEAT in the context of Embodied Evolution. The algorithm uses high-resolution clocks at the robot level to mark structural innovations. However, such clocks may drift, thus requiring synchronization, especially if the robots are deployed for long periods of time, which is one of the objectives of EE. In a sense, this requirement is in contradiction with the distributed aspect of Embodied Evolution.

Our main motivation in the following work is to propose an algorithm to mark structural innovations that does not rely on a centralized clock or synchronizations. This algorithm that we name Gene Clocks (GC), is inspired from logical clock algorithms used in asynchronous distributed system models. By only using local exchanges and with a small computational and communication overhead, GC tracks the historical chronology of genes in the population. In this way the distributed nature of Embodied Evolution algorithms is preserved.

In the following sections, we will start by describing different algorithms that addressed neuroevolution of topologies in the embodied and distributed context and discuss the way innovations are marked. We will then detail our approach, and explain its links with asynchronous distributed system models and how innovation marking is performed in a distributed way. In section 4, we describe the experiments and discuss the results obtained. Finally, we present some open questions and research directions that could extend our work.

2 Related Work

Several neuroevolution algorithms dealing with the evolution of neural topology along with the synaptic weights, have been proposed ([Angeline et al., 1994], [Gruau, 1993], [Stanley and Miikkulainen, 2002]). Neural networks can be encoded as a set of neuron genes and connection genes. In this case, mutation operators usually add or remove neurons and connection genes with some probability, to introduce topological innovations in the neural networks. In this context, the goal of cross-over operators is to accurately combine the best functional characteristics, or building blocks, of two parent networks, to produce offspring that inherit these building blocks. This raises some problems (*e.g. Competing Conventions* problem [Schaffer et al., 1992]), because similar topologies can be represented with very different genetic encodings. Classical cross-over operators tend to be disruptive and break existing functional structures, thus hindering evolution. If cross-over is to be applied between two of such networks, their respective structural elements have to be matched and recombined correctly. This is the main reason why many topology-evolving neuroevolution algorithms do not use the cross-over operator [Angeline et al., 1994].

NEAT [Stanley and Miikkulainen, 2002] proposed an elegant solution to reduce the effect of the competing conventions problem and perform meaningful cross-over. The algorithm marks each new neuron or connection with an unique *innovation number*, which is a historical marker that identifies the new gene and keeps track of the chronology of its appearance in the evolutionary process. These innovation numbers are inherited by offspring networks. The purpose of innovation numbers is to detect the genes expressing the same neural elements between two genomes (genes having the same innovation number) without the need of matching the topology of the two networks. As such, the impact of the competing conventions problem is reduced, because genes having the same innovation number in two parent genomes are not repeated when applying cross-over. These historical markers are implemented in NEAT by keeping a global counter of innovations and sequentially assigning it to new genes in the genomes of the population. Furthermore, the algorithm uses a niching mechanism to divide the population in non-overlapping species, based on genotypic similarity. To promote topological diversity, NEAT uses intra-species fitness sharing, *i.e.* networks in the same niche share the same fitness. This results in the protection of new structures that could lead to fitter individuals but require time to optimize their parameters. The algorithm is a centralized offline EA with one global population initialized with minimal fully-connected perceptrons, and where all new genes can be tracked since all information is centralized.

When evolving both weight and topology in an EE context, the same problems raised in offline centralized setups (competing conventions, see above) occur. However, a global counter as in NEAT's innovation marking mechanism is not directly transferable to Embodied Evolution, because different robots cannot be simultaneously aware of each other's innovations. Two topology-evolving EE algorithms, and more specifically, two mechanisms to mark topological innovations in a distributed manner, have been proposed to date: odNEAT [Silva et al., 2014] and IM-($\mu + 1$) [Schwarzer et al., 2012].

In IM-($\mu + 1$), each robot draws a random number (between 1 and 1000 in their experiments) to assign it as identifier of a new neuron gene added in any robot's genome. The authors state that, if the probability of drawing the same number, which is referred as an identifier collision, is sufficiently low, the system is overall not disturbed, and the impact of colliding innovation numbers is reduced by selection. Although this may be true in relatively short experiments with few robots, when dealing with long-term adaptation EE setups, this probability increases, and genealogy can not be tracked.

odNEAT (Online Distributed NeuroEvolution of Augmenting Topologies) translated NEAT characteristics to a distributed setup. Each local population is speciated based on a genotypic distance, and the genomes in a species share the fitness of the species, as in NEAT. Innovation markers are assigned using local, high-resolution timestamps. Robots mark new genes using their respective local clocks. Offspring are produced by selecting two parent genomes, recombining them as in NEAT, and mutating the result with some probability. Mutations either probabilistically add a new neuron, a connection or add a normal random variable to every weight in the network. The innovation numbers are used to sort the genes in a genome to facilitate the matching of common structures with another genome. As in NEAT, this alignment is used to distinguish between matching genes and non-matching genes in both genomes, either to compute a genotypic distance or to mate two parent genomes without repeating structures. According to the authors, the use of high-resolution timestamps practically guarantees identifier uniqueness and allows different robots to retain the chronology of the innovations across the distributed system.

Since odNEAT runs on all robots that are simultaneously evolving, their respective local clocks have a strong probability of drifting w.r.t. each other. This is specially true in open-ended long-term adaptation setups, where robots run for a long time and clock drifting may be considerably more important. In this case, innovation timestamps would not be coherent, and thus would not correctly keep track of the actual chronology of innovations. If odNEAT is to be run on physical robots, it would require a periodical clock synchronization between the robots, which may be impossible when robots are scattered.

