GESwarm: Grammatical Evolution for the Automatic Synthesis of Swarm Robotics Collective Behaviors

Author 1
Some lab
something@mail.com

Author 2
Some lab
something@mail.com

Author 3
Some lab
something@mail.com

Author 4
Some lab
something@mail.com

ABSTRACT

In this paper we propose a novel methodology for automatically synthesizing collective behaviors for swarms of autonomous robots. This methodology automatically derives the microscopic rules and interactions among robots needed to achieve the desired macroscopic behavior. Evolutionary robotics typically relies on artificial evolution for tuning the weights of an artificial neural network that is then used as microscopic behavior representation. The main caveat of neural networks is that they are very difficult to reverse engineer, meaning that once a suitable solution is found, it is very difficult to analyze, to modify, and to tease apart the inherent principles that lead to the desired collective behavior.

In this paper we propose GESwarm, a novel tool that, paired with artificial evolution, is used to automatically synthesize completely readable and analyzable microscopic rules that lead to the desired macroscopic collective behavior. The core of our method is a grammar that can generate a rich variety of collective behaviors. We test GESwarm by evolving a foraging strategy using a realistic swarm robotics simulator. We then systematically compare the evolved behavior against a hand-coded behavior for performance, scalability and flexibility, and show that GESwarm systematically outperforms the hand-coded one.

Categories and Subject Descriptors

I.2.11 [**Computing Methodologies**]: Artificial Intelligence—*Distributed Artificial Intelligence*; I.2.9 [**Computing Methodologies**]: Artificial Intelligence—*Robotics*; I.2.8 [**Computing Methodologies**]: Artificial Intelligence—*Problem Solving, Control Methods, and Search*

General Terms

Algorithms

Keywords

swarm robotics, evolutionary robotics, genetic programming

1. INTRODUCTION

Swarm robotics is the study and the design of collective behaviors for swarms of autonomous robots [?, ?]. The desired collective behavior is the result of local interactions among robots and between robots and the environment. The goal in swarm robotics is to design collective behaviors that are flexible (to different environments), robust (to robot failures) and scalable (to different swarm and problem sizes). Swarm robotics relies on principles such as self-organization and local interaction rather than on centralized coordination and global communication. Despite all these advantages, swarm robotics suffers from the so-called *design problem* [?]: given the desired macroscopic collective behavior, it is not trivial to design the corresponding microscopic behaviors and interaction rules.

Several approaches have been proposed to tackle the design problem in swarm robotics. A common approach is to use automatic design methods to derive automatically microscopic behaviors and interactions given some macroscopic description of the collective behavior or task [?]. Evolutionary robotics [?] is the most commonly used automatic design framework in swarm robotics. In evolutionary robotics, evolutionary algorithms are used to find collective behaviors that maximize a fitness function, which is used to evaluate the performance of the entire swarm. To represent the microscopic behavior of an individual robot, artificial neural networks (ANNs) are typically used, which directly map the robots' sensory inputs into actuators values. Despite their generality, ANNs suffer from a major drawback: they consist of black-box models of individual behaviors that, once obtained, are very difficult to reverse engineer. Thus, it is not easy to either obtain insights on the evolved principles responsible for the self-organized collective behavior or to modify the latter in order to comply with different requirements.

In this paper, we make a step forward into the realization of an automatic design tool for swarm robotics able to produce analyzable and modifiable collective behaviors. We propose GESwarm, a method based on Grammatical Evolution (GE) [?] with a novel generative grammar that is able to synthesize a rich variety of swarm robotics collective behaviors. In GESwarm, it is possible to provide a set of low level individual behaviors, well tested in simulation and potentially also on real robots, that through artificial evolution are combined until more complex behaviors are generated. The evolved behavior is a set of rules responsible for switching from one low level behavior to another in response to internal states and local environmental conditions. These evolved rules can be easily read, analyzed and modified due to their intuitive representation.

We present a case study of foraging task as validation for this method. Robots have to collect objects present at a source location and drop them at the goal location. After analyzing the evolutionary performance of the algorithm, we perform systematic experiments in which we compare the evolved behaviors against a hand-coded one. Additionally, we perform experiments in different conditions than the ones used during evolution, showing that the evolved behavior generalizes well.

The remaining of the paper is organized as follows. Section 2 introduces GESwarm. Section 3 presents the experimental setup and the results of our experiments. Section 4 presents a discussion in which we enframe GESwarm in the literature of automatic design methodologies for swarm robotics. In Section 5, we conclude the paper and we discuss possible future directions.

