Cooperative Coevolution of Partially Heterogeneous Multiagent Systems

Jorge Gomes
BioMachines Lab &
Instituto de Telecomunicações
& Faculdade de Ciências,
Universidade de Lisboa, BioISI
Lisboa, Portugal
jgomes@di.fc.ul.pt

Pedro Mariano
Faculdade de Ciências,
Universidade de Lisboa, BioISI
Lisboa, Portugal
plmariano@fc.ul.pt

Anders Lyhne Christensen
BioMachines Lab &
Instituto de Telecomunicações
& Instituto Universitário de
Lisboa (ISCTE-IUL)
Lisboa, Portugal
anders.christensen@iscte.pt

ABSTRACT

Cooperative coevolution algorithms (CCEAs) facilitate the evolution of heterogeneous, cooperating multiagent systems. Such algorithms are, however, subject to inherent scalability issues, since the number of required evaluations increases with the number of agents. A possible solution is to use partially heterogeneous (*hybrid*) teams: behaviourally heterogeneous teams composed of homogeneous sub-teams. By having different agents share controllers, the number of coevolving populations in the system is reduced. We propose *Hyb-CCEA*, an extension of cooperative coevolution to partially heterogeneous multiagent systems. In *Hyb-CCEA*, both the agent controllers and the team composition are under evolutionary control. During the evolutionary process, we rely on measures of behaviour similarity for the formation of homogeneous sub-teams (*merging*), and propose a stochastic mechanism to increase heterogeneity (*splitting*). We evaluate *Hyb-CCEA* in multiple variants of a simulated herding task, and compare it with a fully heterogeneous CCEA. Our results show that *Hyb-CCEA* can achieve solutions of similar quality using significantly fewer evaluations, and in most setups, *Hyb-CCEA* even achieves significantly higher fitness scores than the CCEA. Overall, we show that merging and splitting populations are viable mechanisms for the cooperative coevolution of hybrid teams.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence

Keywords

Cooperative coevolution; emergent team composition; heterogeneity; scalability; partially heterogeneous teams.

1. INTRODUCTION

Cooperative coevolution algorithms (CCEAs) evolve solutions that consist of interacting, coadapted components [22]. CCEAs are commonly used for the evolution of multiagent behaviours [23]: the natural decomposition of the problem into sub-components makes the domain a good fit for cooperative coevolution, as each agent can represent a component of the solution. CCEAs are thus capable of evolving a heterogeneous set of agent behaviours, where each agent can have a specialised behaviour [23, 29]. CCEAs are, however, associated with inherent scalability issues [21, 28], since each agent behaviour typically evolves in a separate population. The computational complexity therefore increases at least linearly with the number of agents. Moreover, credit assignment issues might also arise in large multiagent systems: it can be hard to assess the contribution of each individual agent to the performance of the team as a whole [1].

In multiagent systems with large numbers of agents, successful solutions may contain agents with similar behaviours [16]. Nonetheless, if each population is isolated, as it is typically the case in CCEAs, similar agent behaviours might have to be learned multiple times in different populations. One way to increase the scalability of multiagent learning is through the reduction of the heterogeneity in the system [21]. Partially heterogeneous multiagent systems, also known as *hybrids*, are composed of multiple homogeneous sub-teams — sub-teams in which all agents have identical controllers. By allowing partial heterogeneity, the number of agent controllers that need to be evolved is reduced, improving the scalability of the learning process [14]. Most previous studies on the evolution of partially heterogeneous teams are focused on *team learning*, where one single genome encodes the behaviour and/or the composition of the whole team [2, 11, 13]. The cooperative coevolution of hybrid multiagent systems is still unexplored.

We propose an extension to the traditional cooperative coevolution architecture, *Hyb-CCEA*, that facilitates the emergence of partially heterogeneous teams. In *Hyb-CCEA*, both the team composition and the agent controllers are under evolutionary control. By forming homogeneous sub-teams inside the larger heterogeneous team, agent controllers can be shared by multiple agents in the team. This approach has potential to significantly improve scalability, as the number of populations can be much lower than the number of agents in the team, which translates in fewer individuals to evaluate. In *Hyb-CCEA*, populations can be *merged* if they are converging to the same region of the agent behaviour space, thereby avoiding the evolution of similar behaviours in separate populations. When two populations merge, a new and larger homogeneous sub-team is created, composed

Appears in: *Proceedings of the 14th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2015), Bordini, Elkind, Weiss, Yolum (eds.), May 4–8, 2015, Istanbul, Turkey.*

of the agents that were assigned to each former population. We also propose a stochastic *splitting* mechanism to complement the merging mechanism. The number of populations in the system is therefore dynamic, while the number of agents remains fixed. Multiple team compositions and levels of heterogeneity are thus explored throughout evolution.

We study the proposed approach in a simulated herding task, where each agent is controlled by a neural network. We explore multiple variations of the herding task, that involve different numbers of agents, require different degrees of specialisation, and that either benefit from the existence of sub-teams or not. We evaluate the performance improvements of *Hyb-CCEA* over a fully-heterogeneous CCEA. We additionally study the evolutionary dynamics of *Hyb-CCEA*, evaluating the dynamic of populations throughout evolution, the impact of the merge and split procedures, and the choice of the algorithm's main parameters.

2. RELATED WORK

2.1 Heterogeneous Multiagent Systems

Heterogeneous multiagent systems are characterised by the morphological and/or behavioural diversity of their agents. In morphologically heterogeneous systems, agents have varied actuation and sensing capabilities, and through collaboration, the system takes advantage of the collective set of capabilities [7]. In behaviourally heterogeneous systems, agents share the same morphology and capabilities, but have different controllers. In such systems, specialisation and division of labour are often paramount [23, 29, 4].

Heterogeneity in a multiagent system may significantly increase its capabilities, at the price of added complexity [25]. Heterogeneity complicates the multiagent learning process, as the search space becomes significantly larger [21]. In a heterogeneous system, behavioural control must integrate the abilities of different agents to work in synergy towards achieving a common goal. To solve the behavioural control problem, it is necessary to pursue a holistic approach, in which interactions between different agents are taken into account from the very beginning [7, 21]. Artificial evolution has proven to be a powerful tool for the synthesis of controllers for multiagent systems, both in homogeneous systems [3, 26] and in heterogeneous systems [23, 29].