3 odNEAT with logical markers

In our proposed innovation marking algorithm, we view an evolving set of robots in Embodied Evolution as a distributed computer system. In the asynchronous distributed system model [Coulouris et al., 2011], several interconnected processors communicate with each other and perform local computations in order to solve a common problem. Events in such systems consist of either local computations or message exchanges. In this work, we consider robots as processors, computational processes as the evolutionary process of each robot, events as topological innovations in the system and message exchange between processors as genome exchange between robots.

One of the fundamental problems in distributed systems is the dating and ordering of events. Since distributed processors do not have a common clock, ordering events w.r.t. each other requires a specific mechanism. Logical Clock algorithms (LC) [Lamport, 1978] were proposed to identify and date events in the system in a decentralized way. In this mechanism, every processor has an unique identifier and a monotonically increasing local counter initialized to 0 to keep track of the events. When an internal event or a message exchange takes place, the event is marked with the current local counter and the counter is incremented by one. Whenever a processor receives a message, it updates its local counter with the maximum value between the received counter and its own. This way, events in the system can be ordered with respect to the moment they occurred. However, two events may have identical counters if they originated in two different processors and no communication took place between the two processors during the period in between events. These events are considered concurrent events. Whenever this case occurs, authors order the concurrent events by convention w.r.t. the processor identifier to define a total order between events in the system.

3.1 Decentralized Marking of Innovations

In our proposed method, each robot has a unique identifier, r, as well as a sequential *local innovation counter*, c_r, in the same manner as processors have identifiers and event counters in a distributed system. We consider innovation numbers as pairs $< r, c_r >$. Each robot increments its counter c_r by one each time a mutation adds a new gene, as processors do when they date events. When a robot broadcasts its active genome, all its topological innovations are marked with such innovation numbers.

Using these innovation numbers, genes in the genome can be ordered, based on their gene counters, since they keep a chronological sequence of the innovations on each individual robot. However, innovations originated in different robots can not be chronologically ordered since all robots maintain independent sequences of innovations. To palliate this issue, in addition to sending its active genome, a robot broadcasts its current c_r, and upon reception, robots update their respective c_r's with the maximum of the received value and their own, as it is done in LC. Concurrent innovations are still possible, *i.e.* two innovations with the same c_r, but originating in different robots that do not communicate in between. In this case, innovation numbers will be ordered by convention with respect to the robot's identifier so all robots sort genes in the exact same manner.

3.2 Evolutionary Algorithm

In our experiments, we use odNEAT as Embodied Evolution algorithm to evolve the topology and weights of neural networks. The main steps of the algorithm, that runs on every robot, are shown in Algorithm 1. At all times, each robot has an active genome that is initialized with a fully-connected single-layer perceptron. Robots locally broadcast their active genomes at every timestep, with a probability proportional to the fitness of the species of the genome (see below). Received genomes are stored in a limited-size local population, which also contains previous active genomes. When the population is full, the addition of a fit genome implies the removal of the worst genome in the population.

Each robot has an internal virtual energy level that is initialized to a default task-dependent value. This value represents the performance of the active controller in the task at hand, and it increases and decreases according to robot behavior. As such, the energy level measures how well a robot solves the given task. Every control cycle, a robot executes its active controller and updates its energy level according to its behavior. One of the inherent problems of EE is that evaluation is noisy, given that different controllers can be evaluated in very different conditions. For example, if a fit controller starts its evaluation surrounded by obstacles, its energy level will drop until it is able to escape the area. To alleviate this issue and record a more precise approximation of the actual performance, the robots periodically measure their respective energy levels at regular intervals. The energy measures are averaged to compute the controller's fitness $f_r(t)$ of robot r at instant t. Finally, whenever the energy level drops below a given threshold, a new offspring is produced and replaces the active genome, and the energy level is reset to its default initial value.

Algorithm 1: odNEAT algorithm run by every robot.

```
$g_a \leftarrow random\_genome()$
$//$Add also to a new species
$\mathbf{P} \leftarrow \{g_a\}$
$e \leftarrow e_{init}$
while $True$ do
    if $do\_broadcast?$ then
        $send(g_a, e, neighbors)$
    forall the $g \in received$ do
        if Tabu.$approves(g)$ and P.$accepts(g)$ then
            $//$Add also to corresponding species
            $\mathbf{P} \leftarrow \mathbf{P} \bigcup \{g\}$
            $adjust\_species\_fitness()$
    $execute(g_a)$
    $e \leftarrow e + \frac{\Delta E}{\Delta t}$
    if $e \leq e_{threshold}$ and $\neg$ $in\_maturation\_period$ then
        $\mathbf{Tabu} \leftarrow \mathbf{Tabu} \bigcup \{g_a\}$
        if $random() < p_{mate}$ then
            $p_1, p_2 \leftarrow select\_parents(select\_species())$
            $g_{offsp} \leftarrow mate(p_1, p_2)$
        else
            $g_{offsp} \leftarrow select\_parent(select\_species())$
        if $random() < p_{mutate}$ then
            $g_{offsp} \leftarrow mutate(g_{offsp})$
        $//$Add also to corresponding species
        $\mathbf{P} \leftarrow \mathbf{P} \bigcup \{g_{offsp}\}$
        $g_a \leftarrow g_{offsp}$
        $e \leftarrow e_{init}$
```

To maintain diversity in local populations, genomes are divided into species based on a genotypic distance, *i.e.* the genomes in the population are grouped in non-overlapping species by genotypic similarity, and fitness sharing is applied in the same manner as in NEAT.

