Embodied Evolution versus Cooperative Coevolution in Multi-Robot Optimization: a practical comparison

SUPPLEMENTARY MATERIAL: EXPERIMENT DETAILS

P. Trueba
Integrated Group for Engineering Research
Universidade da Coruña
Spain
pedro.trueba@udc.es

A. Prieto
Integrated Group for Engineering Research
Universidade da Coruña
Spain
abprieto@udc.es

F. Bellas
Integrated Group for Engineering Research
Universidade da Coruña
Spain
francisco.bellas@udc.es

DETAILED ALGORITHM DESCRIPTION

WE describe the three main algorithms that have been used in the comparison, and several variations of them. First of all, a state-of-the-art CCEA approach, the Differential Evolution Cooperative Coevolution (DECC). Second, an encapsulated Embodied Evolution approach with two variations, and finally, a distributed Embodied Evolution approach. This way we can compare the two extreme options of EE, encapsulated and distributed, with a CCEA algorithm.

DECC

We have resorted to general CCEA bibliography to find the state-of-the-art approach in high-dimensional optimization. The selected algorithm has been the Differential Evolution Cooperative Coevolution (DECC), which is an adaptation of the DECC-G algorithm from Yang et al. [1]. The motivation of the DECC-G algorithm is to apply cooperative coevolution to decompose high dimensional and non-separable problems. In high dimensional problems, it is beneficial to apply cooperative coevolution for decomposing the problem in smaller problems but previous cooperative coevolution techniques failed when the problem is non-separable, because they do not account for the relations between the variables. The DECC-G algorithm change the decomposition strategy to solve non-separable problems. The proposed decomposition strategy is to make groups of interdependent variables that improve the optimization and uses an advanced evolutionary algorithm like Differential Evolution (DE) to optimize each component. Another innovation is that previous cooperative coevolutionary algorithms did not use a state of the art evolutionary technique like DE but instead simpler genetic algorithms. One algorithm that extends the DECC-G algorithm is the MLCC (Multilevel Cooperative Coevolutionary) algorithm, that adjust automatically the group size [2].

The DECC algorithm combines concepts from both approaches (grouping strategy and DE recombination operator) to apply it to the coordination of a team of robots. The algorithms proceed as follows:

- First of all, there is a vector of all the variables of the problem that will be divided in groups, with each group evolving in isolated instances of a differential evolution algorithm. The number of components depends on the grouping size. In each cycle, the first step of the algorithm is to permute the vector of variables. This step can be omitted and the groups will be assigned initially and remain fixed in the case of the variation named DECC-FG (DECC with fixed groups). Or it can be randomly assigned for each evolutionary generation as it is the case of the version shown in Algorithm 1 (DECC-RG).

- After the permutation, new populations of DE are created with the corresponding groups of variables and they are evolved for a number of evaluations. One cycle of the algorithm consists of evolving each of these components sequentially. Given that each component has an internal population, there are a lot of combinations but here the process is sequential, with an off-line evolution, evolving one group of variables each time.

- For each evaluation, the current variables are transferred to the robots and they try to perform the task for a number of iterations. After the evaluation, the environment is reset. The evaluation of a group of variables provides the global fitness of the team in the task.

- If the groups of variables are fixed and assigned exactly to the control parameters of one robot, the evaluation of the group and the robot private fitness in the task would coincide. This version of the algorithm using private fitness is referred here as DECC-FG-Private.

- After one component has evolved for the specified time, the best variables for this group are modified in the variable vector, and the next component will evolve with these new solutions.

- Finally, when all components had evolved, new groups are created and evolution of all components starts again until a number of cycles are completed.

