Distributed On-line Learning in Swarm Robotics with Limited Communication Bandwidth

Nicolas Fontbonne, Olivier Dauchot, Nicolas Bredeche

To cite this version:

Nicolas Fontbonne, Olivier Dauchot, Nicolas Bredeche. Distributed On-line Learning in Swarm Robotics with Limited Communication Bandwidth. IEEE Congress on Evolutionary Computation, 2020, Glasgow (virtual), United Kingdom. 10.1109/CEC48606.2020.9185697 . hal-03175237

HAL Id: hal-03175237
https://hal.sorbonne-universite.fr/hal-03175237v1

Submitted on 19 Mar 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed On-line Learning in Swarm Robotics with Limited Communication Bandwidth

Nicolas Fontbonne
Sorbonne Université, CNRS
Institut des Systèmes Intelligents et de Robotique
Paris, France
nicolas.fontbonne@sorbonne-universite.fr

Olivier Dauchot
ESPCI, CNRS
Gulliver Lab
Paris, France
olivier.dauchot@espci.fr

Nicolas Bredeche
Sorbonne Université, CNRS
Institut des Systèmes Intelligents et de Robotique
Paris, France
nicolas.bredeche@sorbonne-universite.fr

***Abstract*—This paper presents a new algorithm for distributed on-line evolutionary learning in swarm robotics. The challenge we address is to cope with the limited computation and communication capabilities of low cost robots, which are often used in swarm robotics. In order to do so, the algorithm decouples computation and communication and ensures learning of efficient control policies even when only a limited amount of information can be exchanged between neighbouring robots. We show experimentally that this algorithm is both remarkably robust with respect to its meta-parameter values, and able to adapt automatically to the available communication bandwidth.**

***Index Terms*—online distributed learning, swarm robotics, black-box objective function, embodied evolution, social learning, evolutionary robotics**

I. Introduction

Swarm robotics is a sub-domain of collective robotics. This field is characterised by the use of a generally large number of robots with limited communication and computation capabilities, and an objective defined at the level of the whole swarm (e.g. exploring the environment, searching for resources, collectively transporting heavy objects). The challenge is to design the rules of microscopic interactions between the robots in order to achieve a relevant swarm organisation at the macroscopic level [1]–[3], [9].

There are several bio-inspired methods to address this problem of distributed programming, which can be grouped into two main classes: manual programming and automatic design methods. In the first case, the aim is to explicitly reproduce at a more or less abstract level behaviours observed in nature, but whose macroscopic result is sometimes difficult to foresee in advance.

In the second case, the use of evolutionary optimisation algorithms makes it possible to automate the design of individual behaviours according to a global objective function [19]. Indeed, since the structure of the environment is not known in advance, writing the objective function makes it possible to specify the general objective of the task (e.g. maximizing the number of objects picked up for a foraging task), without giving any precision on the structure of the strategy to be deployed. This is a well-known problem framework in evolutionary robotics: the structures of the research space and the fitness landscape have a tenuous relationship, for which stochastic optimisation methods are a good fit.

However, an important assumption that underlies manual or automatic design methods is that behavioural rules are obtained in the laboratory, and then deployed in a real-world situation as is without any further tuning. In other words, the environment and the nature of the task are considered stationary between the conditions known at the time of design and the conditions actually encountered afterwards.

In this paper, we are interested in a different class of problem. We consider that the environment in which the robots will be deployed is not known in advance. Provided a general objective, that can be written down as an objective function defined at the level of the individual (e.g. given a foraging task, each robot should try to get the largest number of items), we implement a distributed on-line learning algorithm to allow the swarm of robots to progressively acquire the necessary skills to perform the target task. The behaviours learned depend on the nature of the task, but also on the particularities of the environment. For example, picking up objects in a foraging task requires different strategies depending on whether the objects are grouped together in a specific area or distributed in the environment.

The class of evolutionary algorithms that addresses this problem is known as either embodied evolution or social learning. Whether biological or cultural evolution is considered, both methods can be seen as instances of designing algorithms inspired by evolutionary dynamics for swarm robotics. These methods stand as on-line distributed learning that takes into account interactions between hardware limited robots distributed over a possibly large space with local communication. On the one hand, these evolutionary dynamics methods are similar to the classical evolutionary robotics approach as the goal is to optimised a black-box objective function whose analytical form is not known. On the other hand, they differ in that the problem of the transition to reality is simply non-existent in evolutionary dynamics methods: the actual learning starts at the time of operational deployment of the robots, and not before it. In other words, the goal is to design robust on-line learning algorithms as much as robust solutions.

To date, several evolutionary dynamics algorithms have been validated on real robots, addressing various scientific and practical issues. However, the number of robots is often in the order of ten(s) due to the hardware and CPU requirements of such robots to implement learning algorithms (e.g. 20 e-Puck robots in [6], 6 Thymio-2 robot with a Raspberry board in [13]). This is in stark contrast with other works in swarm robotics that do not implement learning capabilities, where the number robots is often counted by the hundreds [14], [20] and even up to slightly more than one thousand robots [15]. The challenge remains open as to implementing online distributed learning on such a large swam of robots.

In this paper, we propose a new distributed on-line learning algorithm inspired by evolutionary dynamics. The originality of this algorithm is to minimise the requirements in terms of memory and communication cost, in order to be deployed on a swarm of very low cost robots with limited computing and communication capabilities (e.g. a Kilobot robot). The proposed algorithm is based on the *horizontal gene transfer* mechanism observed in bacteria: when two robots interact, a part of the control parameters is transferred to the robot's memory and communication system. Thus, the amount of information transferred can take into account time and bandwidth limitations, regardless of the number of control parameters governing decision making.

