Fine-Grained Cooperative Coevolution in a Single Population: Between Evolution and Swarm Intelligence

Evelyne Lutton, Shatha F. Al-Maliki, Jean Louchet, Alberto Tonda, Franck Vidal

Fine-Grained Cooperative Coevolution in a Single Population: Between Evolution and Swarm Intelligence

É. Lutton[1], S. Al-Maliki[2,3], J. Louchet[4], A. Tonda[1], and F. P. Vidal[2]

[1] UMR 518 MIA, INRAE, Palaiseau, France
evelyne.lutton@inrae.fr,alberto.tonda@inrae.fr
[2] School of Computer Science and Electronic Engineering, Bangor University, UK
f.vidal@bangor.ac.uk shatha.f.almaliki@bangor.ac.uk
[3] Computer Science department, College of Science, Basrh University, Iraq
[4] EFREI, Villejuif, France, jean.louchet@gmail.com

Abstract. Particle Swarm Optimisation (PSO) and Evolutionary Algorithms (EAs) differ in various ways, in particular with respect to information sharing and diversity management, making their scopes of applications very diverse. Combining the advantages of both approaches is very attractive and has been successfully achieved through hybridisation. Another possible improvement, notably for addressing scalability issues, is cooperation. It has first been developed for co-evolution in EA techniques and it is now used in PSO. However, until now, attempts to make PSO cooperate have been based on multi-population schemes almost exclusively. The focus of this paper is set on single-population schemes, or fine-grained cooperation. By analogy with an evolutionary scheme that has long been proved effective, the fly algorithm (FA), we design and compare a cooperative PSO (coPSO), and a PSO-flavoured fly algorithm. Experiments run on a benchmark, the Lamp problem, show that fine-grained cooperation based on marginal fitness evaluations and steady-state schemes outperforms classical techniques when the dimension of the problem increases. These preliminary results highlight interesting future directions of research on fine-grained cooperation schemes, by combining features of PSO and FA.

1 Introduction

Swarm intelligence is a source of inspiration for many optimisation algorithms, for instance for Particle Swarm Optimisation (PSO) proposed by J. Kennedy and R. Eberhart in 1995 [22], Ant Colony Algorithms [16], Artificial Bee Colony Algorithms [20] or Bacterial Foragings [13]. The idea is to exploit the collective behaviour of a set of entities, the same way as natural populations (flocks of birds or ant colonies) search for food.

There is actually a proliferation of new techniques based on analogies to animal behaviour [38]. With respect to the ongoing debate about the originality and relevance of such a proliferation, we stress the fact that this contribution is

not proposing yet another novel optimisation methodology, but making a point on two established heuristics that date back over 20 years and might look similar at first glance.

PSO is based on social interactions. The emerging collective behaviour results from a balance between following a leader and following an individual focus, thanks to inter-individual communications [31]. This mechanism is different from Evolutionary Algorithms (EAs) that rely on genetic transmission and natural selection analogies (birth, death and inheritance within a population). An important difference between them is how they manage diversity and share information[5], making them best fitted to different optimisation tasks [19].

Among other desirable features, scalability is a major concern. A way to deal with it is co-evolution, which was first developed for EA techniques [33] and starts to be experimented for PSO [6]. There are two major existing co-evolution schemes: mono- and multi-population [30], but as far as we know, only multi-population schemes are used in PSO [18,42].

This study investigates the differences and commonalities between intra-population communication in PSO and cooperative-co-evolution [12] as implemented in the Fly Algorithm (FA) [26,41,3]. This paper is organised as follows. After a rapid overview of the state of the art for PSO and cooperative PSO (Section 2), mono-population cooperative co-evolution and FA (Section 3), we propose a mono-population cooperative PSO (coPSO) and a new operator for the FA in Section 4. These schemes are compared on a cooperative-coevolution benchmark, the Lamp test case [39] in Section 5. The discussion and conclusions are given in Section 6.

2 From PSO to cooperative PSO

Each entity of a PSO, called a particle, has a position in space and a velocity, that determines a random movement depending on the context. Velocities and positions are updated at each iteration using rules taking into account local and collective memories, mimicking respectively a cognitive and a social behaviour.

Similar to evolutionary techniques, the theoretical understanding of swarm intelligence is a formidable challenge: with very simple mechanisms, interactions of a large number of elements produce a nontrivial global dynamic. Besides experimental evidence that such a system is able to concentrate the population into optimal areas of a search space [35], theoretical results for convergence and convergence rates [31] exist and are based on simple PSO models. The parameter settings and the structure of the update rules clearly have a crucial influence on performance [37].

A canonical PSO can be described as follows [22]: each particle keeps track of its own best known position *pbest* and has also access at any time to the global swarm best known position *gbest*. An iteration loop is then implemented:

1. Particles are initialised with random positions and velocities.

[5] via inter-individual communications in PSO or genetic inheritance in EAs

2. Best known positions are computed (according to the function to be optimised): $pbest_i$ for each particle i and $gbest$ for the whole swarm.
3. For each particle i, velocity v_i and position x_i are then updated (in vector notation, valid for any dimension of the search space):
$$v_i(t+1) = \omega v_i(t) + \varphi_p r_p(pbest_i - x_i) + \varphi_g r_g(gbest - x_i) \quad (1)$$
$$x_i(t+1) = x_i(t) + v_i(t) \quad (2)$$
where r_p and r_g are random values uniformly distributed between 0 and 1, ω is the inertia weight, φ_p and φ_g are the cognitive and social learning factors.
4. The process is repeated from Step 2 until a stopping criterion is met (e.g. stagnation, predefined level of fitness, or max. number of iterations).