Algorithm 1 DECC

```
$variables \leftarrow random()$ { robots*genomeSize }
$bestVariables \leftarrow variables$
while $cycles \leq maxCycles$ do
  create groups of variables { random or fixed groups}
  initialize DE populations with groups
  for all population in DE populations do
    while $evaluations < maxEvaluations$ do
      generate new individual in this population
      evaluate team with new individual
      $f_{new} \leftarrow$ fitness {Global or private}
      if $f_{new} > f_{ancestor}$ then
        Replace ancestor with new
        if globalFitness > bestGlobalFitness then
          $bestGlobalFitness \leftarrow globalFitness$
          $bestParameters \leftarrow team$
        end if
        $variables \leftarrow bestParameters$
      end if
    end while
  end for
end while
```

Encapsulated Embodied Evolution

As for the encapsulated version of EE algorithms, several implementations can be found [3][4], which basically follow the same operating principles with minor changes in the evaluation of individuals. To take part in this comparison, we have followed the operation scheme of those algorithms and implemented them with some structural changes in the evaluation in order to make the results comparable with those of DECC approaches. In other encapsulated algorithms, the time for evaluation is shared between the individuals of each population but in our encapsulated algorithm a high fitness controller will have more evaluations than a bad controller.

The encapsulated EE algorithm follows the pseudocode shown in Algorithm 2. Each robot has an internal DE population that evolve independent from each other. Because of this, multiple combinations of individuals are possible for forming teams. In our implementation, before starting any evaluation, a tournament is made in each robot to select the controller. After that, some of the populations are chosen to create new individuals. The team with these new individuals is evaluated for a fixed period of time. If the global evaluation with the new individuals is higher than previous global evaluations of their ancestor, the ancestor is replaced. Also, the individuals that formed the team use the new global fitness to update their fitness as an average of all the evaluations that include them. There is no migration between DE populations, so in this case the populations work as isolated islands. This operation sequence implies that this algorithm is synchronous and off-line, since the evaluations are fixed to a period of time.

Two main differences set encapsulated EE and DECC algorithms apart: on the one hand in this algorithm each robot corresponds always to the same DE population, with as many DE populations as robots and a group size that matches the number of parameters of the controller of one robot, unlike DECC-FG where the group is fixed but the group size could correspond to several robots. On the other hand, in DECC, only one group creates new individuals at a time, but in this algorithm, more than one population can create new individuals at each cycle since several populations are selected randomly for creating new individual at each cycle. This means that this algorithm should converge faster (higher exploitations, lower exploration) than the previous one because a larger portion of the variables are modified in each evaluation. Also, a high number of active populations is not beneficial for the stability of the evaluations, because with the tournament it can be considered that the robots whose populations are not active will perform well with one fit controller.

There is no migration between DE populations, so in this case the population work as isolated islands. This operational sequence implies that this algorithm is synchronous and off-line, since the evaluations are fixed to a period of time.

Algorithm 2 Encapsulated DE

```
for all robot in robot population do
  robot ⟵ DE population
end for
while $cycles \leq maxCycles$ do
  choose team with tournament {in each robot}
  choose randomly active populations
  generate individuals in active populations
  modify team with generated
  evaluate team
  $f_{new} \leftarrow$ globalFitness
  if $f_{new} > f_{ancestor}$ then
    replace ancestor with new
  end if
  update fitness of team members
end while
```

We also found interesting to implement an asynchronous version of the algorithm and include it in the comparison. In the asynchronous encapsulated algorithm, the evolution is on-line and the controllers are evaluated until they run out of energy instead of being evaluated for a fixed number of iterations. The new individuals are generated when the controller runs out of energy and the change of the controller only affect that robot, without resetting the environment or changing any other controller. Because the populations inside the robot are large and all individuals need to be evaluated, instead of creating a new individual every time the controller runs out of energy, according to a probability, a tournament is made to select the next controller for the corresponding robot.

Distributed Embodied Evolution

Among the existing EE algorithms found in the literature, three distributed implementations can be noted: mEDEA [5], PGTA [6], and ASiCo [7]. They have been applied with success to different multi-robot tasks requiring coordination and adaptation in real time. In a previous work [8], authors have developed and analyzed a canonical version of a distributed EE algorithm that generalizes

the operational principles of the three algorithms highlighted above. In order to capsulize this category, this canonical version will be used to represent it.
The canonical distributed embodied evolution (dEE) algorithm parametrizes the basic processes of the embodied evolution paradigm. It is not the scope of this work to analyze this parametrization but compare this paradigm versus other cooperative coevolutionary algorithms like the previously described. The pseudocode for the dEE algorithm is shown in Algorithm 3.