Though similar ideas have been explored in evolutionary computation [11], it has never been employed in the context of distributed on-line learning, where the possibility of modulating the amount of information transmitted between robots makes it possible to take into account hardware and environmental constraints. Obviously, the price to pay for such an algorithm is a reduced convergence speed. But it also makes it possible to adjust the amount of information exchanged to account from practical limitations at hand. Indeed, the quality of the communication bandwidth depends both on technical characteristics and environmental contingencies (number of packet collisions that increase with the number of robots, disturbances due to materials used in the environment, etc.).

The rest of the paper is organised as follow : Section 2 presents the algorithm and evolutionary operators. Section 3 describes the experimental setup. Section 4 presents results on a classic foraging task, first by comparing the proposed algorithm with a state-of-the-art counterpart, and second, by performing a sensitivity analysis of the various hyper-parameters of the Algorithm. We then present an extension of the Algorithm to automatically adapt to the available communication bandwidth.

II. Algorithm

Both embodied evolution [4], [21] and social learning [12] implement an evolutionary algorithm scheme, adapted to perform distributed on-line learning as illustrated in Figure 1. Each robot optimises a policy to maximise a score, G, that is locally defined in each robot.

Robots follow a *sense-act loop* commonly used in robotics for reactive agents. Each robot i runs a deterministic policy $\pi_{\boldsymbol{\theta}_i}$. At initialisation, a robot is initialised with a random control parameter set $\boldsymbol{\theta}_i$, referred to as *active parameters*. These parameters are then used by the policy to determine the actions to be taken by the agent in reaction to its observations.

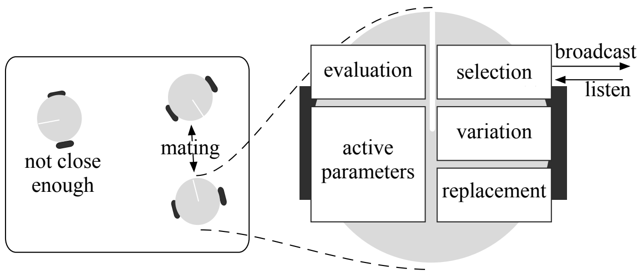


Fig. 1. Principle of embodied evolution algorithms. The robotic agent sends and receives parameters of a policy function, and uses a evolutionary algorithm to improve it own active parameters.

The typical algorithm goes as follow : as a robot moves through its environment, the quality of its behaviour is assessed using a *score* function which depends on the user-defined task to be achieved (e.g. number of items gathered for a foraging task, which is also sometimes referred to as *fitness value*, *reward*, *performance* or *utility*). When two robots are within communication range, they exchange their active parameters and the (current) assessment of their policies. This operation is called *mating*. The choice of accepting a partner may be purely based on the environmental contingencies but other consideration may play a role such as performance or similarity. Mating does not automatically imply a change in the active parameters of one agent, but incoming data is generally stored for later use in a *reservoir*.

The renewal of a robot's active parameters typically occurs after some predefined amount of time. At this point, the robot will use information stored in the reservoir to update its current active parameter set, which imply constructing a new candidate set of parameters from the reservoir using typical selection and variation evolutionary operators [5], [8], [10], [12], [17].

A. Horizontal Information Transfer

All embodied evolutionary algorithm to date assume that the whole set of active parameters (and current score) can be sent as a single communication packet. This is generally true where the bandwidth is virtually infinite (as in simulation) or where it is order of magnitude larger than the size of messages to be sent (as with Linux-running robots using WIFI exchanging a few hundreds neural network weights) [4]. Using hardware-limited robots such as Kilobots [18] then raise the question of limiting the number of control parameters to stay within the limits of the communication bandwidth, which can be due to either technical limitations or environmental contingencies. For example, Kilobots not only use slow IR communication (a few octets per seconds), but also are limited when the number of Kilobots closeby increases due packet collisions.

In order to decouple communication and computation constraints, we introduce the HIT algorithm. HIT stands for Horizontal Information Transfer, and can be seen as an instance of either embodied evolution or social learning algorithms. It uses evolutionary operators in a distributed on-line fashion, and manage communication between robots by exchanging a part, rather than all, of the robot's control parameters. By exchanging only partial information, we expect two benefits:

- the possible recombination of behavioural skills (i.e. efficient subsets of the policy parameters) obtained by separate robots, which would not be possible in a winner-take-all approach. This will be shown is Section IV-B;
- the decoupling of the number of control parameters that can be *used* for control and *sent* to other robots. In other words, it makes it possible to exploit the computation and memory capabilities of the robot, without being limited by its communication capability. This will be shown is Section IV-C.

In addition, HIT differs from other similar algorithms as it does not require a reservoir to store incoming information. Upon receiving control parameters and current score from a nearby robot, a robot will immediately integrate the new parameter values (i.e. overwriting the current corresponding values) if its interlocutor's score is better.