To evolve controllers for heterogeneous systems, two main approaches can be considered [21]: *team learning* and *concurrent learning*. In *team learning*, all controllers for the whole team are encoded in a single genome [2, 27]. In *concurrent learning*, multiple learning processes for different parts of the team run in parallel, with the objective of improving the team as a whole. The most prominent *concurrent learning* algorithm is cooperative coevolution [22], where the controllers for the different agents are coevolved in separate populations that interact during the evaluation process. In this study, we focus on cooperative coevolution algorithms, which will be presented in the following section.

2.2 Cooperative Coevolution

Cooperative coevolution has a number of advantages over the non-coevolutionary approaches for the design of heterogeneous multiagent systems, such as: (i) the capability of working on a problem decomposition, comprised by more manageable sub-problems [12, 22], and (ii) the emergence of agent roles and specialisations [23, 29]. Some of the previous applications of CCEAs in the domain of embodied multiagent systems include the cooperative predator-prey task [29, 9, 18], multi-rover task [19], collective construction [16], herding [23], and keepaway soccer [9].

In a typical application of a CCEA, each agent is evolved in a separate population [23]. At every generation of the evolutionary algorithm, each population is evaluated in turn. To evaluate an individual from one population, teams are formed with representatives from the other populations. The teams are then evaluated by a fitness function in the problem domain, and the individual being evaluated receives the fitness score obtained by the team as a whole. The fitness differential is thus strictly a function of the individual's contribution to the problem-solving effort within the context of the other team members.

Cooperative coevolution has to deal with the intricate dynamics that stem from having multiple coevolving populations [28]. Some of the main challenges include premature convergence to *mediocre stable states* [20, 9], and the generally poor scalability of the algorithms [21]. The classic CCEA architecture is associated with inherent scalability issues with respect to the number of agents in the team. If each agent evolves in a separate population, the number of populations increases linearly with the number of agents, which opens the search space and increases computational complexity. Moreover, in large teams, the impact of a single agent in the behaviour of the team can be almost imperceptible, which can cause the fitness gradients to vanish [1].

The scalability issue is also related with the problem of *reinvention* [21]. CCEAs commonly separate agents into different populations, creating a strict separation between agents. In many multiagent tasks, however, there can be a high degree of overlap between the policies of each agent [16]. The evolutionary process can potentially waste many resources learning the same behaviour in multiple populations. In previous works, the problem of reinvention has been addressed by pre-programming the shared skillset in the robots [23], or by implementing a shaping phase to evolve the basic skillset for all robots [17]. An alternative approach was adopted in CONE-2 [16], where evolution starts with only one agent, and more agents, based on existing ones, can be added throughout evolution. CONE-2 thereby adapts the number of agents to the task at hand and mitigates the problem of reinvention.

2.3 Partially Heterogeneous MAS

One way of improving the scalability of multiagent learning is through the reduction of heterogeneity in the system [21]. By reducing heterogeneity, the number of agent controllers that need to be learned decreases, thus improving scalability [14]. With partial heterogeneity, the problem of reinvention can also be addressed: if different agents perform the same task, they can belong to the same homogeneous sub-team, thus avoiding the evolution of similar solutions in different populations.

Evolution of partially heterogeneous multiagent systems is a relatively unexplored field of research [27]. Previous works on the evolution of hybrid teams have been mostly restricted to team learning. Luke [14] evolved hybrid teams for the RoboCup challenge, with the team composition manually specified beforehand. Each genome was a forest of GP trees that encoded the behaviour of the multiple sub-teams. Hara [11] proposed a GP-based grouping technique, *Auto-*

matically Defined Groups (ADG), that automatically discovers the optimal number of groups and their compositions. Also based on genetic programming, Bongard [2] proposed the *Legion System*, where the genome encodes the composition of the team and one sub-tree for each behaviour class. In the context of team learning, Lichocki et al. [13] studied crossover operators for the evolution of hybrid teams, focusing on the exchange of agents between teams. A different neuroevolution approach was taken in [5]: the agent controllers were encoded in a single genome, and an indirect encoding technique, *HyperNEAT*, was used to exploit similarities in agents' policies. HyperNEAT allowed for sharing of policies between agents, while still exhibiting variations.

While encoding the whole team in one genome can facilitate the evolution of team compositions, such approaches lack the advantages provided by coevolutionary algorithms: the decomposition of the large search space into more tractable sub-problems. To cope with large heterogeneous multirobot systems, Nitschke [19] proposed CONE, a cooperative coevolution method that allows regulated breeding between different populations, each corresponding to a different agent. Crossover between different populations is allowed if the individuals of those populations share the same specialisation. The possible specialisations are specified manually by the experimenter. It was shown that CONE can improve the performance of coevolution when the tasks require a high degree of specialisation. The evolved teams are, however, still fully heterogeneous, as complete controllers are not shared by different agents.

In this paper, we propose a cooperative coevolution method that is capable of evolving partially heterogeneous teams. Both the team composition and the agent controllers are under evolutionary control. Our method draws inspiration from CONE [19], in the sense that it relies on agent specialisations to regulate interactions between different populations. In contrast to CONE, however, our method uses these specialisations to build homogeneous sub-teams, rather than only relying on them for the exchange of genetic material. Our approach is described in the following section.

3. HYB-CCEA: COOPERATIVE COEVOLUTION OF HYBRID TEAMS

We propose *Hyb-CCEA*, an extension of a CCEA to evolve agent controllers for physically homogeneous, behaviourally heterogeneous multiagent systems. In traditional applications of CCEAs in multiagent systems, there is a one-to-one mapping between agents and populations. Our approach departs from this concept: we allow population individuals to encode a controller that can be used by multiple agents, thus effectively forming homogeneous sub-teams inside heterogeneous teams. In *Hyb-CCEA*, each population is assigned to a subset of the agents in the team. The number of population individuals is constant across all populations, and two different populations cannot be assigned to the same agent.