2. GESWARM

GESwarm combines concepts derived from three different disciplines: formal language theory [?], evolutionary computation [?] and swarm robotics [?]. The space of strings of a formal grammar are used as the search space for the evolutionary algorithm. These strings represent an abstract syntax tree which is used as the controller for the robots. A GA is used to evaluate the collective behaviors produced out of these individual behaviors, until a solution with good performance is found. These strings represent a set of rules that are used to combine behaviors taken from a repertoire of low-level swarm robotics behaviors.

In this section we provide details on how this process is carried out: first we explain the type of behavioral representation that can be generated by the grammar (Section 2.1), then we describe the GESwarm grammar and how it can generate such behaviors through evolution (Section 2.2).

2.1 Behavioral representation

We assume that each robot in the swarm is executing the same behavior. Each evolved behavior is a set $\mathcal{R}$ composed of an arbitrary number n_R of *rules* R_i:

$$\mathcal{R} = \{R_i\}, i \in \{1, \ldots, n_R\}.$$

Each rule has the following form:

$$R_i = \mathcal{P}_i \times \mathcal{B}_i \times \mathcal{A}_i,$$

that is, is composed of an arbitrary number of *preconditions* $P_j \in \mathcal{P}_i$, *behaviors* $B_k \in \mathcal{B}_i$ and *actions* $A_l \in \mathcal{A}_i$, with $j \in \{1, \ldots, n_{iP}\}, k \in \{1, \ldots, n_{iB}\}, l \in \{1, \ldots, n_{iA}\}$. The intuitive interpretation of a rule is: if all the preconditions in $\mathcal{P}_i$ are met, and if the robot is executing any of the behaviors contained in $\mathcal{B}_i$, all actions contained in $\mathcal{A}_i$ are executed. We now explain what preconditions, behaviors and actions are.

A precondition P_j is something that can be true or false about the environment. Currently, GESwarm allows for boolean preconditions, but can be extended to include preconditions comparing variables to natural numbers. Example of preconditions are: P_{ON_AREAX} (which is true if the robot is on top of a given area $AREAX$ and false otherwise), $P_{SEES\_OBSTACLE}$ (which is true if the robot sees obstacles close-by or false otherwise), $P_{IS\_CLOSE\_TO(k)}$ (which is true if the robot sees k objects or other robots and false otherwise), etc. Preconditions are meant to be simple sensorial pre-processing steps, depending on the available robot technology. For the experiments in this paper, and to be faithful to the swarm robotics philosophy, we only use preconditions that can be implemented using the on-board sensors of the robots we considered. The precondition of rule R_i, is assumed to be the conjunction $\bigwedge_{P \in \mathcal{P}_i} P$ (logical "and") of all preconditions in $\mathcal{P}_i$, that is, rule R_i is activated (and the corresponding actions executed) only if all preconditions are met. Consistent with the above definition, if the set of preconditions is empty ($\mathcal{P}_i = \emptyset$), the precondition is assumed to be satisfied.

A behavior B_k is a low level primitive that a robot can execute for a given amount of time. Examples of low level behaviors are: $B_{PHOTOTAXIS}$ (the robot moves in the direction of the highest light intensity), B_{RANDOM_WALK} (the robot executes a directed random walk), $B_{FOLLOW\_COLOR(c)}$ (the robot moves in the direction of a given color c, which could be static or signaled by other robots), etc. In a rule R_i, $\mathcal{B}_i$ can contain more than one behavior. Rule R_i is activated only if the robot is executing any of the behaviors in $\mathcal{B}_i$. If the set of behaviors is empty ($\mathcal{B} = \emptyset$), the rule is considered activated regardless of the behavior the robot is currently executing.

Each action A_l is associated with a probability value p_l. Allowing probabilistic actions has been extensively used in swarm robotics in order to increase flexibility [?, ?]. Provided that preconditions $\mathcal{P}_i$ are met and the robot is executing one of the behaviors in $\mathcal{B}_i$, the action is executed with probability p_l in any given timestep. Delayed execution of actions is achieved by values of p_l smaller than 1. Actions can be one of two types. The first type is *behavior change*, or A_B. With A_B actions, the robot can switch from one behavior to another. Therefore, A_B actions are always followed by an argument indicating the behavior to switch to. The other type is *internal state change*, or A_{IS}. These actions are used to either produce some immediate response by the robot (i.e. turning on its LEDs or dropping an object that has been collected earlier) or to change its propension to do something (i.e. to collect more objects or to go past a given obstacle in the environment). A_{IS} actions are followed by two arguments, the first indicating the internal state variable to change and the second indicating the variable's new value.

