Swarm and Evolutionary Computation 2 (2012) 25–38

Contents lists available at SciVerse ScienceDirect

Swarm and Evolutionary Computation

journal homepage: www.elsevier.com/locate/swevo

Regular paper

Evolving behavioral specialization in robot teams to solve a collective construction task

G.S. Nitschke [a,*], M.C. Schut [b], A.E. Eiben [b]

[a] Computational Intelligence Research Group, Computer Science Department, University of Pretoria, Pretoria, 0002, South Africa

[b] Computational Intelligence Group, Computer Science Department, Vrije Universiteit, Amsterdam, De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands

ARTICLE INFO

Article history:
Received 31 December 2010
Received in revised form
5 August 2011
Accepted 17 August 2011
Available online 14 September 2011

Keywords:
Neuro-evolution
Collective construction
Specialization

ABSTRACT

This article comparatively tests three cooperative co-evolution methods for automated controller design in simulated robot teams. *Collective Neuro-Evolution* (CONE) co-evolves multiple robot controllers using emergent behavioral specialization in order to increase collective behavior task performance. CONE is comparatively evaluated with two related controller design methods in a collective construction task. The task requires robots to gather building blocks and assemble the blocks in specific sequences in order to build structures. Results indicate that for the team sizes tested, CONE yields a higher collective behavior task performance (comparative to related methods) as a consequence of its capability to evolve specialized behaviors.

1. Introduction

The automated design and adaptation of collective behavior in simulated (agent) or situated and embodied (robot) groups [1] often use biologically inspired design principles. Collective behavior refers to group behaviors that result from the interaction of individual agents or robots [1]. The objective of such systems is to replicate desirable collective behaviors exhibited in biological systems such as social insect colonies [2], multi-cellular organisms [3], and economies of a nation and companies [4].

As an essential part of survival in nature, there is a balance of cooperation versus competition for resources between and within different species. An individual's ability to survive (its fitness) changes over time since it is coupled to the fitness of other individuals of the same and different species inhabiting the same environment. Co-adaptation between species is referred to as co-evolution [5] and has manifest itself in the form of increasingly complex competitive and cooperative behaviors [6]. Natural co-evolution has provided a source of inspiration for the derivation of co-evolution algorithms [7]. Co-evolution algorithms work via decomposing a given task into a set of composite sub-tasks that are solved by a group of artificial species. These species either *compete* (competitive co-evolution) or *cooperate* (cooperative co-evolution) with each other in order to solve a given task. Co-evolution provides a natural representation for many collective behavior tasks, since each species is equatable with the behavior of individual agents or robots.

In certain biological systems, behavioral specializations have evolved over time as a means of diversifying the system in order to adapt to the environment [8]. For example, honey bees efficiently divide labor between specialized individuals via dynamically adapting their foraging behavior for pollen, nectar, and water as a function of individual preference and colony demand [9]. That is, in many biological systems, specialization is the fundamental mechanism necessary for the group to adapt to task and environment constraints and achieve optimal efficiency.

This research proposes combining *Neuro-Evolution* (NE) [10] and cooperative co-evolution [11] to adapt *Artificial Neural Network* (ANN) [12] agent controllers in order that a group of simulated agents solve a collective behavior task.

This research falls within the purvey of evolutionary robotics research [13]. Within the larger taxonomy of cooperative multi-robot systems [14]. The robot teams simulated in this research are defined as being *cooperative*, *heterogeneous*, *aware*, *weakly coordinated*, and *distributed*. That is, the teams are *cooperative* in that the robots co-operate in order to perform some global tasks [15]. The teams are *heterogeneous* since each robot is initialized with and develops a different behavior [16] over the course of a cooperative co-evolution process. The teams are *aware* in that robots take into account the actions performed by other robots in order to accomplish their own task [17]. The teams are *weakly coordinated* in that they do not employ any explicit or predefined coordination protocol [17]. In this research, coordination and cooperation are emergent properties resulting from the interaction between robots and their environment [18].

* Corresponding author.
E-mail addresses: gnitschke@cs.up.ac.za (G.S. Nitschke), schut@cs.vu.nl (M.C. Schut), gusz@cs.vu.nl (A.E. Eiben).

2210-6502/$ – see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.swevo.2011.08.002

1.1. Neuro-Evolution, cooperative co-evolution, and collective behavior

NE is the adaptation of ANNs using artificial evolution [10]. The main advantage of NE is that details about how a task is to be solved does not need to be specified *a priori* by the system designer. Rather, a simulator is used to derive, evaluate and adapt controller behaviors for a given task [19]. For a comprehensive review of NE methods the reader is referred to Floreano et al. [20].

Cooperative co-evolution methods use cooperation between multiple species (populations of genotypes) and competition between genotypes within a species to derive solutions. Applied to a collective behavior task, genotypes within a species constitute candidate *partial* solutions to a *complete* solution. That is, genotypes within the same species compete for the role of the fittest genotype (a candidate partial solution). Individuals selected from each species are co-evolved in a task environment where they collectively form complete solutions. Genotypes from each species that work well together (as complete solutions) are selected for recombination. The fittest complete solutions are those that yield the highest collective behavior task performance, when tested in a given simulator.

The advantages of cooperative co-evolution include versatility and applicability to many complex, continuous, and noisy tasks [21]. The use of multiple species provides a natural representation for many collective behavior tasks [22–24], and specialization is often effectuated in the behaviors evolved by species (partial solutions) in response to task and environment constraints [25,26].

An overview of all methods that combine NE and cooperative co-evolution is beyond the scope of this article. Recent reviews of NE based cooperative and competitive co-evolution, applied to solve collective behavior tasks, can be found in [20,27].

Given the success of previous cooperative co-evolution methods that use NE to solve collective behavior tasks, this research proposes to apply the *Collective Neuro-Evolution* (CONE) method, detailed in Section 2 to solve a complex collective construction task. The goal of this article is to evaluate the task performance of CONE in comparison with two related controller design methods that use NE. The two other controller design methods tested are *Multi-Agent Enforced Sub-Populations* (MESP) [28], and *Cooperative Co-Evolutionary Algorithm* (CCGA) [29]. CCGA and MESP were selected since both methods have been demonstrated as being appropriate for facilitating specialization in the behaviors of ANN controlled agents and for solving collective behavior tasks [25,28]. All methods are evaluated in a *Gathering and Collective Construction* task (Section 3).

1.2. Collective construction

This article investigates a collective construction task (Section 3). Collective construction tasks require that agents coordinate their behaviors or cooperate in order to build structures in the environment. Most research that applies adaptive methods to solve collective construction tasks has been studied in the context of simulated agent groups [30–32].

The collective construction task studied in this article, requires that agents first gather resources and place them in a construction zone. Collective gathering tasks require that agents search for, and transport resources from given locations to another part of the environment [2]. Collective gathering tasks typically require that agents divide their labor amongst sub-tasks to derive a collective behavior that maximizes the quantity of resources gathered. Thus collective gathering (and by extension collective construction) tasks are interpretable as optimization problems and have been studied with mathematical models [33–35]. There are numerous examples of adaptive methods applied to simulated agent groups in order that collective gathering [36–38,35,39] or construction [30,40,41,31,32] tasks are solved.

For example, Theraulaz and Bonabeau [30] proposed a controller for an agent team given a collective construction task in a three-dimensional simulation environment. Agents moved randomly on a cubic lattice and placed building blocks whenever they encountered a *stimulating configuration* in the structure being built. Agent behaviors were guided by previous work, since each time an agent placed a building block, it modified the shape of the configuration that triggered its building action. The new configuration then stimulated new building actions by other agents, which resulted in the emergence of a collective construction behavior. Results indicated that these local stigmergic agent interactions succeeded in building multiple complete structures that resembled nests built by social insects such as wasps.

Guo et al. [41] used a controller based on a *Gene Regulatory Networks* (GRN) to derive behaviors for a simulated multi-robot team given a collective construction task. The collective construction task was for the robots to self assemble into various shapes and patterns. Local interactions of the robots were represented by biologically inspired reaction–diffusion model. Results indicated that the GRN inspired multi-robot controller was effectively able to balance two different (specialized) behaviors in each robot. First, to approach and join a predefined shape, and second, to avoid collisions with other robots.

Expanding upon previous research using reactive ANN controllers for agents that build structures in two dimensional simulation environments [42], Panangadan and Dyer [32] introduced a *connectionist action selection mechanism* (ConAg) agent controller. An agent team was given the task of collecting colored discs (building blocks) and transporting them to a particular location to build a structure with a given configuration. A Reinforcement Learning [43] method was used so as each agent learnt a sequence of behaviors necessary for it to perform the construction task, and a heuristic controller comparatively tested. Results indicated that the behavior and success of heuristic-based controller was dependent upon the shape of the structure being built, and sensitive to disc locations in the environment. This was not the case for the ConAg controller, which was sufficiently robust so as to continue building the desired structure even if discs were moved during construction.

As with these related research examples, this article studies a simulated agent group that must solve a collective construction task. In this article, *collective* refers to the increase in task performance that results from the division of labor and agents working concurrently.

1.3. Research objectives and hypotheses

Objective 1: To extend previous work [44], and investigate the efficacy of CONE as a controller design method for a more complex collective construction task (robots must build up to 10 structures, each structure containing up to 100 components). Nitschke [44] demonstrated that CONE was effective at evolving specialized robot controllers that complemented each other to form collective construction behaviors that built one object using 10–30 components.

Objective 2: Test CONE for evolving controllers in teams that contained a greater number of robots (teams of 50 or 100). Nitschke [44] described a collective construction task using teams of 30 robots.

Hypothesis 1: For the given collective construction task, CONE evolved teams yield a statistically significant higher task performance, for all environments and teams tested, compared to CCGA and MESP evolved teams.

Hypothesis 2: CONE *Genotype* and *Specialization Difference Metrics* (GDM and SDM, respectively), that adaptively regulate inter-population recombination, evolve behavioral specializations that result in statistically significant higher task performances, comparative to CCGA and MESP evolved teams. Without these specializations the higher task performance of CONE evolved teams could not be achieved.

1.4. Contributions

This research extends the collective construction task described in [44]. However, there are four key differences in this article's research.

