Chapter 1
Lamps: A Test Problem for Cooperative Coevolution

Alberto Tonda and Evelyne Lutton and Giovanni Squillero

Abstract We present an analysis of the behaviour of Cooperative Co-evolution algorithms (CCEAs) on a simple test problem, that is the optimal placement of a set of lamps in a square room, for various problems sizes. Cooperative Co-evolution makes it possible to exploit more efficiently the artificial Darwinism scheme, as soon as it is possible to turn the optimisation problem into a co-evolution of inter-dependent sub-parts of the searched solution. We show here how two cooperative strategies, Group Evolution (GE) and Parisian Evolution (PE) can be built for the lamps problem. An experimental analysis then compares a classical evolution to GE and PE, and analyses their behaviour with respect to scale.[1]

Keywords : Cooperative co-evolution, Group Evolution, Parisian Evolution, Benchmark Problem, Experimental Analysis, Scalability.

1.1 Introduction

Cooperative co-evolution algorithms (CCEAs) share common characteristics with standard artificial Darwinism-based methods, i.e. Evolutionary Algorithms (EAs), but with additional components that aim at implementing collective capabilities.

Alberto Tonda
ISC-PIF, CNRS CREA, UMR 7656, 57-59 rue Lhomond, Paris France. e-mail: Alberto.Tonda@gmail.com

Evelyne Lutton
AVIZ Team, INRIA Saclay - Ile-de-France, Bat 490, Université Paris-Sud, 91405 ORSAY Cedex, France. e-mail: Evelyne.Lutton@inria.fr

Giovanni Squillero
Politecnico di Torino - Dip. Automatica e Informatica, C.so Duca degli Abruzzi 24 - 10129 Torino - Italy. e-mail: Giovanni.Squillero@polito.it

[1] Aknowledgements for the funding received from the European Community's Seventh Framework Programme (FP7/2009-2013) under grant agreement DREAM n. 222654-2.

For optimisation purpose, CCEAs are based on a specific formulation of the problem where various inter- or intra-population interaction mechanisms occur. Usually, these techniques are efficient as optimiser when the problem can be split into smaller interdependent subproblems. The computational effort is then distributed onto the evolution of smaller elements of similar or different nature, that aggregates to build a global solution.

Cooperative co-evolution is increasingly becoming the basis of successful applications [1, 6, 8, 15, 19], including learning problems, see for instance [3]. These approaches can be shared into two main categories: co-evolution process that happens between a fixed number of separate populations [5, 13, 14] or within a single population[7, 10, 18].

The design and fine tuning of such algorithms remain however difficult and strongly problem dependent. A critical question is the design of simple test problem for CCEAs, for benchmarking purpose. A first test-problem based on Royal Road Functions has been proposed in [12]. We propose here another simple problem, the Lamps problem, for which various instances of increasing complexity can be generated, according to a single ratio parameter. We show below how two CCEAs can be designed and compared against a classical approach, whith a special focus on scalability.

The paper is organised as follows: the Lamps problem is described in section 1.2, then the design of two cooperative co-evolution strategies, Parisian Evolution and Group Evolution, is detailed in sections 1.3 and 1.4. The experimental setup is described in section 1.5: three strategies are tested, a classical genetic programing approach, (CE for Classical Evolution), the Group Evolution (GE) and the Parisian Evolution (PE). All methods are implemented using the μGP toolkit [16]. Results are presented and analysed in section 1.6, and conclusions and future work are given in section 1.7.

1.2 The Lamps problem

The optimisation problem chosen to test cooperative coevolution algorithms requires to find the best placement for a set of lamps, so that a target area is fully brightened with light. The minimal number of lamps needed is unknown, and heavily depends on the topology of the area. All lamps are alike, modeled as circles, and each one may be evaluated separately with respect to the final goal. In the example, the optimal solution requires 4 lamps (Figure 1.1, left): interestingly, when examined independently, all lamps in the solution waste a certain amount of light outside the target area. However, if one of the lamps is positioned to avoid this undesired effect, it becomes impossible to lighten the remaining area with the three lamps left (Figure 1.1, right). Since lamps are simply modeled as circles, the problem may also be seen as using the circles to completely cover the underlying area, as efficiently as possible.