During evaluation, all individuals of a given population are assigned to the same set of agents (see Figure 1). Each population thus become responsible for the evolution of a homogeneous sub-team, not just one specific agent. The rest of the coevolutionary evaluation operates the same way as a traditional CCEA [22]: individuals are joined with representative individuals from the other populations for evaluation, and the individual being evaluated receives the fitness

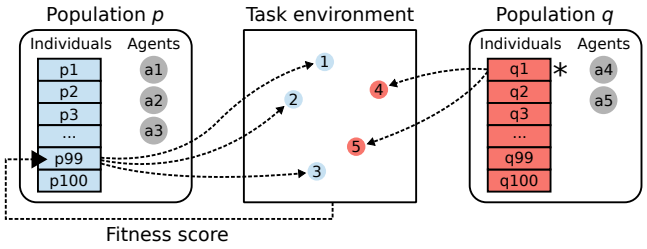

Figure 1: Illustration of the evaluation phase. In this example, the individual *p99* of population *p* is being evaluated. The representative individual of *q* is *q1*. The population individuals (controllers) are assigned to the respective agents, and the fitness of the whole team is assigned to individual *p99*.

Algorithm 1 *Hyb-CCEA* algorithm.

1: Let P be a set of populations, and A the set of all agents.
2: $P \leftarrow createPopulations()$
3: **for** each generation **do**
4: **for** $p \in P$ **do**
5: **for** each individual $i \in p$ **do**
6: $t \leftarrow$ Form a team with i and the representative r_q from each other population $q \neq p$.
7: $fit_i \leftarrow evaluate(t)$, assigning each individual to the respective set of agents.
8: $P \leftarrow attemptSplit(P)$
9: $P \leftarrow attemptMerge(P)$
10: **for** $p \in P$ **do**
11: $r_p \leftarrow$ individual $i \in p$ with maximum *fit* score.
12: Breed p, based on the fitness scores *fit* of the individuals.

score that the team as a whole obtained. The outline of the algorithm is shown in Algorithm 1. Throughout this study, only one collaboration is formed to evaluate each individual. Evaluations could, however, rely on more collaborations, for example using randomly chosen collaborators [20].

The distinctive aspect of *Hyb-CCEA* is that it does not assume that the optimal number of sub-teams and their composition are known beforehand: we extend the CCEA so that the number of composition of the sub-teams is also under evolutionary control. Different levels of heterogeneity can thus be explored throughout the evolutionary process. To this end, we propose: (i) a procedure for merging two populations (step 9), which creates a new population assigned to the agents of the two former populations, thus decreasing the heterogeneity of the system; and (ii) a procedure for splitting a population (step 8), which creates two populations assigned to different sets of agents, thus increasing heterogeneity. Both the split and merge procedures were implemented with the objective of minimising any immediate negative impact in the behaviour and performance of the teams. The evolutionary process can therefore follow the fitness gradients without major disruptions caused by sudden changes in team compositions. The following sections detail the initialisation, merge, and split procedures.

3.1 Initialisation Procedure

We consider two alternatives for the creation of the initial populations, which are studied and compared in Section 5.3.

Homogeneous start: There is only one population initially. All the agents of the multiagent system are assigned to this population.

Heterogeneous start: There is one population assigned to each agent of the multiagent system.

Another alternative would be to initialise the algorithm with a partially heterogeneous configuration, based on the experimenter's domain knowledge for instance. This alternative is fully compatible with the proposed algorithm, but we do not address it in this paper due to space restrictions.

3.2 Merge Procedure

Previous works have shown that cross-breeding between agents that share similar specialisations [19], or belong to the same sub-team [15], can be beneficial for the emergence of specialisations. We go beyond this concept by allowing behaviourally similar agents to become genetically homogeneous. If two separate populations are evolving similar agent behaviours, they can be merged into one population, see Figure 2. To identify behaviour similarities between agents, we rely on an *agent behaviour characterisation* [10] provided by the experimenter. This characterisation is a real-valued vector $\beta(a)$, composed of features that describe the state of the agent a during task execution, and/or the relation of the agent with the environment and other agents [10]. Characterisations are obtained during the evaluation phase: besides obtaining the fitness score of the individual, the behaviour of the respective agent(s) is also recorded.

The concept behind the *merge procedure*, described in Algorithm 2, is to first obtain a set of agent behaviours that is representative of each population, then measure the distance between these sets of behaviours, and finally merge populations with a high degree of behavioural similarity.

The behavioural set B of a population p is obtained (steps 2 and 5) by aggregating the agent behaviours recorded during the evaluation of p, considering the agents to which that population is currently assigned (A_p):

$$agentBehaviours(p) = \left\{\beta(i_a) : i \in p' \wedge a \in A_p\right\} \quad . \qquad (1)$$

Since the objective is to assess the behaviour space region to which a population is converging, we only consider the high-fitness portion of the population (p').[1]

The distance between two populations, including the distance between a population and itself, is then given by the mean pairwise distance between the respective sets of agent behaviours, $\mathcal{A}$ and $\mathcal{B}$:

$$pairwiseDistance(\mathcal{A}, \mathcal{B}) = \frac{1}{|\mathcal{A}||\mathcal{B}|} \sum_{x \in \mathcal{A}} \sum_{y \in \mathcal{B}} d(x, y) \quad , \qquad (2)$$

where d is the Euclidean distance between the behaviour characterisation vectors.

The distance between each two populations is then normalised (step 8), by dividing it by the respective intra-population distances (steps 3 and 6). The normalised distance measure $\delta_{p,q}$ represents the separation of two populations p and q in agent behaviour space. If two populations are converging to similar behaviours, their distance will be close to 1. The most similar populations are merged if the distance between them is inferior to the *merge threshold* T_D (step 10). Only populations older than a *stability threshold* T_G are allowed to merge. This threshold enforces a stability period that provides the population time to adapt to the current set of assigned agents. The new population replaces the two parent populations, and is comprised by the respective high-fitness individuals (steps 11–14). The number of individuals that are drawn from each population is proportional to the number of agents assigned to that population.

[1] In the experiments presented in this paper, we considered the high-fitness portion as the top 20% of each population, which is the same as the *survival threshold* of the NEAT algorithm we use. Preliminary experiments showed this parameter is robust to moderate variation.

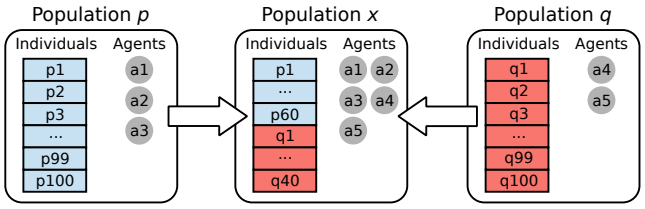

Figure 2: Merge procedure: the new population x replaces the two parents, p and q. The population x is formed by a subset of their individuals, and is assigned to all the parents' agents.