2.2 Grammatical evolution

GESwarm is based on Grammatical Evolution (GE) [?]. GE is related to Genetic Programming [?] (GP), which deals with the automatic synthesis of computer programs using artificial evolution. The main difference between the two is that GE employs a grammar to evolve programs for arbitrary languages [?] as opposed to GP which is bound to a specific language such as LISP. GESwarm uses GE and a formal grammar to evolve a set of rules $\mathcal{R}$ as the ones described in Section 2.1. The GESwarm grammar is the following:

$$
\begin{aligned}
S &\rightarrow \mathcal{R} && (1)\\
\mathcal{R} &\rightarrow R\,\mathcal{R}|R && (2)\\
R &\rightarrow \mathcal{P}\,\mathcal{B}\,\mathcal{A} && (3)\\
\mathcal{P} &\rightarrow P\,\mathcal{P}|\varepsilon && (4)\\
\mathcal{B} &\rightarrow B\,\mathcal{B}|\varepsilon && (5)\\
\mathcal{A} &\rightarrow A\,\mathcal{A}|A && (6)\\
P &\rightarrow P_{name} == true|P_{name} == false && (7)\\
B &\rightarrow B_{name} && (8)\\
A &\rightarrow A_B|A_{IS} && (9)\\
A_B &\rightarrow p = \langle value\rangle, B_{name} && (10)\\
A_{IS} &\rightarrow p = \langle value\rangle, IS_{name} = \langle new\_value\rangle && (11)
\end{aligned}
$$

To understand this grammar, recall that formal grammars are made of production rules, each consisting in a left-hand side and a right-hand side separated by the symbol "→". The left-hand side contains one *non-terminal* symbol. The right-hand side contains one or more terminal or non-terminal symbols that can be either concatenated or separated by the special "or" ("|") symbol. The difference between terminal and non-terminal symbols is that non-terminal symbols are further expanded by a production rule whereas terminal symbols are not.

The first rule of a formal grammar always expands the special non terminal S. In our case, S is expanded in the non-terminal $\mathcal{R}$ that represents a set of rules (Expansion 1). In the following expansions, each non terminal can be expanded to either groups of non-terminal and terminal symbols or to groups of terminal symbols alone. The grammar has completed producing a valid string only when all non-terminals have been expanded. We now explain how the GESwarm grammar can produce valid swarm robotics behaviors.

The production rule 2 denotes a list-type expansion: it says that $\mathcal{R}$ can only be expanded as a list of rules R of arbitrary length. To produce a list of n rules, one applies the $R\,\mathcal{R}$ part of the rule recursively $n-1$ times, and lastly expand $\mathcal{R}$ into the sole terminal R, thus terminating the list. Production rule 3 defines that a rule $\mathcal{R}$ is a set of preconditions $\mathcal{P}$ followed by a set of behaviors $\mathcal{B}$ followed by a set of actions $\mathcal{A}$. Production rules 4, 5 and 6 are also list-type expansions that correspond to $\mathcal{P}$, $\mathcal{B}$ and $\mathcal{A}$ being lists of preconditions (P), behaviors (B) and actions (A), respectively. Since both preconditions and behaviors can be empty, ε is interpreted both as the empty list, and the end of a list. Production rules 7 and 8 only produce terminal symbols: they describe what a precondition (a logical evaluation of a given precondition variable P_{name}) and a behavior (a behavior variable B_{name}) are. Production rule 9 expands to one of two possible non-terminal symbols, thus selecting between behavior change actions (A_B) and internal state change actions (A_{IS}). Finally, production rules 10 and 11 describe what these two action are. In both cases, the first part of the expansion sets the value of the probability p associated with the action to the provided real number $\langle value\rangle$. Fir A_B, this assignment is followed by the behavior B_{name} to change to, while for A_{IS} it is followed by an internal state assignment.

GESwarm uses the above grammar to generate candidate solutions that can be evolved via GE. Each of the initial set of candidate solutions is produced from the grammar by successively selecting a random production rule to apply (starting from S) until a valid string has been produced. In Section 3.1 we give more details on the experimental setup used for GE and for the swarm robotics simulations.

3. EXPERIMENTS WITH EVOLUTION OF FORAGING

In this section, we show how GESwarm can be used to successfully generate swarm robotics collective behaviors. As a case study, we use GESwarm to evolve a collective foraging behavior.