At each iteration, robots broadcast their active genome with a probability proportional to the fitness of the genome species. When a broadcast occurs, the robot's active genome, its current energy level, and c_r in the case of GC, are sent to all neighboring robots. If a given robot receives the same genome twice, the energy level is averaged with the previous value to provide a better estimator of the performance of the stored genome.

Whenever the energy level drops below the threshold, the active controller is replaced by a new genome generated as follows: First, a species is selected with a probability proportional to its average adjusted fitness. Then, with a probability p_{mate} two parents are selected within the chosen species and are recombined. Finally, the resulting offspring is mutated with a probability p_{mutate}. With a probability $(1 - p_{mate})$, only one parent is selected from the species, and is mutated with probability p_{mutate}.

Mutation can be of 2 types, structural mutation and parameter mutation. Structural mutation includes adding a neuron with a probability p_{node} by splitting a random existing connection or adding a connection with probability p_{conn} between two random unconnected existing neurons. When adding a connection, there is a probability p_{recur} for this connection to be recurrent. On the other hand, concerning parameter mutation, there is a probability p_w of mutating all weights by adding a normal random variable with mean 0 and a standard deviation σ. Synaptic weights are maintained in $[-10, +10]$. The resulting offspring replaces the previous active genome, its energy level is reset to the default value, and the new genome starts being executed.

A new genome is given some time T_m to be executed and evaluated even if its energy level drops below the threshold level. This is known as maturation period [Bredeche et al., 2010], and it aims firstly at protecting potentially fit controllers that start their evaluation in difficult conditions, and secondly, at setting a lower bound to the time a given genome is evaluated to obtain a sufficiently accurate estimate of its performance.

Taking in account all the aforementioned variation probabilities, standard deviation of the weight mutation σ, the maturation time T_m, and the local population size, $|\mathbf{P}|$, we need to set values to 9 parameters. For that, we use a tuning procedure that is described in the next section.

odNEAT also maintains a tabu list to keep track of recent poor genomes, either genomes dropped because they depleted their energy, or the genomes that are not competitive enough and are dropped from the population because it was full. This is used to filter out received genomes that are similar to one or more individuals in the tabu list, before adding them to the local population, *i.e.* to avoid evaluating candidates similar to known poor solutions.

Both the tabu list filtering and the speciation mechanism rely on computing a genotypic distance, that is measured as follows:

$$d(g_1, g_2) = \frac{c_1 \cdot E}{N} + \frac{c_2 \cdot D}{N} + c_3 \cdot \overline{W} \tag{1}$$

where $\overline{W}$ is the average of the weight difference between matching genes, *i.e.* genes corresponding to the same structural elements in g_1 and g_2. Non-matching genes are considered in two fashions, depending on the chronology of the compared genes: D is the number of *disjoint* genes, that is non-matching genes in the middle of the genome, and E is the number of *excess* genes, non-matching genes in the end of the largest genome. N is the number of genes of the largest of both genomes, and c_1, c_2, c_3 are coefficients weighting the relative importance of each of the three elements. The way to distinguish between matching, disjoint and excess genes consists in aligning two genomes using the innovation numbers presented above. Furthermore, the cross-over operator in odNEAT (as NEAT) also uses historical markers to distinguish between matching, disjoint and excess genes between two parent genomes (see [Stanley and Miikkulainen, 2002] for details).

4 Experiments

In our experiments, we compare the results of odNEAT using Gene Clocks and using timestamps as innovation numbers. Timestamps in a system where local clocks are synchronized allow for a perfect sorting of genes in a genome, provided that no drift occurs, whereas, in the case of concurrent innovations, GC sorts genes by convention. Our main hypothesis is that Gene Clocks approximate sufficiently the chronology of innovations to lead to performance levels similar to those obtained using timestamps while relying exclusively on local information and exchanges. Our experiments, presented below, corroborate this hypothesis, in terms of the quality of learned behaviors and in terms of the architecture of the evolved controllers.

4.1 Experimental Setup

We conducted our experiments on two well-studied tasks in ER [Nolfi and Floreano, 2000], navigation with obstacle avoidance and item collection. In the navigation with obstacle avoidance task, robots must learn to move as fast and as straight as possible in a bounded environment, while avoiding both moving and static obstacles (other robots and walls). In the item collection task, food items are placed in the environment and robots must collect as many of them as possible.

We performed all our experiments in the RoboRobo simulator [Bredeche et al., 2013], which simulates a team of robots deployed in a bounded environment containing obstacles (black lines in Figure 1), as well as food items in the case of the item collection task (green circles). All robots have the same morphology, sensors and motors. 8 obstacle proximity sensors, that measure the distance to obstacles and other robots, are evenly distributed around the robot. An additional sensor measures the current energy level of the robot. In the case of the item collection task, robots perceive food using 8 additional food item sensors with the same range as the obstacle sensors. The robots move using two differential drive wheels. Being morphologically homogeneous, behavior difference between robots originates from having different controllers. A robot's neural controller is initialized with a bias neuron and as many inputs as there are sensors (9 in the case of navigation and 17 in the case of item collection). The two outputs of the neurocontroller correspond to the right and left wheel velocities.