Algorithm 2 Merge procedure.

1: **for** each population $p \in P \wedge age(p) \geq T_G$ **do**
2: $B_p \leftarrow agentBehaviours(p)$
3: $\Delta_p \leftarrow pairwiseDistance(B_p, B_p)$
4: **for** each $q \in P \wedge q \neq p \wedge age(q) \geq T_G$ **do**
5: $B_q \leftarrow agentBehaviours(q)$
6: $\Delta_q \leftarrow pairwiseDistance(B_q, B_q)$
7: $\Delta_{p,q} \leftarrow pairwiseDistance(B_p, B_q)$
8: $\delta_{p,q} \leftarrow \frac{2\Delta_{p,q}}{\Delta_p + \Delta_q}$
9: Select p and q with the minimum $\delta_{p,q}$.
10: **if** $\delta_{p,q} \leq T_D$ **then**
11: Create a pop. x, assigned to the agents $A_p \cup A_q$.
12: Add to x the highest-fit individuals of p and q, in the proportions $\frac{|A_p|}{|A_q \cup A_q|}$ and $\frac{|A_q|}{|A_p \cup A_q|}$, respectively.
13: Remove p and q from P.
14: Add x to P.

3.3 Split Procedure

The *split procedure* increases the heterogeneity of the system by breaking up a homogeneous sub-team: two clones of a population are created, and each one is assigned to a different set of agents, see Figure 3. Since all agents assigned to a given population share the same controllers, they will most likely display very similar behaviours. Therefore, contrary to the merging process, we cannot rely on agent behaviour characterisations to regulate splits. We resort to a stochastic approach to splitting, described in Algorithm 3.

In the splitting process, each population is assigned a splitting pressure S_p (step 2). The splitting pressure increases linearly with the age of the population (number of generations since it was created) and the fraction of agents it is assigned to. The fraction of agents has a limited influence on the splitting pressure, in order to guarantee a healthy frequency of splits in multiagent systems of any size. The population with the highest pressure is split if S_p is above the stability threshold T_G (steps 3–4). The individuals of

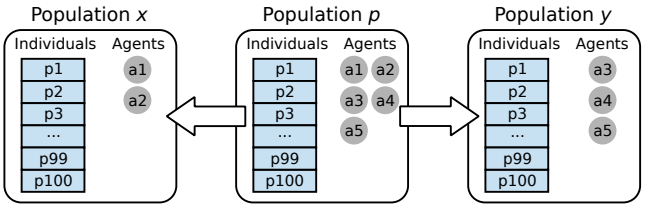

Figure 3: Split procedure: two new populations, x and y, replace the parent p. The populations x and y are copies of p, but each one is assigned to a disjoint set of agents.

Algorithm 3 Split procedure.

1: **for** each population $p \in P \wedge |A_p| \geq 2$ **do**
2: $S_p \leftarrow \frac{age(p)}{2}\left(1 + \frac{|A_p|-2}{|A|-2}\right), |A| > 2$
3: Select the population p with the maximum S_p.
4: **if** $S_p \geq T_G$ **then**
5: Randomly split the set of agents A_p into A_x and A_y such that: $A_x \cup A_y = A_p \wedge A_x \cap A_y = \varnothing$.
6: Create two copies of p, x and y, assigned to the set of agents A_x and A_y.
7: Remove p from P.
8: Add x and y to P.

the two resulting populations are exact copies of the individuals of the parent population (step 6). The set of agents of the parent population is randomly split into two (step 5), and each of the child populations is assigned to one of those subsets (step 6).

Stochastic splits are feasible because splits can be reverted later by the *merge procedure*. As both the merge and split procedures use the stability threshold T_G, new populations cannot be removed for at least T_G generations. This stability period provides the new populations time to converge to different regions of the agent behaviour space. If two recently split populations did not converge to different behaviours after the stability period, the distance between them will be relatively small, and they can therefore be merged again.

4. EXPERIMENTAL SETUP

Our experiments are based on a simulated version of the herding task [23]. In this task, a group of physically homogeneous shepherds must corral one or more sheep. To make the task more challenging, one or more foxes are present, which try to capture the sheep, and must be kept away by the shepherds. Only the controllers for the shepherds are evolved, and the other agents have a preprogrammed behaviour. The herding task requires behavioural heterogeneity in the team of shepherds [23], since there are different tasks that must be performed simultaneously: corral the sheep and keep away the foxes. We use multiple variants of the task in our experiments. The number of agent specialisations required to solve the task is varied by using different numbers of sheep and foxes. Additionally, we introduce two fox types: a *weak fox*, which can be kept away by just one shepherd, and an *evasive fox*, which requires more than one shepherd to be stopped. By introducing these variations to the herding task, we can study the algorithms in setups that require different sub-teams inside the larger team.

Table 1: Task setups.

Setup	Sheph.	Sheep	Foxes	Fox type	Eval. budget
W5	5	1	2	Weak	150K
E5	5	1	1	Evasive	150K
W7	7	2	2	Weak	300K
E7	7	1	2	Evasive	300K
W10	10	2	3	Weak	400K
E10	10	1	3	Evasive	400K

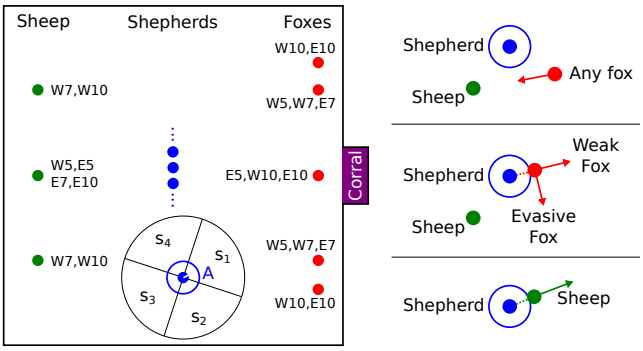

Figure 4: Left: Initial conditions for each setup. The labels on the sheep and foxes indicate the initial positions that are used in each setup. The shepherd on the bottom illustrates the sensor setup. Right: Illustration of the pre-programmed behaviour of the sheep and the different types of foxes.