The most common scheme, also called *gbest* **strategy**, corresponds to "fully informed" particles aware of the state of the whole population. In another important trend, called *lbest* **strategy**, each particle may only access local information [31]. The update rule is the same, except that in equation (1) *lbest*, a best *local* position, is used instead of *gbest*. This is more time-consuming as the neighbours of each particle (according to a given topology) have to be identified. This *lbest* scheme allows various subtleties to preserve diversity; neighbourhood topology has a strong influence on the performance of the algorithms [23]. Topology may vary: the neighbourhood can be gradually enlarged according to a topological distance or a graph hierarchy, sometimes using adaptive strategies [31]. The *lbest* scheme is particularly useful in parallel implementations when communication between processors is limited [42]. However, it may cause trouble with high dimensional search spaces, as it relies on a distance measure which may become computationally expensive with large swarms, besides the fact that distance functions get less useful in high dimension spaces [27]. In this paper, we will focus on *gbest* strategies only.

Diversity is an important issue in PSO, to avoid premature convergence. For instance, dispersion and collision-avoiding mechanisms or repulsion mechanisms have been proposed [42]. It has to be noted that multi-population approaches have been developed for improving the management of diversity.

Cooperative PSO and multi-swarm models[6] have been developed for different purposes: to improve diversity [45,29], track multiple optima in multimodal or dynamic multimodal landscapes [31,11], address multi-objective problems [42], perform dynamic optimisation using adaptive strategies [8], handle constrained optimisation [36], or deal with large search spaces, by explicitly splitting the problem into interdependent sub-problems with smaller dimensions [42].

Bergh and Engelbrecht [6] were the first to use a cooperative scheme, in the style of Potter and De Jong [33,14] with separate sub-populations. Cooperation comes from the exchange of information between sub-populations, to build a composite fitness in the high-dimension problem. Usually the *gbest* particles of other sub-swarms are used to evaluate the particles of a sub-swarm.

[6] [18] defines cooperative search for any method as strategies that have several search modules running and exchanging information to improve search capability.

Fine tuning these algorithms is difficult [32], the choice of information to be exchanged and the synchronisation strategies deeply affect performance. It has been observed that "Increasing the number of cooperating swarms helps in improving the performance up to a certain limit, after which, the solution starts to deteriorate" [18].

Note that cooperative PSO developed until now corresponds to what we may call coarse-grained cooperation, *i.e.* the swarms or sub-swarms are explicitly separated: cooperation occurs at swarm-, not particle-level[7].

3 Fine grained cooperative co-Evolution

Co-evolution is an extension of standard EAs [30] that "distributes" the encoding of a solution onto several individuals. As a consequence the fitness of each individual depends on other individuals. An early example of this technique is the "Michigan approach" [44] for classifier systems, in which a single population of individuals, each being a rule, is evolved to collectively achieve a given task (rule-based machine learning). Another pioneering work is the multi-population approach of Potter and De Jong [33], later transferred to the PSO model. Co-evolution has actually been structured and exploited in optimisation in quite different ways, according the interacting behaviour, competitive versus cooperative [12,14,43,9,5,40] or the granularity of interaction: a single population of interbreeding individuals versus multiple interacting populations [30].

Various versions of **fine grained single-population cooperation** have been proposed: "Parisian Evolution"[12,17] in 2000 and more recently "Kaizen programming"[15], "FFX"[28] or "ϵ-lexicase survival"[24]. In [12], all individuals share the same representation, can exchange genetic material thanks to genetic operators and evolve together inside a single population. The EA loop then embeds an additional step at each generation for aggregating individuals to build a solution, evaluate it and distribute rewards to individuals. The idea is to exploit the evolution mechanism in a more parsimonious manner: where a traditional Evolutionary Algorithm (EA) only keeps the best individual as an optimum solution at the end of the evolution (forgetting all precious information gathered by the population during its exploration of the search space), a Parisian approach tries to capitalise the full potential of an evolved population. It possess all usual features (e.g. mutation, crossover, and selection), but with two possible levels of fitness: **a local fitness** to assess the performance of a single individual (partial evaluation or local information) and **a global fitness** to assess the collective performance of the whole population. Maintaining diversity helps avoid degenerate solutions, e.g. when individuals gather in

[7] However, an application to the generation of improvised music [7] implements both types of cooperation, coarse and fine grained (this is not quite an optimisation, but rather an exploration task). It was performed with multi-swarms: each particle being a note (loudness, pulse and pitch of a MIDI event), each swarm a voice or instrument, and the whole system being considered as an improvising ensemble. Coherence is reached by self-organisation of particles and swarms.

only a few areas of the search space. Finally, a solution is built from a collation of individuals (sometimes with the concatenation of whole population). The way the fitness functions are constructed and the solution is extracted, are of course problem-dependent. Parisian Evolution has been successfully applied to various optimisation problems, such as text-mining [25], hand gesture recognition [21], complex interaction modelling in industrial agrifood processes [4,5], imaging problems such as computer stereo vision in robotics [26], tomography reconstruction in medical physics [2], and computer art [1].