An active genome's life cycle, or genome's evaluation, consists on executing it, updating the energy level and probabilistically broadcasting it to nearby robots, until its energy level is below a threshold. At this point, the genome is considered unfit to the task, and is replaced by a new one, generated as presented in Section 3.2. The energy level of the new genome is reset to the default value of 100 units, its maximal value being 200 units. A robot updates its energy level every control cycle as follows:

- **Navigation**:

$$\frac{\Delta E}{\Delta t} = f_n(|v_l + v_r| \cdot (1 - \sqrt{|v_l - v_r|}) \cdot (1 - d)) \tag{2}$$

  where v_l and v_r are the left and right wheel velocities and d is the distance to the closest obstacle. The f_n function is a linear mapping from the interval $[0, 1]$ to $[-1, 1]$. Thus, if a robot moves fast, straight and far from obstacles it will gain energy, and it will lose energy otherwise. This is the same

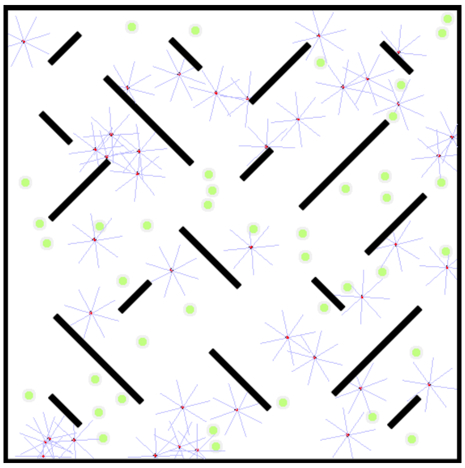

Figure 1: The simulation environment containing robots (red dots with thin hair-lines representing sensors), obstacles (dark lines) and food items (green dots).

energy update function in which odNEAT was studied [Silva et al., 2014], and it is inspired from [Nolfi and Floreano, 2000].

- **Item collection**:
$$\frac{\Delta E}{\Delta t} = -0.1 + 10 \cdot c(t) \tag{3}$$
where $c(t)$ is a boolean function indicating if an item was collected at t. At each time step, the robot loses 0.1 energy units, and when it picks up an item, it gains 10 units. In the experiments, 75 items are randomly placed in the environment. When a robot collects an item, it disappears, and a new one is randomly placed in the environment to keep the number of items constant.

In all experiments, 100 robots were deployed in the environment. Each robot runs Algorithm 1 for 40000 cycles in the navigation task and 60000 cycles in the collection task. We performed 64 independent runs for both innovation marking methods in both tasks. Every 100 cycles, we measure:

- The average fitness of all robots, or swarm fitness $F_s(t) = \frac{1}{|team|}\sum_{r \in team} f_r(t)$. It represents the overall performance of the evolving robots at instant t.
- The average number of species over the local populations of all robots. The number of species provides information on the overall topological diversity in the robots population.
- The average size of active controllers of all robots. In our work, we consider the number of connections and neurons as the size of a given neural controller. This gives an idea of the complexity of the evolved solutions.

In Embodied Evolution, the best fitness reached during evolution is not a reliable measure of the overall performance of the robots. Measures must emphasize the online nature of the algorithms, *i.e.* since robots adapt during the actual execution, an instantaneous measure would not be a reliable indicator of their performance. To consider this in the analysis of our experiments, we use the metrics of performance that were previously introduced in [naki Fernández Pérez et al., 2014]. These metrics reflect information on the evolutionary process in general by integrating information spanning over time (see Figure 2).

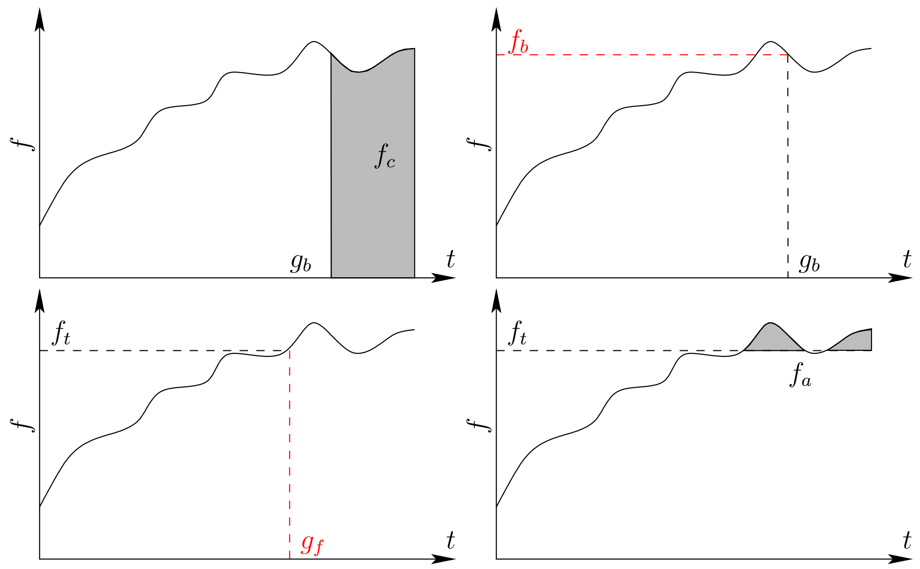

Figure 2: A pictorial description of the four comparison measures. From top to bottom and left to right: the average accumulated swarm fitness, f_c, the fixed budget swarm fitness, f_b, the time to reach target g_f, and the accumulated fitness above target, f_a.