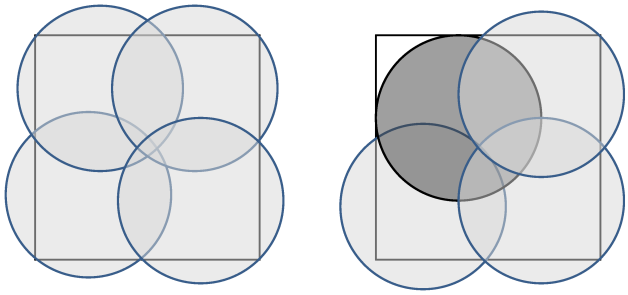


Fig. 1.1 Placement of a set of lamps. The aim is to enlighten all the square area. It is interesting to notice how a solution where some of the light of each lamp is wasted outside the area (left) overall performs better than a solution where the grayed lamp maximises its own performance (right).

This apparently simple benchmark exemplifies a common situation in real-world applications: many problems have an optimal solution composed of a set of homogeneous elements, whose individual contribution to the main solution can be evaluated separately. Note that, in this context, *homogeneous* is used to label elements sharing the same base structure.

A similar toy problem has been sketched in [17], but the structure of the benchmark has been improved and parametrised, and the fitness function has been modified, to increase the complexity and the number of local optima on the fitness landscape.

1.2.1 Size of the problem

It is intuitive that the task of enlightening a room with a set of lamps can be more or less difficult, depending on the size of the room and the cone of light of each lamp. If small sources of light are used to brighten a large room, surely a greater number of lamps will be required, and the number of possible combinations will increase dramatically.

With these premises, the complexity of the problem can thus be expressed by the ratio between the surface to be enlightened and the maximum area enlightened by a single lamp:

$$problem\_size = \frac{area\_room}{area\_lamp}$$

as this ratio increases, finding an optimal solution for the problem will become harder.

It is interesting to notice how variations in the shape of the room could also influence the complexity of the task: rooms with an irregular architecture may require more intricate lamp placements. However, finding a dependency between the shape of the room and the difficulty of the problem is not trivial, and results might be less

intuitive to analyze. For all these reasons, the following experiments will feature square rooms only.

1.2.2 Fitness value

Comparing different methodologies on the same problem requires a common fitness function, to be able to numerically evaluate the solutions obtained by each approach.

Intuitively, the fitness of a candidate solution should be directly proportional to the area fully brightened by the lamps and inversely proportional to the number of lamps used, favoring solutions that cover more surface with light using the minimal number of lamps. The first term in the fitness value will thus be proportional to the ratio of the area enlightened by the lamps,

$$\frac{area\_enlightened}{total\_area}$$

To increase the complexity of the fitness landscape, a further contribution is added: the area brightened by more than one lamp is to be minimised, in order to have as little overlapping as possible. The second term will be then proportional to:

$$-\frac{area\_overlap}{total\_area}$$

It is interesting to note that minimising the overlap also implies an optimisation of the number of lamps used, since using a greater number would lead to more overlapping areas.

The final fitness function will then be:

$$fitness = \frac{area\_enlightened}{total\_area} - W \cdot \frac{area\_overlap}{total\_area} =$$

$$= \frac{area\_enlightened - W \cdot area\_overlap}{total\_area}$$

where W is a weight associated to the relative importance of the overlapping, set by the user.

Using this function with $W = 1$, fitness values will range between $(0,1)$, but it is intuitive that it is impossible to reach the maximum value: by problem construction, overlapping and enlightenment are inversely correlated, and even good solutions will feature overlapping areas and/or parts not brightened. This problem is actually multi-objective, the fitness function we propose corresponds to a compromise between the two objectives.

1.3 Parisian Evolution

Initially designed to address the inverse problem for Iterated Function System (IFS), a problem related to fractal image compression[7], this scheme has been successfully applied in various real world applications: in stereovision [4], in photogrammetry [9, 11], in medical imaging [18], for Bayesian Network Structure learning [2], in data retrieval[10].