A typical fine-grained cooperation is the Fly Algorithm (FA) [26]. First designed for stereovision applications, the Fly algorithm evolves a population of individuals called "flies". It uses an "inverse problem" approach where conventional approaches to stereovision use primitive extraction, pattern matching and calculation of disparities [26]. In the original version, a fly is defined as a 3-D point (x, y, z). A population of flies is initialised in the field of view common to at least two cameras, then evolved using a classical Evolutionary Strategy, guided by the flies' fitness values. The solution is given by the whole population (or a subset of the population), concentrated on the visible surfaces of the objects in the scene [10]. The fitness of a fly is a measurement of the consistency of its projections on the cameras. Classical operators – mutation, optional CMX crossover, immigration (introducing brand new flies) and tournament selection – are most commonly used.

4 Fine-grained optimisation based on PSO and FA

Particle Swarm Optimisation versus Fly Algoritm
Besides the narrative attached to each scheme (communications and social behaviour versus genealogical features transmission and selection mechanisms), PSO and FA share obvious features, and a parallel can be drawn between flies mutations and particle movements, but this actually leads to a different balance between diversification and intensification [19]. In particular, selection is not used in PSO, although it is an explicit intensification mechanism. Additionally, diversity preservation mechanisms are more explicit and tunable in FA, with the help of an "immigration" operator that introduces a proportion of purely random flies in the current population. We propose hereafter two different lines for mutual cross-fertilisation (i) implementing the PSO algorithm using the Parisian approach, and (ii) introducing the same information sharing mechanism as in PSO into the FA.

A cooperative PSO: coPSO
A Cooperative Particle Swarm Optimisation (coPSO), in terms of fine grained approach, consists in evolving, within a single swarm, particles that carry only a small part of a solution. At each iteration of the algorithm it is necessary to aggregate the particles of the swarm (or a selected part of it) to build the problem solution. As for FA, there are now two levels of objective functions, an optional global one computed on the whole swarm and a local one computed for each particle. The local fitness function is used to update *pbest*. Due to the

distributed nature of the approach, the social learning factor (φ_g in Eq. 1) is set to 0 as it makes no sense to follow the global best particle (*gbest*). Eq. 2 remains the same. In the experiments below, a marginal fitness[8] is used at the local level.

FA as a Swarm: SFA

To introduce a "PSO-like" information sharing mechanism within a FA, we built an additional operator, the *genealogical mutation*. The idea is, for each individual, to keep track of the best of its ancestors, according to the genealogy due to the genetic operators. Additionally, an extra vector similar to the velocity in PSO is attached to each fly. When a genealogical mutation is triggered, the velocity and position of offspring are updated using Equations 1 and 2. For the same reason as above, φ_g is set to zero. Note that this operator tends to focus the search of a fly into the direction of its *pbest*. However, it may be too restrictive (i) at the start of the optimisation when no knowledge is available, and (ii) at the end of optimisation when the result needs to be refined. This is why an an adaptive mutation scheme has been built.

Adaptive mutation

The adaptive *genetic bi-operator*, concurrently assesses two different genetic operators (here Gaussian mutation and genealogical mutation) so that the most successful operator in generating good offspring is favoured. Both operators are initially given an equal probability of occurrence. Their success rates are checked at regular intervals to adjust their probabilities. The update rule is multiplicative as for the famous $1/5^{\text{th}}$ rule [34].

Each operator has i) a counter to keep track of how many times it has been applied and ii) an accumulator that keeps track of how many times it has been successful. This accumulator is incremented if the marginal fitness of the newly created fly is positive, decremented if negative. The success rate of an operator is its accumulator divided by its counter. The probability of the most successful operator over the last period is increased at the expense of the other one. The probabilities are then clamped in the range 10%-90% to make sure that the least successful operator retains a chance to be picked up.

5 Experimental analysis on a toy problem

A toy problem for cooperative-coevolution: the Lamps

There are few benchmarks designed for cooperative co-evolutionary algorithms. *The Lamps* [39] is one of the toy problems available: the basic premise is to optimally place a set of circles (*lamps*) of given radius, so that they completely cover a square field. The fitness function rewards each lamp separately, and also provides a global reward that depends on the overall placement of all lamps. While each single lamp can be optimally placed on the square field, so that it

[8] Positive or negative contribution of the individual to the global fitness, *i.e.* the difference between the fitness of the population, when complete or deprived from this particular individual. This concept has been successfully used in various applications, see for instance [2]. In the absence of additional information at the local level for building a specific "local fitness", marginal fitness is a convenient option.

lits as much area as possible, it is interesting to notice that sometimes individual lamps with sub-optimal positions (e.g. part of their area falls outside of the field) can significantly improve the global reward (see Fig. 1). This simple toy problem only has one parameter, the ratio between the radius of a circle/lamp and the side of the square field (*i.e.* the ratio between the surface of a lamp and the surface of the field), $problem\_size = \frac{area\_room}{area\_lamp}$. With higher parameter values, more lamps are needed with more placement possibilities, making the benchmark more challenging. A further difficulty can be added by introducing penalties for overlapping lamps.