In order to set the appropriate values for the 9 parameters of the algorithm (see 3.2), we performed a tuning procedure. For each task the parameters were independently tuned using Iterated Race for Automatic Algorithm Configuration (irace) [López-Ibáñez et al., 2011], a parameter tuning algorithm that optimizes a given quality measure of complete runs of the evolutionary algorithm. The quality measure we used in irace is f_c, as presented above.

The tuning procedure was performed using timestamps as innovation numbers, and a total budget of 1200 runs was allotted to irace to find parameters maximizing f_c, for both tasks. At the end of these procedures, we obtain two sets of parameters, one for each task. All the experiments presented below used these sets of parameters, that are summarized in Table 1.

4.2 Results and analysis

In summary, we performed four experiments (two innovation marking mechanisms in two tasks), in order to test if our proposed method for innovation marking leads to similar results as the original timestamps-based mechanism of odNEAT. For each experiment, we performed 64 independent runs to provide statistically sound results. Figure 3 (resp. Figure 4) shows the swarm fitness F_s

Table 1: Summary of the tuned and fixed parameters of odNEAT in our experiments, as described in section 3.2.

	Tuned		Fixed	
	Navigation	Collection		
p_{mate}	0.842	0.212	c_1	0.5
p_{mutate}	0.154	0.244	c_2	1.5
p_w	0.575	0.559	c_3	0.4
p_{node}	0.057	0.422	$\#Robots$	100
p_{conn}	0.184	0.275	$\#Items$	75
p_{recur}	0.376	0.526		
σ	0.442	0.114		
T_m	19 cycles	27 cycles		
$\|\mathbf{P}\|$	5	7		

(the average fitness of the robots) during evolution, for both innovation marking algorithms in the navigation task (resp. the item collection task).

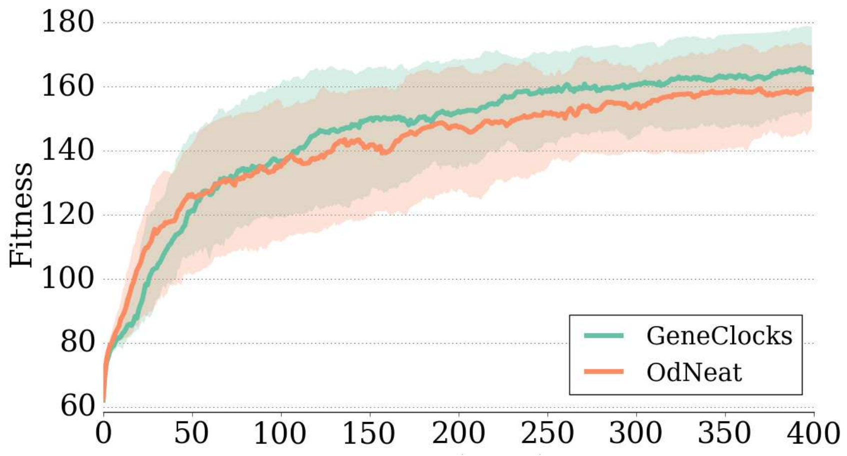

Figure 3: Swarm fitness, F_s, over time of the 64 runs for the navigation and obstacle avoidance task. The lines correspond to the median value between all runs, and the shaded areas show the interquartile range.

For the navigation task (Figure 3), both dating methods lead to the learning of proper behaviors, achieving around the end of the experiments values between 75% and 90% of the maximum level of swarm fitness (fixed in the experiments at 200). Upon inspection of some of the evolved behaviors, we observed that, while having a limited range of perception, the robots are able to rapidly react to other incoming robots, avoid each other and keep moving straight and fast. As for the comparison between GC and the timestamp based dating method, the trend of F_s for both is roughly the same in median value and interquartile range.

Similarly, in the item collection task (Figure 4) both experiments managed to evolve controllers that searched the environment and effectively collected items, with performances falling between 70% and 80% of the maximum value around

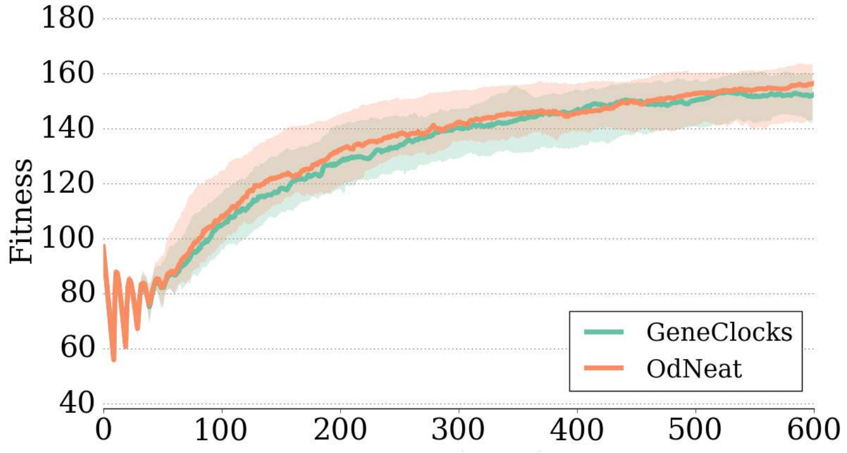

Figure 4: Same as Figure 3 for the item collection task.