Algorithm 3 Canonical dEE algorithm

```
for all robot in population do
    $\vec{g} \leftarrow$ generate random genotype {Initialize}
end for
loop
    for all robot in population do
        Fitness assignment {Interaction with the task}
        if random < $P_{mating}$ then
            candidates $\leftarrow$ $findCandidatesForReproduction$
            $\vec{s} \leftarrow selectCandidates$ {$P_{elegibility}$}
            offspring $\leftarrow recombination(\vec{g}, \vec{s})$
        end if
        if random < $P_{replacement}$ then
            $\vec{g} \leftarrow$ offspring
            reset robot state
        end if
    end for
end loop
```

EXPERIMENTAL SETUP

To compare the algorithms, we have designed a multi-robot experiment with all the features established in the introduction for a collective task inspired in a real-world problem. The experiment requires to optimize a high-dimensional search space which is dynamic, decentralized and with a strong interdependence between controllers, promoting the emergence of specialized individuals. Moreover, it generalizes other typical tasks solved in evolutionary robotics, like searching points of interest, area coverage and so on. Thus, it constitutes a prototypical application case in this domain, which can be very useful to analyze the practical response of these algorithms.

Multi-robot surveillance with location accuracy degradation

In this experiment, we deal with a fleet of Unnamed Aerial Vehicles (UAVs) that has to collectively survey an indoor scenario. To do it properly, the UAVs need to locate themselves accurately to share information with other robots coherently. It is well-known that the accuracy in the localization is a key aspect in navigation tasks with autonomous robots, so we propose to include it as a part of the problem that must be optimized.
The determination of the UAV positions will be performed using their IMU, the position of other UAVs in sight, and artificial landmarks that can be sensed using the onboard camera. The control of each of the UAVs will be obtained by evolution using the different algorithms presented above, that are in charge of coordinating the UAVs in the scenario in order to increase the accuracy of the fleet location, and consequently, the speed at which a new point of interest is reached.

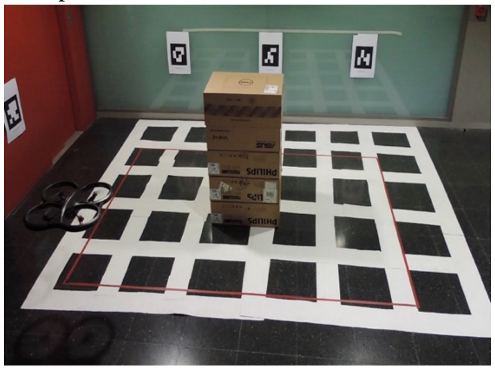

Figure 1. Parrot ARDrone 2.0 and the visual fiducial markers

The experimental setup has been defined in simulation, but based on a real indoor gathering task performed by real UAVs. The specific UAV that has been modeled is the Parrot ARDrone 2.0 (displayed in Figure 1). The most important aspect of the simulation is the model of the response of the location sensors when a certain maneuver is carried out. Regarding the artificial landmarks, we have used visual fiducial markers, with a model that represents the AprilTags created by The APRIL Robotics Laboratory at the University of Michigan. Therefore, the location estimation provided by the markers is based on a real accuracy model which was produced in our laboratory using an ARDrone 2.0 and 40 cm long AprilTags, and that can be formulated as a function $\boldsymbol{\varphi}$ which relates the variance of the estimation $Var(\vec{p}_{drone})$ with the relative distance ($\|\vec{p}_{drone} - \vec{p}_{tag}\|$) and orientation ($yaw_{drone:tag}$) between camera and tag:

$$Var(\vec{p}_{drone}) = \boldsymbol{\varphi}(\|\vec{p}_{drone} - \vec{p}_{tag}\|, yaw_{drone:tag})$$

These tags (*permanent tags*) provide an absolute and potentially accurate position estimation, which does not degrade with time. In order to improve the performance of the navigation by improving the accuracy of the UAVs, the same type of tags is also attached to the body of the quadcopters (*mobile tags*), which will make up a hybrid location sensor. Therefore, the detection provided by a mobile tag still constitutes a direct location estimation but, unlike in the case of permanent tags, their accuracy degrades as the accuracy of the carrier of the tag decreases. The accuracy degradation is, however, proportional to the velocity of the UAV. Therefore, the mobile tags should tend to stay still to be more efficient as accuracy exchangers. The use of this type of mobile tags leads to accuracy becoming a resource that UAVs can get and share to be able to slow down its degradation, and therefore, to accomplish their main task more efficiently. It could also allow some of the UAVs not to require visits to static tags, which are frequently non optimally located, in order to improve their location accuracy, being 'nourished' by mobile tags.