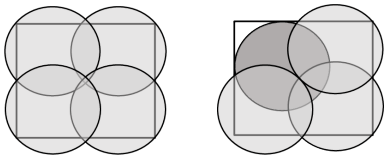

Fig. 1. Arrangement of a set of four lamps to enlighten a square field. (**left**) The lamps completely cover the square field, but part of their own area is outside of the square itself. (**right**) One of the lamps is now completely inside the square, but the global solution is unable to completely cover the square.

$$fitness = \frac{area\_enlightened}{total\_area} - W.\frac{area\_overlap}{total\_area} \tag{3}$$

The fitness of a candidate involves the total area enlightened and the number of lamps used. A weight W sets the balance with the overlapping term, see eq. (3). Best solutions maximise the illuminated area whilst minimising the number of lamps to cover the whole area. Tonda *et al.* showed that traditional approaches based on genetic operators are competitive when the search space is relatively small, *i.e.*, for Lamps problem size less than 10 [39]. For more complex problems, the Parisian approach outperformed the other algorithms tested.

Experimental setup

The Lamps problem with increasing sizes (3, 5, 10, 20, 100, and 500) has been used for benchmarking the scalability of six algorithms:

- A traditional PSO with no algorithmic enhancement, as a baseline for comparison (labelled PSO in the tables and figures below);
- The coPSO algorithm (labelled coPSO in the tables and figures);
- A steady state FA with marginal fitness, threshold selection, varying population size using mitosis and slaughtering/culling, 30% of immigration and 70% of Gaussian mutation (labelled FA);
- A steady state FA as above but with 30% of immigration, 35% of Gaussian mutation, and 35% of Genealogical mutation (labelled SFA35);
- As above but with 30% of immigration, and 70% of Genealogical mutation (labelled SFA70);
- As above but with 30% of immigration, and 70% of genetic bi-operator with both Gaussian and Genealogical mutation (labelled SFA-bi operator).

The lamp radius is 8 and $W = 1$ (Eq. 3) to match the value initially used in [39]. Algorithms 1 and 2 show the skeleton of FA and coPSO implementations, displayed side-by-side to highlight similarities and differences. The structure of the algorithms is fairly similar but coPSO lacks natural selection for killing and breeding. The mutation in FA and the position update in coPSO are similar in the sense that they both move an individual or particle from its current position.

```
          // Read problem specific data
                    // Set the algorithm
Initialisation
// Create the initial population of n
   individuals
repeat n times
    Create a fly at a random position in
      the search space;
    Add the fly to the population;
    Add the fly's contribution to the
      population's fitness;
end
Compute the global fitness;
repeat                  // Optimisation loop
    repeat n times
        repeat          // Select a bad fly
            i ← Random(0, n − 1);
            MF(i) ← Marginal fitness of
              Fly i;
        until MF(i) ≤ 0;
        Remove Fly(i)'s contribution
          from the population's;
        Compute the global fitness;
        Select genetic operator;
        if Genetic operator is
          immigration then
            Replace Fly(i) with a
              random fly in the search
              space;
        else            // Mutation is used
            repeat // Select a good fly
                j ← Random(0, n − 1);
                MF(j) ← Marginal
                  fitness of Fly j;
            until MF(j) > 0;
            Copy Fly(j)'s genes into
              Fly(i)'s;
            Randomly mutate Fly(i)'s
              genes;
        end
        Add Fly(i)'s contribution to
          global fitness;
        Compute the global fitness;
    end
until Convergence;
Iteratively eliminate bad flies;
Convert the population of flies into
  problem specific answer;
```

Algorithm 1: Steady state FA

```
          // Read problem specific data
                    // Set the algorithm
Initialisation
// Create the initial swarm of n
   particle
repeat n times
    Create a particle at a random
      position in the search space;
    Initialise the particle's velocity;
    Add the particle to the swarm;
    Add the particle's contribution to
      the swarm's;
end
Compute the global fitness;
repeat                  // Optimisation loop
    foreach Particle p_i ∈ Swarm do
        Remove p_i's contribution from
          the swarm's;
        Update the p_i's velocity;
        Update the p_i's position;
        Compute the global fitness;
        Compute p_i's local fitness
          (Marginal fitness)
        Update p_i's lbest if needed
    end
until Convergence;
Iteratively eliminate bad particles;
Convert the swarm of particles into
  problem specific answer;
```

Algorithm 2: Cooperative PSO.

Algorithmic enhancements such as varying population or swarm size are not shown to improve the readability of the pseudocode. In our experiment, we added an extra loop so that each time stagnation is detected slaughtering/culling and mitosis are alternatively triggered. In the slaughtering/culling step, bad flies or particles are eliminated so that there are only good flies or particles left. If triggering slaughtering/culling and mitosis does not help the population or swarm improve the global fitness over N iterations (stagnation), the optimisation ends and the problem solution is extracted. Our main stopping criterion is thus stagnation.

N is set to 5 for coPSO, FA and SFA. However, it was empirically determined that this number was far too low for PSO, which is why we use 50 in our experiments. An additional stopping criterion is the maximum number of iterations in case an algorithm fails to converge towards a solution. All parameters are provided in Table 1.

Each experiment is repeated 100 times using a supercomputer to gather statistically meaningful results[9]. That is 100 runs $\times$ 6 problem sizes $\times$ 6 algorithms = 3600 optimisation processes in total. Each algorithm records the global fitness of the solution it provided, and how many lamps needed to be created and tested before a solution was accepted. This number is linearly proportional to the computational power that was required to find the solution.