3.1 Experimental setup

We now explain the setup used to carry out the experiments, by describing the swarm robotics experimental setup (Section 3.1.1) used for executing the foraging simulations and the configuration of the evolutionary algorithm (Section 3.1.2).

3.1.1 Robotic setup

We consider a foraging scenario in the arena depicted in Figure 1a. A swarm of N robots has to collect items from a region of the arena that we call *source* and bring them to another region that we call *nest*. In the source, 5 objects are placed. New objects are generated at a random location every time one is picked up by a robot, so that there are always 5 items present at the source. A light source is located far north, beyond the borders of the arena and allows robots to execute the phototaxis and anti-phototaxis behaviors necessary to navigate the arena.

The experiments were carried out using the ARGoS simulator [**?**]. ARGoS is an open-source simulator[1] capable of simulating up to tens of thousands of robots in real time by exploiting modern multi-core computers and a plugin-based architecture. The robot involved in the experiments is a simulated version of the foot-bot robot [**?**], which is a differential-drive, non-holonomic, mobile robot. The physical foot-bot robot is depicted in Figure 1b.

We implemented the following low-level behaviors that, recombined by GESwarm, can produce the desired higher level foraging strategy. These behaviors exploit only the sensors and actuators shown in Figure 1b. The output of these behaviors are vectors that point to a target direction.

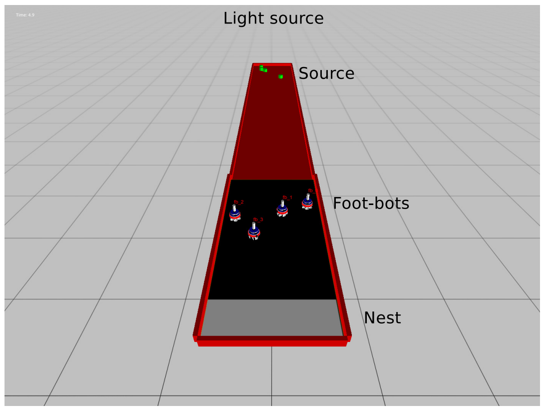
$B_{PHOTOTAXIS}$ This behavior only uses the light sensor. The output vector points toward the direction with the highest perceived light intensity.

$B_{ANTI-PHOTOTAXIS}$ This behavior also only uses the light sensor. The output vector points toward the lowest perceived light intensity.

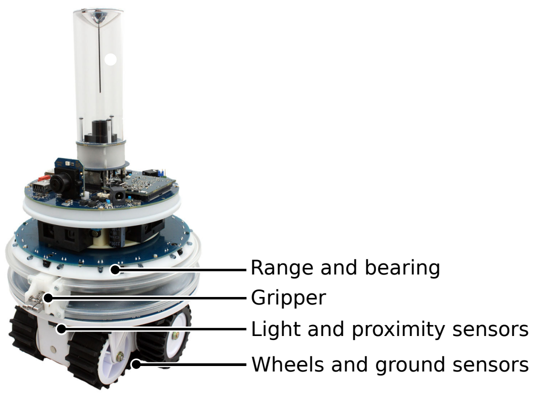
B_{RANDOM_WALK} This behavior doesn't use any sensor. The output is a random unit vector that remains constant for a random amount of time.

These output vectors are post-processed by adding an obstacle avoidance vector. The obstacle avoidance vector is computed using two sensors: the proximity sensors for sensing the walls and the range and bearing sensors to sense other robots. The obstacle avoidance vector points away

[1]http://iridia.ulb.ac.be/argos



(a)



(b)

Figure 1: Experimental setup. (a) A snapshot of the ARGoS simulator that explains our experimental setup. (b) The real foot-bot robot and the sensor/actuators that were used.

R_1:	If not holding an object and not at the source, go to the source		
$\mathcal{P}_1$	$P_{ON\_SOURCE} == false$	$P_{HAS\_OBJECT} == false$	
$\mathcal{B}_1$	ε		
$\mathcal{A}_1$	A_B	$p = 1$	$B_{PHOTOTAXIS}$
R_2:	If arrived at the source and not holding an object, start looking for objects		
$\mathcal{P}_2$	$P_{ON\_SOURCE} == true$	$P_{HAS\_OBJECT} == false$	
$\mathcal{B}_2$	$B_{PHOTOTAXIS}$		
$\mathcal{A}_2$	A_B A_{IS}	$p = 1$ $p = 1$	B_{RANDOM_WALK} $IS_{WANT\_OBJECT} \leftarrow true$
R_3:	If holding an object, go back to the nest		
$\mathcal{P}_3$	$P_{HAS\_OBJECT} == true$		
$\mathcal{B}_3$	B_{RANDOM_WALK}	$B_{PHOTOTAXIS}$	
$\mathcal{A}_3$	A_B	$p = 1$	$B_{ANTI-PHOTOTAXIS}$
R_4:	If arrived at the nest and still holding an object, drop it and start random walk		
$\mathcal{P}_4$	$P_{ON\_NEST} == true$	$P_{HAS\_OBJECT} == true$	
$\mathcal{B}_4$	B_{RANDOM_WALK}	$B_{ANTI-PHOTOTAXIS}$	
$\mathcal{A}_4$	A_B A_{IS}	$p = 1$ $p = 1$	B_{RANDOM_WALK} $IS_{DROP\_OBJECT} \leftarrow true$

