odNEAT: An Algorithm for Decentralised Online Evolution of Robotic Controllers

Fernando Silva fsilva@di.fc.ul.pt
Bio-inspired Computation and Intelligent Machines Lab, 1649-026 Lisboa, Portugal
Instituto de Telecomunicações, 1049-001 Lisboa, Portugal; BioISI, Faculdade de Ciências, Universidade de Lisboa, 1749-016 Lisboa, Portugal

Paulo Urbano pub@di.fc.ul.pt
BioISI, Faculdade de Ciências, Universidade de Lisboa, 1749-016 Lisboa, Portugal

Luís Correia luis.correia@ciencias.ulisboa.pt
BioISI, Faculdade de Ciências, Universidade de Lisboa, 1749-016 Lisboa, Portugal

Anders Lyhne Christensen anders.christensen@iscte.pt
Bio-inspired Computation and Intelligent Machines Lab, 1649-026 Lisboa, Portugal
Instituto de Telecomunicações, 1049-001 Lisboa, Portugal; Instituto Universitário de Lisboa (ISCTE-IUL), 1649-026 Lisboa, Portugal

doi:10.1162/EVCO_a_00141

Abstract

Online evolution gives robots the capacity to learn new tasks and to adapt to changing environmental conditions during task execution. Previous approaches to online evolution of neural controllers are typically limited to the optimisation of weights in networks with a prespecified, fixed topology. In this article, we propose a novel approach to online learning in groups of autonomous robots called odNEAT. odNEAT is a distributed and decentralised neuroevolution algorithm that evolves both weights and network topology. We demonstrate odNEAT in three multirobot tasks: aggregation, integrated navigation and obstacle avoidance, and phototaxis. Results show that odNEAT approximates the performance of rtNEAT, an efficient centralised method, and outperforms IM-$(\mu + 1)$, a decentralised neuroevolution algorithm. Compared with rtNEAT and IM-$(\mu + 1)$, odNEAT's evolutionary dynamics lead to the synthesis of less complex neural controllers with superior generalisation capabilities. We show that robots executing odNEAT can display a high degree of fault tolerance as they are able to adapt and learn new behaviours in the presence of faults. We conclude with a series of ablation studies to analyse the impact of each algorithmic component on performance.

Keywords

Artificial neural networks, decentralised algorithms, multirobot systems, neurocontroller, online evolution.

1 Introduction

Evolutionary computation has been widely studied and applied as a means to automate the design of robotic systems (Floreano and Keller, 2010). In evolutionary robotics, controllers are typically based on artificial neural networks (ANNs). The parameters of the ANNs, such as the connection weights and occasionally the topology, are optimised by an evolutionary algorithm, a process termed *neuroevolution*. Neuroevolution has

Manuscript received: August 8, 2013; revised: May 13, 2014 and October 29, 2014; accepted: November 21, 2014.

been successfully applied to tasks in a number of domains; see Yao (1999) and Floreano et al. (2008) for examples.

In traditional evolutionary approaches, controllers are synthesised offline. When a suitable neurocontroller is found, it is transferred to real robots. Once deployed, the controllers are thus specialised to a particular task and environmental conditions. Even if the controllers are based on adaptive neural models (Beer and Gallagher, 1992; Harvey et al., 1997), they are fixed solutions and exhibit limited capacity to adapt to conditions not seen during evolution.

Online evolution is a process of continuous adaptation that potentially gives robots the capacity to respond to changes and unforeseen circumstances by modifying their behaviour. An evolutionary algorithm is executed on the robots themselves while they perform their tasks. The main components of the evolutionary algorithm (evaluation, selection, and reproduction) are carried out autonomously on the robots without any external supervision. This way, robots have the potential for long-term self-adaptation in a completely autonomous manner. The first example of online evolution in a real mobile robot was provided by Floreano and Mondada (1994). A contribution by Watson et al. (1999, 2002) followed, in which the use of multirobot systems was motivated by an anticipated speedup of evolution due to the inherent parallelism in such systems.

Over the last decade, different approaches to online evolution in multirobot systems have been developed; see, for instance, Bianco and Nolfi (2004); Bredèche et al. (2009); Haasdijk et al. (2010); Prieto et al. (2010); Karafotias et al. (2011). Notwithstanding, in such contributions online approaches have been limited to the evolution of weighting parameters in fixed-topology ANNs. Fixed-topology methods require the system designer to decide on a suitable topology for a given task, which usually involves intensive experimentation. A nonoptimal topology affects the evolutionary process and consequently the potential for adaptation.

In this article, we present a novel algorithm for online evolution of ANN-based controllers in multirobot systems called Online Distributed NeuroEvolution of Augmenting Topologies (odNEAT). odNEAT is completely decentralised and can be distributed across multiple robots. odNEAT is characterised by maintaining genetic diversity, protecting topological innovations, keeping track of poor solutions to the current task in a tabu list, and exploiting the exchange of genetic information between robots for faster adaptation. Moreover, robots executing odNEAT are flexible and potentially fault-tolerant, as they can adapt to new environmental conditions and to changes in the task requirements.

This article offers a comprehensive presentation and analysis of odNEAT, initially introduced in Silva et al. (2012). To the best of our knowledge, only one other online method that optimises neural topologies and weights in a decentralised manner has previously been proposed: the island model-based $(\mu + 1)$ algorithm of Schwarzer et al. (2011), henceforth IM-$(\mu + 1)$, an implementation of the $(\mu + 1)$-online algorithm by Haasdijk et al. (2010) with ad hoc transmissions of randomly chosen controllers. In addition, the real-time NeuroEvolution of Augmenting Topologies (rtNEAT) algorithm by Stanley et al. (2005), an online version of NeuroEvolution of Augmenting Topologies (NEAT) (Stanley and Miikkulainen, 2002) designed for video games, is a prominent *centralised* evolutionary algorithm that has a number of features in common with odNEAT. The goal of rtNEAT is similar to that of odNEAT in online evolution. rtNEAT was developed to allow nonplayer characters to optimise their behaviour during a game. We experimentally compare the performance of odNEAT with the performance of rtNEAT and IM-$(\mu + 1)$ in three simulation-based experiments involving groups of e-puck-like robots (Mondada et al., 2009). The first experiment, the aggregation task, requires individual search and coordinated movement to form and remain in a single group. The task is challenging given the robots' limited sensing capabilities. The second experiment, the integrated navigation and obstacle avoidance task, is used as a means to analyse odNEAT's behaviour when two conflicting objectives have to be learned. The task implies an integrated set of actions, and consequently a trade-off between avoiding obstacles and maintaining speed and forward movement. The third experiment, the phototaxis task, is performed in a dynamic environment with changing task parameters. The main conclusion is that odNEAT is an efficient and robust online neuroevolution algorithm. odNEAT yields performance levels comparable to rtNEAT despite being completely decentralised, and odNEAT significantly outperforms IM-($\mu + 1$). We furthermore show that (1) odNEAT allows robots to continuously adapt to different environmental conditions and to the presence of faults injected in the sensors, and (2) odNEAT evolves controllers with relatively low complexity and superior generalisation performance.

The article is organised as follows. In Section 2 we discuss previous studies on online evolution in multirobot systems and introduce NEAT, rtNEAT, and IM-($\mu + 1$). In Section 3 we present and motivate the features of odNEAT. In Section 4 we explain our experimental methodology and the three tasks used in this study. In Section 5 we present and discuss our experimental results. Section 6 is dedicated to the exploration and analysis of odNEAT in terms of long-term self-adaptation, fault tolerance, and the influence of each algorithmic component on performance. Concluding remarks and directions for future research are provided in Section 7.

2 Background and Related Work

In this section, we review the main approaches in the literature for online evolution of ANN-based controllers in multirobot systems, and the main characteristics of NEAT, rtNEAT, and IM-($\mu + 1$).

2.1 Online Evolutionary Robotics

Existing approaches to online evolution in multirobot systems fall into three categories (Eiben et al., 2010a): (1) distributed evolution, (2) encapsulated evolution, and (3) a hybrid approach, similar to a physically distributed island model.

2.1.1 Distributed Evolution

In distributed evolution, each robot carries a single genotype. The evolutionary process takes place when robots meet and exchange genetic information. The first attempt at truly autonomous continuous evolution in multirobot systems was performed by Watson et al. (1999, 2002) and entitled *embodied evolution*. Robots probabilistically broadcast a part of their stochastically mutated genome at a rate proportional to their fitness. Robots that receive gene transmissions incorporate this genetic material into their genome with a probability inversely proportional to their fitness. This way, selection and variation operators are implemented in a distributed manner through the interactions between robots.

Following Watson et al.'s initial publications on embodied evolution, a number of improvements and extensions were proposed; see Bianco and Nolfi (2004) and Karafotias et al. (2011) for examples. The distribution of an evolutionary algorithm in a population of autonomous robots offered the first demonstration of evolution as a continuous adaptation process. The main disadvantage of embodied evolution approaches is that the improvement of existing solutions is only based on the exchange of genetic information between robots. In large and open environments, where encounters may be rare or frequent transmission of information may be infeasible, the evolutionary process is thus prone to stagnation.

2.1.2 Encapsulated Evolution

A complementary approach, encapsulated evolution, overcomes stagnation, as each robot maintains a population of genotypes stored internally and runs its self-sufficient instance of the evolutionary algorithm locally. Alternative controllers are executed sequentially and their fitness is measured. Within this paradigm, robots adapt individually without interacting or exchanging genetic material with other robots.

In order to enable efficient encapsulated evolution, Bredèche et al. (2009) proposed the (1+1)-online algorithm after the classic (1+1)-evolutionary strategy. Montanier and Bredèche (2011) then extended the (1+1)-online algorithm to incorporate a restart procedure as a means to bypass local optima. Between those two studies, Haasdijk et al. (2010) proposed the $(\mu + 1)$-online algorithm. The algorithm was evaluated for the ability to address noisy evaluation conditions and to produce solutions within a satisfactory period of time. With respect to Haasdijk et al.'s algorithm, subsequent studies examined (1) distinct self-adaptive mechanisms for controlling the mutation operator (Eiben et al., 2010b), (2) racing as a technique to cut short the evaluation of poor individuals (Haasdijk et al., 2011), and (3) the impact of a number of parameters on performance (Haasdijk et al., 2012). Although a number of successive contributions have used encapsulated evolution, the main drawback of the approach is that there is no transfer of knowledge or exchange of genetic information between robots, which can accelerate online evolution (Huijsman et al., 2011).

2.1.3 Hybrid Evolution

The two methodologies, encapsulated evolution and distributed evolution, can be combined, leading to a hybrid approach similar to an island model. Each robot optimises the internal population of genomes through intra-island variation, and genetic information between two or more robots is exchanged through inter-island migration.