The scenario was discretized to reduce computational effort as a 768 x 768 (square length units) non-toroidal square arena, which is provided with 4 fixed tags placed randomly. Figure 2 shows a schematic representation of a portion of the arena. Each UAV is associated to both a real and an estimated position. The former is shown with a solid color in the simulation and the later with a softened shadow of the UAV. The UAV has no idea of the real position, this is just an externally obtained value for display purposes. The further the distance between those positions, the higher the location estimation error and the lower the exploration level. Table 1 contains the specific parameters that define the scenario.

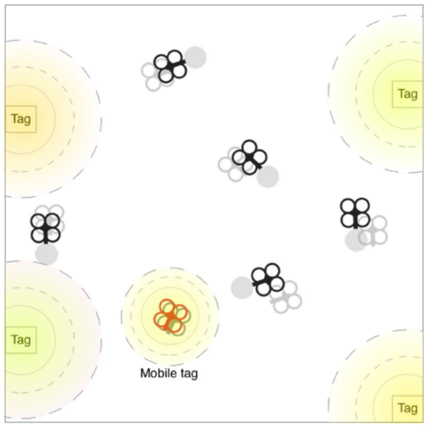

Figure 2. Graphical representation of a portion of the scenario. The distance to each own shadow represents the current estimation error for the UAV. The red UAV represents a mobile tag. The circles around the tags are the different levels of accuracy provided by the tag.

Table 1. Design parameters of the simulated scenario

Parameter	Value
Side of the arena (L)	768
Total area	L^2
Fixed tags detection range	L/4
Mobile tags detection range	L/16
Max velocity (V_{max})	L/50

The final objective of the surveillance task is for the fleet of UAVs to continuously cover the maximum possible area. In order to perform the search of unexplored areas, the UAVs can keep record of the areas they have already explored. This is stored in an *'exploration map'* carried by each UAV and that can also be shared with others when they meet. The exchange of this information allows the task to be solved cooperatively since it enables the distribution of the search among the group of UAVs. However, since the estimation of one's position has a varying accuracy, the updating of the exploration map has to take that into account. The exploration of a cell is modeled as an *exploration probability* (P_{ex}), which indicates the probability that a certain cell has of having been explored. Subsequent UAVs must decide whether or not to *re-explore* that cell based on the guaranteed exploration probability P_{ex}. Therefore, the collaborative exploration map is the only information that a UAV gets from the scenario about the surveillance process.

Algorithm configuration

The UAVs are defined by their real spatial location $[x_r, y_r]$ (they are supposed to fly at a constant height $[z_r]$) and their estimated location $[x_e, y_e]$ (their orientation, given by the yaw angle, is defined by the direction of movement so it is not required explicitly). They also display an estimated accuracy (a_e), which is described by the standard deviation of their estimated spatial coordinates.

Three types of *sensors* are considered, namely, the *explorability* sensor (E), the *available accuracy* (A_a) and the *current exploration level* of the UAV (D_{ex}). The first two sensors are associated to a certain portion of the arena (a group of neighboring cells), where the potential increase of exploration that the UAV can provide if it visits it is calculated (E) together with an estimation of the available accuracy (A_a) as a combination of the distance and accuracy of the closest tag. Several groups of neighboring cells of different sizes are considered (1, 3x3, 6x6 and 12x12, up to 36 groups of cells in total) and their sensed parameters are calculated. Those values together with the current exploration level of the UAV (the exploration probability it will set if it visits a cell) will constitute the inputs to the control unit. As *actuators*, the UAVs can execute one of five predefined behaviors: *near exploration*, *distant exploration*, *accuracy sharing*, *increase accuracy* or *tag avoidance*. Thus, only their motion is controlled by the control unit, since the rest of actions are performed automatically based on the relative position between the UAVs and between UAVs and tags.The *control unit* is in charge of defining the actuation of the UAVs, which is, in a nutshell, their motion. It defines its operation based on a set of parameters (8) that will be used as *weighting*, *threshold* and *duration* coefficients to define the criteria to select the target area of the arena from a set of neighboring candidates.