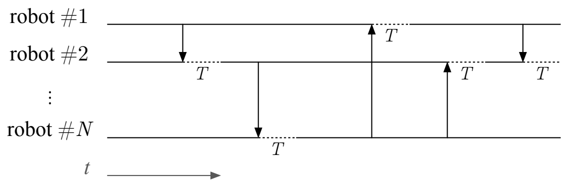


Fig. 2. Communication or mating between the agents. Each robot has a maturation period of time T where communication with neighbours is turned off. A new maturation period is initiated after each modification of the active parameters. During this period, the robot only evaluate it policy

Algorithm 1 describes HIT as implemented in each robot. It includes both evolutionary learning and decision-making. It can be decomposed in two cycles:

- *lines 4-7, sense-act cycle*: The agent retrieves information from the environment via its sensors using the sense() function. It retrieves an observation vector $\mathbf{o}$ and a reward scalar r. The reward is stored in a queue, here $G[.]$, of size T for later use. As for the observation vector, it is used by the deterministic policy $\pi_{\boldsymbol{\theta}}$ to compute the action vector $\mathbf{a}$. Finally, the act() function is responsible for transmitting the commands to the actuators.
- *lines 9-15, evolutionary cycle*: After an evaluation period of fixed size T, the agent can enter the evolutionary cycle. It broadcasts a random subset of its control parameter set to its neighbours with an evaluation of its quality (i.e. the score). The subset size is controlled by the transfer rate α. When it receives a new message (indicated by the $new\_message$ variable), if the received parameters have a better score, then it replaces its own control parameters by the sender's selected parameters (function TRANSFER). Then it applies a Gaussian mutation of variance σ on all its parameter set (function MUTATION).

In the following, we provide more details on the selection and variation operators.

Algorithm 1: The HIT algorithm (*Horizontal Information Transfer*)

```
Data:
  $\alpha$ : transfer rate $\in [0,1]$,
  $T$ : evaluation time,
  $\pi$ : Policy function,
  $\boldsymbol{\theta}$ : Random uniform initialisation of policy
  parameters, dim($\boldsymbol{\theta}$) $= m$,
  $R[T]$ : Empty reward buffer of size $T$,
  $r$ : Null reward scalar,
  $\mathbf{a}$ : Null action vector,
  $\mathbf{o}$ : Null observation vector
1  begin
2      $t = 0$
3      while True do
4          $\mathbf{o}, r$ = sense()
5          $R[t \mod T] = r$
6          $\mathbf{a} = \pi(\mathbf{o}|\boldsymbol{\theta})$
7          act($\mathbf{a}$)
8          if $t > T$ then
9              $G = \sum_{k=0}^{T-1} R[k]$
10             Create the Idx array by drawing randomly
                 $\alpha m$ integers in range $[0, m-1]$ without
                 replacement
11             broadcast($\boldsymbol{\theta}$[Idx], Idx, $G$)
12             if $new\_message$ then
13                 $\boldsymbol{\theta}$ = TRANSFER($\boldsymbol{\theta}$, $G$,
                   Idx$_{\text{message}}$,$\boldsymbol{\theta}_{\text{message}}$, $G_{\text{message}}$)
14                 $\boldsymbol{\theta}$ = MUTATION($\boldsymbol{\theta}$)
15                 $t = 0$
16             end
17         end
18         $t = t + 1$
19     end
20 end
```

B. Evaluation and Selection

The assessment of individuals depends mainly on the quality of the policy but can be very noisy due to non-stationary stochastic variabilities in the environment and behaviour of other agents. Depending on the definition of the score function and the distribution of the rewards in the environment, some individuals may find themselves naturally favoured by chance and thus spread misguided policies.

HIT uses a sliding time window of size T to assess the robot's performance. The window size depends on the task and the environment at hand, and must be set so that sufficient information is gathered to provide a relevant estimate of the quality of current policy. In order to compare policies in a fair manner, robots can only exchange information after the sliding window has been completely filled. We call this *maturation period*, which duration corresponds to the time required for a full self-evaluation, i.e. the evaluation time T.

Figure 2 illustrates the dynamics of the algorithm. Whenever two robots meet, the worst-scoring robot replace part of its control parameter with those of the best-scoring robot. Then, the updated robot resets its score and enters a maturation period during which communication is disabled.

At each step, the agent receives a reward R_t . We call score G_t, the cumulated reward obtained by the agent during T steps (sliding window). The evaluation time T, the score G_t and reward R_t at step t are linked by the relation:

$$G_t = \sum_{k=0}^{T-1} R_{t-k} \qquad (1)$$

While a longer evaluation time should allow for more accurate assessment of a given policy's quality, there is of course a cost in terms of convergence speed.

C. Transfer operator

HIT introduces a *transfer operator* with rate $\alpha \in [0, 1]$ that defines the amount of information that will be transferred during communication between two robots. As an example, a transfer rate of $\alpha = 0.5$ means that half of the control parameters will be randomly selected to be sent. From one interaction to another, a different subset of parameters can be selected for sending, and two interacting robots will send the same quantity of possibly different parameters.

There can be different methods to randomly pick the parameters to be sent. A basic method, which we use afterwards, is to send n parameters stored in 32-bit float, along withan additional n bytes to send the indexes of these parameters. However, several strategies can be used to compress or limit the quantity of information without compromising overall optimisation. It can also possible to send a segment of parameters whose offset changes randomly in-between each new message.

Algorithm 2 details the transfer mechanism during the mating operation. Unlike other embodied evolution algorithms, HIT does not store incoming information in a reservoir. Whenever the score of the sender is greater or equals to the score of the receiver, the received parameter are directly used to overwrite the corresponding local parameter.