Distinct approaches to hybrid online evolution have been proposed. An example of such a method is one introduced by Elfwing et al. (2005), in which task-oriented and survival-oriented criteria are combined. In this approach, robots evaluate their internal population of solutions in a time-sharing manner. During the evaluation period, the performance of a controller is assessed based on its ability to solve the task and to sustain the robot's energy reserve by finding and replacing battery packs.

With the purpose of enabling online self-organisation of behaviour in multirobot systems, Prieto et al. (2010) proposed the real-time Asynchronous Situated Coevolution (r-ASiCo) algorithm. r-ASiCo is based on a reproduction mechanism in which individual robots carry an embryo. When robots meet, the embryo is genetically modified. If a controller is unable to solve the task, the embryo is used to produce a new controller. Huijsman et al. (2011) introduced an approach based on the encapsulated $(\mu + 1)$-online algorithm and on a peer-to-peer evolutionary algorithm designed by Laredo et al. (2010). The hybrid method consistently yielded a better performance than both pure distributed evolution and pure encapsulated evolution because of its ability to leverage parallelism in multirobot systems. Despite the new approaches and algorithmic advances, one of the main limitations of existing approaches to online evolution, including all methods just described, is that neuroevolution solely adjusts the weights of the ANN, while the topology must be specified a priori by the experimenter and remains fixed during task execution.

2.2 NeuroEvolution of Augmenting Topologies

The NeuroEvolution of Augmenting Topologies (NEAT) method (Stanley and Miikkulainen, 2002) is one of the most prominent offline neuroevolution algorithms. NEAT optimises both network topologies and weighting parameters. NEAT executes with global and centralised information like canonical evolutionary algorithms, and has been successfully applied to distinct problems, outperforming several methods that use fixed topologies (Stanley and Miikkulainen, 2002; Stanley, 2004). The high performance of NEAT is due to three key features: (1) the tracking of genes with *historical markers* to enable meaningful crossover between networks with different topologies, (2) a *niching scheme* that protects topological innovations, and (3) the incremental evolution of topologies from simple initial structures, that is, *complexification*.

In NEAT the network connectivity is represented through a flexible genetic encoding. Each genome contains a list of neuron genes and a list of connection genes. Connection genes encompass (1) references to the two neuron genes being connected, (2) the weight of the connection gene, (3) one bit indicating if the connection gene should be expressed or not, and (4) a *global innovation number*, unique for each gene in the population. Innovation numbers are assigned sequentially, and they therefore represent a chronology of the genes introduced. Genes that express the same feature are called *matching* genes. Genes that do not match are either *disjoint* or *excess*, depending on whether they occur within or outside the range of the other parent's innovation numbers. When crossover is performed, matching genes are aligned. The NP-hard problem of matching distinct network topologies is thus avoided, and crossover can be performed without a priori topological analysis. In terms of mutations, NEAT allows for classic connection weight and neuron bias perturbations, and structural changes that lead to the insertion of either a new connection between two previously unconnected neurons or to a new neuron. A new neuron gene, representing the new neuron in the ANN, is introduced in the genome by splitting an old connection gene into two new connection genes.

NEAT protects new structural innovations by reducing competition between genomes representing differing structures and network complexities. The niching scheme is composed of speciation and fitness sharing. *Speciation* divides the population into nonoverlapping sets of similar genomes based on the amount of evolutionary history they share. *Explicit fitness sharing* dictates that individuals in the same species share the fitness of their niche. The fitness scores of members of a species are first *adjusted*, that is, divided by the number of individuals in the species. Species then grow or shrink in size depending on whether their average adjusted fitness is above or below the population average. Since the size of the species is taken into account in the computation of the adjusted fitness, new smaller species are not discarded prematurely, and one species does not dominate the entire population.

NEAT starts with a population of simple networks with no hidden neurons. New neurons and new connections are then progressively added to the networks as a result of structural mutations. Because NEAT speciates the population, the algorithm effectively maintains a variety of networks with different structures and different complexities over the course of evolution. In this way, NEAT can search for an appropriate degree of complexity to the current task.

2.3 rtNEAT: Real-Time NEAT

Real-time NeuroEvolution of Augmenting Topologies (rtNEAT) was introduced by Stanley et al. (2005) with the purpose of evolving ANNs online. rtNEAT is a centralised real-time version of NEAT originally designed for video games. rtNEAT differs from NEAT in a number of aspects, namely, (1) rtNEAT is a steady state algorithm, while NEAT is generational, (2) rtNEAT produces one offspring at regular intervals, every n time steps, and (3) unlike NEAT, in which the number of species may vary, rtNEAT attempts to keep the number of species constant. To that end, rtNEAT adjusts a threshold C_t that determines the degree of topological similarity necessary for individuals to belong to a species. When there are too many species, C_t is increased to make species more inclusive; when there are too few, C_t is decreased to make species less inclusive. Despite these differences, rtNEAT has been shown to preserve the dynamics of NEAT, namely, complexification and protection of innovation through speciation (Stanley et al., 2005).

Even though rtNEAT only creates one offspring at a time, it approximates NEAT's behaviour, in which a number of offspring n_k is assigned to each species. In rtNEAT, a parent species S_k is chosen proportionally to its average adjusted fitness, as follows:

$$Pr(S_k) = \frac{\overline{F}_k}{\overline{F}_{\text{total}}}, \tag{1}$$

where $\overline{F}_k$ is the average adjusted fitness of species k, and $\overline{F}_{\text{total}}$ is the sum of all species' average adjusted fitness. As a result, the expected number of offspring for each species in rtNEAT is, over time, proportional to n_k in traditional NEAT. The remaining process of reproduction is similar to NEAT's: Two parents are chosen, and crossover and mutation operators are applied probabilistically. The individual with the lowest adjusted fitness is then removed from the population and replaced with the new offspring.

Like rtNEAT, odNEAT is based on steady state dynamics. We therefore use rtNEAT instead of the original, generational NEAT for comparisons of features and performance.

2.4 IM-($\mu + 1$) Algorithm

To the best of our knowledge, only one method has previously been proposed (Schwarzer et al., 2011) that optimises both the weighting parameters and the topology of ANN-based controllers according to a physically distributed island model. The algorithm is a variant of the ($\mu + 1$)-online algorithm, proposed by Haasdijk et al. (2010), with random ad hoc exchange of genomes between robots. We henceforth refer to this method as IM-($\mu + 1$). We first describe the similarities between IM-($\mu + 1$) and rtNEAT and then the execution loop of IM-($\mu + 1$).

The IM-($\mu + 1$) algorithm mimics a number of features of rtNEAT. First, new genes are assigned innovation numbers. An important difference with respect to rtNEAT is that in IM-($\mu + 1$) innovation numbers are local and randomly chosen from a predefined interval to enable a decentralised implementation. The interval $[1, 1000]$ was used in Schwarzer et al. (2011). Second, as in rtNEAT, crossover is performed between similar genomes to maximise the chance of producing meaningful offspring. Explicit speciation is not used in IM-($\mu + 1$). Instead, genomes that have a high degree of *genetic similarity* are recombined. Genetic similarity is computed based on the ratios of matching connection genes and matching neuron genes.

Third, IM-($\mu + 1$) follows rtNEAT's complexification policy and usually starts evolution from simple topologies, although it can be seeded with a parametrisable number of hidden neurons (Schwarzer et al., 2011). IM-($\mu + 1$) allows for nonstructural mutations similarly to rtNEAT, namely, connection weight and neuron bias perturbations, but it employs different methods for optimising the topology of the ANN. Structural mutation of a connection gene can either remove the gene or introduce a new connection gene in a random location and with a randomly assigned weight. In the same way, structural mutation of a neuron gene can either delete the gene or produce a new neuron gene with a random innovation number and a random bias value. In addition, insertion of new neuron genes and connection genes can be performed through duplication and differentiation of an existing neuron gene, and of its incoming and outgoing connection genes.

2.4.1 Execution Loop

In the IM-($\mu + 1$) algorithm, as in the encapsulated ($\mu + 1$)-online, each robot maintains an internal population of μ genomes. Robots in close proximity can exchange genomes according to a migration policy that presupposes bilateral and synchronous communication. When two robots meet, they exchange genomes if neither of them has had a migration within a predefined period of time. Each robot randomly selects and removes one genome from its internal population and probabilistically applies crossover and mutation. The resulting genome is transmitted to the neighbouring robot, and the genome received is incorporated in the internal population.

During task execution, $\lambda = 1$ new offspring is produced at regular time intervals. One genome from the population of μ genomes maintained by the robot is randomly selected to be a parent. A second parent is used if any genome in the population is considered genetically similar. The offspring resulting from crossover and mutation is decoded into an ANN that controls the robot. The controller operates for a fixed amount of time called the *evaluation period*. When the evaluation period elapses, the evaluated genome is added to the population of the robot. The genome with the lowest fitness score is subsequently removed to keep the population size constant.

3 Description of odNEAT

In this section, we present Online Distributed NeuroEvolution of Augmenting Topologies (odNEAT), an online neuroevolution algorithm for multirobot systems in which both the weighting parameters and the topology of the ANNs are under evolutionary control. One of the motivations behind odNEAT is that by evolving neural topologies, the algorithm bypasses the inherent limitations of fixed-topology online algorithms. As in rtNEAT, odNEAT starts with simple controllers, in which the input neurons are connected to the output neurons. The ANNs are progressively augmented with new neurons and new connections, and a suitable network topology is the product of a continuous evolutionary process. In addition, odNEAT is distributed across multiple robots that evolve in parallel and exchange solutions.

odNEAT adopts rtNEAT's matching and recombination of neural topologies and niching scheme, and presents a number of differences when compared with IM-($\mu + 1$). The main differences between the three algorithms are summarised in Table 1. odNEAT differs from rtNEAT and IM-($\mu + 1$) in a number of key aspects:

- The internal population of each robot is constructed in an incremental manner.
- odNEAT maintains a local tabu list of recent poor solutions, which enables a robot to filter out genomes representing controllers similar to those that failed recently.

Table 1: Summary of the main differences between odNEAT, rtNEAT, and IM-($\mu + 1$).

	odNEAT	rtNEAT	IM-($\mu + 1$)
Population	Distributed	Centralised	Distributed
Niching scheme	Yes	Yes	No
Tabu list	Yes	No	No
Migration policy	Unilateral	n/a	Synchronous
Innovation numbers	High-res. timestamps	Sequential	Random
Controller replacement	When previous fails	Periodically	Periodically
Maturation period	Yes	No	No

- odNEAT follows a unilateral migration policy for the exchange of genomes between robots. Copies of the genome encoding the active controller are probabilistically transmitted to nearby robots if they have a competitive fitness score and represent topological innovations with respect to the robot's internal population of solutions.
- odNEAT uses local, high-resolution timestamps. Each robot is responsible for assigning a timestamp to each local innovation, be it a new connection gene or a new neuron gene. The use of high-resolution timestamps for labels practically guarantees uniqueness and allows odNEAT to retain the concept of chronology.
- A controller remains active as long as it is able to solve the task. A new controller is only synthesised if the current one fails, that is, when it is actually necessary.
- New controllers are given a minimum amount of time controlling the robot, a *maturation period*.