1. *Larger team sizes*. Nitschke [44] tested only team sizes of 30 robots, where as this research tests team sizes of 50 and 100 robots. The team sizes used in this research are comparable to team sizes tested in swarm robotics experiments [45].
2. *Increased task complexity*. Nitschke [44] evaluated robot teams for the task of collectively building structures that consist of between 10 and 30 building blocks, where only a single structure had to be built. This article's experiments evaluate teams that concurrently build one to 10 structures. Each structure comprises 10–100 building blocks.
3. *Increased fidelity in multi-robot simulator*. Nitschke [44] simulated robot teams using a simple low-fidelity multi-robot simulator implemented using the MASON simulation toolkit [46]. In the MASON simulator, robot sensors and actuators, and the physics of robot movement only had relevance within the simulation. This research uses an extension of the high-fidelity *EvoRobot* Khepera robot simulator, which allows teams of up to 100 Kheperas to be simulated. EvoRobot was used so as robot behaviors evolved in simulation could potentially be transferred to physical Khepera robots.
4. *Self Regulating Difference Metrics*. Nitschke [44] used a version of CONE with static values for the *Genotype* and *Specialization Difference Metrics* (GDM and SDM, respectively). The GDM and SDM regulate inter-population recombination based on average weight differences and degrees of specialization exhibited by controllers (Section 2.3). In this article's research, the GDM and SDM are self-regulating meaning that inter-population recombination is adaptively regulated.

For the reader's convenience, a list of abbreviated terms and symbols used throughout this article are presented in Table 7, at the end of the article.

2. Methods: collective neuro-evolution (CONE)

CONE is an automated controller design method that uses cooperative co-evolution to adapt a team of ANNs (agent controllers). Given n genotype populations (species), n controllers are evolved (one in each population). Controllers are collectively evaluated (as a team) according to how well they solve a given task. Each controller is a recurrent feed-forward ANN with one hidden layer. The hidden layer is fully connected to the input and output layers, with recurrent connections to the input layer. Each hidden layer neuron is encoded as a genotype. CONE evolves the input–output connection weights of hidden layer neurons, and within each species combines the fittest of these neurons into complete controllers.

CONE extends the work of [28] on *Multi-Agent Enforced Sub-Populations* (MESP), with the inclusion of two novel contributions. First, CONE solves collective behavior tasks using emergent behavioral specialization in agents. Second, CONE uses genotype and specialization metrics to regulate inter-population genotype recombination and facilitate behavioral specialization appropriate for each agent. When these specialized agent behaviors interact, the team is able to increase task performance and solve collective behavior tasks that could not otherwise be solved.

Unlike related cooperative co-evolution methods, including CCGA [11], ESP [47], and MESP [28], CONE uses *Genotype* and *Specialization Difference Metrics* (GDM and SDM, respectively), to regulate genotype recombination *between* and *within* populations. Based upon genotype similarities and the success of evolving behavioral specializations, the GDM and SDM control recombination and direct the evolution of collective behaviors in an agent team.

For succinctness, this section describes only CONE's representation (Section 2.1), how behavioral specialization is measured (Section 2.2), the GDM and SDM (Section 2.3) and CONE's iterative evolutionary process (Section 2.4). Nitschke [27] presents a comprehensive description of CONE.

The design choices for CONE's representation and iterative process were motivated by two sets of previous research. First, the work upon which CONE is based [47], which successfully solved collective behavior tasks [22,28]. Second, research on genetically heterogeneous agent teams (agents have different genotypes) indicates that such teams are amenable to evolving specialized behaviors [48,49], especially in cooperative co-evolutionary systems [50].

The use of the GDM and SDM as mechanisms to regulate inter-population recombination was motivated by research on *partially heterogeneous* agent groups. A partially heterogeneous group is an agent group comprised of sub-groups that are, on average, more genetically similar (but not identical) to individuals of their given sub-group, comparative to individuals of the rest of the population [51]. In this article's research, such sub-groups are defined as *species*. The impact of partial genetic heterogeneity on the evolution of group behaviors, especially with respect to the evolution of multiple, complementary specialized behaviors has received little investigation in evolutionary multi-agent [52], or swarm robotics [45] research. However, Luke et al. [48] and Luke [53] suggest that partial genetic heterogeneity in an evolving agent group can lead to specialized behaviors. This is supported by studies in biology Hamilton [54] and Lehmann and Keller [55]. Furthermore, Perez-Uribe et al. [56,51] indicated that increases in team fitness were related to selection within genetically related agents. As an extension, the GDM and SDM were derived with the supposition that recombining genetically *and* behaviorally related agents, would increase the team's task performance, or allow the team to solve tasks that could not otherwise be solved.

2.1. Representation: multi-population structure

As with related NE methods [29,47], CONE segregates the genotype space into n populations so as to evolve n controllers. CONE mandates that ANN_i ($1 \leq i \leq n$) is derived from population P_i, where P_i contains u_i (initially) or w_i ($u > 0$, due to controller size adaptation) sub-populations. Fig. 1 exemplifies the use of sub-populations in CONE. ANN_1 and ANN_2 (evolved from populations 1 and 2, respectively) has three hidden layer neurons, whilst ANN_3 (evolved from population 3) has four hidden layer neurons. Thus, populations 1 and 2 consist of three sub-populations, for evolving the three neurons in ANN_1 and ANN_2. Where as, population 3 uses four sub-populations for evolving the four neurons in ANN_3. ANN_i is derived from P_i via selecting one genotype from each sub-population and decoding these genotypes into hidden layer neurons (Fig. 2). ANN_i consists of w input neurons, and v output neurons, fully connected to all hidden layer neurons. In this research, CONE uses a fixed number of input, output and hidden layer neurons.

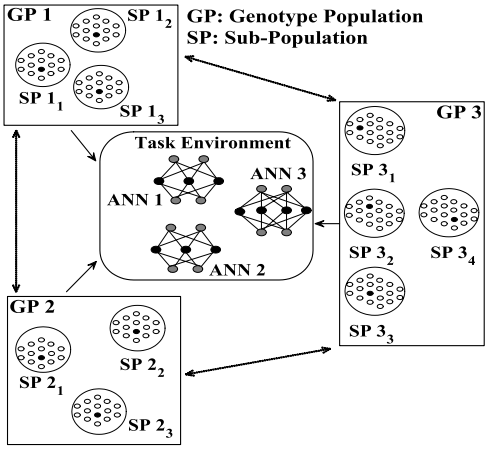

Fig. 1. *CONE example*. A controller is evolved in each population. All controllers are evaluated in a collective behavior task. Double ended arrows indicate self regulating recombination occurring between populations. *ANN*: Artificial Neural Network (controller). *GP X*: Genotype Population *X*. *SP* X_z: Sub-Population z in Genotype Population *X*.

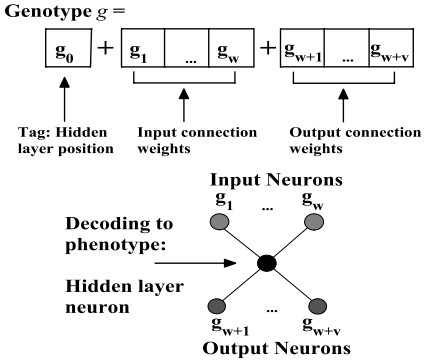

Fig. 2. *CONE genotype*. There is a direct mapping between a genotype and a hidden layer neuron. A genotype has w genes indicating the neuron's input connection weights, and v genes for the neuron's output weights. A tag (g_0) specifies the neuron's position in a controller's hidden layer, and hence the sub-population to which the genotype belongs.

The CONE process is driven by mechanisms of cooperation and competition within and between sub-populations and populations. Competition exists between genotypes in a sub-population that compete for a place as a hidden layer neuron in the fittest controller. Cooperation exists between sub-populations, in that fittest genotypes selected from each sub-population must cooperate as a controller. Cooperation also exists between controllers since controllers must cooperate to accomplish a collective behavior task.

2.2. Behavioral specialization

An integral part of CONE is defining and measuring controller behavioral specialization. The *degree of behavioral specialization* (S) exhibited by a controller is defined by the frequency with which the controller switches between executing distinct motor outputs (actions) during its lifetime. The S metric used is an extension of that defined by Gautrais et al. [35], and was selected since it is applicable to individual controller behaviors, accounts for a partitioning of a controller's work effort among different actions, and is simple enough to work within CONE. The metric is also general enough to define specialization as the case where controllers regularly switch between different actions, spending an approximately equal portion of its lifetime on each action, but where there is a slight preference for one action.

Eq. (1) specifies the calculation of S, which is the frequency with which a controller switches between each of its actions during its lifetime. Eq. (1) assumes at least two distinct agent actions and that an agent executes an action during the same simulation iteration of an action switches. In Eq. (1), A is the number of times the controller switches between different actions, and N is the total number of possible action switches.

$$S = \frac{A}{N}. \tag{1}$$

An S value close to zero indicates a high degree of specialization. In this case, a controller specializes to primarily one action, and switches between this and its other actions with a low frequency. An S value close to one indicates a low degree of specialization. In this case, a controller switches between some or all of its actions with a high frequency. A perfect specialist ($S = 0$), is a controller that executes the same action for the duration of its lifetime ($A = 0$). An example of a non-specialist ($S = 0.5$) is where a controller spends half of its lifetime switching between two actions. For example, if $A = 3, N = 6$, then the controller switches between each of its actions every second iteration.

Controllers are labeled as *specialized* if S is less than a given *behavioral specialization threshold* (for this study a 0.5 threshold was selected). Otherwise, controllers are labeled as *non-specialized*. If a controller is specialized, then it is given a specialization label *action x*, where x is the action executed for more than 50% of the iterations of the controller's lifetime. If multiple controllers are specialized, then controllers are grouped according to their common specialization label. In the case that an agent performs a low number of action switches ($S < 0.5$), such that it executes at least two actions for approximately equal durations, then the agent is assumed to have multiple specializations, since continuous and equal periods of time are spent on each action. If an agent performs a low number of action switches ($S < 0.5$), and is able to execute at least two actions simultaneously, then the agent is assumed to have one specialization defined by the interaction of these actions and named by the experimenter. This is the case in this study, since robot controllers can execute two actions simultaneously (Section 3.3).

2.3. Regulating recombination and adaptation of algorithmic parameters

The purpose of the genotype and specialization difference metrics (GDM and SDM, respectively) is to adaptively regulate genotype recombination between different populations as a function of the fitness progress of all controllers. As part of the regulation process, two dynamic algorithmic parameters, the *Genetic Similarity Threshold* (GST), and *Specialization Similarity Threshold* (SST), are used by the GDM and SDM, respectively.

The initial GST and SST values are floating point values randomly initialized in the range: [0.0, 1.0]. Whenever the GST value is adapted by the GDM, a static value (δGST) is either added or subtracted to the GST value. Similarly, when the SST value is adapted by the SDM, a static value (δSST) is either added or subtracted to the SST value. The remainder of this section describes the mechanisms used to adapt the SST and GST values.