Table 1: The hand-coded behavior

from the aggregate of the relative positions of all sensed objects. The integrated output vector is then used to obtain the two output speeds which are then applied to robot's wheels, as in [**?**].

Additionally, a combination of ground and light sensor are also used to detect whether the robot is on the source, on the nest or somewhere else. This information is used to evaluate the preconditions P_{ON_SOURCE} and P_{ON_NEST} used in the rule set. Finally, a virtual simulated gripper sensor is used to let the robot pick-up and drop objects. When a robot picks-up an object, the precondition P_{HAS_OBJECT} evaluates as true.

To evolve the behavior, we use 4 robots. When assesing scalability and flexibility, we use up to 20 robots. The total allotted time for the foraging task is 5000 simulated seconds.

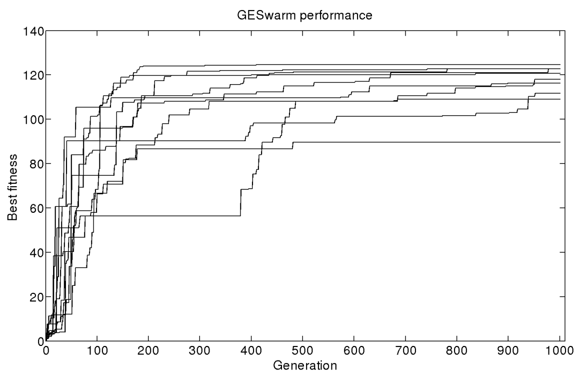
3.1.2 Evolutionary setup

We use an existing library called GEVA [**?**] for GE. GEVA maps the GESwarm rules into strings of integers. We execute a total of 10 evolutionary runs. Each evolutionary run lasts 1000 generations and involves 100 individuals. Since swarm robotics simulations are stochastic, each individual is evaluated 3 times. We use a single-point crossover with probability 0.3 and a mutation probability of 0.05. Crossover and mutation are applied directly to the string of integers representing the rules. We choose a generational-type of replacement with 5% elitism. Generational-type is used instead of steady-state type because it allows for massive parallelization of the simulations on a computer cluster. We use roulette-wheel selection, that is, the probability that an individual is selected for reproduction (which may involve mutation and crossover) is proportional to its fitness relative to the fitness of all individuals.

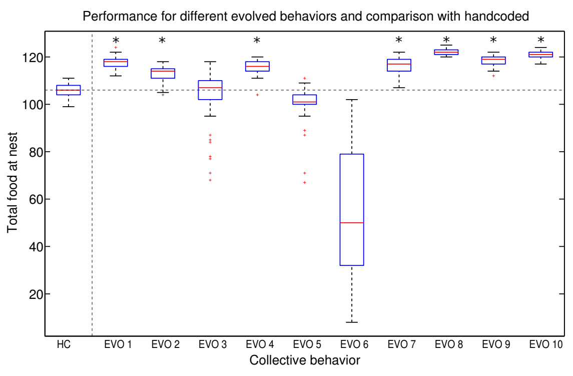
The fitness function used is simply the total number of objects collected during one simulation. The same quantity was also used to validate the evolved behavior (Section 3.2.1 and Section 3.2.2).

3.2 Results

We now analyze the ten evolved behaviors (Section 3.2.1) against the hand-coded behavior shown in Table 1, and



(a)



(b)

Figure 2: Results obtained with GESwarm used to evolve a foraging behavior. (a) shows the value of the fitness of the best individual found so far as a function of generation time, for the the 10 evolutionary runs. A good foraging behavior was evolved in all runs but one. (b) shows the results of evaluating 50 times all evolved (EVO X) and the hand-coded (HC) behaviors. We report the total number of collected objects in the allotted time. The dashed horizontal line represents the baseline performance (i.e. the mean performance of the HC behavior). Seven out of the ten evolved behaviors outperform significantly ($p-$ value < 0.001) the hand-coded behavior, denoted by $*$.

further analyze EVO 8, the best evolved behavior (Section 3.2.2).