3.1 Measuring Individual Performance

In odNEAT robots maintain a virtual energy level reflecting the individual task performance of the current controller. Controllers are assigned a default and domain-dependent virtual energy level when they start executing. The energy level increases and decreases as a result of the robot's behaviour. If a controller's energy level reaches a minimum threshold, a new controller is produced.

One common issue, especially for highly complex tasks, is that online evaluation is inherently noisy. Very dissimilar conditions may be presented to different genomes when they are translated into a controller and evaluated. The location of a robot within the environment and the proximity to other robots, for instance, are factors that may have a significant influence on performance and behaviour. With the purpose of obtaining a more reliable fitness estimate, odNEAT distinguishes between the fitness score of a solution and its current energy level. The fitness score is defined as the average of the virtual energy level, sampled at regular time intervals.

3.2 Internal Population, Exchange of Genomes, and Tabu List

In odNEAT each robot maintains a local set of genomes in an internal population. The internal population is subject to speciation and fitness sharing. The population contains genomes generated by the robot, namely, the current genome and previously active genomes that have competitive fitness scores, and genomes received from other robots.

Every control cycle, a robot probabilistically broadcasts a copy of its active genome and the corresponding virtual energy level to robots in its immediate neighbourhood, an *inter-robot reproduction event*, with a probability computed as follows:

$$P(event) = \frac{\overline{F}_k}{\overline{F}_{\text{total}}}, \tag{2}$$

where $\overline{F}_k$ is the average adjusted fitness of local species k to which the genome belongs, and $\overline{F}_{\text{total}}$ is the sum of all local species' average adjusted fitnesses. The broadcast probability promotes the propagation of topological innovations with a competitive fitness.

The internal population of each robot is updated according to two principles. First, a robot's population does not allow for multiple copies of the same genome. Because of the exchange of genetic information between the robots, a robot may receive a copy of a genome that is already present in its population. Whenever a robot receives a copy C' of a genome C, the energy level of C' is used to incrementally average the fitness of C and thus to provide the receiving robot with a more reliable estimate of the genome's fitness. Second, whenever the internal population is full, the insertion of a new genome is accompanied by the removal of the genome with the lowest adjusted fitness score. If a genome is removed or added, the corresponding species has either increased or decreased in size, and the adjusted fitness of the species is recalculated.

Besides the internal population, each robot also maintains a local tabu list, a short-term memory that keeps track of recent poor solutions, namely, (1) genomes removed from the internal population because it got full, or (2) genomes whose virtual energy level reached the minimum threshold and therefore failed to solve the task. In the latter case, the genome is added to the tabu list but is maintained in the internal population as long as its fitness is comparatively competitive.

The tabu list filters out solutions broadcast by other robots that are similar to those that have already failed. The purpose of the tabu list is (1) to avoid flooding the internal population with poor solutions, and (2) to keep the evolutionary process from cycling around in an unfruitful neighbourhood of the solution space. Received genomes must first be checked against the tabu list before they become part of the internal population. Received genomes are only included in the internal population if they are topologically dissimilar from all genomes in the tabu list.

3.3 New Genomes and the Maturation Period

When the virtual energy level of a controller reaches the minimum threshold because it is incapable of solving the task, a new genome is created. In this process, an *intra-robot reproduction event*, a parent species is chosen proportionally to its average adjusted fitness, as defined in Equation (2). Two parents are selected from the species, each one via a tournament selection of size 2. The offspring is created based on crossover of the parents' genomes and mutation of the new genome. Once the new genome is decoded into a new controller, it is guaranteed a maturation period during which no controller replacement takes place. The new controller can continue to execute after the maturation period if its energy level is above the threshold.

odNEAT's maturation period is conceptually similar to the recovery period of the $(1+1)$-online algorithm (Bredèche et al., 2009), and its purpose is twofold. First, the maturation period is important because the new controller may be an acceptable solution to the task but the environmental conditions in which it starts to execute may be unfavourable. Thus, the new controller is *protected* for a minimum period of time.

Algorithm 1 Pseudocode of odNEAT that runs independently on every robot. The experimental parameters of odNEAT and the robot capabilities are detailed in Section 4.1.

```
$genome \leftarrow create\_random\_genome()$
$controller \leftarrow assign\_as\_controller(genome)$
$energy \leftarrow default\_initial\_energy$
**loop**
    **if** $broadcast?$ **then**
        $send(genome, robots\_in\_communication\_range)$
    **end if**
    **if** $has\_received?$ **then**
        **for all** $c$ **in** $received\_genomes$ **do**
            **if** $tabu\_list\_approves(c)$ **and** $population\_accepts(c)$ **then**
                $add\_to\_population(c)$
                $adjust\_population\_size()$
                $adjust\_species\_fitness()$
            **end if**
        **end for**
    **end if**
    $operate\_in\_environment()$
    $energy \leftarrow update\_energy\_level()$
    **if** $energy \leq minimum\_energy\_threshold$ **and** $not(in\_maturation\_period?)$ **then**
        $add\_to\_tabu\_list(genome)$
        $offspring \leftarrow generate\_offspring()$
        $update\_population(offspring)$
        $genome \leftarrow replace\_genome(offspring)$
        $controller \leftarrow assign\_as\_controller(genome)$
        $energy \leftarrow default\_initial\_energy$
    **end if**
**end loop**
```

Second, the maturation period defines a lower bound of activity in the environment that should be sufficiently long to enable a proper estimate of the quality of the controller. This implies a trade-off between a minimum evaluation time and how many candidate solutions can be evaluated within a given amount of time. A longer maturation period increases the reliability of the fitness estimate of solutions that fail, that is, of the stepping stones of the evolutionary process, while a shorter maturation period increases the number of solutions that can be evaluated within a given period of time. Algorithm 1 summarises odNEAT as executed by each robot.

4 Experimental Methodology

In this section, we define our experimental methodology, including the simulation platform and robot model, and we describe the three tasks used in the study: aggregation, phototaxis, and integrated navigation and obstacle avoidance. Our experiments serve as a means to determine if and how odNEAT evolves controllers for solving the specified tasks, the complexity of solutions evolved, and the efficiency of odNEAT when compared with rtNEAT and IM-($\mu + 1$).

4.1 Experimental Setup

We use JBotEvolver (Duarte et al., 2014) to conduct our simulated experiments. JBotEvolver is an open source, multirobot simulation platform and neuroevolution framework. The simulator is written in Java and implements 2D differential drive kinematics.

In our experimental setup, the simulated robots are modelled after the e-puck (Mondada et al., 2009), a 7.5 cm in diameter differential drive robot capable of moving

Table 2: Summary of experimental details.

Group size	5 robots
Broadcast range	25 cm
Control cycle frequency	100 ms
Arena size	3 x 3 meters
Simulation length	100 hours
Runs per configuration	30
odNEAT population size	40 genomes per internal population
IM-($\mu + 1$) population size	40 genomes per internal population
rtNEAT population size	200 genomes
Neural network weight range	$[-10, 10]$
Inputs range	$[0, 1]$
Outputs range	$[0, 1]$, rescaled to $[-1, 1]$
Activation function	Logistic function

at speeds up to 13 cm/s. Each robot is equipped with infrared sensors that multiplex obstacle sensing and communication between robots at a range of up to 25 cm.[1] The sensors of the robots are used in the three tasks to detect walls and other robots. In real e-pucks, the wall sensors can be implemented with active infrared sensors, while the robot sensors can be implemented using the e-puck range and bearing board (Gutiérrez et al., 2008). In the phototaxis task, robots are also given the ability to detect the light source, which can be provided by the fly-vision turret or by the omnidirectional vision turret.[2] Each sensor and each actuator are subject to noise, which is simulated by adding a random Gaussian component within ±5% of the sensor saturation value or actuation value. Each robot also has an internal sensor that enables it to perceive its current virtual energy level.

Each robot is controlled by an ANN produced by the evolutionary algorithm being tested. The controllers are discrete-time recurrent neural networks with connection weights $\in [-10, 10]$. The inputs of the ANN are the normalised readings $\in [0, 1]$ from the sensors. The output layer has two neurons, whose values are linearly scaled from $[0, 1]$ to $[-1, 1]$. The scaled output values are used to set the signed speed of each wheel (positive in one direction, negative in the other). The activation function is the logistic function. The three algorithms start with simple networks with no hidden nodes, and with each input neuron connected to every output neuron. Evolution is therefore responsible for adding new neurons and new connections, both feedforward and recurrent.

A group of five robots operates in a square arena surrounded by walls. The size of the arena was chosen to be 3 x 3 meters. Every 100 ms of simulated time, each robot executes a control cycle. Table 2 summarises the characteristics common to the three tasks. Regarding the configuration of the evolutionary algorithms, the parameters are the same for odNEAT, rtNEAT, and IM-($\mu + 1$): crossover rate, 0.25; mutation rate, 0.1; add neuron rate, 0.03; add connection rate, 0.05; and weight mutation magnitude, 0.5.

[1]The original e-puck infrared range is 2–3 cm (Mondada et al., 2009). In real e-pucks, the *liblrcom* library, available at http://www.e-puck.org, extends the range up to 25 cm and multiplexes infrared communication with proximity sensing.

[2]See the Extensions section at http://www.e-puck.org/.

The parameters were found experimentally. New connections have to be added more frequently than new neurons, and evolution tends to perform better if neural networks are recombined and augmented in a parsimonious manner. odNEAT is configured with a maturation period of 500 control cycles (50 seconds). Preliminary experiments showed this parameter value to provide an appropriate trade-off between the reliability of fitness estimates and the speed of convergence toward good solutions. A solution that was previously found to be poor is removed from the tabu list if a similar genome is not among the last 50 genomes received by the robot. Performance is robust to moderate variations in the parameters.

4.2 Preliminary Performance Tuning

We conducted a series of preliminary tests across all three tasks as a means to analyse the algorithms' dynamics. First, we tested both odNEAT and rtNEAT with (1) a dynamic compatibility threshold and a target number of species from 1 to 10, and (2) a fixed compatibility threshold $\delta = 3.0$ for speciating the population, with corresponding coefficients set as in previous studies using NEAT (Stanley and Miikkulainen, 2002; Stanley, 2004). Results were found to be similar, and we therefore decided to use a fixed compatibility threshold.