2.3.1. Genotype difference metric (GDM)

The GDM is a heuristic that adaptively regulates recombination of similar genotypes in different populations. Any two genotypes $\bar{a}$ and $\bar{b}$ are considered similar if the average weight difference [57] between $\bar{a}$ and $\bar{b}$ $<$ GST. Regulating genotype recombination between populations is integral to the CONE process. That is, controllers must not be too similar or dissimilar so as the (specialized) behaviors of controllers properly complement each

other in accomplishing a collective behavior task. The GST value, and hence inter-population genotype recombination, is adapted as a function of the number of previous recombinations and a team's average fitness progress (of the fittest n controllers). The following rules were used to regulate the GST value and thus the number of inter-population recombinations, with respect to average team fitness and the number of inter-population recombinations that occurred over the previous V generations.

1. If recombinations between populations have increased over the previous V generations, and fitness has stagnated or decreased, then decrement the GST value, so as to restrict the number of recombinations.
2. If recombinations between populations have decreased or stagnated, and fitness has stagnated or decreased over the last V generations, then increment the GST value, to increase the number of recombinations.

Similar genotypes in different populations may encode very different functionalities, recombining similar genotypes may produce neurons that do not work well in the context of a controller. The *Specialization Difference Metric* (SDM) addresses this problem.

2.3.2. Specialization difference metric (SDM)

The SDM adaptively regulates genotype recombination based on behavioral specialization similarities exhibited by controllers. The SDM ensures that only the genotypes that constitute controllers with sufficiently similar behaviors are recombined. If the behavior of two controllers are calculated to have sufficiently similar specializations, the GDM is applied to regulate inter-population recombination. The SDM measures the similarity between the specialized behaviors of controllers ANN_i and ANN_j. Controllers are considered to have similar specializations if the following conditions are true:

1. $|S(ANN_i) - S(ANN_j)| < \text{SST}$, where, S (Eq. (1) in Section 2.2) is the *degree of behavioral specialization* exhibited by ANN_i and ANN_j.
2. If ANN_i and ANN_j have the same *specialization label*.

The SST value is adapted as a function of behavioral specialization similarities and a team's average fitness (of the fittest n controllers) progress. The following rules are used to regulate the SST value, and hence the number of inter-population recombinations with respect to the average degree of behavioral specialization (S) and fitness of a team.

1. If the S of at least one of the fittest controllers has increased over the last W generations, and average fitness stagnates or is decreasing over this same period, then decrement the SST value. Thus, if the fittest controllers have an average S that is too high for improving team fitness, then recombination between populations is restricted.
2. If the S of at least one of the fittest controllers has decreased over the last W generations, and average fitness stagnates or is decreasing over this same period, then increment the SST value. Thus, if the fittest controllers have an average S that is too low to improve team fitness, then allow for more recombination between populations.

2.4. Collective neuro-evolution (CONE) process overview

This section overviews CONE's iterative evolutionary process. Nitschke [27] presents a comprehensive description of each step of CONE's process.

1. *Initialization*. n populations are initialized. Population P_i ($i \in \{1, \ldots, n\}$) contains u_i sub-populations. Sub-population P_{ij} contains m genotypes. P_{ij} contains genotypes encoding neurons assigned to position j in the hidden layer of ANN_i (ANN_i is derived from P_i).
2. *Evaluate all genotypes*. Systematically select each genotype g in each sub-population of each population, and evaluate g in the context of a complete controller. This controller (containing g) is evaluated with $n-1$ other controllers (where, n is the number of controllers in a team). Other controllers are constructed via randomly selecting a neuron from each sub-population of each of the other populations. Evaluation results in a fitness being assigned to g.
3. *Evaluate elite controllers*. For each population, systematically construct a fittest controller via selecting from the fittest genotypes (elite portion) in each sub-population. Controller fitness is determined by its *utility*. Utility is the average fitness of the genotypes corresponding to a controller's hidden layer. Groups of the fittest n controllers are evaluated together in task simulations until all genotypes in the elite portion of each population have been assigned a fitness. For each genotype, this fitness overwrites previously calculated fitness.
4. *Parent selection*. If the two fittest controllers ANN_i and ANN_j constructed from the elite portions of P_i and P_j are calculated as having sufficiently similar *behavioral specializations* (Section 2.2) then P_i and P_j become candidates for recombination. For P_i and P_j to be recombined, both ANN_i and ANN_j must have the same specialization label (Section 2.2). That is, both ANN_i and ANN_j must be behaviorally specialized to the same action. Between P_i and P_j each pair of sub-populations is tested for *genetic similarity* (average weight difference is less than GST). Genetically similar sub-populations are recombined. For sub-populations that are not genetically similar to others, recombination occurs *within* the sub-population. Similarly, for populations that are not behaviorally similar to other populations, recombination occurs *within* all sub-populations of the population.
5. *Recombination*. When pairs of sub-populations are recombined, the elite portion of genotypes in each sub-population is ranked by fitness. Genotypes with the same fitness rank are recombined. For recombination *within* a sub-population, each genotype in the sub-population's elite portion is systematically selected and recombined using one-point crossover [58], with another randomly selected genotype from the sub-population's elite portion.
6. *Mutation*. *Burst mutation* with a *Cauchy* distribution [47] is applied to each gene of each genotype with a given probability.
7. *Parameter adaptation*. If the fitness of one of the fittest controllers has not progressed in:
   (a) V generations: Adapt *Genetic Similarity Threshold* (GST).
   (b) W generations: Adapt *Specialization Similarity Threshold* (SST).
8. *Stop condition*. Reiterate steps [2, 7] until a desired collective behavior task performance is achieved, or the process has run for X generations.

3. Task: Gathering and Collective Construction (GACC)

The *Gathering And Collective Construction* (GACC) task requires that robots place building blocks in a *construction zone* in a specific sequence to build a predefined structure. This GACC task extends previous work, that demonstrated that behavioral specialization is beneficial for accomplishing a collective construction task [44]. The GACC task presented in this article extends Nitschke [44] via increasing task complexity, using more robots and building blocks, and imposing a constraint that teams must concurrently construct multiple objects. The motivation for increasing task complexity, and testing larger team sizes was to thoroughly test the efficacy of CONE as a controller design method. Table 1 presents the GACC task parameters. The calibration of these parameters is discussed in Section 4.2.

Table 1
Simulation and neuro-evolution parameters. For the GACC task.

Simulation and Neuro-Evolution Parameter settings	
Robot movement range	0.01 (of environment width/length)
Light/proximity detection sensor range	0.05 (of environment width/length)
Initial robot positions	Random (Excluding construction zone)
Construction zone location	Environment's Center
Environment width/height	10 m/10 m
Block distribution (initial locations)	Random (Excluding construction zone)
Simulation runs (evolution/testing phases)	20
Iterations per epoch (robot lifetime)	1000 Iterations
Generations	100
Epochs	10
Mutation (per gene) probability	0.05
Mutation type	Burst (Cauchy distribution)
Mutation range	$[-1.0, +1.0]$
Fitness stagnation V/W	5/10 Generations (CONE)
Population elite portion	20%
Weight (gene) range	$[-1.0, +1.0]$
Crossover	Single point
ANN sensory input neurons	20
ANN hidden layer neurons (CONE evolved)	4
ANN motor output neurons	4
Genotype	Vector of floating point values
Genotype length	24 (CONE/MESP)/96 (CCGA)
Team size	50/100
Genotype populations	50/100
Genotypes per population	200/100
Total genotypes	10 000

A GACC task was selected since it is a simulation with potential collective behavior applications including multi-robot gathering and collective construction in hazardous or uninhabitable environments, such as underwater habitats or space stations [59]. This GACC task is *collective* in that robots must coordinate their behaviors so as to concurrently gather building blocks and then deliver the building blocks to a construction zone in a correct sequence. This GACC task consists of three sub-tasks.

1. *Search for building blocks*[1]: The environment contains type *A* and *B* blocks. A light on each block makes it detectable by robot light sensors.
2. *Transport blocks*: Robots must *grip* and *transport* blocks to a *construction zone*, a predefined space in the environment. All robots have *a priori* knowledge of the location of the construction zone, and thus do not need to discover the construction zone.
3. *Build structure*: In the construction zone, robots must *place* the blocks in a specific sequence of block types required for structure assembly. This sequence is specified by a *construction schema*. The construction schemas tested in this GACC task are presented in Section 3.2.

Team task performance is the number of blocks placed in the construction zone (in a correct sequence) during a team's lifetime. Fig. 3 presents an example of the GACC task being accomplished by a team of 10 robots. In Fig. 3, the construction schema used is labeled: *Assembled Block Structure*.

Assembled Block Structure = *A*(East), *B*(North, South, East), *A*(End), *A*(End), *B*(East), *A*(North, South, East), *A*(End), *A*(End), *A*(East), *B*(End);

Where, for a given block type (*A* and *B*), *North*, *South*, *East*, *West* denotes the block side to which the next block type in a sequence is to be connected. *End* denotes a final block to be connected to another block's side.

[1] The terms *building block* and *block* are used interchangeably.

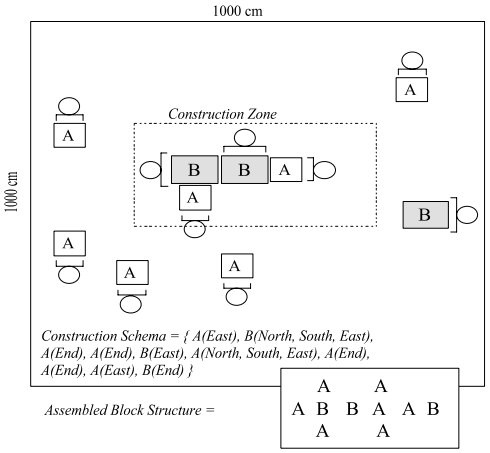

Fig. 3. *Gathering and Collective Construction (GACC) Example*. Seven type *A*, and three type *B* blocks are randomly distributed throughout the environment. In the construction zone, five blocks are connected as a partially assembled structure. The construction schema and target structure are given at the bottom. The team uses 10 robots.

3.1. Simulation environment

The environment is a 1000 cm × 1000 cm continuous area, simulated using an extended version of the *EvoRobot* simulator [60], and contains:

- Q building blocks ($Q \in \{2, \ldots, 100\}$, type $\{A, B\}$). Type *A* and *B* blocks have a low and high intensity light on their tops, respectively.
- N robots ($N \in \{2, \ldots, 100\}$). Table 4 presents light detection sensor and gripper settings for block detection and transport, respectively.
- A construction zone (100 cm × 100 cm) at the environment's center.

Initially, blocks (1.0 cm × 1.0 cm) and robots (5.5 cm in diameter) are randomly distributed in the environment, except in the construction zone.

3.2. Assembling structures

A structure is defined as a combination of m blocks (where, $m \in \{2, \ldots, Q\}$). A construction schema specifies *how* blocks must be *assembled* in order for a structure to be built. That is, a construction schema defines which block sides (for a given block type) must connect to the next block in a sequence.