The task variants are listed in Table 1, and the initial conditions are depicted in Figure 4 (left). The agents' initial positions for each variant are fixed. The task environment is a bounded square arena of size 150x150 units, with no obstacles. Each shepherd has the following sensors, which correspond to the inputs of the neural network: (i) four sensors that return the distance of the nearest shepherd ($s_1 - s_4$), with a range of 25 units; and (ii) $2(f + s + 1)$ sensors that return the distance and relative orientation for each of the f foxes, each of the s sheep, and the centre of the corral. The two neural outputs control respectively the shepherd's linear speed (maximum of 1 unit/step) and turning speed (maximum of 22.5°/step).

When a shepherd approaches a sheep (distance less than 5 units), the sheep moves 1 unit in the opposite direction of that shepherd (see Figure 4, right). The sheep are otherwise passive. The behaviour of the foxes is preprogrammed: each fox tries to intercept the closest sheep by estimating its future position and by moving 1 unit in that direction. If a shepherd gets close to a fox (distance less than 5 units), the fox moves away from the shepherd. If the foxes are of the *weak* type, the fox moves in the opposite direction. If the foxes are of the *evasive* type, the fox moves in a perpendicular direction, choosing the side where the closest sheep is present. Figure 4 (right) illustrates the fox behaviours. A trial ends when all sheep enter the corral, all sheep are captured by a fox, or when 500 time steps have elapsed.

The fitness function F rewards the shepherds for getting the sheep closer to the corral, and in case a sheep is successfully corralled, for the amount of time it took to corral it. To bootstrap the evolutionary process, the fitness function

also rewards the shepherds for getting close to the sheep at any point in the simulation trial.

$$F = \sum_{s \in S} \left[\frac{1 - m_s}{|S|} + \begin{cases} 2 - \tau_s/T & \text{if } s \text{ is corralled} \\ 1 - d_{f,s}/d_{i,s} & \text{otherwise} \end{cases} \right], \quad (3)$$

where S is the set of sheeps, m_s is the minimum distance of any shepherd to s (normalised to $[0, 1]$), τ_s is the number of time steps it took to corral s, T is the maximum trial length, $d_{f,s}$ is the final distance of s to the corral, and $d_{i,s}$ is the initial distance.

The agent behaviour characterisation $\beta(a)$ is a vector that describes the relation of shepherd a with the environment: (i) distance of a to every sheep, averaged over the simulation trial; (ii) mean distance of a to the corral, and (iii) mean distance of a to every fox. The length of the characterisation varies from 3 to 6, depending on the number of sheep and foxes. All features are normalised to the range [0,1].

The agent controllers are neural networks evolved by NEAT [24], a state-of-the-art neuroevolution algorithm that evolves both the weights and topology of the networks, and has been extensively used in evolutionary robotics. We use the NEAT4J[2] implementation and most of the default parameter values: each population has a fixed size of 100 individuals, the mutation probability is 25%, crossover probability is 20%, the probability of adding a connection is 5%, the probability of adding a neuron is 3%, and the target number of species is 5. The same evolutionary parameters were used in the experiments with *Hyb-CCEA* and *CCEA*. The task is implemented in MASON[3], and the evolutionary algorithms are implemented over the ECJ framework[4].

Although the two evolutionary algorithms are generational, we rely on the number of evaluations (i.e., number of evolved individuals) for a fair comparison, since *Hyb-CCEA* has a dynamic number of population individuals. The two algorithms were therefore given a fixed budget of evaluations for each task setup (see Table 1). Every evolutionary treatment is repeated in 30 independent evolutionary runs.

5. RESULTS

5.1 Comparison with Fully Heterogeneous Teams

To evaluate the potential improvements brought by *Hyb-CCEA*, we compare its performance with the underlying cooperative coevolution algorithm (*CCEA*), in which the teams are fully heterogeneous. The *CCEA* is implemented as described in Algorithm 1, but each agent is assigned to a separate population, and there are no split and merge procedures. *Hyb-CCEA* used the same parameter values in all tasks: $T_G = 20$ and $T_D = 1.25$ (these parameters are studied in Section 5.4). Figure 5 shows the highest fitness scores achieved after a given number of evaluations, averaged over 30 evolutionary runs for each method and task. The highest fitness scores achieved by *Hyb-CCEA* were significantly superior to those achieved by *CCEA* in all task setups (Mann-Whitney test, $p < 0.05$), except *W5* ($p = 0.14$). The results show that *Hyb-CCEA* not only achieves higher fitness scores in the end, it also displays a steeper initial increase in fitness scores across all tasks.

[2] `http://neat4j.sourceforge.net`
[3] `http://cs.gmu.edu/~eclab/projects/mason`
[4] `http://cs.gmu.edu/~eclab/projects/ecj`

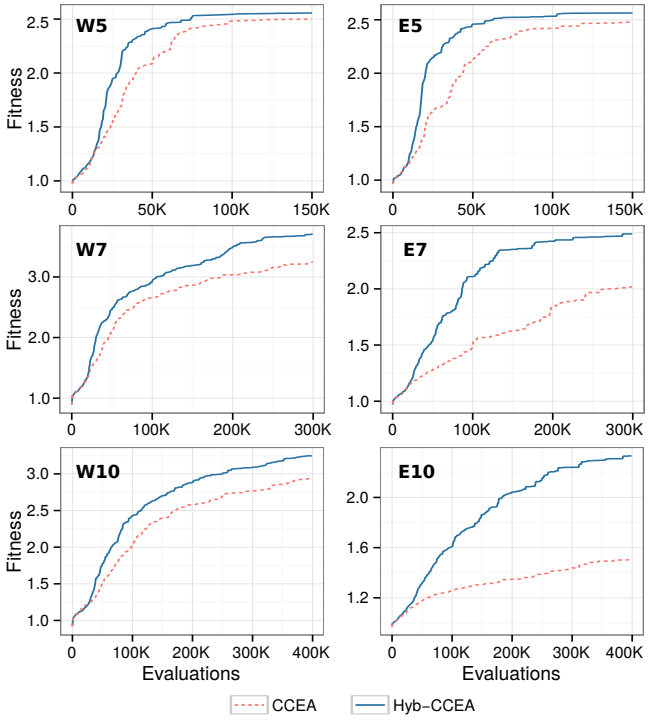

Figure 5: Mean of the highest fitness scores achieved after a given amount of evaluations.