Table 2: p-values of the Mann-Whitney two-sided tests between the four measures for Gene Clocks and timestamp-based dating mechanisms and both navigation and item collection tasks. The differences are not statistically significant for any comparison.

	f_c	f_b	t_f	f_a
Navigation	0.134	0.175	0.489	0.074
Collection	0.698	0.956	0.373	0.697

the end of the experiments. When comparing GC to timestamps, we observe an even clearer overlap between the curves. The approximation of the global time with our decentralized method seems to have no impact with respect to the swarm fitness.

To further support this claim, we computed the four aforementioned measures on the 64 runs of each experiment. Box-and-whisker diagrams are presented in Figures 5 and 6 for each task. We performed two-sided Mann-Whitney tests comparing both dating mechanisms in both tasks, and over all four measures. The p-values of all tests are all over 0.05, *i.e.* there is no statistical difference between the dating methods with respect to any of the measures (see Table 2). This confirms our main hypothesis that Gene Clocks approximate a perfect innovation ordering sufficiently to have no impact on the reached fitness level.

We also investigated the size of the architecture of evolved controllers and the size of the species in local populations to compare both methods in terms of the controllers they produce. The average number of species over time is presented in Figure 7 for the navigation task and in Figure 8 for the item collection task. The average size of the neural controller over time, measured as the sum of neurons and connections, is presented in Figures 9 (navigation) and 10 (item collection).

The average number of species per local population follows a similar trend over time in both methods, especially in the navigation task, in which they are almost undistinguishable. As for the item collection task, the median value (around 3.75 species per robot) is the same for both experiments, while the vari-

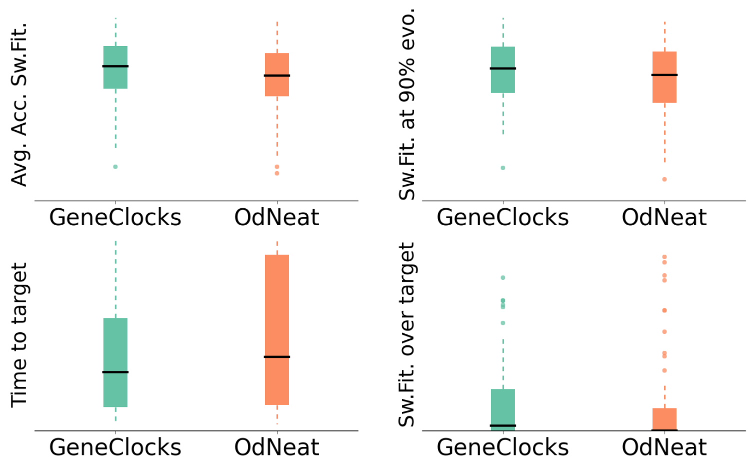

Figure 5: Performance measures of the 64 runs for the navigation task. From left to right and from top to bottom, f_c, f_b, t_f and f_a.

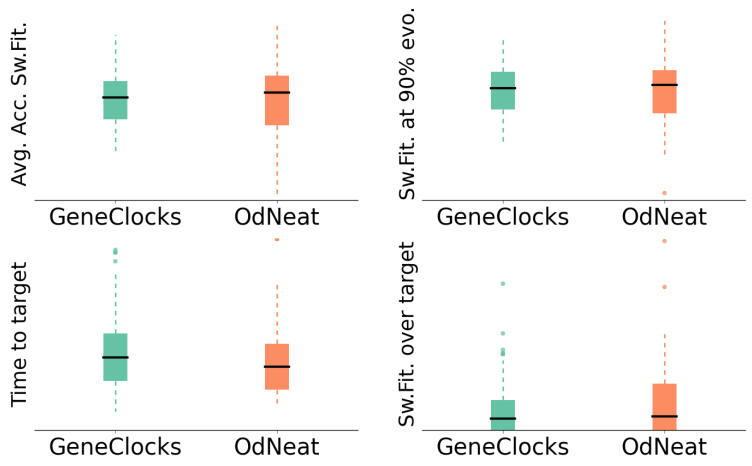

Figure 6: Same as Figure 5 for the collection task.

ance is slightly larger through evolution for the experiment with timestamps (interquartile range of around $\pm$ 0.40 species) than for GC (around $\pm$ 0.25 species). Although the difference is slim, the timestamps marking mechanism seems to induce slightly more variability w.r.t. the number of species. This suggests that the timestamps create more topological diversity in the local populations than our method that shows less variability. We believe that the cause for the disparity is related to innovations being sorted by convention during alignment in GC. In the experiments, the coefficients c_1 for excess genes, and c_2 for disjoint genes, used when computing genotypic distance, were set to 0.5 and 1.5 respectively, which means that excess genes have less impact on genome distance. Given that species are computed based on this value, one possible explanation for the lower dispersion of number of species with GC may be a bias towards alignments resulting in more genes at the end of the genome. Indeed, this requires further investigation regarding the effect that the coefficients have in the alignments resulting from the sorting using Gene Clocks.

With respect to size of the evolved controllers, the use of GC creates networks with around 3 $\pm$ 1.25 added connections or neurons in the navigation task, against around 2.25 $\pm$ 1 for the timestamp mechanism. In the item collection size, 3.75 $\pm$ 1 neural elements were added to controllers when using Gene Clocks, and 3 $\pm$ 0.8 in the case of the timestamps. GC learns controllers with approximately 0.75 more neural elements in both tasks. We computed the average accumulated size of the neural controllers during the last 10% of evolution (as for the swarm fitness above). The results in Figure 11 show that the difference between both dating methods is not statistically significant in both tasks (p-values > 0.9). This requires further investigation to ascertain if there is a bias induced by GC towards slightly larger networks compared to marking with a global clock shared by all robots.