The global fitness (F) is defined by the task as the average of the exploration level (e_i) of each area of the arena. The higher the global fitness the faster and more exhaustive the exploration the fleet is performing. Being M the number of areas in which the arena is divided:

$$F = \frac{\sum_{i=1}^{M\ areas} e_i}{M}$$

The private fitness (f_j) defined in this scenario for each of the individuals is calculated using the sum of the increment in exploration it provides to each cell it visits $\left(\Delta e_i^j\right)$. However, we can find individuals (or actions of individuals) that collaborate to the success of the global aim but which do not explore the scenario. Those are the individuals that provide location accuracy (s_i^j, shared accuracy from the ith individual to the jth) to the rest of the UAVs. Therefore, the reward assigned for the exploration to one individual will be shared with the individuals that provided the accuracy used to perform that exploration (if that was the case) according to a *trade fee* (t_f). This fee will set the percentage of the exploration level achieved that will be returned to the 'accuracy provider', and of course subtracted from the explorer, as a function of the level of accuracy exchanged. Using this fitness-sharing scheme we allow an adjustable credit assignment policy:

$$f_j = \sum_{i=0}^{M\ areas} (\Delta e_i^j - \sum_{k=0}^{N\ mavs} (s_k^j t_f)) + \sum_{k=0}^{N\ mavs} (s_j^k t_f)$$

Table 2: Parameterization of algorithms

Parameter	Value
DECC	
DE populations	40
Group size	4
DE: population size	10
DE: F	0.5
DE: CR	0.5
Evaluations in cycle	100
Evaluation time	1000
Iterations in cycle	$4x10^6$
Cycles	10
Total iterations	$4x10^7$
Encapsulated EE	
DE populations	40
DE: population size	10
DE: F	0.5
DE: CR	0.5
Evaluation time	1000
Total iterations	$4x10^7$
Canonical dEE	
Maximum lifetime	1000
Maturity time	1
Selection criteria	F
Tournament size	40
Local search probability	0.99
Mediocrity coefficient	0.01
Total iterations	$4x10^7$

The task is performed by 40 robots controlled with a feedforward network encoded in a genotype of 4 real values. In the off-line algorithms (DECC and encapsulated) each team is evaluated during 1000 steps and the environment is reset for each evaluation. Specific parameters for the three types of algorithms are shown in Table 2, which have been adjusted to produce the best solutions according to the authors' recommendations.

RESULTS

In this section, a comparison regarding the performance of all algorithms is presented. This performance is evaluated with respect to: the global fitness obtained by each algorithm after a limited number of iterations and the time required to obtain a predefined satisfactory fitness level. The limit of iterations is set to a very high number in order to let all the algorithms enough time to converge to their best solutions. To evaluate this fitness, it should be noted that these results compare algorithms that evolve on-line with algorithms that evolve off-line. Given that the DECC algorithm and its variants and the encapsulated algorithm evolve off-line, they need to reset the scenario in order to evaluate new teams that could perform very differently in the task. For this reason, instead of an instantaneous measure of fitness in the task that could fluctuate between evaluations, a more representative estimation of fitness is given by the best fitness achieved by the best team found. The global fitness employed represents the exploration level of the environment, that ranges, theoretically, from 0 (totally unexplored environment) to 8 (totally explored environment). Practically, with the defined scenario dimensions, number of robots, and other task parameters, the maximum attainable exploration level is around 80-90% of the theoretical one (around 6-7).