Parisian Evolution (PE) is based on a two-level representation of the optimisation problem, meaning that an individual in a Parisian population represents only a part of the solution. An aggregation of multiple individuals must be built to complete a meaningful solution to the problem. This way, the co-evolution of the whole population (or of a major part of it) is favoured over the emergence of a single best individual, as in classical evolutionary schemes.

This scheme distributes the workload of solution evaluations at two levels. Light computations (e.g. existence conditions, partial or approximate fitness) can be done at the individual's level (local fitness), while the complete calculation (i.e. global fitness) is performed at the population level. The global fitness is then distributed as a bonus to individuals who participate the global solution. A Parisian scheme has all the features of a classical EA (see figure 1.2) with the following additional components:

- A grouping stage at each generation, that selects individuals that are allowed to participate to the global solution.
- A redistribution step that rewards the individuals who participate to the global solution : their bonus is proportional to the global fitness.
- A sharing scheme, that avoids degenerate solutions where all individuals are identical.

Efficient implementations of the Parisian scheme are often based on partial redundancies between local and global fitness, as well as clever exploitation of computational shortcuts. The motivation is to make a more efficient use of the evolution of the population, and reduce the computational cost. Successful applications of such a scheme usually rely on a lower cost evaluation of the partial solutions (i.e. the individuals of the population), while computing the full evaluation only once at each generation or at specified intervals.

1.3.1 Implementation of the lamps problem

For the lamps problem, the PE has been implemented as follows. An individual represents a lamp, its genome is its (x,y) position, plus a third element, e, that can assume values 0 or 1 (on/off switch). Lamps with $e = 1$ are "on" (expressed) and contribute to the global solution, while lamps with $e = 0$ do not.

Global fitness is computed as described in subsection 1.2.2. In generation 0 the global solution is computed simply considering the individuals with $e = 1$ among

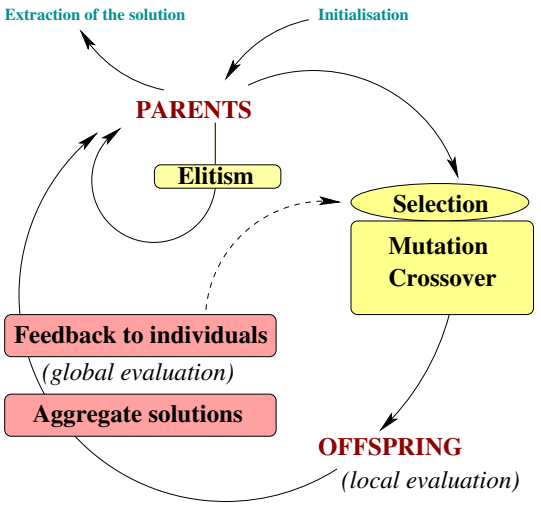


Fig. 1.2 A Parisian EA: a monopopulation cooperative-coevolution. Partial evaluation (local fitness) is applied to each individual, while global evaluation is performed once a generation.

the μ initial ones. Then, at each step, λ individuals are generated. For each new individual with $e = 1$, its potential contribution to the global solution is computed. Before evaluation, a random choice ($p = 0.5$) is performed: new individuals are either considered in addition to or in replacement of the existing ones.

If individuals are considered for addition, the contribution to the global solution of each one is computed. Otherwise, the less performing among the old individuals is removed, and only then the contribution to the global solution of each new individual is evaluated. If the addition or replacement of the new individuals leads to an improvement over the previous global fitness, the new individual selected is rewarded with a high local fitness value ($local\_fitness = 2$), together with all the old individuals still contributing to the global solution. New expressed individuals ($e = 1$) that are not selected for the global solution are assigned a low fitness value ($local\_fitness = 0$). Non-expressed individuals ($e = 0$) have an intermediate fitness value ($local\_fitness = 1$).