We compared our proposed decentralized innovation marking method with the timestamps mechanism originally proposed for odNEAT in two multi-robot tasks, navigation with obstacle avoidance and item collection. The results comfort our motivation in proposing a completely decentralized innovation marking method that does not hinder the performance of the timestamp method, and that behaves in the same fashion. Furthermore, our method is completely decentralized, and only requires the addition of one integer to broadcasted genomes (the current robot's c_r), a considerably small communication overhead.

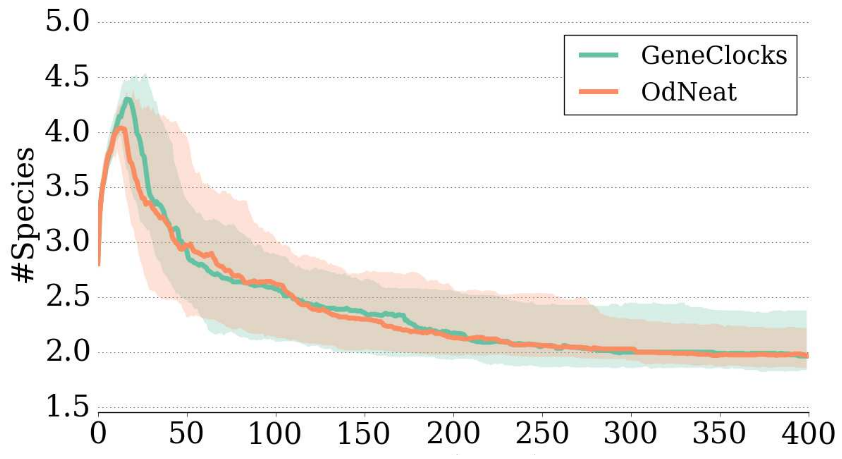

Figure 7: Mean number of species on all the robots, w.r.t. time, over the 64 runs for the navigation task.

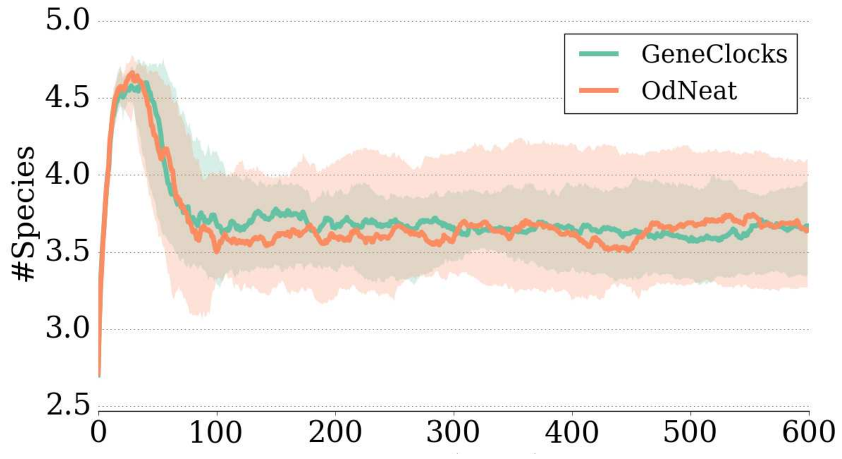

Figure 8: Same as Figure 7 for the collection task.

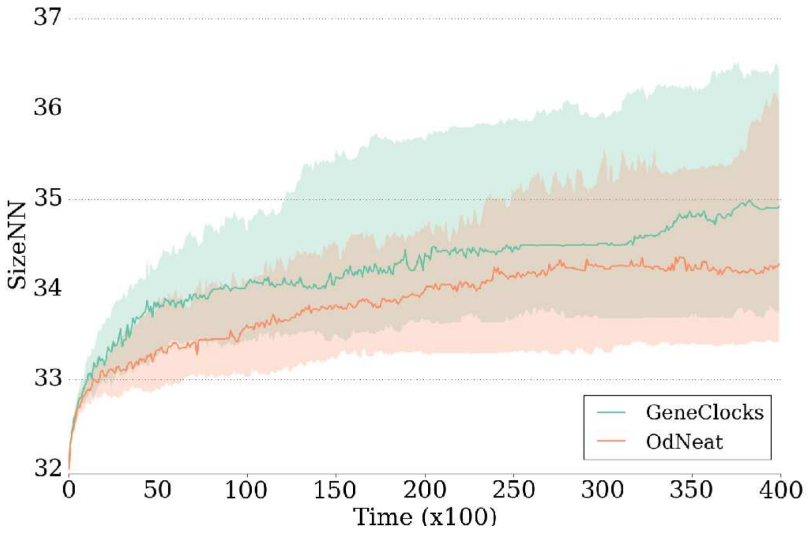

Figure 9: Mean size of the active neural controllers among the robots, measured as the sum of neurons and connections of active genomes, w.r.t. time, over the 64 runs for the navigation task.