Second, we verified that the periodic production of new controllers in rtNEAT and in IM-$(\mu + 1)$ led to incongruous group behaviour. Regular substitution of controllers caused the algorithms to perform poorly in collective tasks that explicitly required continuous group coordination and cooperation, such as the aggregation task. In IM-$(\mu + 1)$ rupture of collective behaviour was caused by having every robot changing to a new controller at the same time. In rtNEAT rupture of collective behaviour was still significant but less accentuated, as only one controller was substituted at regular time intervals. We compared the average fitness score of consecutive groups of controllers executed on the robots. In the majority of the substitutions, the performance of new controllers was worse than the performance of the previous ones. In these cases, differences in the fitness scores were statistically significant in the aggregation task for rtNEAT and for IM-$(\mu + 1)$ ($\rho < .01$, Mann-Whitney) and in the phototaxis task for IM-$(\mu + 1)$ ($\rho < .05$).

To provide a fair and meaningful comparison of performance, we modified the condition for creating new offspring. rtNEAT and IM-$(\mu + 1)$ were modified to produce offspring when a robot's energy level reaches the minimum threshold (zero in our experiments), that is, when a new controller is necessary. In the IM-$(\mu + 1)$ algorithm, we also observed collisions in the assignment of innovation numbers. To reduce the chance of collisions, we replaced the random assignment of innovation numbers by high-resolution timestamps, as in odNEAT. We verified that the modified versions of rtNEAT and IM-$(\mu + 1)$ consistently outperformed their original, nonmodified counterparts by evolving final solutions with higher fitness scores ($\rho < .05$, Mann-Whitney in the aggregation and phototaxis tasks) and that were found in fewer evaluations on average.

4.3 Aggregation

In an aggregation task, dispersed agents must move close to one another so that they form a single cluster. Aggregation plays an important role in a number of biological systems (Camazine et al., 2001). For instance, several social animals use aggregation to increase their chances of survival or as a precursor of other collective behaviours. We study aggregation because it combines different aspects of multirobot tasks, namely, distributed individual search, coordinated movement, and cooperation. Aggregation is

Table 3: Aggregation controller details.

Input neurons: 18
8 for infrared robot detection (range: 25 cm)
8 for infrared wall detection (range: 25 cm)
1 for energy level reading
1 for reading the number of different genomes received in the last $P = 10$ control cycles
Output neurons: 2
Left and right motor speeds

also related to a number of real-world problems. For instance, in robotics, self-assembly and collective transport of heavy objects require aggregation at the site of interest (Gross and Dorigo, 2009).

In the initial configuration, robots are placed in random positions at a minimum distance of 1.5 meters between neighbours. Robots are evaluated based on a set of criteria that include the presence of robots nearby and the ability to explore the arena and move fast. The initial virtual energy level of each controller E is set to 1,000 and limited to the range [0, 2,000] units. At each control cycle, E is updated according to the following equation:

$$\frac{\Delta E}{\Delta t} = \alpha(t) + \gamma(t), \tag{3}$$

where t is the current control cycle, and $\alpha(t)$ is a reward proportional to the number n of different genomes received in the last $P = 10$ control cycles. In our experiments, we use $\alpha(t) = 3 \cdot n$ energy units. As robots executing odNEAT exchange candidate solutions to the task, the number of different genomes received is used to estimate the number of robots nearby. The second component of the equation, $\gamma(t)$, is a factor related to the quality of movement computed as

$$\gamma(t) = \begin{cases} -1 & \text{if } v_l(t) \cdot v_r(t) < 0, \\ \Omega_s(t) \cdot \omega_s(t) & \text{otherwise,} \end{cases} \tag{4}$$

where $v_l(t)$ and $v_r(t)$ are the left and right wheel speeds, $\Omega_s(t)$ is the ratio between the average and maximum speed, and $\omega_s(t) = \sqrt{v_l(t) \cdot v_r(t)}$ rewards controllers that move fast and straight at each control cycle. Robot and controller details are summarised in Table 3. To determine when robots have aggregated, we sample the number of robot clusters at regular intervals throughout the simulation, as in Bahgeçi and Sahin (2005). Two robots are part of the same cluster if the distance between them is less than or equal to their sensor range (25 cm).

4.4 Integrated Navigation and Obstacle Avoidance

Navigation and obstacle avoidance is a classic task in evolutionary robotics. Robots have to simultaneously move as straight as possible, maximise wheel speed, and avoid obstacles. The task is typically studied using only one robot. In multirobot experiments, each robot is an additional, moving obstacle for the other robots. In a confined environment, such as our enclosed arena, this task implies an integrated set of actions and consequently a trade-off between avoiding obstacles in sensor range and maintaining speed and forward movement. We study navigation and obstacle avoidance because it is an essential feature for autonomous robots operating in real-world environments, and

Table 4: Navigation and obstacle avoidance controller details.

Input neurons: 17
8 for infrared robot detection (range: 25 cm)
8 for infrared wall detection (range: 25 cm)
1 for energy level reading
Output neurons: 2
Left and right motor speeds

because it provides the basis for more sophisticated behaviours such as path planning or traffic rules (Cao et al., 1997).

Initially, robots are placed in random locations and with random orientations, drawn from a uniform distribution. The virtual energy level E is limited to the range $\in [0, 100]$ units. When the energy level reaches zero, a new controller is generated and assigned the default energy value of 50 units. During task execution, E is updated every 100 ms according to the following equation, adapted from Floreano and Mondada (1994):

$$\frac{\Delta E}{\Delta t} = f_{\text{norm}}(V \cdot (1 - \sqrt{\Delta v}) \cdot (1 - d_r) \cdot (1 - d_w)), \tag{5}$$

where V is the sum of rotation speeds of the two wheels, with $0 \leq V \leq 1$; $\Delta v \in [0, 1]$ is the normalised absolute value of the algebraic difference between the signed speed values of the wheels; and d_r and d_w are the highest activation values of the infrared sensors for robot detection and for wall detection, respectively. Both d_r and d_w are normalised to a value between 0 (no robot/wall in sight) and 1 (collision with a robot or wall).

The four components encourage respectively, motion, straight line displacement, robot avoidance, and wall avoidance. The function f_{norm} maps linearly from the domain $[0, 1]$ into $[-1, 1]$. Robot and controller details are summarised in Table 4.

4.5 Phototaxis

In a phototaxis task, robots have to search for and move toward a light source. We use a dynamic version of the phototaxis task. Every 5 minutes of simulated time, the light source is moved instantaneously to a new random location. In this way, robots have to continuously search for and reach the light source, which eliminates behaviours that find the light source by chance. As in the navigation task, robots start at random locations and with random orientations drawn from a uniform distribution. The virtual energy level $E \in [0, 100]$ units, and controllers are assigned an initial value of 50 units. At each control cycle, the robot's virtual energy level E is updated as follows:

$$\frac{\Delta E}{\Delta t} = \begin{cases} S_r & \text{if } S_r > 0.5, \\ 0 & \text{if } 0 < S_r \leq 0.5, \\ -0.01 & \text{if } S_r = 0, \end{cases} \tag{6}$$

where S_r is the maximum value of the readings from light sensors, between 0 (no light) and 1 (brightest light). Light sensors have a range of 50 cm, and robots are therefore only rewarded if they are close to the light source. Remaining IR sensors detect nearby walls and robots within a range of 25 cm. Details are summarised in Table 5.

Table 5: Phototaxis controller details.

Input neurons: 25
8 for infrared robot detection (range: 25 cm)
8 for infrared wall detection (range: 25 cm)
8 for light source detection (range: 50 cm)
1 for energy level reading
Output neurons: 2
Left and right motor speeds

5 Results

In this section, we present and discuss the experimental results. We compare the performance of odNEAT with the performance of rtNEAT and of IM-$(\mu + 1)$ in the three tasks. We analyse (1) the number of evaluations, that is, the number of controllers tested by each robot before one that solves the task is found, (2) the task performance in terms of fitness score, (3) the complexity of solutions evolved, and (4) their generalisation capabilities. The number of evaluations is measured because it is independent of the potentially different evaluation times used by the algorithms and thus ensures a meaningful comparison. We use the two-tailed Mann-Whitney test to compute statistical significance of differences between sets of results because it is a nonparametric test, and therefore no strong assumptions need to be made about the underlying distributions. Success rates are compared using the two-tailed Fisher's exact test, a nonparametric test suitable for this purpose (Fisher, 1925).

For each of the three tasks considered and each algorithm evaluated, we conduct 30 independent evolutionary runs. Because multiple comparisons are performed using the results obtained in each set of 30 runs (number of evaluations, fitness scores, complexity of solutions evolved, and their generalisation capabilities), we adjust the ρ value using the two-stage Hommel method (Hommel, 1988), a more powerful and less conservative modification of the classic Bonferroni correction method (Blakesley, 2008).

5.1 Comparing odNEAT and rtNEAT

We start by comparing the performance of odNEAT and rtNEAT to evaluate the quality of our algorithm with respect to a centralised state-of-the-art neuroevolution method. Table 6 summarises the number of evaluations and fitness scores of controllers that solve the task for both odNEAT and rtNEAT. In terms of evaluations, results show comparable performance even though there is a slight advantage in favour of rtNEAT. On average, odNEAT requires each robot to evaluate approximately 6 to 8 controllers more than rtNEAT. Differences in the number of evaluations are not statistically significant ($\rho \geq .05$, Mann-Whitney).

With respect to the fitness scores, both odNEAT and rtNEAT typically evolve high-performing controllers, as summarised in Table 6.[3] In two of the three tasks assessed, odNEAT tends to outperform rtNEAT. In the aggregation task and in the navigation task, odNEAT produces better-performing controllers ($\rho < .05$, Mann-Whitney). In the phototaxis task, rtNEAT evolves superior controllers ($\rho < .05$, Mann-Whitney).

[3]The fitness scores for the aggregation task shown in Table 6 are linearly mapped from [0, 2,000] to [0, 100], which is the range of the fitness score for the other two tasks.

Table 6: Comparison of the number of evaluations and fitness scores of solutions to the task (out of 100) between odNEAT and rtNEAT.

Task	Method	No. of Evaluations	Fitness Score
Aggregation	odNEAT	103.7 ± 80.9	89.2 ± 4.8
	rtNEAT	95.5 ± 60.4	83.4 ± 11.9
Navigation	odNEAT	23.6 ± 19.2	93.0 ± 9.2
	rtNEAT	17.6 ± 11.1	89.9 ± 11.4
Phototaxis	odNEAT	40.9 ± 24.1	85.7 ± 6.4
	rtNEAT	34.8 ± 16.3	89.6 ± 6.3

Values listed are the average ± standard deviation over 30 independent runs for each experimental configuration.

odNEAT differs from rtNEAT in the sense that it has a distributed and decentralised nature and therefore does not assess the group-level information from the global perspective. To unveil the evolutionary dynamics of the two algorithms, we first analyse the stepping stones that lead to the final solutions, that is, how the evolutionary search proceeds through the high-dimensional genotypic search space for the respective algorithms. A number of methods have been proposed to perform such analysis (Kim and Moon, 2003; Vassilev et al., 2000), and they are mainly based on visualisations of the structure of fitness landscapes or the fitness score of the best individuals over time. To visualise the intermediate genomes produced by odNEAT and rtNEAT, and how the algorithms traverse the search space with respect to each other, we use *Sammon's nonlinear mapping* (Sammon, 1969). Unlike visualisation and dimensionality reduction techniques such as principal components analysis (Kittler and Young, 1973) and self-organising maps (Kohonen, 1982), Sammon's mapping aims to preserve the original distances between elements in the mapping to a lower dimension.