Algorithm 2: *Transfer function*

```
Data:
  $\boldsymbol{\theta}$: active parameters,
  $G$: current evaluation,
  $\mathrm{Idx}_{\mathrm{message}}$: received parameter indexes,
  $\boldsymbol{\theta}_{\mathrm{message}}$: received parameters,
  $G_{\mathrm{message}}$: received evaluation
1 begin
2     if $G_{message} > G$ then
3         for $i \in Idx_{message}$ do
4             $\boldsymbol{\theta}[i] = \boldsymbol{\theta}_{\mathrm{message}}[i]$
5         end
6     end
7 end
```

D. Mutation operator

HIT implements a classic *Gaussian mutation* operator. It applies a perturbation centred on the current parameter value, with a variance σ. It is defined as follow, for the m parameters:

$$\theta[i] \leftarrow \mathcal{N}(\theta[i], \sigma) \quad \forall i \in [1, m]$$

Mutation allows to introduce and maintain some level of diversity during the exploration of the parameters space. While it may not be useful in the first steps of evolution, it eventually maintain some level of diversity for exploring the parameters space as using the transfer operator can only leverage what is already present in the initial population.

III. Experimental setup

A. Task and Environment

In order to study the dynamics of HIT, we devise a foraging task, similar to tasks solved by many species of insect collectives. It is also a good abstraction of a search and retrieve robotic task, where an unknown environment must be explored to retrieve specific objects or resources.

The goal here is for each robot to collect as many items as possible. Both robots and items are initially randomly placed in the arena. Whenever a robot picks up an object at iteration t, it gets a reward of $r_t = 1$, and a new item appears at a random location in the arena to maintain a constant number of resources.

B. Simulation environment

We used the Roborobo3 simulator [7], which is a pseudo-realistic, light and fast multi-agent simulation environment developed in C++. It provides a pseudo-realistic physics robotic model similar to the seminal Khepera2 and e-Puck 2-wheeled mobile robots while still ensuring fast enough simulation to allow for extensive experimental work involving hundreds of robots.

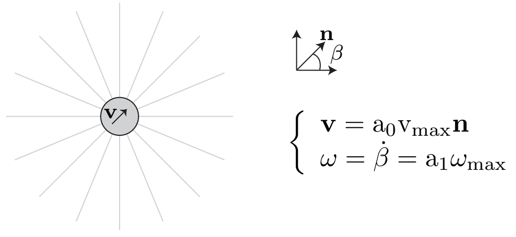
Robots, objects and the environment are physically represented by bitmap images, which allows roborobo3 to manage collisions at the pixel level, though location, perception and displacement are handled in the continuous domain.

Robots have a size of 5px $\times$ 5px . They move in an environment of 1400px $\times$ 800px that is uniformly filled with objects.

C. Robot Model

Robots are subject to a kinematic model. It is, therefore, a question of controlling a velocity vector. Thus, the robots have a 2-dimensional action space A where the two dimensions represent speed ($\mathrm{a}_0 \in [-1, 1]$ for [backward, forward]) and angular speed ($\mathrm{a}_1 \in [-1, 1]$ for [clockwise, anti-clockwise]).

Their observation space O is composed of 16 range sensors that get information about the surrounding of the agent. They are ray-casting sensors that have a maximum range of three times the robot length (15px). For each of these sensors, four information are extracted: the distance to contact if an object is detected, and three other Boolean information for each sensor

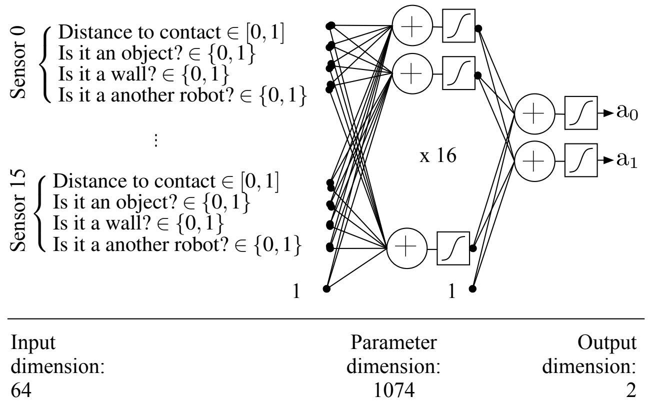


$$\begin{cases} \mathbf{v} = \mathrm{a}_0 \mathrm{v}_{\max} \mathbf{n} \\ \omega = \dot{\beta} = \mathrm{a}_1 \omega_{\max} \end{cases}$$

Fig. 3. Robot model with 16 sensors, a velocity $\mathbf{v}$, a maximal speed $v_{\max}$, an angular speed ω and a maximal angular speed $\omega_{\max}$. The control variables are then a_0 and a_1.

to explicit the type of information (object, wall or agent). Thus, we obtain an observation vector of $\dim(O) = 64$ dimensions.

The robot's policy maps observations $\mathbf{o} \in O$ to actions $\mathbf{a} \in A$. For that purpose, we use a multi-layered Perceptron (MLP) as the main policy structure. Figure 4 details the full topology between inputs and outputs, and the number of parameters.



Sensor 0
Distance to contact ∈ [0, 1]
Is it an object? ∈ {0, 1}
Is it a wall? ∈ {0, 1}
Is it a another robot? ∈ {0, 1}
Sensor 15
Distance to contact ∈ [0, 1]
Is it an object? ∈ {0, 1}
Is it a wall? ∈ {0, 1}
Is it a another robot? ∈ {0, 1}
x 16
a0
a1
1
1
Input dimension: 64
Parameter dimension: 1074
Output dimension: 2

Fig. 4. Architecture of the policy function. Each sensor gives information about the distance to obstacle and the type of object detected, if any. The architecture represented here is a multi-layered Perceptron, used for the experiments. It has 64 dimensions as input, a hidden layer of 16 dimensions plus a bias term for each layer. The action dimension is 2. The total number of parameters is 1074.