Results and discussion[10]

Quantitative results are given in Table 2. It highlights for each problem size which algorithm(s) provides solutions significantly better ($p < 0.05$) than the other algorithms. From the table it is clear that PSO performs best with small problem sizes but collapses rapidly. It is also computationally intensive compared to FAs. Figures 2 and 3 are a visualisation of these data in terms of global fitness and computing time versus problem size in log scale.[11]

With small problem sizes, FA does not perform quite as well as PSO; Swarm Fly Algorithm (SFA) is comparable to FA though a little less performing, and coPSO does not perform well at all. With larger problem sizes, Figure 2 shows that PSO collapses; FA stabilises, taking advantage of its scalability; coPSO starts a shy improvement, showing that communication is only beneficial with a larger problem size. Figure 3 clearly shows that PSO and coPSO are not as efficient as FA and SFA. On both figures FA and the 3 variants of SFA are hard to distinguish when the problem size increases. A zoomed scatterplot (Figure 4) of global fitness versus computational effort (number of lamps created) gives a more precise comparison for FA and SFA. FA's performance decreases when the problem size increases to become quite close to SFA35's and SFA70's. SFA-bi operator is, however, more consistent and starts to slightly outperform FA in terms of computing requirements ($p < 0.05$).

[9] Except for the largest instance (size 500) for which only 50 runs were done.

[10] **Reproducibility:** code available at `http://doi.org/10.5281/zenodo.7101160`

[11] A synthetic scatterplot is also provided in `https://evelyne-lutton.fr/Lutton_EA2022-Additional.pdf` for assessing the balance between both measurements.

Table 1. Summary of the algorithms' parameters.

	PSO	**FA**	**coPSO**	**SFA35**	**SFA70**	**SFA-bi operator**
Initial number of particles/individuals:	$\sqrt{\frac{\overline{FA}}{3\times \text{pb size}}}$	3 × pb size	3 × pb size	3 × pb size	3 × pb size	3 × pb size
Lamps per particle/individual:	3 × pb size	1	1	1	1	1
W in Eq. 3:	1	1	1	1	1	1
Immigration probability (%):	N/A	30	N/A	30	30	30
Gaussian mutation probability (%):	N/A	70	N/A	35	0	varying
Genealogical mutation probability (%):	N/A	0	N/A	35	70	varying
Initial Gaussian mutation factor (pixels):	N/A	16	N/A	16	N/A	16
Decrease of mutation factor per generation:	N/A	0.016 pixel	N/A	0.016 pixel	N/A	0.016 pixel
ω in Eq. 1:	$\frac{1}{2\times\log(2)}$	N/A	$\frac{1}{2\times\log(2)}$	$\frac{1}{2\times\log(2)}$	$\frac{1}{2\times\log(2)}$	$\frac{1}{2\times\log(2)}$
φ_p in Eq. 1:	$\frac{1}{2}+\log(2)$	N/A	$\frac{1}{2}+\log(2)$	$\frac{1}{2}+\log(2)$	$\frac{1}{2}+\log(2)$	$\frac{1}{2}+\log(2)$
φ_g in Eq. 1:	$\frac{1}{2}+\log(2)$	N/A	0	0	0	0
Stopping criteria 1) No improvement over the last :	50 iterations	5 iterations	5 iterations	5 iterations	5 iterations	5 iterations
2) Max # of gen. or iterations:	500	500	500	500	500	500

For each problem size, $\overline{FA}$ is the average number of lamps created over 100 runs of FA to reach the problem solution.

6 Conclusions

The previous experiments are a first try with fine-grained cooperative swarms. For the moment this has been reached with a single swarm in which social communications have been cut (the *gbest* position has no influence on local rules). Together with a convenient formulation of the problem (which information is carried by a particle), this rough strategy (coPSO) is able to drive the full swarm into a good solution, while having better scalability than the classical *gbest* PSO. Marginal fitness is actually an indirect way to implement some social communication, as it evaluates the contribution of a particle with respect to the whole swarm. Less efficient than FA, a mature technique of fine-grained cooperative based on EA, this simple coPSO however exhibits interesting scalability properties (positive slope on Fig. 2 for large problem sizes).

Future improvements of this strategy can follow different lines. A first one could be distance-based *lbest* strategies, but possibly limited in high dimensions. Another line, sketched in this paper, is a combination of features from Evolutionary Algorithms (life and death, genetic transmission) and swarms (internal memory and social communication in the group). SFA is an attempt to add a memory to the flies, in the same spirit as coPSO, as an inter-generational transmission of information. The experiments displayed above prove that these

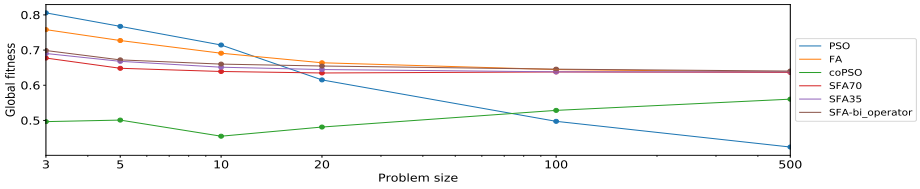

Fig. 2. Comparison in terms of global fitness (maximisation).

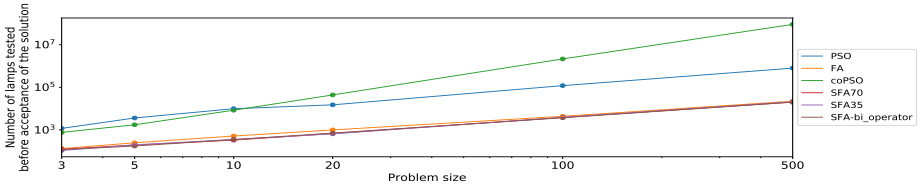

Fig. 3. Comparison in terms of computational requirement. This is a value that should be as small as possible.