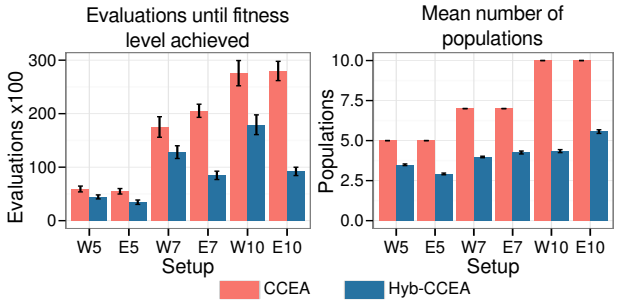

Figure 6: Left: Number of evaluations necessary to reach the following fitness levels – *W5* and *E5*: 2.5; *W7*: 3.0; *E7*: 1.75; *W10*: 2.75; *E10*: 1.5. Right: mean number of populations throughout evolution.

An analysis of how many evaluations were required on average to reach solutions with certain fitness levels is provided in Figure 6 (left). The fitness levels were chosen to be as high as possible while still being achieved by at least half the evolutionary runs of both *CCEA* and *Hyb-CCEA*. In all task setups, *Hyb-CCEA* requires significantly fewer evaluations to reach the same fitness levels (Mann-Whitney test, $p < 0.05$). Figure 6 (right) shows that in all task setups, *Hyb-CCEA* was able to form homogeneous sub-teams, since it used on average significantly fewer populations than the number of agents in the team. By reducing the number of populations, *Hyb-CCEA* could therefore achieve solutions with similar fitness scores with fewer evaluations, when compared with a fully heterogeneous *CCEA*.

The differences in both the number of evaluations and highest fitness scores are more pronounced in the setups with *evasive* foxes. In these setups, the shepherd team can bene-

fit from the existence of sub-teams, since multiple shepherds are required to deal with each fox. This suggests that *Hyb-CCEA* is especially effective in domains where sub-teams should be formed. Nonetheless, in setups that do not require sub-teams (*weak* foxes), *Hyb-CCEA* still managed to achieve good solutions faster, without any sacrifice in terms of solution quality. These results highlight that *Hyb-CCEA* not only reduces the redundancy in the evolutionary process and improves scalability, it also takes advantage of the existence of homogeneous sub-teams to evolve higher quality solutions. In the following sections, we study the mechanisms contributing to the performance of *Hyb-CCEA*.

5.2 Evolutionary Dynamics Analysis

In this section, we analyse the evolutionary dynamics of *Hyb-CCEA* induced by the merge and split procedures. Figure 7 shows the mean number of populations throughout evolution in each task. In the early stages of evolution, there is a steep decrease in the number of populations, which suggests a low level of heterogeneity. This initial decrease is explained by the lack of agent specialisations in the early generations: since the agents display similar behaviours initially, the respective populations are merged. As evolution progresses, populations begin to converge to specific regions of the behaviour space: specialisations emerge. The number of populations therefore tends to increase. It is noteworthy that for setups with the same number of agents, evolution can converge to different number of populations (see *W5* and *E5*, and *W10* and *E10*), which confirms that *Hyb-CCEA* can adapt the level of heterogeneity to the task at hand.

The merging and splitting of populations should not lead to significant disruptions in the evolutionary process. This implies that the fitness of the highest scoring solutions in the system should not decrease substantially after a merge or a split. Table 2 lists the immediate impact of the merge and split procedures. In all setups, the immediate change in fitness after a split occurs is not significantly different from the change when no split or merge occurs (Mann-Whitney test, $p > 0.25$). This suggests that splits have no immediate impact in the performance of the teams, which is desirable, and explained by the fact that both new populations are initialised with the individuals of the parent population.

After a merge occurs, fitness scores tend to decrease slightly ($p < 0.001$ in all setups). In the merge procedure, only populations with *similar* agent behaviours are merged. There might, however, exist subtle differences in the behaviour of different agents that are beneficial for the behaviour of the team. If those agents become genetically homogeneous, these subtle differences disappear, potentially decreasing the performance of the teams. This impact, however, is relatively small on average (around 2% of the fitness score), and the results suggest that it does not compromise the effectiveness of *Hyb-CCEA*.

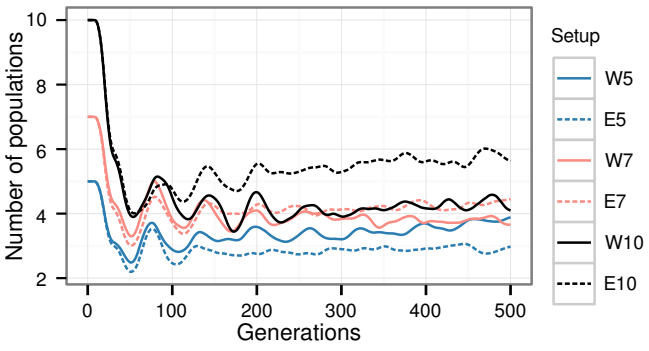

Figure 7: Mean number of populations at each generation. Results are only shown up to the 500th generation, the values remain relatively stable after.

Table 2: Mean variation of the highest fitness score after a merge/split, and mean number of merges/splits in each run. The *Stable* column shows the variation when no merge/split occurs.

Setup	Merges		Splits		Stable
	Num.	Var.	Num.	Var.	Var.
W5	13.26	-1.96%	12.20	+1.50%	+1.02%
E5	16.97	-3.01%	14.93	+0.60%	+0.55%
W7	25.57	-2.00%	22.43	+1.20%	+0.74%
E7	23.27	-2.67%	20.60	+1.00%	+0.76%
W10	41.50	-1.80%	35.03	+0.71%	+0.87%
E10	34.06	-0.76%	29.66	+0.27%	+0.51%

5.3 Initialisation Procedure

As discussed in Section 3.1, we consider two possible initialisations procedures: (i) each agent is assigned to a separate population (*heterogeneous start*); and (ii) all agents are assigned to the same population (*homogeneous start*).