3.2.1 *The evolved behavior*

In Figure 2a we show the fitness of the evolved behaviors as a function of the generations. For each run, we report the fitness of the best individual obtained so far. The behaviors obtained by the end of the runs are all functional solutions for the foraging problem, as can be seen by their ability to successfully collect a significant number of objects. This highlights a strength of GESwarm: all evolutionary runs obtain a functional behavior.

We compare the performance of all evolved behaviors against the hand-coded behavior. We evaluated the performance of all 10 evolved behaviors (EVO X) and the hand-coded behavior (HC) 50 times each. These experiments were performed in the same environmental setting used in the evolutionary runs. Results are reported in Figure 2b. Here, HC is significantly ($p-$ value < 0.001) outperformed by seven out of the ten evolved behaviors, has performance comparable to two other behaviors, and performs better than only one of them ($EVO6$). In this latter case, the evolutionary process could not get past a local optimum (Figure 2a). However, this happened only in one out of the ten cases.

3.2.2 *Scalability and flexibility analysis*

We further analyze the performance of the best evolved behavior (EVO 8, reverse-engineered in Table 2). We executed two sets of experiments. In both sets, we analyzed the performance of the system with larger swarm sizes compared to the one used during evolution (scalability analysis). In the first set, we keep the robot density the same (≈ 0.23 robots$/m^2$), increase the environment width and the object availability accordingly. In the second set, we increased the robot density by a factor of 4 (≈ 0.9 robots$/m^2$) and reduced the object availability at the source by a factor of 2 (flexibility analysis). Both the width and the length of the environment had to be modified to achieve this. As a side effect, the overall performance of the scalability and flexibility analyses cannot be directly compared. For statistical strength, 50 runs for each setting were executed.

Figure 3a and Figure 3b summarize the results of the scalability and flexibility analyses, respectively. The first thing that can be noticed is that the best evolved controller continues to outperform the hand-coded one in all cases. Evolved behaviors generalize well to different conditions other than the ones used during the evolutionary process. Further observations can also be made: in both sets, and for both behaviors, performance scale approximately linearly with respect to the swarm size. This is also true for experiments with higher density of robots (Figure 3b), showing that both behaviors are not strongly affected by robot-to-robot interference and decreased availability of resources.

4. DISCUSSION AND RELATED WORK

Automatic design methods have been, for decades, the holy grail for roboticists and swarm roboticists. Automatic generation of behaviors is very useful with tasks that have not been completely defined in advance or that can change over time. Furthermore, they are even more useful in swarm robotics, as they could ease the derivation of microscopic behaviors and interaction rules given the macroscopic objective. While significant progress has been achieved in automatic design methods, there is still no consensus on which is the best method that can be used within the swarm robotics setting. We classify automatic design methodologies in two main frameworks: reinforcement learning and evolutionary robotics.

In Reinforcement Learning (RL), a single agent or robot learns a behavior by trial-and-error interactions with the environment and receiving rewards and punishment from it. An elegant and unified mathematical framework has been developed [?]. In the multi-agent and multi-robot case, however, less success has been reported. A review of such studies was conducted in [?] and is outside the scope of this

R_1:	The switch to random walk for object exploration is probabilistic and can happen any time		
$\mathcal{P}_1$	ε		
$\mathcal{B}_1$	$B_{PHOTOTAXIS}$		
$\mathcal{A}_1$	A_B	$p = 0.05$	$B \leftarrow B_{RANDOM\_WALK}$
R_2:	If going to the nest and reached the nest, drop any object and go back to the source		
$\mathcal{P}_2$	$P_{ON\_NEST} == true$		
$\mathcal{B}_2$	B_{RANDOM_WALK}	$B_{ANTI-PHOTOTAXIS}$	
$\mathcal{A}_2$	A_B A_{IS}	$p = 1$ $p = 1$	$B_{PHOTOTAXIS}$ $IS_{DROP\_OBJECT} \leftarrow true$
R_3:	Whenever holding an object, the robot should go back to the nest		
$\mathcal{P}_3$	$P_{HAS\_OBJECT} == true$		
$\mathcal{B}_3$	B_{RANDOM_WALK}	$B_{PHOTOTAXIS}$	
$\mathcal{A}_3$	A_B	$p = 0.1$	$B_{ANTI-PHOTOTAXIS}$
R_4:	The default action is to go to look for objects at the source		
$\mathcal{P}_4$	$P_{ON\_NEST} == true$	$P_{HAS\_OBJECT} == true$	
$\mathcal{B}_4$	B_{RANDOM_WALK}	$B_{PHOTOTAXIS}$	
$\mathcal{A}_4$	A_B A_B	$p = 1.0$ $p = 0.001$	$B_{PHOTOTAXIS}$ $B_{ANTI-PHOTOTAXIS}$

Table 2: The best evolved behavior EVO 8. Evolved rules can be easily explained by looking at them.