Construction schema $= \{B_i(c), \ldots, B_j(c)\}$;
Where: $i, j \in \{1, \ldots, Q\}$, $c \in$ {North, South, East, West, End}, $B =$ Block.

To keep construction simple, it is assumed that any block can be initially placed in the construction zone. However, the next block must be connected to one side of the initially placed block. Consider the example in Fig. 3. If a type *B* block is the first to be placed in the construction zone, then (for the given construction schema) the next block placed must be a type *A* block connected to the north, west, or south face, or a type *B* block connected to the east face. Alternatively, another type *B* block can be connected to the west face, or a type A block to the east face. The task is complete when all blocks have been transported to the construction zone and connected according to the sequence defined by the construction schema.

Table 2 presents the construction schemas used for the simulation environments tested. For each environment, one construction schema is used. Table 3 presents, for each environment, the number of type *A* and *B* blocks, and the number of structures that must be assembled from these blocks.

Table 2
Construction schemas. For given environments, the block type sequence (*A*, *B*) to build a structure. *E*: East, *W*: West, *N*: North, *S*: South. *End*: No more connections.

Environment	Construction schema
1, 6	**A**(*E*)**B**(*E*)**B**(*S*)**B**(*E*)**B**(*E*)**B**(*E*)**B**(*S*)**B**(*E*)**B**(*S*)**A**(*End*)
2, 7	**A**(*E*)**B**(*E*)**B**(*S*)**A**(*E*)**B**(*E*)**B**(*E*)**A**(*S*)**B**(*E*)**B**(*S*)**A**(*End*)
3, 8	**B**(*S*)**A**(*S*)**B**(*E*)**A**(*E*)**B**(*N*)**A**(*N*)**B**(*E*)**A**(*E*)**B**(*S*)**A**(*End*)
4, 9	**B**(*E*)**A**(*E*)**A**(*S*)**B**(*E*)**A**(*E*)**A**(*E*)**B**(*S*)**A**(*E*)**A**(*S*)**B**(*End*)
5, 10	**B**(*E*)**A**(*E*)**A**(*S*)**B**(*E*)**A**(*E*)**A**(*E*)**A**(*S*)**A**(*E*)**B**(*S*)**B**(*End*)

Table 3
Simulation environments: Block type distribution and structures to be built.

Environment	Type *A* blocks	Type *B* blocks	Structures
1	2	8	1
2	14	6	2
3	20	10	3
4	22	18	4
5	30	20	5
6	45	15	6
7	52	18	7
8	68	12	8
9	80	10	9
10	94	6	10

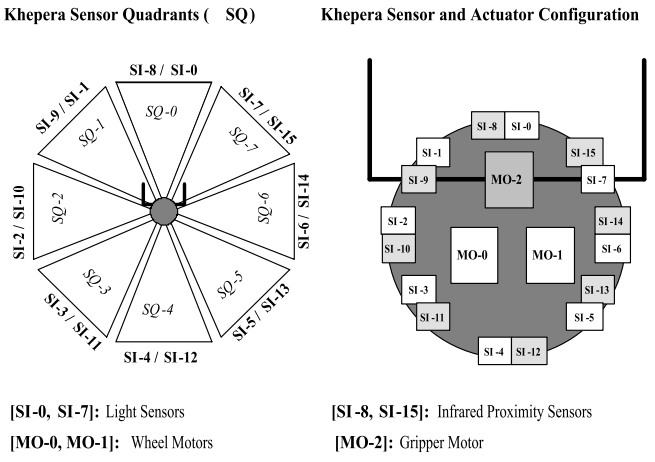

Fig. 4. *Robot sensors and actuators*. Each simulated *Khepera* has eight light ([S-0, S-7]) and eight infra-red proximity ([S-8, S-15]) sensors on its periphery. Each robot also has three actuators: two wheel motors ([MO-0, MO-1]), and one gripper motor (MO-3).

3.3. Robots

Robots are simulated *Kheperas* [61]. Each robot is equipped with eight light sensors (for block detection) and *Infra-Red* (IR) proximity (for obstacle avoidance) detection sensors, providing each a 360 degree *Field of View* (FOV). Also, each robot has two wheel motors for controlling speed and direction, and a gripper for transporting blocks. Fig. 4 depicts the sensor and actuator configuration of each robot. Detection sensor values are normalized in the range [0.0, 1.0], via dividing a sensor reading by the maximum sensor value. Table 1 presents the robots' light and proximity detection sensor ranges, and maximum movement range.

3.3.1. Light detection sensors

Eight light detection sensors enable each robot to detect blocks in eight sensor quadrants ([S-0, S-7] in Fig. 4). Type *A* blocks have a low intensity light on top. Type *B* blocks have a high intensity light on top. Table 4 presents the detection sensor settings required to detect type *A* and *B* blocks. When light sensors are activated, all eight sensors are simultaneously activated with a given setting. Sensors remain active until the setting is changed with the next activation. Detection sensor q returns a value inversely proportional the distance to closest block in sensor quadrant q, multiplied by the intensity of the light on top of the closest block.

3.3.2. Infrared (IR) proximity detection sensors

Eight IR proximity detection sensors ([S-8, S-15] in Fig. 4) covering eight sensor quadrants, enable robots to detect and avoid obstacles (other robots and the environment's walls). IR detection sensor q returns a value inversely proportional to the distance to the closest obstacle in sensor quadrant q. The IR proximity detection sensors are constantly active, and are initialized with random values. The IR sensor values are updated every simulation iteration that the robot is within sensor range of an obstacle.

3.3.3. Movement actuators

Each robot is equipped with two movement actuators (wheel motors) that control its speed and heading in the environment. Wheel motors need to be explicitly activated. In a simulation iteration of activation, the robot will move a given distance (calculated according its current speed), and then stop. A robot's heading is calculated by normalizing and scaling the two motor output values (MO-0 and MO-1) in order to derive vectors dx and dy.

$$dx = d_{\max}(\text{MO-0} - 0.5).$$
$$dy = d_{\max}(\text{MO-1} - 0.5).$$

where, $d_{\max}$ is the maximum distance a robot can traverse per iteration. A minimum distance δ^2 is to prevent singularities [62] in the simulator when a robot is very close to a block or obstacle. To calculate the distance between robot r, and blocks or obstacles o, the squared Euclidean norm, bounded by δ^2 is used. Eq. (2) presents the distance metric.

$$\delta(r, o) = \min(\|x - y\|^2, \delta^2). \tag{2}$$

3.3.4. Block gripper

Each robot is equipped with a gripper turret (Fig. 4) for gripping blocks, transporting them, and placing them in the construction zone. The gripper is activated with the value of motor output MO-2. In order to grip block type *A* or *B*, specific output values must be generated (Table 4). These values correspond to the gripper setting necessary to grip type *A* and *B* blocks.

3.3.5. Artificial neural network (ANN) controller

Each robot used a recurrent ANN controller (Fig. 5), which fully connecting 20 sensory input to four hidden layer (sigmoidal) and four motor output neurons. Prior to controller evolution (Section 4.4), n controllers were placed in a shaping phase (Section 4.2). The shaping phase incrementally evolved block gripping, transportation and obstacle avoidance behaviors necessary to accomplish the CGAC task. Also, prior to the evolution phase, the number of hidden layer neurons was derived during a parameter calibration phase (Section 4.3). Sensory input neurons [SI-0, SI-7] accepted input from each of eight IR detection sensors, neurons [SI-8, SI-15] accepted input from each of eight light detection sensors, neurons [SI-16, SI-19] accepted input from the previous activation state of the hidden layer.

Action selection: At each iteration, one of four *Motor Outputs* (MO) are executed. The MO with the highest value is the action executed. If either MO-0 or MO-1 yields the highest value, then the robot moves.

1. *MO*-0, *MO*-1: Calculate dx, dy vectors from MO-0, MO-1 output values, and move in direction derived from dx, dy (Section 3.3.3).
2. *MO*-2: Activate gripper (Section 3.3.4).
3. *MO*-3: Activate light detection sensors (Section 3.3.1).

Table 4
Block detection and transportation. Block detection and transportation require robots to use different light detection sensors and gripper settings, respectively.

Block type	Required light detection sensor setting	Required gripper setting
A (Low intensity light)	0.1	0.5: Gripper at half width
B (High intensity light)	1.0	1.0: Gripper at maximum width

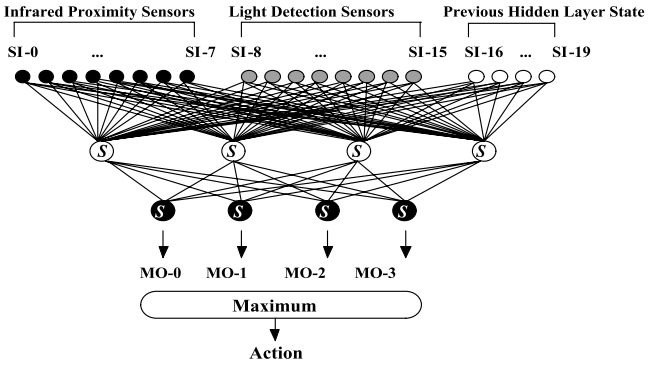

Fig. 5. *Robot artificial neural network (ANN) controller*. A feed-forward ANN with recurrent connections is used to map sensory inputs to motor outputs.

3.4. Behavioral specialization

Each robot performs distinct actions for detecting or gripping blocks, or moving. As such, each robot is able to specialize to *detecting*, or *gripping* or *moving*, or to behaviors that are a composite of the detecting, gripping and moving actions. For example, robots that execute the detect, grip and move actions for type *A* blocks, such that type *A* blocks are placed in the construction zone, are called *Type A Constructors*.

Initially, each robot adopts a search behavior, via moving and using light detection sensors. When a block is found, it uses a gripping action to grip the block. Finally, a robot moves with the gripped block toward the construction zone. The block is then placed in the construction zone, and this process repeats. However, since the GACC task requires that blocks be placed in the construction zone in a specific sequence, robots must concurrently coordinate their search, gripping and block placement behaviors. For example, consider an environment where type *A* blocks are particularly scarce, and type *B* blocks are plentiful, and the number of robots equals the number of blocks. In this case, an appropriate collective behavior would be for most robots to search for, grip, and move type *A* blocks to the construction zone, and concurrently, for relatively few robots to search for, grip and move type *B* blocks to the construction zone. Such a collective behavior would minimize the time for the team to build the structure.