Table 3 shows the highest fitness scores achieved with each approach. We could only find a significant difference in the *W7* setup (Mann-Whitney test, $p = 0.003$). In all other task setups, no differences were found ($p > 0.1$). We analysed the number of populations throughout evolution to discover the reason for the similarities in performance, see Table 4. As expected, there is a large difference in the number of populations in the initial generations (0–99th). As evolution progresses, however, the difference becomes smaller. After the 250th generation, there are no significant differences in the mean number of populations in the two approaches ($p > 0.1$). This result further reinforces that *Hyb-CCEA* can

Table 3: Highest fitness scores achieved with *Hyb-CCEA*, with a heterogeneous (*Het*) and homogeneous (*Hom*) initialisation.

	W5	E5	W7	E7	W10	E10
Het	2.56	2.56	3.71	2.49	3.25	2.33
Hom	2.56	2.57	3.40	2.48	3.07	2.38

Table 4: Mean number of populations at different stages of evolution.

Setup	Gen. 0–99		Gen. 100–249		Gen. 250–	
	Het	Hom	Het	Hom	Het	Hom
W5	3.47	2.22	3.23	3.14	3.61	3.38
E5	3.28	2.00	2.77	2.62	2.90	2.76
W7	4.75	2.33	3.85	3.46	3.87	3.55
E7	4.49	2.34	4.01	3.99	4.30	4.34
W10	5.90	2.44	4.07	3.63	4.18	3.73
E10	5.89	2.41	5.06	4.38	5.66	5.39

adapt the level of heterogeneity in the team to the task at hand, and is not significantly biased by the initial conditions.

5.4 Parameter Sensitivity

In this section, we study the sensitivity of *Hyb-CCEA* to the merge threshold T_D and the stability threshold T_G. The merge threshold dictates the maximum distance allowed between two mergeable populations. The distance between a population and itself is always 1 (see Section 3.2). A value $T_D = 1$ therefore means that two populations should cover exactly the same region of the behaviour space in order to be merged. The higher the threshold, the *easier* it is to merge two populations. The stability threshold T_G dictates the minimum age of a population: how many generations must pass before the population can be removed through a merge or a split. The stability threshold therefore controls the frequency of merges and splits in the evolutionary process. We evaluated the impact of these two parameters in two task setups, *W7* and *E7*, see Figure 8 for results. We evaluated T_D (left) with a fixed value of $T_G = 10$, and we evaluated T_G (right) with a value of $T_D = 1.25$.

The merge threshold T_D has a different impact on the performance of *Hyb-CCEA* in the two setups: in *W7*, there are no significant differences in the range $[1, 1.5]$ and there is a monotonic decrease in $[1.5, 2]$; in *E7*, there is a monotonic increase in the range $[1, 1.5]$, and no significant differences in $[1.5, 2]$. This is explained by the different nature of the two task variants: in *E7*, the existence of sub-teams is beneficial, while in *W7* there is no need of sub-teams, and more specialisations are required. Increasing the merge threshold facilitates the formation of sub-teams, but can hinder the evolution of subtly different specialisations. Considering the two setups, the optimal value of T_D is between $[1.25, 1.5]$.

The stability threshold only has a significant impact in the *E7* setup (Kruskal-Wallis test, $p = 0.50$ in *W7*, and $p = 0.004$ in *E7*). In the *E7* setup, higher values of T_G are preferred, although there are no significant differences in the range $[20, 60]$. Our results are insufficient to draw definite conclusions regarding the stability threshold. Although *Hyb-CCEA* is not overly sensitive to this parameter in our experiments, we are assessing the impact of T_G in different tasks in ongoing work.

Unlike the merge and split thresholds, the agent behaviour characterisation (see Section 3.2) is task-specific. The characterisation should have sufficient behaviour features to distinguish agents with evidently different behaviours. A very detailed characterisation should not be harmful to the algorithm, as the distance between populations is given by a relative measure. In ongoing work, we are empirically evaluating the impact of the behaviour characterisation in *Hyb-CCEA*, and experimenting with task-independent characterisations [6, 8].

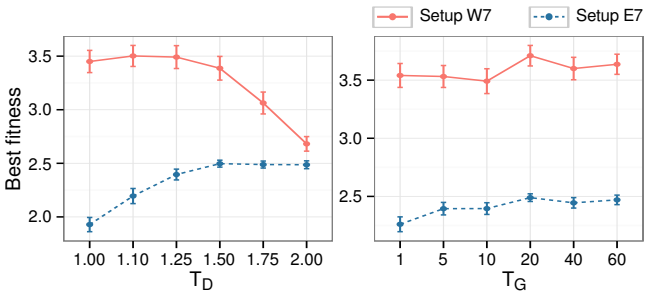

Figure 8: Highest fitness scores achieved at the end of the evolutionary runs, with different values for the merge threshold T_D and stability threshold T_G. Each point is the average of 30 evolutionary runs.

6. CONCLUSION

We proposed *Hyb-CCEA*, a novel method for the cooperative coevolution of partially heterogeneous (*hybrid*) teams. In *Hyb-CCEA*, each population can be assigned to an homogeneous sub-team, which departs from traditional CCEA applications where each agent evolves in a separate population. Both agent controllers and the team composition are under evolutionary control. The level of heterogeneity in the system is therefore dynamic: (i) heterogeneity can be decreased by *merging* two populations into a new one, forming an homogeneous sub-team with the respective agents, and (ii) heterogeneity can be increased by *splitting* one population into two new ones, where each is assigned to a different set of agents. We proposed a merge procedure that merges two populations if they are converging to similar regions of the agent behaviour space, and a stochastic split procedure based on populations' age.

We compared *Hyb-CCEA* with a fully heterogeneous CCEA in a simulated herding task. We used multiple task setups, varying the number of agents from 5 to 10, and varying the number and type of specialisations required. Our results were generally consistent across all tasks: *Hyb-CCEA* could reach good solutions for all tasks in significantly fewer evaluations, and it could also achieve significantly higher fitness scores in the end. We showed that *Hyb-CCEA* can adapt the level of heterogeneity in the system to the task at hand, and that it is especially effective in task variants that require the formation of sub-teams.

We studied the main parameters of *Hyb-CCEA*: the initialisation procedure, the merge threshold, which dictates the distance threshold for merging, and the stability threshold, which controls the minimum age of populations. Our experiments showed that *Hyb-CCEA* is not overly sensitive to variations in the parameter values. Additionally, the same parameter values were used across all the considered task setups, yielding promising results.