Sharing follows the simple formula

$$fitness\_sharing(I_k) = \frac{local\_fitness(I_k)}{\sum_{i=0}^{individuals} sharing(I_k, I_{i \neq k})}$$

with

$$sharing(I_1, I_2) = \begin{cases} 1 - \frac{d(I_1,I_2)}{2 \cdot lamp\_radius} & d(I_1, I_2) < 2 \cdot lamp\_radius \\ 0 & d(I_1, I_2) \geq 2 \cdot lamp\_radius \end{cases}$$

Lamps that have a relatively low number of neighbours will be preferred for selection over lamps with a bigger number of neighbours. In this implementation, sharing

is computed only for expressed lamps ($e = 1$) and used only when selecting the less performing individual to be removed from the population.

1.4 Group Evolution

Group Evolution (GE) is a novel generational cooperative coevolution concept presented in [17]. The approach uses a population of partial solutions, and exploits non-fixed sets of individuals called *groups*. GE acts on individuals and groups, managing both in parallel. During the evolution, individuals are optimised as in a common EA, but concurrently groups are also evolved. The main peculiarity of GE is the absence of *a priori* information about the grouping of individuals.

At the beginning of the evolutionary process, an initial population of individuals is randomly created on the basis of a high-level description of a solution for the given problem. Groups at this stage are randomly determined, so that each individual can be included in any number of different groups, but all individuals are part of at least one group.

Population size $\mu_{individuals}$ is the maximum number of individuals in the population, and it is set by the user before the evolution starts. The number of groups μ_{groups}, the minimum and maximum size of the groups are set by the user as well. Figure 1.3 (left) shows a sample population where minimum group size is 2, and maximum group size is 4.

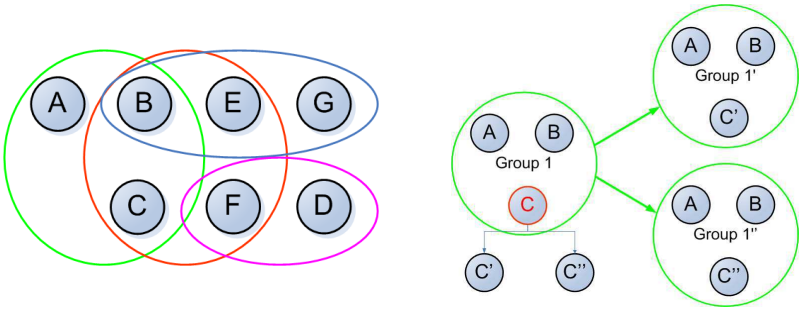


Fig. 1.3 Individuals and Groups in a sample population of 8 individuals (left). While individual A is part of only one group, Individual B is part of 3 different groups. On the right, the effect of a Individual Genetic Operator, applied to individual C. Since individual C is part of Group 1, two groups are created and added to the population.

1.4.1 Generation of new individuals and groups

GE exploits a generational approach: at each evolutionary step, a number of genetic operators is applied to the population. Genetic operators can act on both individuals and groups, and produce a corresponding offspring, in form of individuals and groups.

The offspring creation phase comprehends two different actions at each generation step (see Figure 1.4):

1. Application of *group genetic operators*;
2. Application of *individual genetic operators*.

Each time a genetic operator is applied to the population, parents are chosen and offspring is generated. The children are added to the population, while the original parents are unmodified. Offspring is then evaluated, while it is not compulsory to reconsider the fitness value of the parents again. It is important to notice that the number of children produced at each evolutionary step is not fixed: each genetic operator can have any number of parents as input and produce in output any number of new individuals and groups. The number and type of genetic operators applied at each step can be set by the user.

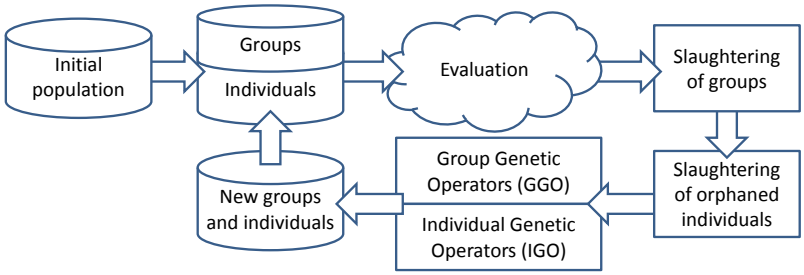


Fig. 1.4 Schema of Group Evolution algorithm.