The degree of behavioral specialization (S) exhibited by each robot (controller) is calculated by the specialization metric (Section 2.2) and applied at the end of each robot's lifetime in the test phase (Section 4.5). If $S < 0.5$, the robot's behavior is *specialized*, otherwise it is *non-specialized*. The 0.5 threshold was selected since if $S = 0.5$, then a robot spends half of its lifetime switching between its move, detect and grip actions, and spends an equal portion of its lifetime on each action. Specialized robots are labeled according to a robot's most executed action, or an aggregate of actions. These specialization labels are *Constructor*, *Mover*, *Gripper*, or *Detector*.

Constructor: Robots that spend more time moving with a gripped block, than executing other actions. *Type A*, *B Constructors* are those specialized to gripping and moving with *Type A*, *B* blocks, respectively.

Mover: Robots that spend more time moving than executing other actions.

Detector: If the most executed action is detecting *type A* or *type B* blocks, a robot is labeled as a *type A* or *type B detector*, respectively.

Gripper: If the most executed action is gripping *type A* or *type B* blocks, a robot is labeled as a *type A* or *type B gripper*, respectively.

4. Experiments

This section describes the *Gathering And Collective Construction* (GACC) experimental setup. Each experiment placed a team (50 or 100 robots) in each simulation environment (Table 3), and applied a controller design method (CCGA, MESP or CONE) to evolve the team's GACC behavior. Experiments measured the impact of a given *team size*, *environment*, and *controller design method* upon the team's *task performance*. The *experimental objective* was to ascertain which controller design method achieves the highest task performance for all environments tested, and to investigate the contribution of behavioral specialization to task performance.

4.1. Experiment phases

Each experiment consisted of the following phases.

- *Shaping phase*: CONE was used to evolve a team in a set of increasingly complex tasks (Section 4.2).
- *Parameter calibration phase*: CONE was used to evolve a team in a set of increasingly complex tasks (Section 4.3).
- *Evolution phase*: Next, the fittest team in the shaping phase was taken as the starting point for CONE, CCGA and MESP evolution (100 generations). One generation is a teams' *lifetime*. Each lifetime lasts for 10 epochs. Each epoch consists of 1000 simulation iterations. An epoch is a simulation scenario that tests different robot starting positions, orientations and block positions in an environment. For each method, 20 simulation runs[2] were performed for each environment (Section 4.4).
- *Test phase*: The fittest team evolved by CCGA, MESP, and CONE was selected and executed (in each environment) for 100 lifetimes. The testing phase did not apply any controller evolution. For the fittest team evolved by CCGA, MESP and CONE, task performance was calculated over 100 lifetimes and 20 simulation runs (Section 4.5).

4.2. Shaping phase

Shaping experiments applied CONE to incrementally evolve collective behaviors in the following set of increasingly complex tasks (*Exp x*). CONE was used in the shaping phase experiments since, compared to CCGA and MESP, it more quickly evolved behavioral solutions to the shaping tasks.

Exp 1: In an environment with two robots, using only IR proximity sensors, an obstacle avoidance (robots and walls) behavior was evolved. Two robots were the minimum for evolving obstacle avoidance.

Exp 2: In an environment with one robot and type *A* block, using IR and light sensors, a type *A* block detection behavior was evolved.

[2] Experiments were run on the *lisa* cluster (www.ka.sara.nl/home/willem/www.sara.nl/). Experiments used 250 nodes (each node has two Intel® Xeon™ 3.4 GHz processors).

Table 5
Parameter calibration. Values tested for the GACC Task.

Parameter	Value range	Range interval
Robot movement range	[0.01, 0.51]	0.05
Light/proximity detection sensor range	[0.01, 0.10]	0.01
Simulation runs	[10, 30]	2
Iterations per epoch (robot lifetime)	[500, 1500]	100
Generations	[50, 150]	10
Epochs	[2, 20]	2
Mutation (per gene) probability	[0.0, 0.11]	0.01
Fitness stagnation V (CONE)	[5, 15]	1
Fitness stagnation W (CONE)	[10, 25]	1
Population elite portion	[5, 55]	5%
Hidden Layer (HL) neurons	[1, 10]	1

Exp 3: In an environment with one robot, using IR and light sensors, and type *B* block a type *B* block detection behavior was evolved.
Exp 4: In an environment with one robot, using IR and light sensors, and a type *A* block, a type *A* block gripping behavior was evolved.
Exp 5: In an environment with one robot, using IR and light sensors, and a type *B* block, a type *B* block gripping behavior was evolved.
Exp 6: In an environment with one robot, using IR and light sensors, and a type *A* block, a block detection and gripping behavior was evolved.
Exp 7: In an environment with one robot, using IR and light sensors, and a type *B* block, a block detection and gripping behavior was evolved.

The fittest controller evolved in shaping experiment 7 was then subjected to parameter calibration experiments (Section 4.3).

4.3. Parameter calibration phase

Parameter calibration experiments were executed for the parameters given in Table 5 for CONE, CCGA, and MESP (team sizes of 50 and 100) in each simulation environment. Table 5 presents the calibrated parameter values.

Each of the parameters (Table 5) was systematically selected and varied within 100% of its value range at 20% intervals. Thus, 10 different parameter values were tested for each parameter. When a given value was selected, other parameter values were fixed at a median value in the range tested. The impact of given parameter values (in CCGA, MESP, and CONE), was ascertained via running each method for 50 generations. An average task performance was calculated (for a given team size and environment) over 10 simulation runs. A low number of generations and runs was used to minimize the time and computational expense of parameter calibration experiments.

Each of the parameters (Table 5) were calibrated independently. Thus, parameter inter-dependences were not taken into account, since the complexities of parameter interactions could not be adequately explored using this parameter calibration scheme. However, investigating the parameter interactions during calibration remains a current research topic [63]. The impact of the *behavioral specialization threshold*, the number of *hidden layer neurons* and *simulation runs* are briefly outlined in the following, since varying these parameters was found to have most affect on CCGA, MESP and CONE evolved team task performance.

Behavioral specialization threshold. Calibration experiments found that decreasing the specialization threshold to below 0.4 resulted in less controllers being classified as specialized and thus less specialized controller recombinations. This reduced the recombination of specialized controllers and beneficial behaviors between populations. Increasing the specialization threshold above 0.6 resulted in more controllers being classified as specialized and thus more controllers being recombined between populations. This resulted in the propagation of specialized behaviors that were not necessarily beneficial. The overall impact of a specialization threshold value outside the range [0.4, 0.6] was a decreasing task performance for all teams tested.

Hidden layer neurons. Calibration experiments determined that for CCGA, MESP, and CONE teams (evolved in all environments), an appropriate number of hidden layer neurons was five, four, and four, respectively. In order to keep method comparisons fair, and evolution time to a minimum, each method used four hidden layer neurons during the evolution phase.

Simulation runs. Calibration experiments determined that 20 runs was sufficient to derive an appropriate estimate of average task performance for evolved teams. Less than 20 evolutionary runs was found to be insufficient, and more than 20 runs consumed too much time and computational expense.

Finally, the parameter values calibrated for CCGA, MESP and CONE were used as the parameter settings for the evolution phase (Section 4.4).

4.4. Evolution phase

The n populations used by CCGA, MESP and CONE were initialized with copies of the fittest *shaped* genotype, where each gene in each genotype was subject to *burst mutation* [64] with a 0.05 probability. Burst mutation uses a Cauchy distribution which concentrates most values in a local search space whilst occasionally permitting larger magnitude values. Thus, the CCGA, MESP, and CONE methods began their behavioral search in the neighborhood of the best shaped solution.

Evolving Collective Behavior with CCGA. For a team of n robots (where, $n \in [50, 100]$), n populations are initialized. Each population is initialized with 400 or 200 genotypes. For a team size of $n = \{50, 100\}$, run for 100 generations, the number of evaluations E, is:

CCGA_E = 400 (genotypes per population) $*n$ (populations) $*100$ (generations) $*10$ (epochs per generation);

$\text{CCGA}_E = \{20\,400\,000(n = 50), 40\,800\,000(n = 100)\}$.

In order that the number of CCGA evaluations equals that of MESP and CONE, the elite portion (fittest 20%) of genotypes in each population are re-evaluated. That is, for each population, elite portion genotypes are systematically selected and evaluated together with genotypes randomly selected from the elite portions of the other populations. The number of evaluations required to evaluate the elite portion of controllers equals 400 000 ($n = 50$) or 800 000 ($n = 100$).

Evolving Collective Behavior with MESP/CONE. MESP and CONE create n ($n \in [50, 100]$) populations from which n robot controllers are evolved. Population i consists of u sub-populations, where u is the number of HL neurons. For teams of 50 and 100 robots (populations), each population is initialized with 400 or 200 genotypes. The process used to select and evaluate controllers is the same for MESP and CONE, and is described in Section 2.4. Specific to CONE is the *Specialization Distance Metric* (SDM) and *Genotype Distance Metric* (GDM). For a team size of $n = \{50, 100\}$, executed for 100 generations, the number of evaluations E, is:

MESP/CONE_E = 400 (genotypes per population) $*n$ (populations) $*100$ (generations) $*10$ (epochs per generation);

$\text{MESP/CONE}_E = \{20\,400\,000(n = 50), 40\,800\,000(n = 100)\}$.

This number of evaluations includes 400 000 evaluations ($n = 50$), or 800 000 evaluations ($n = 100$) required to evaluate controller utility (Section 2.4) of the fittest 20% of controllers.

4.5. Test phase

Finally, the fittest teams evolved by CCGA, MESP, and CONE, for each team size and environment was placed in the test phase. Each test phase experiment was non-adaptive and executed for

100 team *lifetimes* (each lifetime was 1000 simulation iterations), for a given team size and environment. Task performance results are averages calculated over these 100 lifetime runs. Since the test phase did not evolve controllers, the computational expense was marginal compared to a CCGA, MESP, or CONE evolutionary run. Section 5 presents the testing phase results, and statistical tests conducted.

5. Results

This section presents experimental results of applying CCGA, MESP or CONE evolved GACC behaviors, for a given team size (50 or 100 robots) and simulation environment (Table 3). Statistical tests were applied in order to compare task performance differences between teams evolved by each method. For this comparison, the following procedure was followed.

- The Kolmogorov–Smirnov test [65] was applied, and found that all data sets conformed to normal distributions.
- An independent t-test [65] was applied to ascertain if there was a statistically significant difference between the task performances of any two teams. The confidence interval was 0.95.

Bonferroni multiple significance test correction [66] was used to overcome the problem t-tests reporting a spurious significance of difference as a result of being applied for pairwise comparison between multiple data sets. T-tests were applied to test for significant difference between the following data set pairs, for a given team size and environment.

- Task performance results of CCGA versus MESP evolved teams.
- Task performance results of CCGA versus CONE evolved teams.
- Task performance results of MESP versus CONE evolved teams.