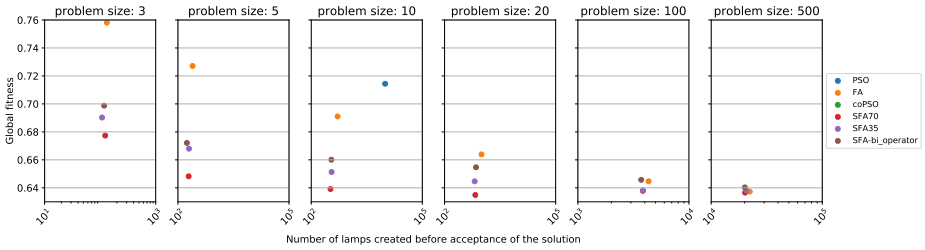

Fig. 4. Performance comparison in terms of effectiveness (highest global fitness) and efficiency (smallest number of tested lamps) zoomed onto FA and SFA. The best algorithms are in the top-left corner of the plots.

inter-generational communications improve the scalability of FA. Making the balance of the SFA mutations adaptive also yields important information about the efficiency of these operators during the runs (see also supplementary material).

Future work on this topic will aim at exploring the combinations of PSO and FA, and extending the experiments to other benchmarks and real problems.

Acknowledgements: We would like to thank Supercomputing Wales for the supercomputer used to generate all the experimental results (`https://www.supercomputing.wales/`).

References

1. Abbood, Z.A., Vidal, F.P.: Fly4Arts: Evolutionary digital art with the Fly algorithm. ISTE Arts & Science **17-1**(1), 11–16 (2017). https://doi.org/10.21494/ISTE.OP.2017.0177
2. Ali Abbood, Z., Lavauzelle, J., Lutton, E., Rocchisani, J.M., Louchet, J., Vidal, F.P.: Voxelisation in the 3-D fly algorithm for PET. Swarm and Evolutionary Computation **36**, 91–105 (Oct 2017). https://doi.org/10.1016/j.swevo.2017.04.001

Table 2. Results of the experiments. Values for algorithms marked in **bold** are significantly better ($p < 0.05$) than the others for the same problem size. Values in *italics* highlight cases where the best performance is equally achieved by two or more algorithms (non separable, with $p > 0.05$).

Problem size	Evolution	Global fitness	Lamps created before acceptance
3	**PSO**	**80.58 % ± 6.84**	1.18e+03 ± 473.47
3	FA	75.81 % ± 4.22	1.32e+02 ± 100.75
3	coPSO	49.66 % ± 11.58	7.63e+02 ± 1560.97
3	SFA70	67.74 % ± 7.59	1.23e+02 ± 100.48
3	SFA35	69.03 % ± 7.56	1.08e+02 ± 79.37
3	SFA-bi operator	69.87 % ± 7.71	1.18e+02 ± 83.20
5	**PSO**	**76.74 % ± 4.98**	3.63e+03 ± 1296.64
5	FA	72.71 % ± 3.51	2.49e+02 ± 138.45
5	coPSO	50.09 % ± 8.51	1.73e+03 ± 1274.77
5	SFA70	64.82 % ± 5.23	1.95e+02 ± 152.83
5	SFA35	66.80 % ± 4.83	1.99e+02 ± 142.94
5	SFA-bi operator	67.21 % ± 4.74	1.75e+02 ± 129.27
10	**PSO**	**71.44 % ± 4.48**	9.89e+03 ± 2843.23
10	FA	69.11 % ± 3.28	5.15e+02 ± 301.90
10	coPSO	45.51 % ± 7.39	8.39e+03 ± 6427.39
10	SFA70	63.91 % ± 4.02	3.31e+02 ± 208.52
10	SFA35	65.13 % ± 3.90	3.55e+02 ± 225.33
10	SFA-bi operator	66.01 % ± 3.22	3.49e+02 ± 205.07
20	PSO	61.52 % ± 4.46	1.50e+04 ± 3893.98
20	**FA**	**66.39 % ± 2.56**	9.97e+02 ± 536.03
20	coPSO	48.12 % ± 6.07	4.37e+04 ± 27148.87
20	SFA70	63.49 % ± 2.81	6.77e+02 ± 290.76
20	SFA35	64.46 % ± 2.69	6.51e+02 ± 322.55
20	SFA-bi operator	65.47 % ± 2.26	7.09e+02 ± 253.34
100	PSO	49.74 % ± 2.26	1.20e+05 ± 34739.96
100	*FA*	*64.47 % ± 1.11*	4.33e+03 ± 1664.40
100	coPSO	52.86 % ± 4.53	2.14e+06 ± 1215718.00
100	SFA70	63.78 % ± 1.29	3.85e+03 ± 854.81
100	SFA35	63.81 % ± 1.42	3.86e+03 ± 912.96
100	*SFA-bi operator*	*64.57 % ± 1.18*	3.71e+03 ± 842.15
500	PSO	42.43 % ± 1.37	7.94e+05 ± 252250.84
500	*FA*	*63.73 % ± 0.56*	2.22e+04 ± 7000.81
500	coPSO	56.04 % ± 3.09	9.00e+07 ± 40418907.90
500	*SFA70*	*63.66 % ± 0.61*	2.02e+04 ± 3852.50
500	*SFA35*	*63.81 % ± 0.57*	2.07e+04 ± 4109.04
500	*SFA-bi operator*	*64.03 % ± 0.54*	2.01e+04 ± 3935.58