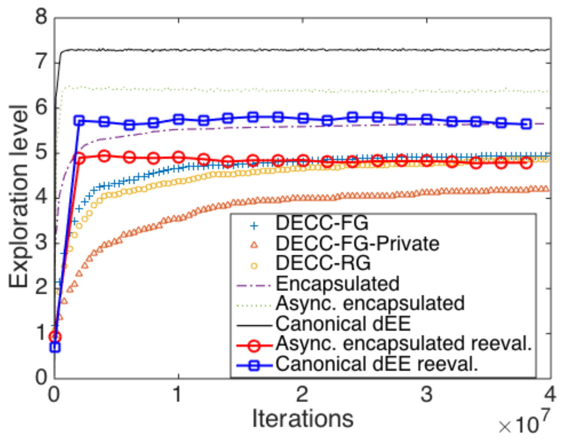

Figure 3: Performance comparison of the algorithms in the exploration task

Fig. 3 contains the average exploration level obtained for the six algorithms throughout iterations. Each algorithm was executed 20 times, each of them implying about 4 hours in an i7 4770s processor, and the resulting exploration level was averaged between them. The most significant result that can be extracted from this figure is the time required to reach stable solutions, which is much lower in the on-line algorithms (asynchronous encapsulated and canonical dEE), as expected. They improve their exploration level quickly, and in around 5 million iterations, which constitutes the 12,5% of the total iterations, they reach a stable range that continues until the final iteration. In contrast, the off-line algorithms (encapsulated and DECC variants) require almost half of the total iterations to reach a stable solution. The encapsulated algorithm provides the best results in the off-line approaches, while the DECC-FG obtains the best performance of the DECC variants.

In order to analyse the exploration level value, it must be pointed out that on-line algorithms cannot be directly compared with off-line ones, because the scenario is not restarted. To solve it, the best teams obtained with the on-line algorithms are stored at a fixed number of iterations for later re-evaluation, like in the case of off-line approaches. The exploration level obtained by the on-line approaches considering this re-evaluation is displayed with the pointed lines in Fig. 3 (the two bottom symbols in the figure legend). As it can be seen, now the on-line algorithms reach a stable level very similar to the off-line encapsulated version, but still better than the DECC variants.

In order to clarify the performance provided by the different algorithms, an analysis of the behaviour of the best populations obtained was performed. To do it, we have introduced a new metric called *behavioural heterogeneity* which will estimate the diversity of behaviours that each individual activates during its lifetime, considering the five possible behaviours described in section 3.2. A high heterogeneity value represents an individual which continuously switches its behaviour, while a low heterogeneity (zero) implies that only one behaviour is activated. High values of heterogeneity are associated to low degree of specialization in the population and conversely low heterogeneity is produced by a high degree of specialization.

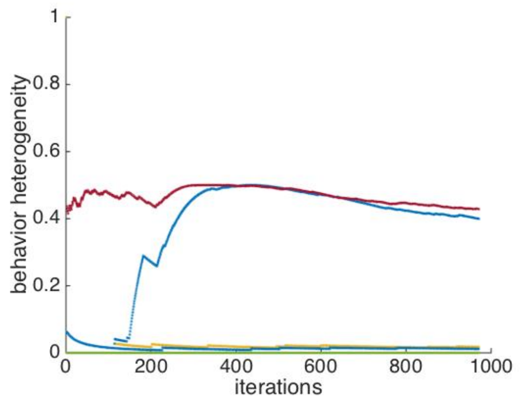

Figure 4: Behavior heterogeneity for the dEE algorithm

In Fig. 4, the behavioural heterogeneity provided by the dEE algorithm is shown. It is calculated by executing the final configuration of controllers obtained after evolution during 1000 iterations. There are 40 lines corresponding to the 40 robots, but most part of the lines (38 out of 40) are just above the horizontal line of heterogeneity 0. As mentioned, heterogeneity 0 represents individuals that had the same behaviour for all the iterations. Only two robots show heterogeneity equal to 0.5 which corresponds to switching between two different behaviours during its lifetime. In Fig. 5, the behavioural heterogeneity for the asynchronous encapsulated algorithm is displayed. The axes are the same as the previous figure and most of the lines are again over heterogeneity 0. In this case, 36 out of 40. Finally, the results provided by the off-line encapsulated algorithm can be seen in Fig. 6. Still most of the lines tend to be close to zero but the effect is much weaker than in the previous cases. For the rest of the off-line algorithms, the tendency to heterogeneity zero is almost non-noticeable so they are not displayed.