5.1. Gathering and collective construction task performance

Figs. 6 and 7 present the average task performances of teams evolved by CCGA, MESP, and CONE in each environment for team sizes 50 and 100, respectively. Task performance is the *number of blocks placed in the correct sequence* in the construction zone, over a team's lifetime. Average task performance was calculated for CCGA, MESP, and CONE evolved teams via executing each, in each environment, for 20 test phase runs (Section 4.5).

Statistical tests indicated that for both team sizes, CONE evolved teams yielded a higher average performance (with statistical significance), compared to CCGA and MESP evolved teams. This result held for environments [4, 10], and supports hypothesis 1 (Section 1.3). That is, CONE evolved teams on average, yield comparatively higher (statistically significant) performances.

Observing the task performance results of CONE evolved teams, it can be noted that as the complexity of the task increases (from the simplest in environment 1, to the most complex in environment 10), the performance of the CONE evolved teams also increases in a linear fashion. In the team of 100 robots there is a statistically significant difference in performance between each of environments [1, 10]. This is also the case for teams of 50 robots tested in environments [1, 7]. However, for environments [8, 10], teams of 50 robots yield no significant performance difference between environments.

This lower task performance for teams of 50 robots is theorized to be a result of the complexity of environments [8, 9, 10] coupled with an insufficient number of *specialized* robots to ensure a team performance comparable to that observed for teams of 100 robots. Consider that, most of the time, a robot would be unable to place the block it was holding, since the block would be *out of sequence*. In CONE evolved teams, one emergent behavior was that robots would drop blocks that they could not place. Such robots would

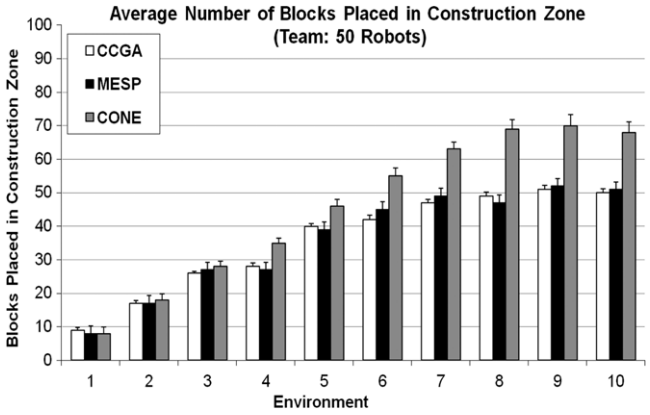

Fig. 6. *Average Gathering and Collective Construction task performance*. For CCGA, MESP, and CONE evolved teams (of 50 robots) for each environment.

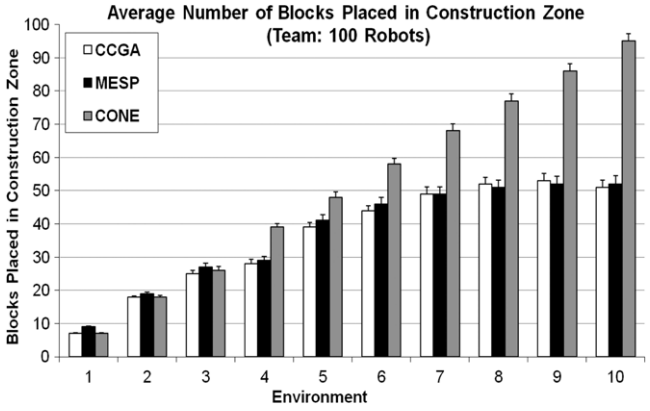

Fig. 7. *Average Gathering and Collective Construction task performance*. For CCGA, MESP, and CONE evolved teams (of 100 robots) for each environment.

then leave the construction zone to continue searching for other blocks. Another emergent behavior in CONE evolved teams was that of *idle* constructor, some of which would be in the construction zone at any given simulation iteration. Blocks dropped within sensor range of an idle constructor would be picked up and their placement attempted. This behavior of idle constructors becoming active and frequently picking up dropped blocks increased the number blocks that were placed in the *correct sequence*. In the case of teams of 50 robots, a relatively low number were in the construction zone at any given simulation iteration. This resulted in a comparatively lower task performance for teams of 50 robots evolved by CONE in environments [8, 9, 10].

Also, to demonstrate that specialized behavior is required to effectively and efficiently place objects in a given sequence, experiments that *did not use a construction schema* were conducted. Experiments that did not use a construction schema did not require a team to place blocks in any particular sequence. Thus, behavioral coordination was not required. These results are not presented here since: (1) statistically comparable performances were attained for the fittest CCGA, MESP and CONE evolved teams, (2) behavioral specialization did not emerge, and investigating the contribution of behavioral specialization to collective behavior is the focus of this study.

These results thus confirm that the task constraints imposed by a construction schema is necessary for specialization to emerge in a team's evolved collective behavior. Section 6 discusses results (of experiments using construction schemas), the contribution of behavioral specialization, and relates this contribution to hypothesis 2 (Section 1.3).

5.2. Emergent behavioral specializations

This section outlines the behavioral specialization that emerged in teams evolved by CCGA, MESP, and CONE in environments [4, 10]. No behavioral specialization emerged in teams evolved by CCGA, MESP, or CONE in environments [1, 3]. Lack of emergent specialization in these environments is supposed to be a result of the relative simplicity of environments [1, 3] (Table 3) compared to environments [4, 10]. Behavioral specialization was identified via applying the specialization metric (Section 2.2) to robot behaviors exhibited during the test phase (Section 4.5). Teams that were not calculated as specialized, were by default classified as *non-specialized*.

5.2.1. Evolved CONE specialization: constructor

In approximately 40% of the fittest teams evolved by CONE, a specialization termed *constructor* emerged. Constructors simultaneously performed the *grip* and *move* actions for more than 50% of their lifetime. Constructors infrequently switched from moving and gripping to the *detector* action.

5.2.2. Evolved CONE specialization: constructor/block dropping

In approximately 25% of the fittest teams evolved by CONE, a behavioral specialization termed *constructor/block dropping* emerged. Robots with this specialization performed the either the constructor (Section 5.2.1) or a block dropping behavior for more than 50% of their lifetime. These robots infrequently switched from either the constructor or block dropping behavior to performing the detector action, but frequently switched between executing constructor and block dropping behavior for most of their lifetime. The block dropping behavior was executed if a robot transporting a block was unable to place it in the construction zone, due to the block being *out of sequence*.

5.2.3. Evolved CCGA/MESP/CONE specialization: constructor/idle

In the fittest teams evolved by CCGA, MESP and CONE, a behavioral specialization termed *constructor/idle* emerged. This specialization emerged in approximately 35%, 55% and 50% of the fittest teams evolved by CCGA, MESP, and CONE, respectively. In the fittest CONE evolved teams, robots with this specialization performed the either the constructor behavior or remained idle for more than 50% of their lifetime. CONE evolved robots would switch from its idle to constructor behavior if a block was dropped within its sensor range. These robots infrequently switched to performing the detector action, but frequently switched between executing constructor and idle behavior for most of their lifetime. However, CCGA and MESP evolved robots simply remained idle for more than 50% of their lifetime, infrequently switching to the detector action during this time.

6. Discussion

This section discusses the contribution of specialized behavior to collective behavior task performance (Sections 6.1 and 6.2). The contribution of the *Genotype* and *Specialization Difference Metrics* (GDM and SDM) to the task performances of CONE evolved teams is also evaluated (Section 6.3).

6.1. Emergent specialization

In the fittest CONE evolved teams, constructors were specialized to gripping, moving with and placing (in the construction zone) type *A* and *B* blocks. Unspecialized robots searched the environment for blocks, and transported them to the construction zone. Upon arriving in the construction zone, unspecialized robots attempted to place the block they were transporting. Most of the time, a block could not be placed, since it was out of sequence. A robot would then drop the block and leave the construction zone to continue searching for other blocks. This block dropping behavior allowed constructors, idle in the construction zone, to place dropped blocks in the correct sequence. This in turn minimized the number of robots in the construction zone and physical interference between robots.

The idle behavior emerged in the fittest CCGA, MESP, and CONE evolved teams for most environments ([4, 10]). However, in the case of CONE evolved teams the idle behavior was coupled with a constructor behavior. Thus, CONE evolved robots switched between the constructor and idle behavior for most of their lifetime. It is theorized that the idle behavior emerged as a means to reduce physical interference between many constructors that concurrently moved toward the construction zone, to attempt to place blocks.

The constructor specialization did not emerge in any of the CCGA and MESP evolved teams, for all team sizes and environments tested. The behaviors of robots in the fittest CCGA and MESP evolved teams were calculated as being unspecialized. In the fittest CCGA and MESP evolved teams, robots that were unable to place blocks in the construction zone at a given simulation iteration would try to place the block at every subsequent iteration. If there were many robots in the construction zone, concurrently attempting to place blocks, where none of these blocks were the next in the construction sequence, the result was physical interference that obstructed collective construction behavior. The degree of interference increased with the team size, resulting in CCGA and MESP (comparative to CONE) evolved teams yielding a statistically lower task performance for environments [4, 10]. The block dropping behavior also emerged in CCGA and MESP evolved teams. However, when a block was dropped by a CCGA or MESP evolved robot, there were no constructor robots to place the block. Instead, the block remained in the construction zone until it was rediscovered by the same or another robot. This resulted in slow structure build times by CCGA and MESP evolved teams, which in turn yielded low task performances.

These results are supported by another collective behavior study [24], which also indicates that CONE is appropriate for evolving collective behavior solutions to tasks where specialization is beneficial, and the type of specialization (that is beneficial) is not known *a priori*.

6.2. Specialization lesion study

To test hypothesis 2 (Section 1.3), this section presents a *specialization lesion study* to evaluates the impact of the constructor specialization upon team task performance. The study was conducted on the supposition that the high task performance of CONE evolved teams, compared to CCGA and MESP evolved teams, results from the interaction between specialized and unspecialized behaviors. To test this supposition, the lesion study removed the constructor controllers and replaced them with unspecialized heuristic controllers. Heuristic controllers were used so as team behavior was unspecialized and teams remained the same size for comparison purposes. Robots were initialized in random positions in the environment and executed the following heuristic behavior. Robots had their light and proximity detection sensors constantly active and moved in a straight line toward the closest block. Otherwise, the robot moved in a straight line, in a random direction, and avoided collisions with the environment boundary. A robot gripped any block it encountered and moved it to the construction zone. If the robot could not place the block it would try again the next simulation iteration.

For each team size and environment, the fittest CONE evolved team (now consisting of unspecialized controllers) was re-executed in 20 test phase simulation runs (Section 4.5), and

Table 6
Average number of blocks placed (lesioned versus unlesioned teams): Columns [2, 5]: A/B is the average task performance of team sizes 50/100. *ENV*: Environment.