3. Ali Abbood, Z., Vidal, F.P.: Basic, dual, adaptive, and directed mutation operators in the fly algorithm. In: Artificial Evolution. pp. 100–114. Springer International Publishing, Cham (2018). https://doi.org/10.1007/978-3-319-78133-4_8
4. Barrière, O., Lutton, E.: Experimental Analysis of a Variable Size Mono-population Cooperative-Coevolution Strategy, pp. 139–152. Springer (2009). https://doi.org/10.1007/978-3-642-03211-0_12
5. Barrière, O., Lutton, E., Wuillemin, P.: Bayesian network structure learning using cooperative coevolution. In: GECCO. pp. 755–762 (2009). https://doi.org/10.1145/1569901.1570006
6. Bergh, F., Engelbrecht, A.: A cooperative approach to particle swarm optimization. Evolutionary Computation, IEEE Transactions on **8**, 225–239 (07 2004). https://doi.org/10.1109/TEVC.2004.826069
7. Blackwell, T.: Swarm music: Improvised music with multi-swarms. In: Symposium on Artificial Intelligence and Creativity in Arts and Science. pp. 41–49 (2003)
8. Blackwell, T., Branke, J.: Multi-swarm optimization in dynamic environments, p. 19–26. Springer-Verlag (2004)
9. Bongard, J., Lipson, H.: Active coevolutionary learning of deterministic finite automata. Journal of Machine Learning Research **6**, 1651–1678 (2005)
10. Boumaza, A.M., Louchet, J.: Mobile Robot Sensor Fusion Using Flies, pp. 357–367 (2003). https://doi.org/10.1007/3-540-36605-9_33
11. Brits, R., Engelbrecht, A., van den Bergh, F.: Scalability of niche PSO. In: Proceedings of the IEEE swarm intelligence symposium, Indianapolis, Indiana, USA, April 24–26. p. 228–234 (2003)
12. Collet, P., Lutton, E., Raynal, F., Schoenauer, M.: Polar IFS + Parisian genetic programming = efficient IFS inverse problem solving. Genetic Programming and Evolvable Machines Journal **1**(4), 339–361 (2000), october

13. Das, S., Biswas, A., Dasgupta, S., Abraham, A.: Bacterial Foraging Optimization Algorithm: Theoretical Foundations, Analysis, and Applications, pp. 23–55 (2009). https://doi.org/10.1007/978-3-642-01085-9_2
14. De Jong, E.D., Stanley, K.O., Wiegand, R.P.: Introductory tutorial on coevolution. In: GECCO '07, London, UK (2007)
15. De Melo, V.V.: Kaizen programming. In: Igel, C., Arnold, D.V., Gagne, C., Popovici, E., Auger, A., Bacardit, J., Brockhoff, D., Cagnoni, S., Deb, K., Doerr, B., Foster, J., Glasmachers, T., Hart, E., Heywood, M.I., Iba, H., Jacob, C., Jansen, T., Jin, Y., Kessentini, M., Knowles, J.D., Langdon, W.B., Larranaga, P., Luke, S., Luque, G., McCall, J.A.W., Montes de Oca, M.A., Motsinger-Reif, A., Ong, Y.S., Palmer, M., Parsopoulos, K.E., Raidl, G., Risi, S., Ruhe, G., Schaul, T., Schmickl, T., Sendhoff, B., Stanley, K.O., Stuetzle, T., Thierens, D., Togelius, J., Witt, C., Zarges, C. (eds.) GECCO '14: Proceedings of the 2014 conference on Genetic and evolutionary computation. pp. 895–902. ACM, Vancouver, BC, Canada (12-16 Jul 2014). https://doi.org/doi:10.1145/2576768.2598264, http://doi.acm.org/10.1145/2576768.2598264
16. Dorigo, M., Birattari, M., Stutzle, T.: Ant colony optimization. IEEE Computational Intelligence Magazine **1**(4), 28–39 (2006)
17. Dunn, E., Olague, G., Lutton, E.: Parisian camera placement for vision metrology. Pattern Recognition Letters **27**(11), 1209–1219 (2006). https://doi.org/https://doi.org/10.1016/j.patrec.2005.07.019, https://www.sciencedirect.com/science/article/pii/S016786550500334X, evolutionary Computer Vision and Image Understanding
18. El-Abd, M., Kamel, M.S.: A taxonomy of cooperative particle swarm optimizers. International Journal of Computational Intelligence Research **4** (01 2008). https://doi.org/10.5019/j.ijcir.2008.133
19. Kachitvichyanukul, V.: Comparison of three evolutionary algorithms: Ga, pso, and de. Industrial Engineering and Management Systems **12**, 215–223 (09 2012). https://doi.org/10.7232/iems.2012.11.3.215
20. Karaboga, D., Gorkemli, B., Ozturk, C., Karaboga, N.: A comprehensive survey: artificial bee colony (abc) algorithm and applications. Artif Intell Rev **42**, 21–57 (2014). https://doi.org/10.1007/s10462-012-9328-0
21. Kaufmann, B., Louchet, J., Lutton, E.: Hand posture recognition using real-time artificial evolution. In: EvoApplications, LNCS 6024. pp. 251–260. Springer (2010)
22. Kennedy, J., Eberhart, R.: Particle swarm optimization. In: Proceedings of ICNN'95 - International Conference on Neural Networks. vol. 4, pp. 1942–1948 (Nov 1995). https://doi.org/10.1109/ICNN.1995.488968
23. Kennedy J, M.R.: Population structure and particle swarm performance. In: CEC, Honolulu, HI, USA, Sept 22–25. p. 1671–1676 (2002)
24. La Cava, M., Moore, J.: A general feature engineering wrapper for machine learning using epsilon lexicase survival. In: J. McDermott, e.a. (ed.) Genetic Programming, pp. 80–95. Springer (2017)
25. Landrin-Schweitzer, Y., Collet, P., Lutton, E.: Introducing lateral thinking in search engines. Genetic Programming and Evolvable Machines **7**(1), 9–31 (Mar 2006). https://doi.org/10.1007/s10710-006-7008-z
26. Louchet, J.: Using an individual evolution strategy for stereovision. Genetic Programming and Evolvable Machines **2**(2), 101–109 (Jun 2001). https://doi.org/10.1023/A:1011544128842
27. Marimont, R.B., Shapiro, M.B.: Nearest Neighbour Searches and the Curse of Dimensionality. IMA Journal of Applied Mathematics **24**(1), 59–70 (08 1979). https://doi.org/10.1093/imamat/24.1.59