5 Conclusion and future work

In this article, we introduced Gene Clocks, a decentralized mechanism to mark genes with innovation numbers in embodied evolution of the topology and weights of neural controllers in a multi-robot distributed context. We have presented our experiments using odNEAT in two experiments involving 100 simulated robots: navigation with obstacle avoidance and item collection. We compared the performances obtained with GC to a method marking genes with local timestamps,

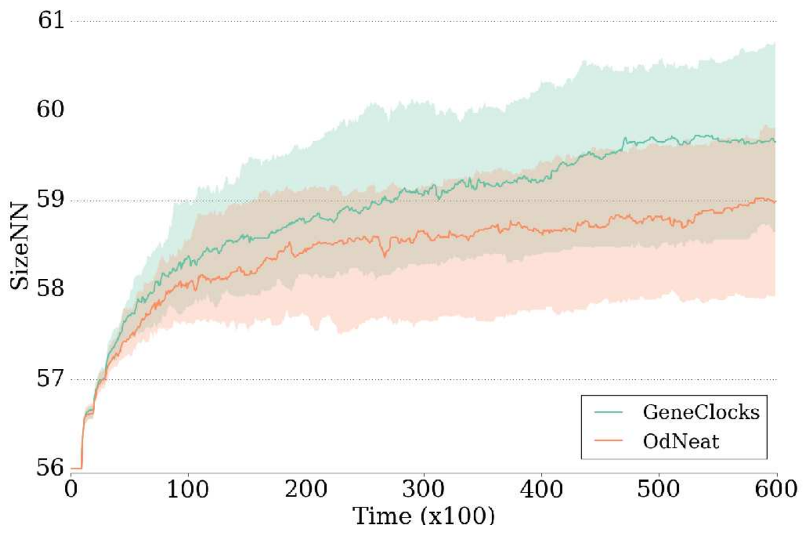

Figure 10: Same as Figure 9 for the collection task.

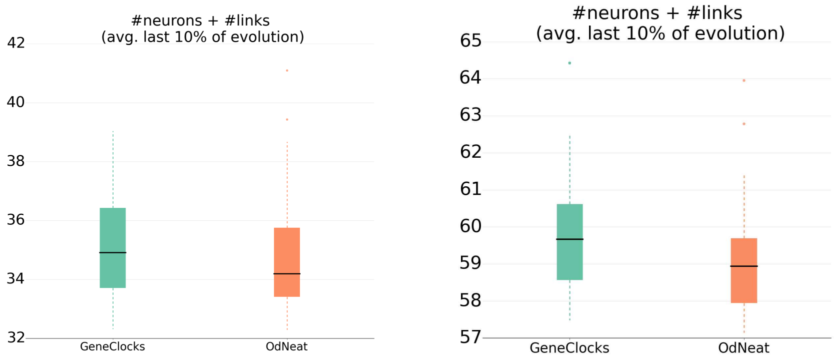

Figure 11: Controller size for the navigation task (left) and the item collection task (right).

which requires periodical synchronizations to be implemented on real robots. Both methods reach the same level of performances in the considered tasks, and the evolutionary dynamics are similar with respect to the number of species and the size of the controllers. In general terms, GC innovation numbers approximate the global time, leading to similar performances and dynamics, while being computed in a completely decentralized manner with a low communication and computational overhead.

One of our motivations to propose a decentralized mechanism to mark innovations in a distributed evolutionary robot system comes from the high cost, or even impossibility, of periodically synchronizing robots in such a framework. This is specially the case in large environments where robot encounters are rare, or more generally, in environments where the robot density is low. Future work aims at evaluating the scalability of the proposed method w.r.t. robot density to test if the innovation marking mechanism keeps functioning correctly when robot encounters are rare.

Innovation numbers contain information on the chronological relative age of genes in the genomes. We are interested in investigating if it is possible to further exploit this information to define more informed cross-over operators or genotypic distance measures.

References

[Angeline et al., 1994] Angeline, P. J., Saunders, G. M., and Pollack, J. B. (1994). An evolutionary algorithm that constructs recurrent neural networks. *Neural Networks, IEEE Transactions on*, 5(1):54–65.

[Bredeche et al., 2010] Bredeche, N., Haasdijk, E., and Eiben, A. (2010). On-line, on-board evolution of robot controllers. In *Artifical Evolution*, pages 110–121. Springer.

[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.

[Coulouris et al., 2011] Coulouris, G., Dollimore, J., Kindberg, T., and Blair, G. (2011). *Distributed Systems: Concepts and Design.* Addison-Wesley, 5th edition.

[Gruau, 1993] Gruau, F. (1993). Genetic synthesis of modular neural networks. In *GECCO'93*, pages 318–325. Morgan Kaufmann.

[Kowaliw et al., 2014] Kowaliw, T., Bredeche, N., and Doursat, R. (2014). *Growing Adaptive Machines: Combining Development and Learning in Artificial Neural Networks.* Springer Publishing Company, Incorporated.

[Lamport, 1978] Lamport, L. (1978). Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565.

[López-Ibáñez et al., 2011] López-Ibáñez, M., Dubois-Lacoste, J., Stützle, T., and Birattari, M. (2011). The irace package, iterated race for automatic algorithm configuration. Technical report, Université Libre de Bruxelles, Belgium.

[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.

[Schaffer et al., 1992] Schaffer, J. D., Whitley, D., and Eshelman, L. J. (1992). Combinations of genetic algorithms and neural networks: A survey of the state of the art. In *COGANN-92*, pages 1–37. IEEE.

[Schwarzer et al., 2012] Schwarzer, C., Schlachter, F., and Michiels, N. K. (2012). Online evolution in dynamic environments using neural networks in autonomous robots. *International Journal On Advances in Intelligent Systems*, 4(3 and 4):288–298.

[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.

[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.