1.4.1.1 Group genetic operators

Group Genetic Operators (GGOs) work on the set of groups. Each operator needs a certain number of groups as parents and produces a certain number of groups as offspring that will be added to the population. GGOs implemented in our approach are:

1. **crossover**: generates offspring by selecting two individuals, one from parent group A and one from parent group B. Those individuals are switched, creating two new groups;

2. **adding-mutation**: generates offspring by selecting one or more individuals from the population and a group. Chosen individuals are added (if possible) to the parent group, creating a single new group;
3. **removal-mutation**: generates offspring by selecting a group and one or more individuals inside it. Individuals are removed from the parent group.
4. **replacement-mutation**: generates offspring by selecting a group and one or more individuals inside it. Individuals are removed from the parent group, and replaced by other individuals selected from the population.

Parent groups are chosen via tournament selection.

1.4.1.2 Individual genetic operators

Individual Genetic Operators (IGOs) operate on the population of individuals, very much like they are exploited in usual GA. The novelty of GE is that for each individual produced as offspring, new groups are added to the group population. For each group the parent individual was part of, a copy is created, with the offspring taking the place of the parent.

This approach, however, could lead to an exponential increase in the number of groups, as the best individuals are selected by both GGOs and IGOs. To keep the number of groups under a strict control, we choose to create a copy only of the highest-fitness groups the individual was part of.

IGOs select individuals by a tournament selection in two parts: first, a group is picked out through a tournament selection with moderate selective pressure; then an individual in the group is chosen with low selective pressure. The actual group and the highest-fitness groups the individual is part of are cloned once for each child individual created: in each clone group the parent individual is replaced with a child. An example is given in Figure 1.3 (right): an IGO, selects individual C as a parent. The chosen individual is part of only one group, Group 1. The IGO produces two children individuals: since the parent was part of a group, a new group is created for each new individual generated. The new groups (Group 1' and Group 1") are identical to Group 1, except that individual C is replaced with one of its children, C' in Group 1' and C" in Group 1" respectively.

The aim of this process is to select individuals from well-performing groups to create new groups with a slightly changed individual, in order to explore the a near area in the solution space.

1.4.2 Evaluation

During the evaluation phase, a fitness value is associated to each group: the fitness value is a number that measures the goodness of the candidate solutions with respect to the given problem. When a group is evaluated, a fitness value is also assigned to all the individuals composing it. Those values reflect the goodness of the solution

represented by the single individual and have the purpose to help discriminate during tournament selection for both IGOs and GGOs.

An important strength of the approach resides in the evaluation step: if there is already a fitness value for an individual that is part of a new group, it is possible to take it into account instead of re-evaluating all the individuals in the group. This feature can be exceptionally practical when facing a problem where the evaluation of a single individual can last several minutes and the fitness of a group can be computed without examining simultaneously the performance of the individuals composing it. In that case, the time-wise cost of both IGOs and GGOs becomes very small.

1.4.3 Slaughtering

After each generation step, the group population is resized. The groups are ordered fitness-wise and the worst one is deleted until the desired population size is reached. Every individual keeps track of all the groups it belongs to in a set of references. Each time a group ceases to exist, all its individuals remove it from their set of references. At the end of the group slaughtering step, each individual that has an empty set of references, and is therefore not included in any group, is deleted as well.

1.4.4 Implementation of the lamps problem

For the lamps problem, GE has been implemented as follows. One individual represents a lamps, its genome is the (x,y) position. The fitness of a single individual, which must be independent from all groups it is part of, is simply the area of the room it enlightens. The group fitness is computed as described in subsection 1.2.2.

1.5 Experimental setup

Before starting the experiments on the cooperative coevolution algorithms, a series of 10 runs for each *problem_size* of a classical evolutionary algorithm is performed, to better understand the characteristics of the problem and to set parameters leading to a fair comparison. The genome of a single individual is a set of lamps, modeled as an array of N couples of values $((x_1,y_1),(x_2,y_2),...,(x_N,y_N))$, where each (x_i,y_i) describes the position of individual i in the room. The algorithm uses a classical $(\mu+\lambda)$ evolutionary paradigm, with $\mu = 20$ and $\lambda = 10$, probability of crossover 0.2 and probability of mutation 0.8. Each run lasts 100 generations.