Sammon's mapping performs a point mapping of high-dimensional data to two- or three-dimensional spaces, such that the structure of the data is approximately preserved. The algorithm minimises the error measure E_m, which represents the disparity between the high-dimensional distances δ_{ij} and the resulting distance d_{ij} in the lower dimension for all pairs of points i and j. E_m can be minimised by a steepest descent procedure to search for a minimum error value, and it is computed as follows:

$$E_m = \frac{1}{\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \delta_{ij}} \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{(\delta_{ij} - d_{ij})^2}{\delta_{ij}}. \tag{7}$$

Using Sammon's mapping, we project the genomes representing intermediate controllers tested over the course of evolution by odNEAT and rtNEAT. In order to obtain a clearer visualisation, we do not map all genomes produced during the course of evolution. Instead, we only map genomes that have a genomic distance $\delta > 4.5$ when compared with already recorded genomes. This criterion was found experimentally to ensure that the two-dimensional space has a representative selection of genomes while being readable. The output of the mapping is x and y coordinates for every genome. The distance in the high-dimensional space δ_{ij} between two genomes i and j is based on genomic distance as used by odNEAT and rtNEAT for speciation. On the other hand, the distance between two points in the two-dimensional visualisation space is computed as their Euclidean distance.

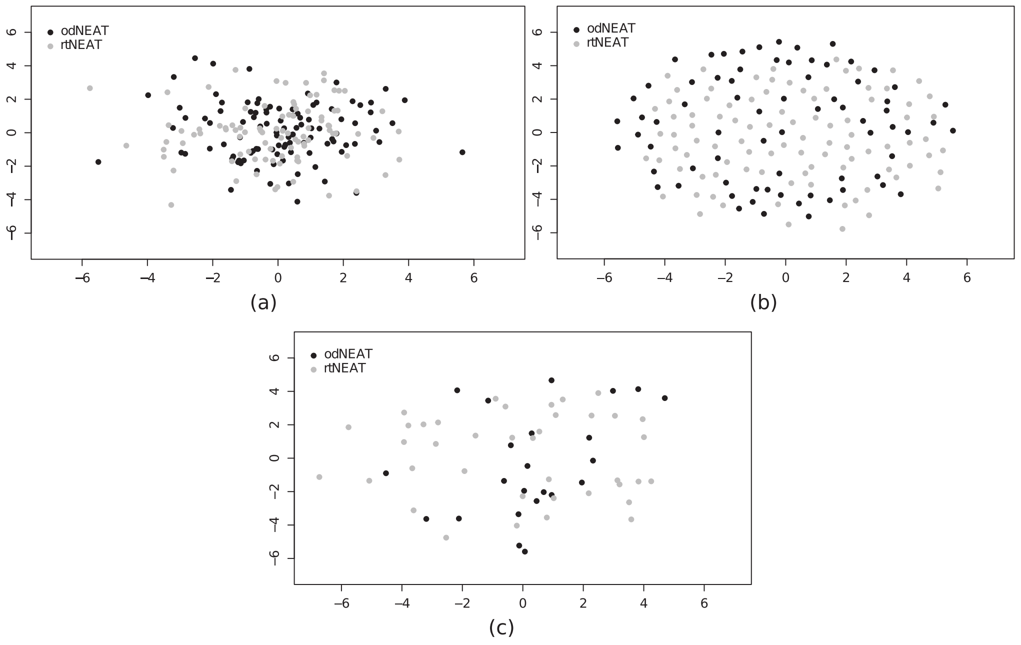

Figure 1: Sammon's mapping. (a) Aggregation task. Sammon's mapping of 103 genotypes evolved by odNEAT (black) and 95 genotypes evolved by rtNEAT (gray). (b) Navigation and obstacle avoidance task. Sammon's mapping of 72 genotypes evolved by odNEAT (black) and 91 genotypes evolved by rtNEAT (gray). (c) Phototaxis task. Sammon's mapping of 21 genotypes evolved by odNEAT (black) and 39 genotypes evolved by rtNEAT (gray).

Figure 1 shows the Sammon's mapping for the three tasks. The error values are $E_m = 0.086$ for the aggregation task, $E_m = 0.075$ for the navigation and obstacle avoidance task, and $E_m = 0.082$ for the phototaxis task. The low error values indicate that the distances between genomes are well preserved by the mapping. In the aggregation task, Sammon's mapping shows a similar exploration of the search space. odNEAT and rtNEAT explore identical regions in the two-dimensional space and evolve a comparable number of genomes matching those regions: 103 evolved by odNEAT versus 95 evolved by rtNEAT. On the other hand, in the navigation task, odNEAT evolves fewer genomes matching the analysed regions of the search space: 72 versus 91 evolved by rtNEAT. That is, rtNEAT covers more regions of the genotypic search space. A similar trend is also observed in the phototaxis task, in which rtNEAT evolves 39 genomes versus 21 evolved by odNEAT.

Since odNEAT relies exclusively on local information, the evolutionary algorithm executing on each robot tends to do a more confined exploration of the search space. In the aggregation task, robots are in close proximity and they therefore continuously exchange genomes. In such a case, the distributed and decentralised dynamics of odNEAT resemble, to some extent, the dynamics of a centralised algorithm. Hence, the exploration of the search space performed by odNEAT and rtNEAT is similar, quantitatively and qualitatively. In the other two tasks, because robots tend to be further apart, there is typically more pressure to evolve solutions by using the information available in the population of each individual robot.

Table 7: Summary of the neural complexity added through evolution from the initial topology by odNEAT and rtNEAT.

Task	Method	Connections Added	Neurons Added	Avg. C_{fp}
Aggregation	odNEAT	6.3 ± 5.6	5.5 ± 5.0	11.8
	rtNEAT	11.3 ± 11.1	9.3 ± 8.9	20.6
Navigation	odNEAT	6.6 ± 0.9	3.1 ± 0.3	9.7
	rtNEAT	10.0 ± 1.4	4.9 ± 0.7	14.9
Phototaxis	odNEAT	2.3 ± 0.7	1.1 ± 0.3	3.4
	rtNEAT	5.4 ± 1.9	2.5 ± 0.9	7.9

Connections Added and Neurons Added refer to the average ± standard deviation over 30 independent runs for each experimental configuration.

5.1.1 Neural Complexity and Generalisation Performance

To the best of our knowledge, there is no general metric for ANN complexity. We use the effective number of parameters in each network, C_{fp}, which is defined as the sum of the number of connections and the number of neurons. This measure of complexity is used in a number of heuristics for the back propagation algorithm to determine, for instance, a suitable size for the training set (Haykin, 1999). Table 7 lists the complexity reached by each evolutionary method in the final successful solutions. odNEAT's evolutionary dynamics also lead to the synthesis of simpler solutions than in rtNEAT, both in terms of neurons and connections added through evolution. Differences in the neural complexity of evolved solutions are significant in all experimental configurations ($\rho < .01$, Mann-Whitney).

In artificial neural network training, it has been shown that among a set of solutions for a given task, less complex networks tend to have better generalisation performance (Schmidhuber, 1997). To compare the generalisation capabilities of odNEAT and rt-NEAT, we restart each task 100 times per original evolutionary run. In the task restarts, each robot maintains its controller, and further evolution is not allowed. Task restarts are generalisation tests that enable us to assess if robots can continuously operate after several redeployments. The generalisation tests involve both the flexibility to solve the task starting from different initial conditions, and the ability to operate in conditions potentially not experienced during the evolutionary phase. A group of robots passes the generalisation test if it continues to solve the task, that is, if the virtual energy level of any of the robots in the group does not reach zero (see Sections 4.3, 4.4, and 4.5). Each generalisation test has a maximum duration of 100 hours of simulated time.

Table 8 lists the generalisation performance of odNEAT and rtNEAT. In general, odNEAT presents an interesting capacity to generalise and execute in different conditions. odNEAT outperforms rtNEAT by approximately 13.5 percentage points in the aggregation task and by 4.0 percentage points in the navigation task, as it successfully solves 406 and 120 tests more, respectively. Differences in successful generalisation tests are statistically significant in the two tasks ($\rho = 1.8 \cdot 10^{-15}$ in the aggregation task, and $\rho = 1 \cdot 10^{-3}$ in the navigation task, Fisher's exact test). In the phototaxis task, rtNEAT yields better generalisation performance and successfully solves 64 tests more than odNEAT, which corresponds to approximately 2.1 percentage points. Differences are considerably smaller than in the other two tasks and are not statistically significant ($\rho = .095$, Fisher's exact test).

Overall, the analysis performed in this section shows that odNEAT yields performance levels comparable to those of rtNEAT in terms of the number of evaluations

Table 8: Generalisation performance of controllers evolved by odNEAT and rtNEAT in three tasks.

Task	Method	Generalisation Performance (%)	Successful Tests
Aggregation	odNEAT	75.1 ± 32.6	2253/3000
	rtNEAT	61.6 ± 32.4	1847/3000
Navigation	odNEAT	73.4 ± 23.3	2202/3000
	rtNEAT	69.4 ± 31.9	2082/3000
Phototaxis	odNEAT	60.2 ± 33.5	1806/3000
	rtNEAT	62.3 ± 19.0	1870/3000

The Generalisation Performance refers to the average ± standard deviation of the success rate for each set of 100 task restarts.

Table 9: Comparison of the number of evaluations and fitness scores of solutions to the task (out of 100) between odNEAT and IM-($\mu + 1$).

Task	Method	No. of Evaluations	Fitness Score
Aggregation	odNEAT	103.7 ± 80.9	89.2 ± 4.8
	IM-($\mu + 1$)	100.8 ± 21.9	75.8 ± 10.5
Navigation	odNEAT	23.6 ± 19.2	93.0 ± 9.2
	IM-($\mu + 1$)	43.6 ± 10.8	89.5 ± 0.4
Phototaxis	odNEAT	40.9 ± 24.1	85.7 ± 6.4
	IM-($\mu + 1$)	91.0 ± 30.6	77.6 ± 9.9

Values listed are the average ± standard deviation over 30 independent runs for each experimental configuration.

necessary to evolve solutions and of the task performance of the final controllers. In addition, odNEAT consistently evolves controllers with relatively low complexity and superior generalisation capabilities that can potentially adapt and operate in different deployment scenarios without further evolution.