ENV	Fittest CONE evolved team	Lesioned CONE evolved team	Fittest CCGA evolved team	Fittest MESP evolved team
1	8/7	5/5	9/9	8/7
2	18/18	9/10	17/19	16/18
3	28/26	17/20	26/27	27/25
4	35/39	22/23	28/29	26/28
5	46/48	26/27	40/41	38/39
6	55/58	31/34	42/46	45/44
7	63/68	35/32	47/50	49/49
8	69/77	36/40	49/49	47/52
9	70/86	39/41	51/52	52/53
10	68/95	40/45	50/52	51/51

Table 7
Nomenclature: Abbreviated terms and symbols. Unless otherwise noted the terms and symbols apply to CCGA, MESP and CONE.

Term/Symbol	Explanation
CONE	Collective Neuro-Evolution (Section 2)
CCGA	Cooperative Co-evolutionary Genetic Algorithm (Section 2)
MESP	Multi-Agent Enforced Sub-Populations (Section 2)
NE	Neuro-Evolution (Section 1)
Species	Genotype population
Generation	1 Robot (Team) lifetime
Robot lifetime	10 Epochs
Epoch	1000 Simulation iterations
Specialization Threshold (CONE)	Defines if a controller's behavior is specialized
V (CONE)	GDM activated after V generations given no fitness increase
W (CONE)	SDM activated after V generations given no fitness increase
GDM (CONE)	Genotype Difference Metric (Section 2.3)
SDM (CONE)	Specialization Difference Metric (Section 2.3)
$[a, b]$	All values between and including a and b
n	Number of genotype populations (controllers) (Section 2.1)
ANN_i	Artificial neural network controller i (Section 2.1)
P_i	Population i (Section 2.1)
S (CONE)	Degree of behavioral specialization (Section 2.2)
SST (CONE)	Specialization Similarity Threshold (Section 2.3)
GST (CONE)	Genetic Similarity Threshold (Section 2.3)
δGST (CONE)	$\pm\delta$ applied to GST (Section 2.3)
δSST (CONE)	$\pm\delta$ applied to SST (Section 2.3)
$S(ANN_i)$	Degree of specialization exhibited by ANN i (Section 2.3)

an average task performance calculated. The contribution of the constructors was ascertained by comparing the average task performance, for each environment and team size, of lesioned versus unlesioned CONE evolved teams. Table 6 presents this task performance comparison. Lesion study results indicate that there is a statistically significant task performance reduction in lesioned teams. That is, lesioned teams were unable to produce collective behaviors with an average task performance comparable to that of CONE evolved teams. This result supports the supposition that CONE evolves an inter-dependency between specialized and unspecialized behaviors, and partially supports hypothesis 2 (Section 1.3). That is, without the constructor specialization, the higher task performance of CONE evolved teams could not be achieved.

6.3. The contribution of the CONE difference metrics

In order to further test hypothesis 2 (Section 1.3), this section presents a study to examine the contribution of the *Genotype* and *Specialization Difference Metrics* (GDM and SDM, respectively). For this GDM and SDM study, CONE was re-executed with the following variant experimental setups.

1. *CONE*-1: Teams were evolved by CONE without the GDM. The SDM for inter-population genotype recombination remained active.
2. *CONE*-2: Teams were evolved by CONE without the SDM. The GDM remained active.

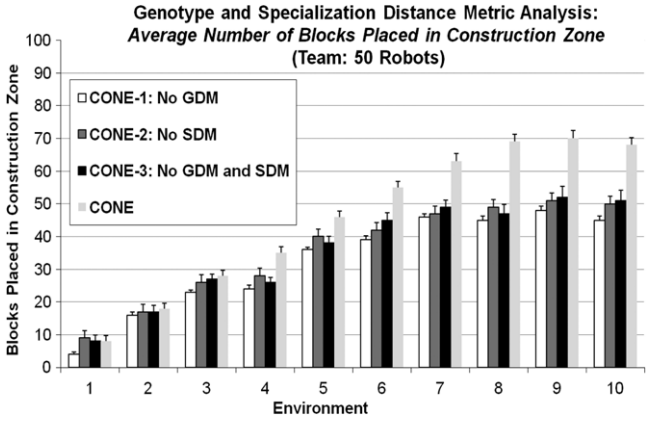

Fig. 8. *Average number of blocks placed in the construction zone* (*Team size*: 50 *robots*). Teams evolved by CONE without the *Genotype Difference Metric* (GDM), *Specialization Difference Metric* (SDM), or both the GDM and SDM.

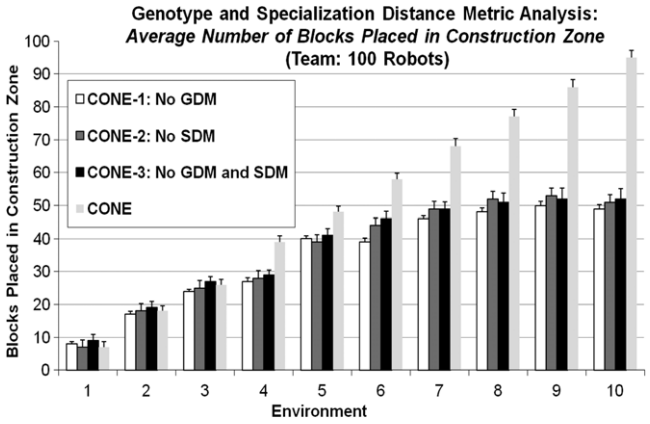

Fig. 9. *Average number of blocks placed in the construction zone* (*Team size*: 100 *robots*). Teams evolved by CONE without the *Genotype Difference Metric* (GDM), *Specialization Difference Metric* (SDM), or both the GDM and SDM.

3. *CONE*-3: Teams were evolved by CONE without the GDM and SDM.

Each of these CONE variants (CONE-1, CONE-2, and CONE-3) was applied to evolve teams in each environment, for team sizes of 50 and 100. The fittest team evolved by CONE-1, CONE-2, and CONE-3 was executed for 20 test-phase simulation runs (Section 4.5). Figs. 8 and 9 present average team task performances yielded by CONE-1, CONE-2, and CONE-3 for team sizes of 50 and 100, respectively. For comparison, the average task performance of the original CONE setup is also presented.

A statistical comparison of these results (Figs. 8 and 9) indicates that teams evolved by CONE without the GDM (CONE-1), SDM (CONE-2), and both the GDM and SDM (CONE-3), yielded a significantly lower task performance compared to CONE evolved

teams for most environments. That is, for team sizes 50 and 100, there was no significant task performance difference between CONE and CONE variant evolved teams for environments [1, 3]. However, CONE evolved teams yielded a significantly higher task performance for environments [4, 10]. Furthermore, teams evolved by the CONE variants yielded task performances comparable to CCGA and MESP evolved teams. That is, there was no statistically significant difference between the average task performances of teams evolved by the CONE variants, CCGA, and MESP for all environments and team sizes tested.

These results further support hypothesis 2 (Section 1.3), since they indicate that both the SDM and GDM were necessary for CONE to evolve teams with the most effective GACC behaviors. That is, when only the GDM or SDM or neither the SDM or GDM were active within the CONE process (CONE-1, CONE-2 or CONE-3), the fittest teams achieved average task performances comparable to that of the fittest CCGA and MESP teams.

7. Conclusions and future directions

This article evaluated controller design methods that coupled cooperative co-evolution and neuro-evolution to solve a collective behavior task. The research goal was to demonstrate that the *Collective Neuro-Evolution* (CONE) method evolves controllers in teams of simulated robots, such that the teams' collective behaviors out-perform that evolved by related methods. The collective behavior task was *Gathering and Collective Construction* (GACC).

Results found genotype and specialization metrics used by CONE for regulating recombination between genotype populations facilitated beneficial specialized behaviors. The interactions between specialized and unspecialized behaviors in CONE evolved teams resulted in a higher GACC task performance, compared to teams evolved by related controller design methods. The CONE metrics regulated inter-population genotype recombination based on the similarity of specialized behaviors exhibited by controllers and the similarity of genotypes. These results are supported by previous work that applied CONE to evolve controllers in a multi-rover task [24]. This article's study, thus, also demonstrates that CONE is appropriate for evolving collective behaviors in tasks where behavioral specialization is beneficial, but the form of specialization is not known *a priori*.

Future work will focus on investigating inter-dependences between the genotype and specialization metrics in CONE evolved teams, and mechanisms that lead to emergent specialization. Furthermore, CONE's capability to evolve collective behavior solutions requiring both behavioral and morphological specialization will be examined. Thus, the principles of CONE to effectuate specialization as a means of increasing collective behavior task performance will be adapted and tested for agents in cooperative co-evolution systems not using artificial neural network controllers. Different controller types, such as rule-based controllers, will be tested in various collective behavior tasks to ascertain if other controller types yield the same benefits.