28. McConaghy, T.: FFX: Fast, scalable, deterministic symbolic regression technology. In: Riolo, R., Vladislavleva, E., Moore, J.H. (eds.) Genetic Programming Theory and Practice IX, chap. 13, pp. 235–260. Genetic and Evolutionary Computation, Springer, Ann Arbor, USA (12-14 May 2011). https://doi.org/doi:10.1007/978-1-4614-1770-5$_1$3, http://trent.st/content/2011-GPTP-FFX-paper.pdf
29. Niu, B., Zhu, Y., He, X.: Multi-population cooperative particle swarm optimization. In: Advances in Artificial Life. pp. 874–883. Springer Berlin Heidelberg, Berlin, Heidelberg (2005). https://doi.org/10.1007/11553090_88
30. Ochoa, G., Lutton, E., Burke, E.K.: Cooperative royal road functions. In: Evolution Artificielle, Tours, France, October 29-31 (2007)
31. Poli, R., Kennedy, J., Blackwell, T.: Particle swarm optimization. Swarm Intelligence **1**(1), 33–57 (Jun 2007). https://doi.org/10.1007/s11721-007-0002-0
32. Popovici, E., Bucci, A., Wiegand, R.P., De Jong, E.D.: Coevolutionary Principles, pp. 987–1033. Springer (2012). https://doi.org/10.1007/978-3-540-92910-9_31
33. Potter, M.A., De Jong, K.A.: A cooperative coevolutionary approach to function optimization. In: Parallel Problem Solving from Nature — PPSN III. pp. 249–257. Springer Berlin Heidelberg, Berlin, Heidelberg (1994)
34. Schwefel, H.P.: Evolution and Optimum Seeking: The Sixth Generation. John Wiley & Sons, Inc., USA (1993)
35. Shi, Y., Eberhart, R.C.: Empirical study of particle swarm optimization. In: Congress on Evolutionary Computation, CEC99. vol. 3, pp. 1945–1950 (July 1999). https://doi.org/10.1109/CEC.1999.785511
36. Shi, Y., Krohling, R.A.: Co-evolutionary particle swarm optimization to solve min-max problems. In: CEC, vol. 2. p. 1682–1687 (2002)
37. Shi, Y., Eberhart, R.C.: Parameter selection in particle swarm optimization. In: Evolutionary Programming VII. pp. 591–600. Springer (1998)
38. Sörensen, K.: Metaheuristics—the metaphor exposed. International Transactions in Operational Research **22**(1), 3–18 (2015)
39. Tonda, A., Lutton, E., Squillero, G.: Lamps: A test problem for cooperative coevolution. In: NICSO. vol. 387, pp. 101–120. Springer (10 2011). https://doi.org/10.1007/978-3-642-24094-2_7
40. Vidal, F.P., Lazaro-Ponthus, D., Legoupil, S., Louchet, J., Lutton, E., Rocchisani, J.M.: Pet reconstruction using a cooperative coevolution strategy. In: Proceedings of the IEEE Medical Imaging Conference 2009. IEEE, Orlando, Florida (Oct 2009)
41. Vidal, F.P., Lutton, E., Louchet, J., Rocchisani, J.: Threshold selection, mitosis and dual mutation in cooperative coevolution: application to medical 3D tomography. In: PPSN. LNCS, vol. 6238, pp. 414–423 (Sep 2010). https://doi.org/10.1007/978-3-642-15844-5_42
42. Wang, D., Tan, D., Liu, L.: Particle swarm optimization algorithm: an overview. Soft Computing (01 2017). https://doi.org/10.1007/s00500-016-2474-6
43. Wiegand, R.P., Potter, M.A.: Robustness in cooperative coevolution. In: Proceedings of GECCO, Seattle, Washington, USA. p. 369–376 (2006). https://doi.org/10.1145/1143997.1144063
44. Wilson, S.T., Goldberg, D.E.: A Critical Review of Classifier Systems . In: Third International Conference on Genetic Algorithms. pp. 244–255 (1989)
45. Zhang, H.: A Newly Cooperative PSO – Multiple Particle Swarm Optimizers with Diversive Curiosity, MPSOα/DC, pp. 69–82. Springer Netherlands, Dordrecht (2011). https://doi.org/10.1007/978-94-007-0286-8_7