5.2 Comparing odNEAT and IM-($\mu + 1$)

In this section, we compare the performance of odNEAT and IM-($\mu + 1$). Experiments conducted with the IM-($\mu + 1$) algorithm serve as a means to compare odNEAT with an algorithm with a similar fundamental characteristic: the decentralised online evolution of neural topologies and weights.

Comparison of performance is shown in Table 9. In the aggregation task, odNEAT and IM-($\mu + 1$) evolve solutions to the task at similar rates. Differences in the number of evaluations between the two algorithms are not statistically significant ($\rho \geq .05$, Mann-Whitney). In the remaining two tasks, the navigation task and the phototaxis task, odNEAT significantly outperforms IM-($\mu + 1$) with respect to the number of evaluations ($\rho < .001$, Mann-Whitney). odNEAT requires approximately 54% of the evaluations needed by IM-($\mu + 1$) in the navigation task, and 45% of the evaluations in the phototaxis task. Furthermore, odNEAT always evolves controllers that yield significantly higher fitness scores ($\rho < 1 \cdot 10^{-4}$, Mann-Whitney).

An analysis of the neural complexity of evolved solutions, shown in Table 10, indicates that ANNs evolved by odNEAT are also less complex than those evolved by IM-($\mu + 1$). Both algorithms evolve networks with recurrent and feed forward connections. Differences in C_{fp} values are statistically significant across the three tasks ($\rho < .001$,

Table 10: Summary of the neural complexity added through evolution from the initial topology by odNEAT and IM-($\mu + 1$). In the IM-($\mu + 1$) algorithm, the size-proportionate addition of new connections and the duplication of neurons, and of their incoming and outgoing connections, leads to networks that consistently have a large number of connections.

Task	Method	Connections Added	Neurons Added	Avg. C_{fp}
Aggregation	odNEAT	6.3 ± 5.6	5.5 ± 5.0	11.8
	IM-($\mu + 1$)	23.6 ± 12.2	2.2 ± 0.4	25.8
Navigation	odNEAT	6.6 ± 0.9	3.1 ± 0.3	9.7
	IM-($\mu + 1$)	36.7 ± 14.4	2.5 ± 0.6	39.2
Phototaxis	odNEAT	2.3 ± 0.7	1.1 ± 0.3	3.4
	IM-($\mu + 1$)	26.0 ± 14.9	1.8 ± 0.5	27.8

Connections Added and Neurons Added refer to the average ± standard deviation over 30 independent runs for each experimental configuration.

Mann-Whitney). Overall, the IM-($\mu + 1$) algorithm is biased toward large networks. Since there is no fitness cost in adding new neurons and connections, IM-($\mu + 1$) consistently generates large neural topologies. The growth is due to the structural mutation operators, as (1) *each* connection gene has a fixed equal probability of generating a new connection gene in the same genome, and (2) insertion of new neuron genes is based on the duplication and differentiation of a neuron gene *and* its incoming and outgoing connection genes. This form of growth leads to networks that consistently have more connections than neurons added through evolution (see Table 10). Since larger networks have more parameters and need more time to be optimised, either by the adjustment of weighting parameters or the removal of unnecessary neurons and connections through mutation, the algorithm tends to require more evaluations to find solutions than odNEAT.

In odNEAT the niching scheme protects topological innovations and also prevents bloating of genomes: species with smaller genomes are maintained in the population as long as their fitness is competitive, and smaller networks are thus not replaced by larger ones unnecessarily. The successive generation of new candidate controllers leads to a progressive optimisation of existing structure in each robot's internal population with parsimonious addition of structure.

In odNEAT the structure of each intermediate solution represents a search space of parameter values that evolution must optimise. The more complex the structure, the higher the number of parameters that must be optimised simultaneously. If the structural complexity can be minimised, the dimensionality of the search spaces explored along the path to a solution is reduced, and evolution can more efficiently optimise the intermediate solutions. Such an approach will generally lead to performance gains in terms of (1) speed of convergence toward the final solution, that is, the number of evaluations, and (2) the task performance of the intermediate and final solutions evolved. This hypothesis is supported by the results in Tables 9 and 10, which show that odNEAT evolves less complex networks that always outperform those evolved by IM-($\mu + 1$) in terms of their ability to solve the task, even when the evaluations necessary to evolve a solution are comparable (as in the aggregation task).

In the generalisation experiments, we observe that odNEAT evolves controllers that also display a generalisation performance superior to the controllers evolved by IM-($\mu + 1$) (see Table 11). Depending on the task, odNEAT outperforms IM-($\mu + 1$)

Table 11: Generalisation performance of controllers evolved by odNEAT and IM-($\mu + 1$) in three tasks.

Task	Method	Generalisation Performance (%)	Successful Tests
Aggregation	odNEAT	75.1 ± 32.6	2253/3000
	IM-($\mu + 1$)	46.1 ± 24.5	1382/3000
Navigation	odNEAT	73.4 ± 23.3	2202/3000
	IM-($\mu + 1$)	55.2 ± 36.2	1657/3000
Phototaxis	odNEAT	60.2 ± 33.5	1806/3000
	IM-($\mu + 1$)	46.0 ± 14.3	1379/3000

The Generalisation Performance refers to the average ± standard deviation of the success rate for each set of 100 task restarts.

between approximately 14 percentage points and 29 percentage points, as robots executing odNEAT successfully solve 427 to 871 generalisation tests more. Differences in the number of successful generalisation tests are statistically significant ($\rho < 1 \cdot 10^{-4}$ in the three tasks, Fisher's exact test). The results of the generalisation tests are coherent with those discussed in Section 5.1.1, which showed that odNEAT evolves neural networks with comparatively high generalisation capabilities. During long-term operation in the field, robots may experience environmental conditions not seen during the evolutionary phase. Therefore, producing robust controllers that can adapt to new circumstances without further evolution is advantageous and has been subject to increasing interest (Lehman et al., 2013). Another important aspect in robotic systems is the robustness to failures (Christensen et al., 2009). The following section is devoted to understanding the properties of odNEAT with respect to the algorithm's ability to adapt to faults in the sensors, and the impact of each algorithmic component on performance.

6 Assessing odNEAT: Fault Injection Experiments and Ablation Studies

In the previous section, we experimentally compared the performance of odNEAT with the performance of rtNEAT and IM-($\mu + 1$). In this section, we further assess odNEAT's robustness and features. We focus on two aspects: (1) odNEAT's ability to address long-term self-adaptation when there are faults in the robot's sensors, and (2) the impact of each algorithmic component on performance.

6.1 Adaptation Performance

The experimental protocol for the fault injection experiments described here is defined in coherence with the results of Carlson et al. (2004), which analysed the reliability of 15 mobile robots in terms of physical failures. For small robots operating in the field, such as the models considered in this study, there is an overall frequency of 0.10 failures per hour, and 12% of failures affect the sensors. In their study, the authors do not distinguish between complete failure and partial failure. In our experiments, we assume sensor failures as damaging the sensor completely and feeding a zero signal into the neural network during subsequent readings.

In the fault injection experiments, we conduct 30 independent runs using a group of five robots. We double the duration of each run to 200 hours of simulated time to examine the long-term effects of injected faults. Every hour of simulated time, faults are injected with probability .10, in which case one randomly chosen operational physical sensor becomes faulty with probability .12. Each robot starts out with the controller

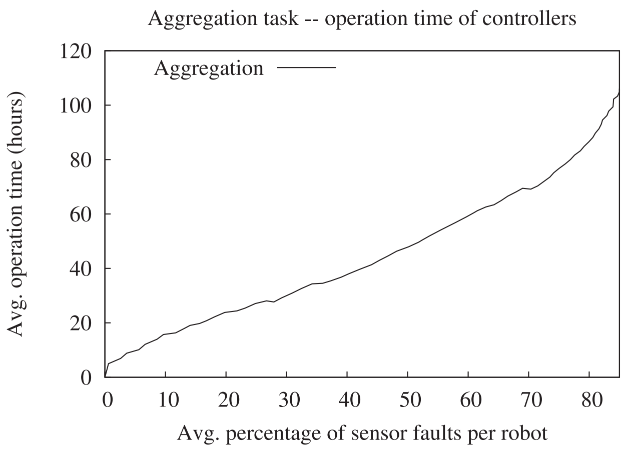

Figure 2: Fault injections during the aggregation task: average operation time (age) of controllers executing at a given time and subject to a given percentage of faults in the sensors.

evolved in the experiments described previously, but further evolution *is* allowed. The goal of using evolved controllers is to separate learning to solve the task from learning to overcome sensor faults.

To assess the effects of faults in the sensors, we analyse (1) the number of controllers produced to cope with a given percentage of faulty sensors, and (2) the operation time (age) of the controllers used by the robots during the experiments. The number of controllers produced is an indicator of the difficulty of the evolutionary process to adapt the behaviour of robots when faults are present. Complementarily, the operation time of controllers relates to the number of faults they can tolerate, thereby indicating the robustness of solutions evolved.

The controllers more robust to faults are those evolved in the aggregation task. On average, robots can sustain faults in approximately 85% of physical sensors, which corresponds to 20 out of 24 physical sensors. After this point, the experiments are terminated because the 200 hour limit is reached. Figure 2 shows the operation time of controllers during the experiments. As more faults are injected, the operation time continues to increase linearly, with a gentle slope, which indicates that new controllers are rarely necessary. In effect, the final controllers operate for more than 100 consecutive hours.

The high robustness to faults in the aggregation task is due to the robot's behaviour and to the task requirements. Robots form a single group and there is therefore considerable sensory information available. As long as robots can sense other robots with one or two sensors, faults in the remaining sensors have virtually no effect on performance. The high degree of tolerance to faults is due to the exchange of genomes between robots. As described in Section 4.3, the number of genomes received by a given robot is used as an estimation of the number of robots nearby, and is part of the virtual energy level and fitness score computations. Because only the physical sensors of the robots are affected by faults, the *virtual* energy level and the "received genomes" sensors function normally and robots continue to solve the task.