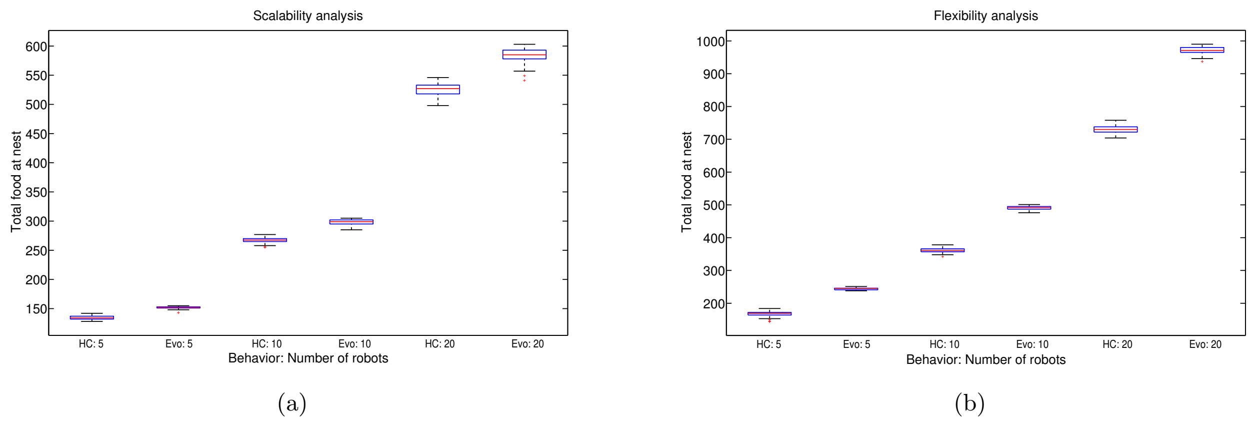


Figure 3: Further analysis of the best evolved behavior. Comparison between the best evolved behavior, Evo 8, against the handcoded behavior. Experiments were performed with 5, 10 and 20 robots. In (a) the density of robots was kept the same to the one where the controller was evolved. In (b) we show results executed in a slightly different environment, in which the robot density is increased and object availability is decreased. As it can be seen, EVO 8 always outperforms the handcoded behavior.

paper. However, we believe RL is not yet mature to be used in swarm robotics. A swarm robotics problem, in fact, can hardly be seen as a RL problem. The user tackles a task at the macroscopic level, but reinforcement learning processes take place at the microscopic level. Hence, the main issue is the *spatial credit assignment*, that is, the decomposition of the global reward into individual rewards [?]. Some works addressed this issue via experiments with few robots (2 to 4), using communication or signaling to share the reward [?, ?].

Evolutionary robotics (ER) has maybe received more success than RL in swarm robotics. This might be due to the different way in which credit assignment is done. In fact, the majority of the works consider homogeneous swarms (all robots use the same individual behavior) and the fitness function evaluates the performance of the entire swarm. Some information about learning dynamics might be lost in this way, but the spatial credit assignment problem is also reduced. Although this is the most common approach, some authors discussed other different ways in which artificial evolution in swarms can take place [?]. They introduced two taxonomies, one categorizing works according to how selection takes place (individual-level vs swarm level) and the other one according to how the swarm is composed (homogeneous vs heterogeneous). For comparison, the experiments considered in this paper utilizes swarm-level selection and homogeneous swarm composition, albeit there is nothing in GESwarm that restricts it to this particular category.