This architecture imposes a large number of free control parameters that need to be optimised. In the present case, this means 1074 parameters.

IV. Result

In this Section, we conduct an experimental study of the HIT algorithm. Values for all experimental parameters can be found in Table I, including environmental, neural network controller and HIT parameters. The rectangular arena used is represented in Figure 5, with 150 robots and 100 items.

A. Qualitative and Quantitative Evaluation

We analyse the dynamics of HIT while solving the foraging task for a well chosen set of meta-parameters (See next Section for an extensive analysis of the α and σ meta-parameters). To analyse the results, we define two notions:

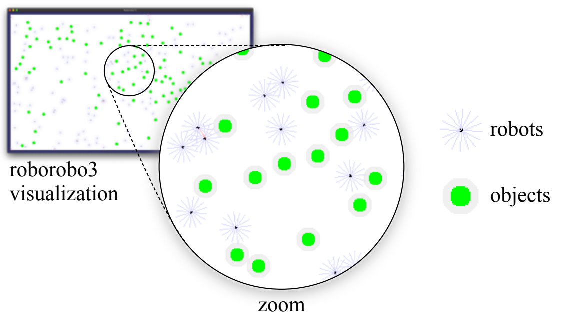


Fig. 5. Arena used for the experiments. It features 150 robots (small blue dots) with 16 short-range sensors and 100 items (green dots). Items disappear when caught, to reappear at a new random location. Robots are never relocated, and the HIT algorithm runs as a distributed on-line fashion.

Parameter	Value
Environment parameters	
Population size	150
Number of objects	100
Arena size	1400px × 800px
Robot size	5px × 5px
Sensor length	15px
Maximum velocity $\mathrm{v}_{\max}$	2 px/steps
Maximum angular velocity $\omega_{\max}$	30 degrees/steps
Controller: multi-layered Perceptron	
Initialisation range	$[-400, 400]$
Sensory inputs	64
Hidden layer	1
Hidden size	16
Control outputs	2
Total number of parameters	1074
Controller: simple Perceptron (only Sec. IV-B)	
Initialisation range	$[-400, 400]$
Sensory inputs	163
Control outputs	2
Total number of parameters	328
HIT parameters	
Evaluation time T	400
Transfer rate α	varying
Mutation size σ	varying

TABLE I
PARAMETERS

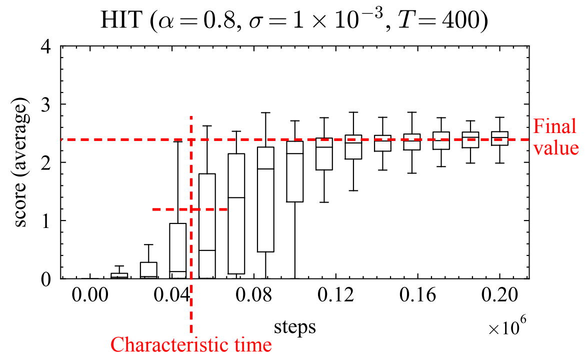


Fig. 6. Results with HIT, $\alpha = 0.8, \sigma = 0.001, T = 400$. The parameters used for these simulation are described on Table I. The Y-axis represents the distribution of the average score among all 150 agents, over all runs. Results are compiled from 128 independent simulation runs.

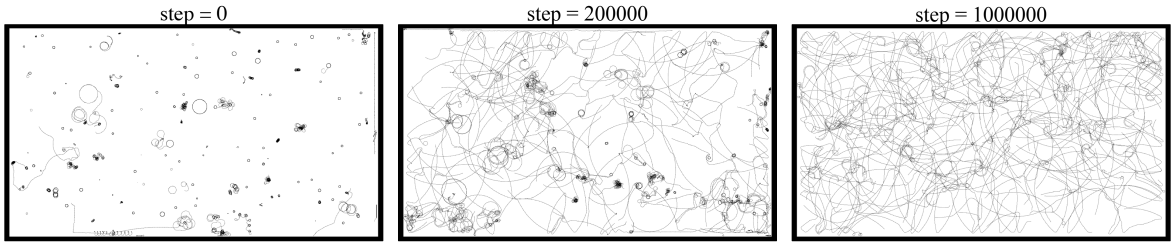


Fig. 7. Trajectories of agents at different steps. Robots are initialized with random policies. We can therefore observe a lot of curved trajectories. Robots that achieve high translation will get more rewards and will better propagate their parameters in the population. After convergence to a unique policy, we observe large translation for all agents.

- the **final score**: to assert the quality of a particular algorithm, we measure the score after convergence;
- the **characteristic time**: to evaluate the speed of convergence, we measure the time at which the average score is half the final score. This is approximately the position of the inflexion point of the data sequence that plots the median score. This was chosen *a posteriori* as in all the experiments we conducted, we *always* observed a sigmoid-like increase of the score when switching from the initial low score values to the final score values.

Figure 6 compiles the results obtained with 128 replicates of the HIT algorithm, with $\alpha = 0.8, \sigma = 0.001, T = 400$. Starting with low values, the score increases between 40000 and 100000 steps (characteristic time ~70000), and then converge to a stable value (final score of ~2.4).

It should also be noted that the variance after convergence is rather small, advocating for the robustness of the algorithm. This is actually confirmed by comparing HIT with a canonical state-of-the-art embodied evolution algorithm, which we refer to as VanillaEE [8], [10], [16].