Figure 3 shows the number of controllers produced and the operation time of the controllers in both the navigation and phototaxis tasks. In these two tasks, odNEAT

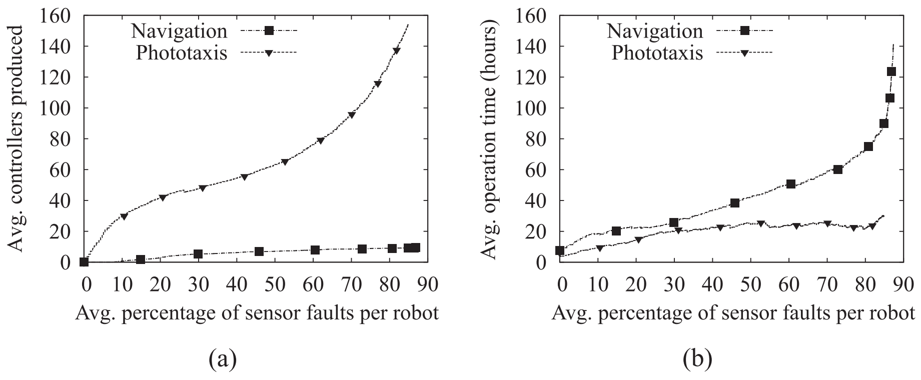

Figure 3: Summary of results concerning the injection of faults in the robots' sensors. (a) Number of controllers produced since the first fault was injected. (b) Operation time (age) of each controller executing at a given time.

can also adapt to cope with failures in approximately 80% to 85% of the sensors, corresponding to a maximum of 14 sensors in the navigation and obstacle avoidance task and 20 sensors in the phototaxis task. However, contrary to the aggregation task, robots have to evolve new controllers more often to handle the new sensory conditions. In the phototaxis task, with the increasing number of faults, odNEAT progressively tests more controllers as a means to synthesise solutions for the task. As shown in Figure 3a, when the percentage of faults reaches 50%, each robot had evolved on average 60 new controllers. For a higher percentage of faults, it becomes increasingly more difficult for odNEAT to evolve a suitable solution. When faults affect 75% and 85% of the sensors, the number of controllers evaluated grows to approximately 100 and 160, respectively. This result indicates that robots experience significant difficulties when more than 50% of the sensors are not functional. However, as supported by the approximately stable average operation time of the group for a percentage of faults greater than 50%, shown in Figure 3b, new controllers are able to sustain a moderate degree of faults in sensors before failing. In terms of neural augmentation, odNEAT continuously adds new topology in response to the sensor faults. odNEAT adjusts and augments neural topologies from an average of 3.4 parameters added through evolution, for solutions not subject to faults, to approximately 28.2 parameters for solutions subject to faults in 85% of the sensors.

In the integrated navigation and obstacle avoidance task, robots are robust to faults. As shown in Figure 3a, the initial controller only becomes unable to solve the task when approximately 10% of the sensors fail. From the first fault until 85% of the sensors fail, each robot tested 12.7 new controllers on average. Thus, even though a significant portion of the sensors are faulty, robots only evaluated relatively few new controllers during the *entire* fault injection experiments. In the synthesis of new controllers, odNEAT does not augment neural topologies substantially, as the complexity of solutions is relatively stable. Neural complexity of solutions is augmented from 9.7 parameters on average added through evolution, for solutions not subject to faults, to 10.8 parameters for solutions with 85% faults in the sensors. Nonetheless, it should be noted that there is a duality in the task. With the progressive injection of faults, robots display a better navigation performance but a worse obstacle avoidance performance. This is due to the obstacle avoidance fitness component relying on sensor readings to determine if there

Table 12: odNEAT ablations summary.

Method	Simulation Time (hours)	No. of Evaluations	Success Rate
Minimal population	16.4 ± 21.2	236.2 ± 213.8	22/30
No exchange of solutions	11.3 ± 12.0	156.0 ± 103.8	24/30
No tabu list	8.9 ± 12.2	133.6 ± 119.2	28/30
No maturation	13.9 ± 22.5	211.3 ± 287.8	29/30
No niching	41.1 ± 17.6	240.2 ± 103.0	25/30
No crossover	7.8 ± 6.5	127.8 ± 91.6	28/30
Full odNEAT	6.2 ± 5.6	103.7 ± 80.9	30/30

Results listed for each configuration are the average ± standard deviation over 30 independent runs.

are obstacles nearby, while the navigation component relies on the speed of the wheels (see Equation (5)), which is not affected by faults. An important consideration is that when the percentage of faults is greater than 67%, the less effective obstacle avoidance performance results in occasional collisions of robots with other robots and with the walls of the arena if the direction of contact is not sensed.

Overall, the analysis in this section shows that odNEAT is able to evolve adaptive solutions that can cope with faults in the robots' sensors. Distinct task requirements lead to different responses in the adaptation process. Robots executing odNEAT (1) are almost unaffected by faults in the aggregation task, (2) progressively change their behaviour in the navigation and obstacle avoidance task, and (3) experience more difficulties in the phototaxis task but are still able to adapt to the new sensory conditions.

6.2 Ablation Studies

In order to determine the impact of each algorithmic component in odNEAT, we perform a series of ablation studies. We use aggregation as the task for ablation studies. Aggregation was found to be the hardest task (see Tables 6 and 9) and thus provides a degree of complexity suitable for comparing the ablated versions of odNEAT with the complete version of odNEAT. We conduct experiments in six distinct experimental setups: (1) with a minimal internal population of size 2, (2) without the exchange of genomes between robots, (3) without the tabu list, (4) without the maturation period, (5) without the niching scheme, and (6) without the crossover operator. Note that the number of genomes received by one robot is directly used in the fitness calculation in the aggregation task (see Section 4.3). To implement the ablation study in which robots do not exchange genomes, we allow robots to communicate but the genomes received are not included in the evolutionary process, that is, they are discarded immediately after the energy level and fitness score calculations have been performed.

Results are shown in Table 12, averaged over 30 independent evolutionary runs for each configuration. Results in this table exclude runs that failed to find sustainable behaviours within 100 hours of simulated time. Simulation time is measured to complement the number of evaluations. In odNEAT, controllers execute as long as they are able to solve the task, that is, the virtual energy level is above the minimum threshold, and the duration of evaluations therefore tends to vary.

The most critical algorithmic component of odNEAT is the internal population. Differences in performance between full odNEAT and odNEAT with a minimal population are statistically significant in terms of the number of evaluations ($\rho < .01$, Mann-Whitney) and success rate ($\rho = 4.6 \cdot 10^{-3}$, Fisher's exact test). The size of the population is important because the population maintains a local view of the system's history and provides the basis for evolution. With a minimal population, evolution is limited to a small set of genomes, in this case 2. This experimental setup causes a significant instability in the evolutionary process, as evolution is much slower and may even be incapable of exploring enough of the solution space to find successful solutions, resulting in the comparatively low success rate and high simulation time and number of evaluations.

The exchange of genomes between robots is also a crucial feature in odNEAT's performance. Differences are statistically significant with respect to the number of evaluations ($\rho < .05$, Mann-Whitney) and the success rate ($\rho = 2.37 \cdot 10^{-2}$, Fisher's exact test). To quantify to what extent a robot is dependent on the genomes it receives from other robots, we analyse the origin of the information stored in each population when executing the full, nonablated odNEAT version. When evolution is ended, 73.4% of the genomes maintained in each internal population originated from other robots, whereas 26.6% of the genomes stored were evolved by the robots themselves. The final solutions executed by each robot to solve the task have on average 77% of matching genes. Moreover, 17.6% of these solutions have more than 90% of their genes in common. The average weight difference between matching connection genes is 4.8, with each weight $w \in [-10, 10]$. Local exchange of genetic information is therefore a crucial part in the odNEAT's evolutionary dynamics that serves as a substrate for speeding up the evolutionary process and for collective problem solving.

Without the tabu list, odNEAT achieves a success rate of 28/30 runs but when evolution is on the right track, it finds solutions relatively fast. Differences in the number of evaluations and in the success rate are not significant. However, the comparatively high standard deviation values in the simulation time and in the number of evaluations reflect the instability of the evolutionary process when the tabu list is not used. The tabu list keeps the evolutionary process from cycling around in an unfruitful neighbourhood of the solution space, which may happen given that odNEAT evolves controllers only based on local information. In effect, by rejecting genomes similar to those that failed, the tabu list promotes topological diversity in the population and positively smooths evolution, thereby providing a relevant contribution to the performance of odNEAT.

The maturation period defines a lower bound of activity in the environment, giving the new offspring a chance to spread their genome. Ablating the maturation period provides results significantly different in terms of the number of evaluations ($\rho < .05$, Mann-Whitney) but not significant with respect to the success rate. Without the maturation period, good solutions are potentially lost forever and evolution is decelerated. Robots are still be capable of solving the task in most experiments but exhibit a clear deficit in performance.

By ablating the niching scheme of odNEAT, performance is considerably affected. Robots solve the task in 25/30 runs. Differences in the success rate are not significant. When odNEAT is able to evolve a solution to the task, it requires substantially more time, and robots evaluate significantly more controllers ($\rho < .001$, Mann-Whitney). Without the crossover operator, odNEAT fails to find solutions for the task in 2 of the 30 runs. When odNEAT is able to find a solution without crossover, each robot is subject to approximately 24 additional evaluations, equivalent to 18.9%. Differences in the number of evaluations are statistically significant ($\rho < .01$, Mann-Whitney). The series of ablation studies involving genetic operators show that both the crossover and the niching scheme have a significant effect on performance, and further support a number of conclusions in the literature: (1) the crossover operator is an advantage for neuroevolution algorithms when performed appropriately, for instance, by resorting to innovation numbers (Stanley and Miikkulainen, 2002), and (2) genotypic diversity mechanisms such as speciation and fitness sharing can improve performance in evolutionary robotics tasks (Mouret and Doncieux, 2012).

The ablation of each component of odNEAT's components leads to a less efficient algorithm. Arguably, the most important conclusion drawn from the ablation studies is that all components of odNEAT contribute to the algorithm's performance as an efficient online, distributed, and decentralised neuroevolution algorithm.

7 Conclusions

In this article, we presented a novel distributed and decentralised neuroevolution algorithm called odNEAT for online learning in groups of robots. odNEAT implements the online evolutionary process according to a physically distributed island model. Each robot optimises an internal population of genomes, and there is exchange of genetic information between robots. An important advantage of odNEAT over the majority of existing online neuroevolution algorithms is that the optimisation of both neural parameters and topology is under evolutionary control, and an appropriate network topology is the result of a continuous evolutionary process.

We compared odNEAT with rtNEAT and with IM-$(\mu + 1)$ in three tasks: (1) aggregation, (2) integrated navigation and obstacle avoidance, and (3) phototaxis. Our study produced three main results. First, odNEAT yields performance levels similar to rtNEAT, a state-of-the-art neuroevolution algorithm, and outperforms the IM-$(\mu + 1)$ algorithm. Second, compared with rtNEAT and IM-$(\mu + 1)$, odNEAT exhibits a higher evolutionary pressure toward neural controllers with low complexity and with superior generalisation capabilities. Third, our experiments showed that individual robots executing odNEAT can successfully adapt to cope with faults in the sensors. Depending on the task requirements, robots can tolerate different degrees of faults and, if necessary, evolve new controllers and progressively modify their behaviour in a completely autonomous manner. Overall, our study showed that odNEAT is an efficient and robust algorithm, and a promising approach for online evolution in multirobot systems.