Therefore, as noticed during the realization of the tests, there is a clear relation between the performance and the specialization of the population in this type of multi-robot optimization problems. Thus, the on-line algorithms, and specially the dEE, exploit their simplicity in this domain in order to specialize individuals in simple tasks so evolution is much faster than in off-line approaches, where the several evolution processes that are executed in parallel makes the optimization slower and more complex.

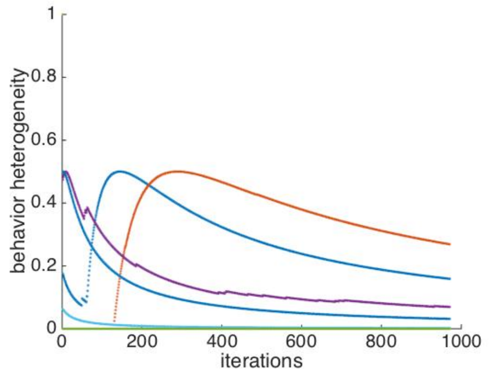

Figure 5: Behavior heterogeneity for the asynchronous encapsulated algorithm

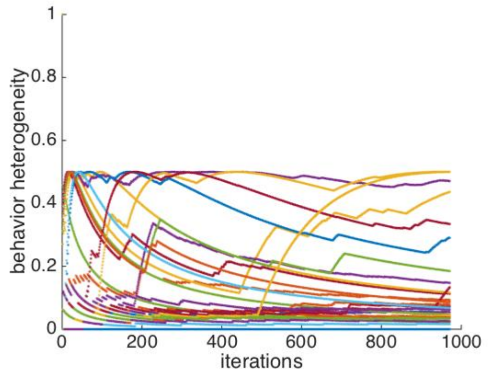

Figure 6: Behavior heterogeneity for the encapsulated algorithm

DETAILED CONCLUSIONS

This work has provided some insights into the problem of obtaining and evolutionary approach that can optimize wide-range multi-robot problems. We have shown through a high-dimensional and realistic experiment that Embodied Evolution approaches have a high potential in this domain, mainly due to their design principles: on-line, on-board evolution with heterogeneous genotypes. These three features allow them to obtain specialized individuals that can solve complex tasks without resorting to high time-consuming evaluations, which results in a much lower number of iterations to reach the optimal solution. In the future, we will continue exploring the application of Embodied Evolution approaches to general multi-robot problems and pursuing a better understanding of its operational mechanisms to improve its performance

REFERENCES TO ALGORITHM DESCRIPTION

[1] Z. Yang, K. Tang, X. Yao, Large scale evolutionary optimization using cooperative coevolution. 2008. *Informat. Sciences* 178-15, Elsevier, 2985-2999

[2] Z. Yang, K. Tang and Xin Yao. 2008. Multilevel cooperative coevolution for large scale optimization. *IEEE Conf on Evolutionary Computation,* 1663-1670

[3] S. Elfwing, E. Uchibe, K. Doya, H. Christensen. 2011. Darwinian embodied evolution of the learning ability for survival. *Adaptive Behavior*, 19(2), SAGE Press, 101–120

[4] E. Haasdijk, A.E. Eiben, G. Karafotias. 2010. On-line evolution of robot controllers by an encapsulated evolution strategy. Proceedings IEEE CEC2010, IEEE Press, 1-7

[5] N. Bredeche, J.M. Montanier, W. Liu, A. Winfield. 2012. Environment-driven Distributed Evolutionary Adaptation in a Population of Autonomous Robotic Agents, *Mathematical and Computational Modelling of Dynamical Systems* 18, 1, 101-129

[6] R. Watson, S. Ficici, J. Pollack. 2002. Embodied evolution: Distributing an evolutionary algorithm in a population of robots, *Robotics and Autonomous Systems*, 39(1), Elsevier, 1-18.

[7] A. Prieto, J.A. Becerra, F. Bellas, R.J. Duro. 2010. Open-ended Evolution as a means to Self-Organize Heterogeneous Multi-Robot Systems in Real Time, *Robotics and Autonomous Systems*, vol. 58, 1282-1291

[8] A. Prieto, F. Bellas, P. Trueba, R.J. Duro. 2015. Towards the standardization of distributed Embodied Evolution, *Information Sciences*, vol 312, 55-77