Figure 8 shows the results in term of characteristic times and final scores of two variants of the HIT algorithm (HIT($\alpha = 0.3$), which is expected converge slower, and HIT($\alpha = 0.8$), as shown before) and of the best-shot of VanillaEE we could find (i.e. $\sigma = 0.001$ mutation rate, elitist selection). To account for the implementation difference between HIT and VanillaEE[1], we re-evaluate the final scores by extracting the control parameters from the last generation, and then running these with the learning algorithm deactivated. All claims are backed using Mann-Whitney U Test.

Firstly, the final score is roughly similar for both HIT variants, which is expected. Both display also an advantage over the VanillaEE control algorithm. Secondly, the characteristic time shows, as expected, that HIT($\alpha = 0.3$) provides the slowest convergence speed. HIT($\alpha = 0.8$) and VanillaEE converge faster, which is actually true *on average*. However, VanillaEE displays a higher variance both in terms of convergence speed and final scores when compared to HIT($\alpha = 0.8$).

Finally, we take a closer look at one typical run of HIT($\alpha = 0.8$). Figure 7 presents the trajectory of the 150 robots during 400 iterations at different steps of the simulation. Starting with randomly initial policy parameters, which produces turning or non-moving behaviours, robots gradually learn to move around to explore their environment, which is both efficient with respect to foraging and mating.

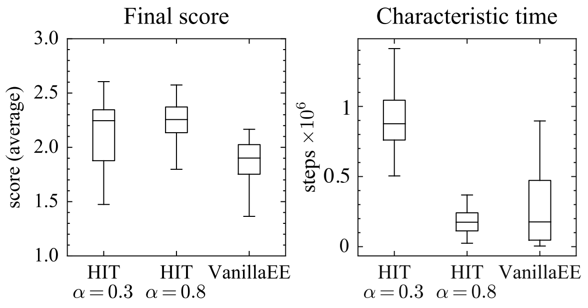


Fig. 8. Comparison between HIT and VanillaEE all with $\sigma = 10^{-3}$. The box plots are computed with 80 independent runs. The characteristic time measures the step at which the average score reaches half its final value. The final score is the average score after saturation.

This change in behaviour is also captured by looking at the evolution of the amount of communication during the course of evolution. For this typical run, Figures 10 and 9 respectively show the evolution of score and number of messages exchanged through time. Comparing the two figures reveals that the increase in communication between robots actually precedes the increase of score values, and remains stable throughout the experiment, even before the final score value is reached.

B. Tradeoff between Speed and Accuracy

Both the transfer rate α and the mutation size σ can have an impact on learning speed and quality. In this Section, we provide an extensive analysis of these two meta-parameters. We measure the characteristic time and the final score the policies obtained by the HIT algorithm, for a large combination of α and σ values.

We explore σ values from 0.2 to 1.0 with a step size of 0.1, and σ from 10^{-2} to 10^{-8}, and 0. This represents a total of 18432 independent runs (9×8 combinations, 256 replicates per combination)[2]. In order to minimise to computational cost,

[1]The original implementation of VanillaEE is synchronous, meaning that all robots update their policy at the same time, which HIT does not do.

[2]Experiments took 12 days using an Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz

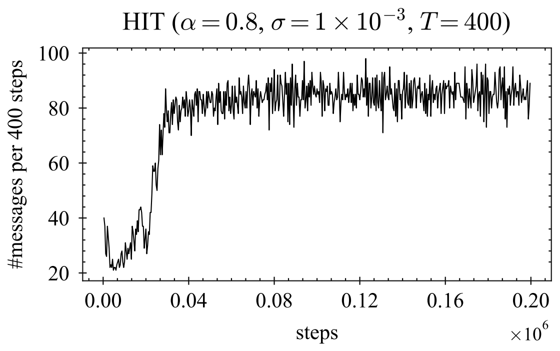


Fig. 9. Evolution of the frequency of messages for one experiment, $\alpha = 0.8, \sigma = 0, T = 400$. We count the number of message sent in a constant interval of 400 iterations and report this value as a function of the step number

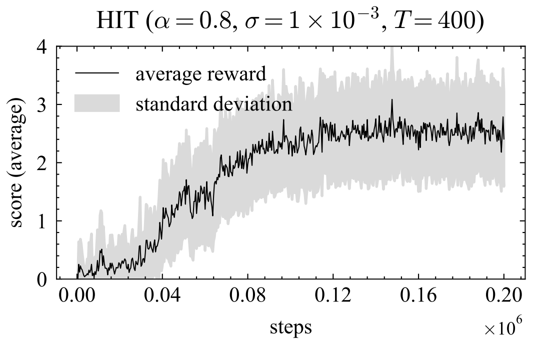


Fig. 10. Evolution of the average score among 150 agents for one experiment of foraging 100 objects, $\alpha = 0.8, \sigma = 0, T = 400$. At every encounter, the best agent share half randomly chosen parameters to the other agent. The worst receives these parameters and use them to improve it own policy. This process can only happen after both agent have spend at least 400 steps to evaluate themselves. During the maturation period, the score before modification of the best is reported for the averaging. No mutations are used here.

we use Perceptron with no hidden layer, but with additional sensory inputs that do not bring useful information.

Results are shown in Figures 11 (final score) and 12 (characteristic time). We can observe that the average score plateaus for mutation rates with $\sigma < 10^2$ and a transfer rate with $\alpha \leq 0.9$. Large mutation rate of $\sigma > 0.01$ as well as the maximum value for transfer rate $\alpha = 1$ are both destructive, whatever the other meta-parameter value. Regarding mutation, it is expected that a too large mutation rate can disrupt selection, and injects noise rather diversity that can be exploited. As for transfer rate, setting $\alpha = 1$ limits convergence to full control parameter sets present in the initial population only, as it is does not allow to benefit from recombination.