The aim of our ongoing work is to study and demonstrate the performance of odNEAT in real multirobot systems. We are therefore studying how to accelerate the online evolutionary process through the use of behavioural building blocks of distinct granularity to enable robots to adapt to dynamic and complex tasks in a timely manner (Silva et al., 2014a, 2014b). In this respect, we intend to investigate the effects of open-ended techniques such as novelty search (Lehman and Stanley, 2011) and behavioural diversity-based methods (Mouret and Doncieux, 2012) in online evolution. In tasks where the environmental conditions or task requirements constantly change, explicitly encouraging behavioural novelty could enable the evolutionary process to more easily produce a large variety of effective solutions, thereby opening a new path toward long-term self-adaptation.

Acknowledgments

This work was partly supported by Fundação para a Ciência e a Tecnologia (FCT) under the grants SFRH/BD/89573/2012, UID/EEA/50008/2013, UID/Multi04046/2013, and EXPL/EEI-AUT/0329/2013, and by the European Union—Information and Communication Technologies project "ASSISIbf," no. 601074. The authors thank Christopher Schwarzer and Florian Schlachter for providing the source code of their algorithm, and the anonymous reviewers for their constructive feedback and valuable comments.

Supplementary Material

The source code of the software, videos of the behaviours evolved, and details of the experiments are available at http://fgsilva.com/?page_id=121. The source code can also be found at: https://github.com/fgsilva/online_evo_jbot.

References

Bahgeçi, E., and Sahin, E. (2005). Evolving aggregation behaviors for swarm robotic systems: A systematic case study. In *Proceedings of the IEEE Swarm Intelligence Symposium*, pp. 333–340.

Beer, R. D., and Gallagher, J. C. (1992). Evolving dynamical neural networks for adaptive behavior. *Adaptive Behavior*, 1(1): 91–122.

Bianco, R., and Nolfi, S. (2004). Toward open-ended evolutionary robotics: Evolving elementary robotic units able to self-assemble and self-reproduce. *Connection Science*, 16(4): 227–248.

Blakesley, R. E. (2008). Parametric control of familywise error rates with dependent ρ-values. Unpublished doctoral dissertation, University of Pittsburgh.

Bredèche, N., Haasdijk, E., and Eiben, A. (2009). On-line, on-board evolution of robot controllers. In *Proceedings of the 9th International Conference on Artificial Evolution*, pp. 110–121.

Camazine, S., Deneubourg, J. L., Franks, N., Sneyd, J., Theraulaz, G., and Bonabeau, E. (2001). *Self-organization in biological systems*. Princeton, NJ: Princeton University Press.

Cao, Y., Fukunaga, A., and Kahng, A. (1997). Cooperative mobile robotics: Antecedents and directions. *Autonomous Robots*, 4(1): 1–23.

Carlson, J., Murphy, R., and Nelson, A. (2004). Follow-up analysis of mobile robot failures. In *Proceedings of the 2004 IEEE International Conference on Robotics and Automation*, pp. 4987–4994.

Christensen, A. L., O'Grady, R., and Dorigo, M. (2009). From fireflies to fault-tolerant swarms of robots. *IEEE Transactions on Evolutionary Computation*, 13(4): 754–766.

Duarte, M., Silva, F., Rodrigues, T., Oliveira, S. M., and Christensen, A. L. (2014). JBotEvolver: A versatile simulation platform for evolutionary robotics. In *Proceedings of the 14th International Conference on the Synthesis and Simulation of Living Systems*, pp. 210–211.

Eiben, A., Haasdijk, E., and Bredèche, N. (2010a). Embodied, on-line, on-board evolution for autonomous robotics. In P. Levi and S. Kernbach (Eds.), *Symbiotic multi-robot organisms: Reliability, adaptability, evolution*, pp. 361–382. New York: Springer.

Eiben, A., Karafotias, G., and Haasdijk, E. (2010b). Self-adaptive mutation in on-line, on-board evolutionary robotics. In *Proceedings of the 4th IEEE International Conference on Self-Adaptive Self-Organizing Systems Workshop*, pp. 147–152.

Elfwing, S., Uchibe, E., Doya, K., and Christensen, H. I. (2005). Biologically inspired embodied evolution of survival. In *Proceedings of the Congress on Evolutionary Computation*, pp. 2210–2216.

Fisher, R. (1925). *Statistical methods for research workers*. Edinburgh: Oliver and Boyd.

Floreano, D., Dürr, P., and Mattiussi, C. (2008). Neuroevolution: From architectures to learning. *Evolutionary Intelligence*, 1(1): 47–62.

Floreano, D., and Keller, L. (2010). Evolution of adaptive behaviour by means of Darwinian selection. *PLoS Biology*, 8(1): e1000292.

Floreano, D., and Mondada, F. (1994). Automatic creation of an autonomous agent: Genetic evolution of a neural-network driven robot. In *Proceedings of the 3rd International Conference on Simulation of Adaptive Behavior*, pp. 421–430.

Gross, R., and Dorigo, M. (2009). Towards group transport by swarms of robots. *International Journal of Bio-Inspired Computation*, 1(1–2): 1–13.

Gutiérrez, A., Campo, A., Dorigo, M., Amor, D., Magdalena, L., and Monasterio-Huelin, F. (2008). An open localization and local communication embodied sensor. *Sensors*, 8(11): 7545–7563.

Haasdijk, E., Arif, A., and Eiben, A. (2011). Racing to improve on-line, on-board evolutionary robotics. In *Proceedings of the 13th Genetic and Evolutionary Computation Conference*, pp. 187–194.

Haasdijk, E., Eiben, A., and Karafotias, G. (2010). On-line evolution of robot controllers by an encapsulated evolution strategy. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pp. 1–7.

Haasdijk, E., Smit, S., and Eiben, A. (2012). Exploratory analysis of an on-line evolutionary algorithm in simulated robots. *Evolutionary Intelligence*, 5(4): 213–230.

Harvey, I., Husbands, P., Cliff, D., Thompson, A., and Jakobi, N. (1997). Evolutionary robotics: The Sussex approach. *Robotics and Autonomous Systems*, 20(2): 205–224.

Haykin, S. (1999). *Neural networks: A comprehensive foundation*. Englewood Cliffs, NJ: Prentice-Hall.

Hommel, G. (1988). A stagewise rejective multiple test procedure based on a modified Bonferroni test. *Biometrika*, 75(2): 383–386.

Huijsman, R., Haasdijk, E., and Eiben, A. (2011). An on-line on-board distributed algorithm for evolutionary robotics. In *Proceedings of the 10th International Conference on Artificial Evolution*, pp. 119–130.

Karafotias, G., Haasdijk, E., and Eiben, A. (2011). An algorithm for distributed on-line, on-board evolutionary robotics. In *Proceedings of the 13th Genetic and Evolutionary Computation Conference*, pp. 171–178.

Kim, Y.-H., and Moon, B.-R. (2003). New usage of Sammon's mapping for genetic visualization. In *Proceedings of 5th Genetic and Evolutionary Computation Conference*, pp. 1136–1147.

Kittler, J., and Young, P. (1973). A new approach to feature selection based on the Karhunen-Loeve expansion. *Pattern Recognition*, 5(4): 335–352.

Kohonen, T. (1982). Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43(1): 59–69.

Laredo, J., Eiben, A., van Steen, M., and Merelo, J. (2010). EvAg: A scalable peer-to-peer evolutionary algorithm. *Genetic Programming and Evolvable Machines*, 11(2): 227–246.

Lehman, J., Risi, S., D'Ambrosio, D., and Stanley, K. O. (2013). Encouraging reactivity to create robust machines. *Adaptive Behavior*, 21(6): 484–500.

Lehman, J., and Stanley, K. O. (2011). Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary Computation*, 19(2): 189–223.

Mondada, F., Bonani, M., Raemy, X., Pugh, J., Cianci, C., Klaptocz, A., Magnenat, S., Zufferey, J., Floreano, D., and Martinoli, A. (2009). The e-puck, a robot designed for education in engineering. In *Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions*, pp. 59–65.

Montanier, J.-M., and Bredèche, N. (2011). Embedded evolutionary robotics: The (1+1)-restart-online adaptation algorithm. In S. Doncieux, N. Bredèche, and J.-B. Mouret (Eds.), *New horizons in evolutionary robotics*, pp. 155–169. Studies in Computational Intelligence, Vol. 341. Berlin: Springer.

Mouret, J., and Doncieux, S. (2012). Encouraging behavioral diversity in evolutionary robotics: An empirical study. *Evolutionary Computation*, 20(1): 91–133.

Prieto, A., Becerra, J., Bellas, F., and Duro, R. (2010). Open-ended evolution as a means to self-organize heterogeneous multi-robot systems in real time. *Robotics and Autonomous Systems*, 58(12): 1282–1291.

Sammon, J., Jr. (1969). A nonlinear mapping for data structure analysis. *IEEE Transactions on Computers*, C-18(5): 401–409.

Schmidhuber, J. (1997). Discovering neural nets with low Kolmogorov complexity and high generalization capability. *Neural Networks*, 10(5): 857–873.

Schwarzer, C., Schlachter, F., and Michiels, N. (2011). Online evolution in dynamic environments using neural networks in autonomous robots. *International Journal on Advances in Intelligent Systems*, 4(3-4): 288–298.

Silva, F., Correia, L., and Christensen, A. L. (2014a). Speeding up online evolution of robotic controllers with macro-neurons. In *Proceedings of the 16th European Conference on the Applications of Evolutionary Computation*, pp. 765–776.

Silva, F., Duarte, M., Oliveira, S. M., Correia, L., and Christensen, A. L. (2014b). The case for engineering the evolution of robot controllers. In *Proceedings of the 14th International Conference on the Synthesis and Simulation of Living Systems*, pp. 703–710.

Silva, F., Urbano, P., Oliveira, S., and Christensen, A. L. (2012). odNEAT: An algorithm for distributed online, onboard evolution of robot behaviours. In *Proceedings of the 13th International Conference on the Simulation and Synthesis of Living Systems*, pp. 251–258.

Stanley, K. O. (2004). Efficient evolution of neural networks through complexification. Unpublished doctoral dissertation, University of Texas, Austin.

Stanley, K. O., Bryant, B. D., and Miikkulainen, R. (2005). Real-time neuroevolution in the NERO video game. *IEEE Transactions on Evolutionary Computation*, 9(6): 653–668.

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

Vassilev, V. K., Fogarty, T. C., and Miller, J. F. (2000). Information characteristics and the structure of landscapes. *Evolutionary Computation*, 8(1): 31–60.

Watson, R. A., Ficici, S. G., and Pollack, J. B. (1999). Embodied evolution: Embodying an evolutionary algorithm in a population of robots. In *Proceedings of the 1999 Congress on Evolutionary Computation*, pp. 335–342.

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 Systems*, 39(1): 1–18.

Yao, X. (1999). Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9): 1423–1447.