In evolutionary swarm robotics, several principles and collective behaviors have been studied. We here summarize some of the most representative work in the field. In [?], the authors evolved successfully coordinated motion of robots capable of assembling to each other. The evolved behavior was ported to real robots, where they showed that four physically-connected robots could navigate together, explore a complex environment and avoid obstacles. In [?], the authors considered the aggregation collective behavior. They performed experiments both with artificial evolution and with probabilistic hand-coded controller, showing that the former can outperform the latter. In [?], the authors evolved both solitary and collective object transport behaviors. They showed that most of the times evolution was favoring a collective transport that involved robots assembling to each other. In [?], the authors successfully evolved a social learning behavior, in which the robots were able to switch between two behaviors either via environmental stimuli or via communication among them. The evolution of communication was also considered in [?], in which the emergence of signaling is needed in order for the robots to categorize two types of environments. In [?], the authors successfully evolved solitary and cooperative foraging also using real robots. The task involved small objects, that could be pushed by single robots, as well as big objects that required at least two robots to be moved. In [?], the authors used an evolutionary algorithm to generate collective behaviors for a team of robot football players, using fully connected recurrent neural network as a behavioral model. In [?], the authors evolved a collective foraging behavior that relies on very simple sensing mechanisms. The robots could not sense object unless in close proximity, but engaged in a dynamic chain behavior having them following each other and creating a chain connecting the source of objects to the nest. The formed chain was also dynamically changing shape when the object source was let free to move in the environment. In [?] the authors evolved an aggregation collective behavior having robots aggregating to different shelters. The evolved behavior was compared to a model of collective decision making displayed by cockroaches based on differential equations.

The evolutionary robotics work described above all utilized artificial neural networks (ANNs) as the representation of the microscopic behavior of each robot. The advantage of ANNs is their generality, in the sense that in principle they do not bias the behavior to a particular class, but the behavior is only restricted by the size of the neural network and by whether it's recurrent or not. Differently from non-recurrent ANNs, recurrent ANNs include weights connecting output neurons back to input neurons, thus encoding internal memory. In swarm robotics, purely reactive, memory-less behaviors are often (although not always) enough to guarantee self-organizing properties (such as [?]), and this justifies why non-recurrent ANNs are more often used compared to recurrent ANNs.

The main drawback of ANNs is their difficulty in being reverse engineered and analyzed. Only few studies have tried to attempt this step, in general with very simple collective behaviors [?] or small neural networks [?]. In addition to this, the size and topology of the neural network is often fixed a-priori, with few recent exceptions [?]. To overcome these difficulty, few other studies followed an alternative approach in which an alternative behavioral representations is used instead of ANNs. For instance, the authors of [?] used evolutionary computation to tune the parameters of a microscopic behavior represented using artificial virtual physics, in order to perform obstacle avoidance with a swarm of robots. Instead, in [?] the authors used artificial evolution to find finite state machines used to evolve collision avoidance and gate trespassing in a swarm of robots. GeSwarm can be placed in the same context of these latter two studies: it represents a further effort in finding an alternative representation that is reverse-engineerable and suitable for evolving collective behaviors for swarm robotics. Additionally, in GESwarm the size of the representation is not fixed in advance, as an arbitrary number of rules made of an arbitrary number of components can in principle evolve.

5. CONCLUSIONS AND FUTURE WORK

In this paper, we presented GESwarm, a new method for the automatic synthesis of swarm robotics collective behaviors that can be reverse engineered. GESwarm uses artificial evolution to combine a set of low-level individual behaviors into more complex strategies potentially capable of self-organization. We presented an experimental validation, where GESwarm was used to successfully evolve ten foraging behaviors out of the ten performed evolutionary runs, whereby 7 out of 10 outperformed a hand-coded foraging behavior, 2 performed comparatively and only 1 performed worse. This showed that GESwarm is capable to generate functional collective behaviors with a high degree of success. The best evolved behavior was further analyzed, showing that it could generalize well to different conditions such as increased number of robots and increased robot density.

From the swarm robotics perspective, it can be argued that the two main behavioral representations used in the literature are finite state machines and virtual physics-based design [?]. The work presented in this paper represents a significant step forward for achieving automatic design of swarm robotics collective behaviors. In fact, similarly to [?], behaviors evolved with GESwarm are equivalent to probabilistic finite state machines and can self-organize into all class of behaviors encoded via finite state machines. Further research directions involve extensions of GESwarm such as the possibility to evolve: rules that represent virtual physics-based interactions; rules that fully exploit the internal state of the robots by allowing for behaviors that adapt to changing environmental conditions; additional real-valued parameters used to further characterize preconditions and behaviors. Such extension would allow to evolve, we believe, almost or all the swarm robotics collective behaviors that so far have only been developed by hand.

6. ACKNOWLEDGMENTS

This work was partially supported by the Vlaanderen Research Foundation Flanders (Flemish Community of Belgium).