Aside from these extremes, HIT is revealed to be remarkably robust in terms of the final score that is reached. The transfer rate appears as the main parameter to modulate the convergence speed, with mutation being either destructive in the worst case $\sigma \geq 10^2$, or of limited interest. In terms of efficiency of policies, the algorithm is rather robust with respect to its meta-parameter values, as long as extreme values are avoided ($\alpha > 0.9$ and $\sigma \geq 10^2$).

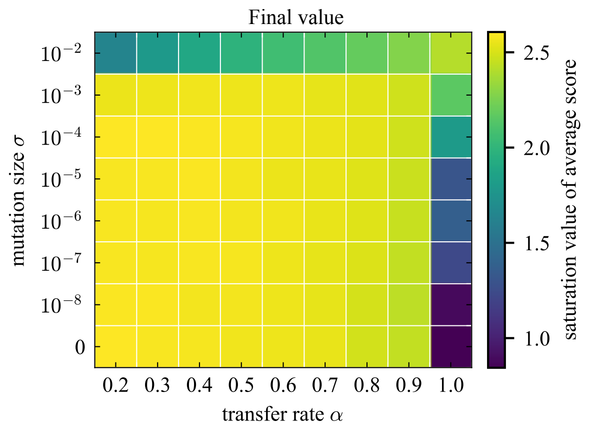


Fig. 11. Final value of the average score for various mutation size and transfer rate. This value is the average of all score on the last 100 measurements (one measurement per 400 iterations). 256 independent simulation run have been used. Lighter colour means better saturation value.

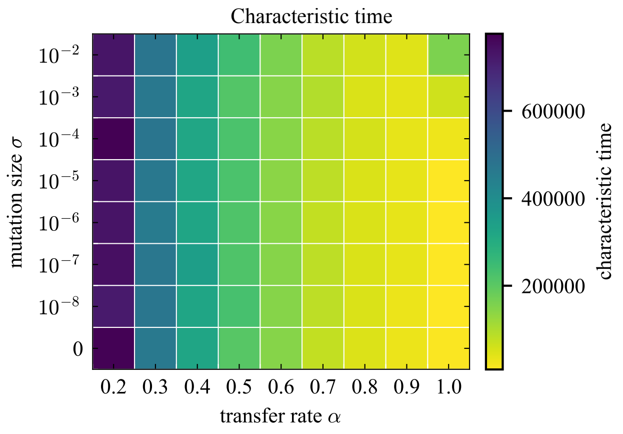


Fig. 12. Characteristic time for various mutation size and transfer rate. The characteristic time is the step at which the average score reach half of it final value. 256 independent simulation run have been used. Lighter colour means shorter convergence time.

C. Learning the Transfer Rate

As stated at the beginning of this paper, our aim is to provide an algorithm that can cope with limited communication capabilities. However, it is hardly possible to know in advance what are the limits. Technical specifications provide a good start, but the impact of robots density on packet collisions or perturbations from other sources makes it difficult to set an appropriate transfer rate beforehand.

Here, we study the HIT algorithm in a pseudo-realistic context where communication is artificially limited. In this experimental setting, all messages larger than $m \times 0.6 \times$ *size(parameter set)*, are lost. The optimal transfer rate is therefore $\alpha_{opt} = 0.6$, but is unknown before deployment[3].

[3]We also tested for $\alpha = 0.3$ with identical results (not shown here).

Resilience to communication failure is addressed by implementing a straight-forward learning method for the transfer rate. The transfer rate value is incorporated in the parameter set of each robot, and initially set to a random value $\alpha \in [0, 1]$. It is then tuned by selection pressure.

We performed 64 experiments, setting a mutation rate to $\sigma = 0.001$ only for the parameter α and $T = 400$ (as earlier). For *all* experiments, the whole robot swarm systematically converged to an α value close to the optimal transfer rate as shown on Figure 13 (left).

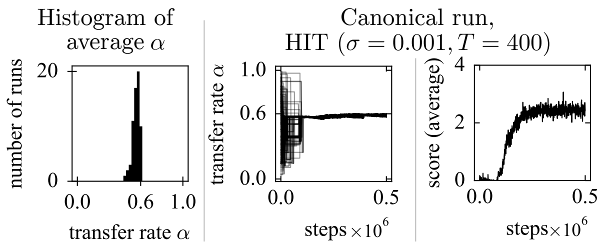


Fig. 13. **Left**: distribution on 64 independents runs of the average transfer rate α after 5×10^5 iterations, with α_{opt} = 0.6. **Center** and **Right**: learning the transfer rate α in a canonical run. Center: each line represent the trajectory of the α value for one specific agent. Right: average score for the population until convergence.

This is illustrated in Figure 13 (center), which represents the distribution of all α from the 150 robots across time steps for a typical run. As expected, α values are initially uniformly distributed between 0 and 1, and then quickly converge to a value close to α_{opt}.

Robots carrying α values which are too large with respect to the communication constraints simply cannot spread, and can only survive as long as it takes to encounter a robot with better score and capable of transmitting its own parameters. On the other hand, robots with small values of α can spread, but may do so at a smaller frequency than robots with relevant α values. This can be viewed as a lexicographic selection: the better performing robot will always fare better, and will diffuse faster if its α value makes the best of the environment at hand.