References

[1] C. Schultz, L. Parker, Multi-Robot Systems: From Swarms to Intelligent Automata, Kluwer Academic Publishers, Washington, DC, USA, 2002.
[2] E. Bonabeau, M. Dorigo, G. Theraulaz, Swarm Intelligence: From Natural to Artificial Systems, Oxford University Press, Oxford, England, 1998.
[3] D. Hawthorne, Genetic linkage of ecological specialization and reproductive isolation in pea aphids, Nature 412 (1) (2001) 904–907.
[4] M. Resnick, Turtles, Termites, and Traffic Jams: Explorations in Massively Parallel Microworlds, MIT Press, Cambridge, USA, 1997.
[5] D. Futuyma, M. Slatkin, in: D. Futuyma, M. Slatkin (Eds.), Coevolution, Sinauer Associates, Sunderland, Massachusetts, USA, 1983.
[6] J. Polechovà, N. Barton, Speciation through competition: a critical review, Evolution 59 (6) (2005) 1194–1210.
[7] R. Wiegand, An analysis of cooperative coevolutionary algorithms, Ph.D. Thesis, Computer Science Department, George Mason University, Fairfax, Virginia, USA, 2004.
[8] H. Seligmann, Resource partition history and evolutionary specialization of subunits in complex systems, BioSystems 51 (1) (1999) 31–39.
[9] N. Calderone, R. Page, Genotypic variability in age polyethism and task specialization in the honey bee. Apis mellifera, Behavioral Ecology and Sociobiology 22 (1) (1988) 17–25.
[10] X. Yao, Evolving artificial neural networks, Proceedings of the IEEE 87 (9) (1999) 1423–1447.
[11] M. Potter, K. De Jong, Cooperative coevolution: an architecture for evolving coadapted subcomponents, Evolutionary Computation 8 (1) (2000) 1–29.
[12] S. Haykin, Neural Networks: A Comprehensive Foundation, 2nd ed., Prentice Hall, Princeton, USA, 1998.
[13] S. Nolfi, D. Floreano, Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines, MIT Press, Cambridge, USA, 2000.
[14] A. Farinelli, R. Farinelli, L. Iocchi, D. Nardi, Multi-robot systems: a classification focused on coordination, IEEE Transactions on Systems, Man, and Cybernetics, Part B 34 (2004) 2015–2028.
[15] F. Noreils, Toward a robot architecture integrating cooperation between mobile robots: application to indoor environment, International Journal of Robotics Research 12 (1) (1993) 79–98.
[16] P. Stone, Layered Learning in Multiagent Systems, MIT Press, Cambridge, USA, 2000.
[17] M. Batalin, G. Sukhatme, Spreading out: a local approach to multi-robot coverage, in: H. Asama, T. Arai, T. Fukuda, T. Hasegawa (Eds.), Distributed Autonomous Robotic Systems, Springer, New York, USA, 2002, pp. 373–382.
[18] L. Steels, Toward a theory of emergent functionality, in: Proceedings of the First International Conference on Simulation of Adaptive Behavior, MIT Press, Cambridge, USA, 1990, pp. 451–461.
[19] R. Miikkulainen, Neuroevolution, in: C. Sammut, G. Webb (Eds.), Encyclopedia of Machine Learning, Springer, New York, USA, 2010, pp. 716–720.
[20] D. Floreano, P. Dürr, C. Mattiussi, Neuroevolution: from architectures to learning, Evolutionary Intelligence 1 (1) (2008) 47–62.
[21] K. Chellapilla, D. Fogel, Evolving neural networks to play checkers without expert knowledge, IEEE Transactions on Neural Networks 10 (16) (1999) 1382–1391.
[22] B. Bryant, R. Miikkulainen, Neuro-evolution for adaptive teams, in: Proceedings of the Congress on Evolutionary Computation, IEEE Press, Canberra, Australia, 2003, pp. 2194–2201.
[23] J. Blumenthal, G. Parker, Competing sample sizes for the co-evolution of heterogeneous agents, in: Proceedings of the International Conference on Intelligent Robots and Systems, IEEE Press, Sendai, Japan, 2004, pp. 1438–1443.
[24] G. Nitschke, M. Schut, A. Eiben, Collective neuro-evolution for evolving specialized sensor resolutions in a multi-rover task, Evolutionary Intelligence 3 (1) (2010) 13–29.
[25] M. Potter, L. Meeden, A. Schultz, Heterogeneity in the coevolved behaviors of mobile robots: the emergence of specialists, in: Proceedings of the International Joint Conference on Artificial Intelligence, AAAI Press, Seattle, 2001, pp. 1337–1343.
[26] L. Li, A. Martinoli, A. Yaser, Learning and measuring specialization in collaborative swarm systems, Adaptive Behavior 12 (3) (2004) 199–212.
[27] G. Nitschke, Neuro-evolution for emergent specialization in collective behavior systems, Ph.D. Thesis, Computer Science Department, Vrije Universiteit, Amsterdam, Netherlands, 2009.
[28] C. Yong, R. Miikkulainen, Coevolution of role-based cooperation in multi-agent systems, Technical Report AI07-338, Department of Computer Sciences, The University of Texas, Austin, USA, 2007.
[29] M. Potter, The design and analysis of a computational model of cooperative coevolution, Computer Science Department, George Mason University, Fairfax, Virginia, USA, 1997.
[30] G. Theraulaz, E. Bonabeau, Coordination in distributed building, Science 269 (1) (1995) 686–688.
[31] J. Werfel, R. Nagpal, Three-dimensional construction with mobile robots and modular blocks, The International Journal of Robotics Research 27 (3–4) (2008) 463–479.
[32] A. Panangadan, M. Dyer, Construction in a simulated environment using temporal goal sequencing and reinforcement learning, Adaptive Behavior 17 (1) (2009) 81–104.
[33] E. Bonabeau, G. Theraulaz, J. Deneubourg, Quantitative study of the fixed threshold model for the regulation of division of labour in insect societies, Proceedings of the Royal Society of London, Series B 263 (1) (1996) 1565–1569.
[34] G. Theraulaz, E. Bonabeau, J. Deneubourg, Fixed response thresholds and the regulation of division of labor in insect societies, Bulletin of Mathematical Biology 60 (1) (1998) 753–807.
[35] J. Gautrais, G. Theraulaz, J. Deneubourg, C. Anderson, Emergent polyethism as a consequence of increased colony size in insect societies, Journal of Theoretical Biology 215 (1) (2002) 363–373.
[36] A. Murciano, J. Millan, Learning signaling behaviors and specialization in cooperative agents, Adaptive Behavior 5 (1) (1997) 5–28.
[37] A. Ijspeert, A. Martinoli, A. Billard, L. Gambardella, Collaboration through the exploitation of local interactions in autonomous collective robotics: the stick pulling experiment, Autonomous Robots 11 (2) (2001) 149–171.
[38] M. Waibel, D. Floreano, S. Magnenat, L. Keller, Division of labor and colony efficiency in social insects: effects of interactions between genetic

architecture, colony kin structure and rate of perturbations, Proceedings of the Royal Society B 273 (1) (2006) 1815–1823.
[39] E. Bonabeau, A. Sobkowski, G. Theraulaz, J. Deneubourg, Adaptive task allocation inspired by a model of division of labour in social insects, in: Bio-Computing and Emergent Computation, World Scientific, Singapore, 1997, pp. 36–45.
[40] G. Thomas, A. Howard, A. Williams, A. Moore-Alston, Multirobot task allocation in lunar mission construction scenarios, in: Systems, Man and Cybernetics, 2005 IEEE International Conference on Volume 1, IEEE Press, 2005, pp. 518–523.
[41] H. Guo, Y. Meng, Y. Jin, A cellular mechanism for multi-robot construction via evolutionary multi-objective optimization of a gene regulatory network, BioSystems 98 (3) (2009) 193–203.
[42] A. Panangadan, M. Dyer, Goal sequencing for construction agents in a simulated environment, in: Proceedings of the International Conference on Artificial Neural Networks, IEEE Press, Las Vagas, USA, 2002, pp. 969–974.
[43] R. Sutton, A. Barto, An Introduction to Reinforcement Learning, John Wiley and Sons, Cambridge, USA, 1998.
[44] G. Nitschke, Neuro-evolution methods for gathering and collective construction, in: Proceedings of the 10th European Conference on Artificial Life, Springer, Budapest, Hungary, 2009, pp. 111–119.
[45] G. Beni, From swarm intelligence to swarm robotics, in: Proceedings of the First International Workshop on Swarm Robotics, Springer, Santa Monica, USA, 2004, pp. 1–9.
[46] S. Luke, C. Cioffi-Revilla, L. Panait, K. Sullivan, G. Balan, MASON: a multiagent simulation environment, Simulation 81 (7) (2005) 517–527.
[47] F. Gomez, Robust non-linear control through neuroevolution, Ph.D. Thesis, Computer Science Department, University of Texas, Austin, USA, 2003.
[48] S. Luke, C. Hohn, J. Farris, G. Jackson, J. Hendler, Co-evolving soccer softbot team coordination with genetic programming, in: RoboCup-97: Robot Soccer World Cup I, Springer-Verlag, Berlin, Germany, 1998, pp. 398–411.
[49] G. Baldassarre, D. Parisi, S. Nolfi, Coordination and behavior integration in cooperating simulated robots, in: Proceedings of 8th Int. Conf. Simulation Adaptive Behavior, MIT Press, Cambridge, USA, 2003, pp. 385–394.
[50] N. Garcia-Pedrajas, C. Hervas-Martinez, D. Ortiz-Boyer, Cooperative coevolution of artificial neural network ensembles for pattern classification, IEEE Transactions on Evolutionary Computation 9 (3) (2005) 271–302.
[51] M. Waibel, L. Keller, D. Floreano, Genetic team composition and level of selection in the evolution of cooperation, IEEE Transactions on Evolutionary Computation 13 (3) (2009) 648–659.
[52] M. Mirolli, D. Parisi, How can we explain the emergence of a language that benefits the hearer but not the speaker, Connection Science 17 (3) (2005) 307–324.
[53] S. Luke, Genetic programming produced competitive soccer softbot teams for robocup 97, in: Proceedings of 3rd Annu. Conf. Genetic Programming, Morgan Kaufmann, San Mateo, USA, 1998, pp. 214–222.
[54] W. Hamilton, The genetical evolution of social behavior i + ii, Journal of Theoretical Biology 7 (1) (1964) 1–52.
[55] L. Lehmann, L. Keller, The evolution of cooperation and altruism: a general framework and a classification of models, Journal of Evolutionary Biology 19 (5) (2006) 1365–1376.
[56] A. Perez-Uribe, D. Floreano, L. Keller, Effects of group composition and level of selection in the evolution of cooperation in artificial ants, in: Advances of Artificial Life: Proceedings of the Seventh European Conference on Artificial Life, Springer, Dortmund, Germany, 2003, pp. 128–137.
[57] M. Wineberg, F. Oppacher, Underlying similarity of diversity measures in evolutionary computation, in: Proceedings of Genetic and Evolutionary Computation Conference, Springer, Chicago, USA, 2003, pp. 1493–1504.
[58] A. Eiben, J. Smith, Introduction to Evolutionary Computing, Springer, Berlin, Germany, 2003.
[59] J. Werfel, R. Nagpal, Extended stigmergy in collective construction, IEEE Intelligent Systems 21 (2) (2006) 20–28.
[60] S. Nolfi, Evorobot 1.1 user manual, Technical Report, Institute of Cognitive Sciences, National Research Council, Rome, Italy, 2000.
[61] F. Mondada, E. Franzi, P. Ienne, Mobile robot miniaturization: a tool for investigation in control algorithms, in: Proceedings of Third International Symposium on Experimental Robotics, IEEE Press, Kyoto, Japan, 1993, pp. 501–513.
[62] A. Agogino, K. Tumer, Efficient evaluation functions for multi-rover systems, in: Proceedings of the Genetic and Evolutionary Computation Conference, Springer, New York, USA, 2004, pp. 1–12.
[63] A. Eiben, S. Smit, Parameter tuning for configuring and analyzing evolutionary algorithms, Swarm and Evolutionary Computation 1 (1) (2011) 19–31.
[64] F. Gomez, R. Miikkulainen, Incremental evolution of complex general behavior, Adaptive Behavior 5 (1) (1997) 317–342.
[65] B. Flannery, S. Teukolsky, W. Vetterling, Numerical Recipes, Cambridge University Press, Cambridge, UK, 1986.
[66] C.W. Dunnett, A multiple comparisons procedure for comparing several treatments with a control, Journal of the American Statistical Association 50 (1955) 1096–1121.