Our experiments revealed that the performance gains of *Hyb-CCEA* become higher as the number of agents increases. In ongoing work, we are evaluating the proposed approach in additional tasks, requiring a higher number of agents. Although the merge and split procedures proposed in this paper worked relatively well and did not disrupt the evolutionary process, we are currently studying alternative implementations of these procedures.

To the best of our knowledge, *Hyb-CCEA* is the first cooperative coevolution algorithm that allows for the emergence of partially heterogeneous teams. *Hyb-CCEA* improves the scalability of cooperative coevolution without sacrificing solution quality. Our study opens new interesting avenues of research, since it brings together the emergence of team compositions and the advantages of cooperative coevolution.

Acknowledgements This research is supported by Fundação para a Ciência e Tecnologia (FCT), with grant SFRH/BD/89095/2012 and projects UID/EEA/50008/2013, UID/Multi/04046/2013, and EXPL/EEI-AUT/0329/2013.

REFERENCES

[1] A. Agogino and K. Tumer. Efficient evaluation functions for evolving coordination. *Evolutionary Computation*, 16(2):257–288, 2008.

[2] J. C. Bongard. The legion system: A novel approach to evolving heterogeneity for collective problem solving. In *Genetic Programming*, volume 1802 of *LNCS*, pages 16–28. Springer, 2000.

[3] M. Brambilla, E. Ferrante, M. Birattari, and M. Dorigo. Swarm robotics: a review from the swarm engineering perspective. *Swarm Intelligence*, 7(1):1–41, 2013.

[4] A. Campbell and A. S. Wu. Multi-agent role allocation: issues, approaches, and multiple perspectives. *Autonomous Agents & Multi-Agent Systems*, 22(2):317–355, 2011.

[5] D. B. D'Ambrosio and K. O. Stanley. Generative encoding for multiagent learning. In *Genetic and Evolutionary Computation Conference (GECCO)*, pages 819–826. ACM Press, 2008.

[6] S. Doncieux and J.-B. Mouret. Behavioral diversity measures for evolutionary robotics. In *Congress on Evolutionary Computation (CEC)*, pages 1–8. IEEE Press, 2010.

[7] M. Dorigo, D. Floreano, L. Gambardella, F. Mondada, et al. Swarmanoid: A novel concept for the study of heterogeneous robotic swarms. *IEEE Robotics & Automation Magazine*, 20(4):60–71, 2013.

[8] J. Gomes and A. L. Christensen. Generic behaviour similarity measures for evolutionary swarm robotics. In *Genetic and Evolutionary Computation Conference (GECCO)*, pages 199–206. ACM Press, 2013.

[9] J. Gomes, P. Mariano, and A. L. Christensen. Avoiding convergence in cooperative coevolution with novelty search. In *International Conference on Autonomous Agents & Multiagent Systems (AAMAS)*, pages 1149–1156. IFAAMAS, 2014.

[10] J. Gomes, P. Mariano, and A. L. Christensen. Systematic derivation of behaviour characterisations in evolutionary robotics. In *International Conference on the Synthesis and Simulation of Living Systems (ALife)*, pages 212–219. MIT Press, 2014.

[11] A. Hara. Emergence of cooperative behavior using adg; automatically defined groups. In *Genetic and Evolutionary Computation Conference (GECCO)*, pages 1039–1046. Morgan Kaufmann, 1999.

[12] T. Jansen and R. P. Wiegand. Exploring the explorative advantage of the cooperative coevolutionary (1+1) EA. In *Genetic and Evolutionary Computation Conference (GECCO)*, volume 2723 of *LNCS*, pages 310–321. Springer, 2003.

[13] P. Lichocki, S. Wischmann, L. Keller, and D. Floreano. Evolving team compositions by agent swapping. *IEEE Transactions on Evolutionary Computation*, 17(2):282–298, 2013.

[14] S. Luke. Genetic programming produced competitive soccer softbot teams for RoboCup97. In *Genetic Programming*, pages 214–222. Morgan Kaufmann, 1998.

[15] S. Luke and L. Spector. Evolving teamwork and coordination with genetic programming. In *Genetic Programming*, pages 150–156. MIT Press, 1996.

[16] G. Nitschke. Behavioral heterogeneity, cooperation, and collective construction. In *Congress on Evolutionary Computation (CEC)*, pages 1–8. IEEE Press, 2012.

[17] G. S. Nitschke. *Neuro-evolution for emergent specialization in collective behavior systems*. PhD thesis, Vrije Universiteit Amsterdam, 2008.

[18] G. S. Nitschke, A. E. Eiben, and M. C. Schut. Evolving team behaviors with specialization. *Genetic Programming and Evolvable Machines*, 13(4):493–536, 2012.

[19] G. S. Nitschke, M. C. Schut, and A. E. Eiben. Collective neuro-evolution for evolving specialized sensor resolutions in a multi-rover task. *Evolutionary Intelligence*, 3(1):13–29, 2009.

[20] L. Panait. Theoretical convergence guarantees for cooperative coevolutionary algorithms. *Evolutionary Computation*, 18(4):581–615, 2010.

[21] L. Panait and S. Luke. Cooperative multi-agent learning: The state of the art. *Autonomous Agents & Multi-Agent Systems*, 11(3):387–434, 2005.

[22] M. A. Potter and K. A. D. Jong. Cooperative coevolution: An architecture for evolving coadapted subcomponents. *Evolutionary Computation*, 8(1):1–29, 2000.

[23] M. A. Potter, L. A. Meeden, and A. C. Schultz. Heterogeneity in the coevolved behaviors of mobile robots: The emergence of specialists. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1337–1343. Morgan Kaufmann, 2001.

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

[25] P. Stone and M. Veloso. Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots*, 8(3):345–383, 2000.

[26] V. Trianni, S. Nolfi, and M. Dorigo. Evolution, self-organization and swarm robotics. In *Swarm Intelligence*, Natural Computing Series, pages 163–191. Springer, 2008.

[27] M. Waibel, L. Keller, and D. Floreano. Genetic team composition and level of selection in the evolution of cooperation. *IEEE Transactions on Evolutionary Computation*, 13(3):648–660, 2009.

[28] R. P. Wiegand. *An Analysis of Cooperative Coevolutionary Algorithms*. PhD thesis, George Mason University, 2003.

[29] C. H. Yong and R. Miikkulainen. Coevolution of role-based cooperation in multiagent systems. *IEEE Transactions on Autonomous Mental Development*, 1(3):170–186, 2009.