From a practical viewpoint, and extrapolating also from results in the previous section, it means that learning the transfer rate is always a good idea as long as it is limited to be strictly inferior to 1.

V. Conclusion

In this paper, we introduced a new algorithm called *Horizontal Information Transfer*, which is at the crossroad of embodied evolution and social learning. We showed that this algorithm is competitive with the state of the art, but is also able to deal with limited communication capabilities that are often met with low-cost robots used in swarm robotics.

The obvious next step is to run full experiments with a swarm of low-cost robots for which preliminary results (not shown here) are still limited but encouraging.

Acknowledgement

This work is funded by the Agence Nationale pour la Recherche under Grant No ANR-18-CE33-0006.

References

[1] Levent Bayindir. A review of swarm robotics tasks. *Neurocomputing*, 172:292–321, 2016.
[2] Gerardo Beni. From Swarm Intelligence to Swarm Robotics. *Robotics*, 3342:1–9, 2005.
[3] Manuele Brambilla, Eliseo Ferrante, Mauro Birattari, and Marco Dorigo. Swarm robotics : A review from the swarm engineering perspective. *Swarm Intelligence*, 7(1):1–41, 2012.
[4] Nicolas Bredeche, Evert Haasdijk, and Abraham Prieto. Embodied evolution in collective robotics: A review. *Frontiers in Robotics and AI*, 5:12, 2018.
[5] Nicolas Bredeche and Jean-marc Montanier. Environment-driven Embodied Evolution in a Population of Autonomous Agents. In *Parallel Problem Solving from Nature (PPSN)*, pages 290–299, 2010.
[6] Nicolas Bredeche, Jean-Marc Montanier, Wenguo Liu, and Alan F T Winfield. Environment-driven Distributed Evolutionary Adaptation in a Population of Autonomous Robotic Agents. *Mathematical and Computer Modelling of Dynamical Systems*, 18(1):101–129, 2012.
[7] Nicolas Bredeche, Jean-Marc Montanier, Berend Weel, and Evert Haasdijk. Roborobo! a fast robot simulator for swarm and collective robotics. *CoRR*, abs/1304.2888, 2013.
[8] Iñaki Fernandez Pérez, Amine Boumaza, and François Charpillet. Comparison of Selection Methods in On-line Distributed Evolutionary Robotics. In *Proceedings of the fourteenth international conference on the synthesis and simulation of living systems*, pages 1–16, 2014.
[9] Heiko Hamann. *Swarm Robotics - A Formal Approach*. Springer, 2018.
[10] Emma Hart, Andreas Steyven, and Ben Paechter. Improving Survivability in Environment-driven Distributed Evolutionary Algorithms through Explicit Relative Fitness and Fitness Proportionate Communication. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pages 169–176, 2015.
[11] Inman Harvey. The microbial genetic algorithm. In *European Conference on Artificial Life*, pages 126–133. Springer, 2009.
[12] Jacqueline Heinerman, Dexter Drupsteen, and A E Eiben. Three-fold Adaptivity in Groups of Robots: The Effect of Social Learning. In Sara Silva, editor, *Proceedings of the 17th annual conference on Genetic and evolutionary computation*, GECCO '15, pages 177–183. ACM, 2015.
[13] Jacqueline Heinerman, Massimiliano Rango, and A. E. Eiben. Evolution, individual learning, and social learning in a swarm of real robots. In *Proceedings - 2015 IEEE Symposium Series on Computational Intelligence, SSCI 2015*, pages 1055–1062. IEEE, 2016.
[14] Noemí Carranza Xaver Diego Fredrik Jansson Jaap A. Kaandorp Sabine Hauert Ivica Slavkov, Daniel Carrillo-Zapata and James Sharpe. Morphogenesis in robot swarms. *Science Robotics*, 3(25), 2018.
[15] Alejandro Cornejo Michael Rubenstein and Radhika Nagpal. Programmable self-assembly in a thousand-robot swarm. *Science*, 345(6198):795–799, 8 2014.
[16] Jean-marc Montanier, Simon Carrignon, and Nicolas Bredeche. Behavioural Specialization in Embodied Evolutionary Robotics: Why so Difficult ? *Frontiers in Robotics and AI*, pages 1–17, 2016.
[17] Abraham Prieto, Francisco Bellas, Andres Faina, and RichardJ. Duro. Asynchronous Situated Coevolution and Embryonic Reproduction as a Means to Autonomously Coordinate Robot Teams. In *Knowledge-Based and Intelligent Information and Engineering Systems SE - 43*, volume 5711 of *Lecture Notes in Computer Science*, pages 351–359. Springer Berlin Heidelberg, 2009.
[18] Michael Rubenstein, Christian Ahler, and Radhika Nagpal. Kilobot: A low cost scalable robot system for collective behaviors. *Proceedings - IEEE International Conference on Robotics and Automation*, pages 3293–3298, 05 2012.
[19] Vito Trianni, Stefano Nolfi, and Marco Dorigo. Evolution, Self-organization and Swarm Robotics. In *Swarm Intelligence*, pages 163–191. 2008.
[20] Gabriele Valentini, Eliseo Ferrante, Heiko Hamann, and Marco Dorigo. Collective decision with 100 kilobots: Speed versus accuracy in binary discrimination problems. *Autonomous Agents and Multi-Agent Systems*, 30(3):553–580, May 2016.
[21] Richard A. Watson, Sevan G. Ficici, and Jordan B. Pollack. Embodied Evolution: Distributing an evolutionary algorithm in a population of robots. *Robotics and Autonomous Systems*, 39(1):1–18, 4 2002.