By examining these first experimental results, it is possible to reach the following conclusions: good solutions for the problem use a number of lamps in the range

$(problem\_size, 3 \cdot problem\_size)$; and, as expected, as the ratio grows, the fitness value of the best individual at the end of the evolution tends to be lower.

In order to perform a comparison with GE and PE as fair as possible, the stop condition for each algorithm will be set as the average number of single lamp evaluations performed by the classical evolutionary algorithm for each *problem_size*. In this way, even if the algorithms involved have significantly different structures, the computational effort is commensurable. Table 1.1 summarises the results.

Problem size	Evaluations	Average best fitness
3	3,500	0.861
5	5,000	0.8216
10	11,000	0.7802
20	22,000	0.6804
100	120,000	0.558

Table 1.1 Average evaluations for each run of the classical evolutionary algorithm.

1.5.1 PE setup

Due to the observation of the CE runs, $\mu = 3 \cdot problem\_size$, while $\lambda = \mu/2$. The probability of mutation is 0.8 and the probability of crossover is 0.2.

1.5.2 GE setup

The number of groups in the population is fixed, $\mu_{groups} = 20$, as is the number of genetic operators selected at each step $\lambda = 10$. The number of individuals in each group is set to vary in the range $(problem\_size, 3 \cdot problem\_size)$. Since the number of individuals in each group will grow according to the problem size, the number of individuals in the population will be $\mu_{individuals} = 3 \cdot \mu_{groups} \cdot problem\_size$. The probability of mutation is 0.8, while the probability of crossover is 0.2, for both individuals and groups.

1.5.3 Implementation in μGP

The two CCEAs used in the experience have been implemented using μGP [16], an evolutionary toolkit developed by CAD group of Politecnico di Torino. Exploiting μGP's flexible structure, with the fitness evaluator completely independent from the evolutionary core, the great number of adjustable parameters, and the modular

framework composed of clearly separated C++ classes, it is possible to obtain the behavior of very different EAs.

In particular, to implement PE, it is sufficient to operate on the fitness evaluator, setting the environment to evaluate the whole population and the offspring at each step. Obtaining GE behavior is slightly more complex, and requires the addition of new classes to manage groups. CE is simply μGP standard operation mode.

1.6 Results and Analysis

In a series of experiments, 100 runs of each evolutionary approach are executed, for a set of meaningful values of *problem_size*. To exploit a further measurement of comparison, the first occurrence of an acceptable solution also appears in the results: here an *acceptable solution* is defined as a global solution with at least 80% of the final fitness value obtained by the CE. Table 1.2 summarises the results for significant values of *problem_size*. For each evolutionary algorithm are reported the results reached at the end of the evolution: average fitness value, average enlightenment percentage of the room, average number of lamps, and average number of lamps evaluated before finding an acceptable solution (along with the standard deviation for each value).

For each *problem_size*, a box plot is provided in figure 1.5. It is noticeable how the performance of each evolutionary algorithm is very close for small values of *problem_size*, while GE and PE gain the upper hand when the complexity increases.

In particular, PE obtains the best performance from *problem_size* $= 10$ onwards. The extra information inserted allows the approach to obtain high enlightenment percentages even when the complexity of the task increases, as shown in Figure 1.6.

On the other hand, GE obtains enlightenment percentages close to CE, but on the average it uses a lower number of lamps, that leads to a lower overlap, as it is noticeable in Figure 1.8 and 1.7.

When dealing with the number of lamp evaluations needed before reaching what is defined an *acceptable* solution, PE is again in the lead, see Figure 1.9.

In Figure 1.10, a profile of the best run for *problem_size* $= 100$ for each algorithm is reported. PE enjoys a rapid growth in the first stages of the evolution, thanks to the extra information it can make use of, while GE proves more efficient than CE in the last part of the evolution, where exploitation becomes more prevalent.

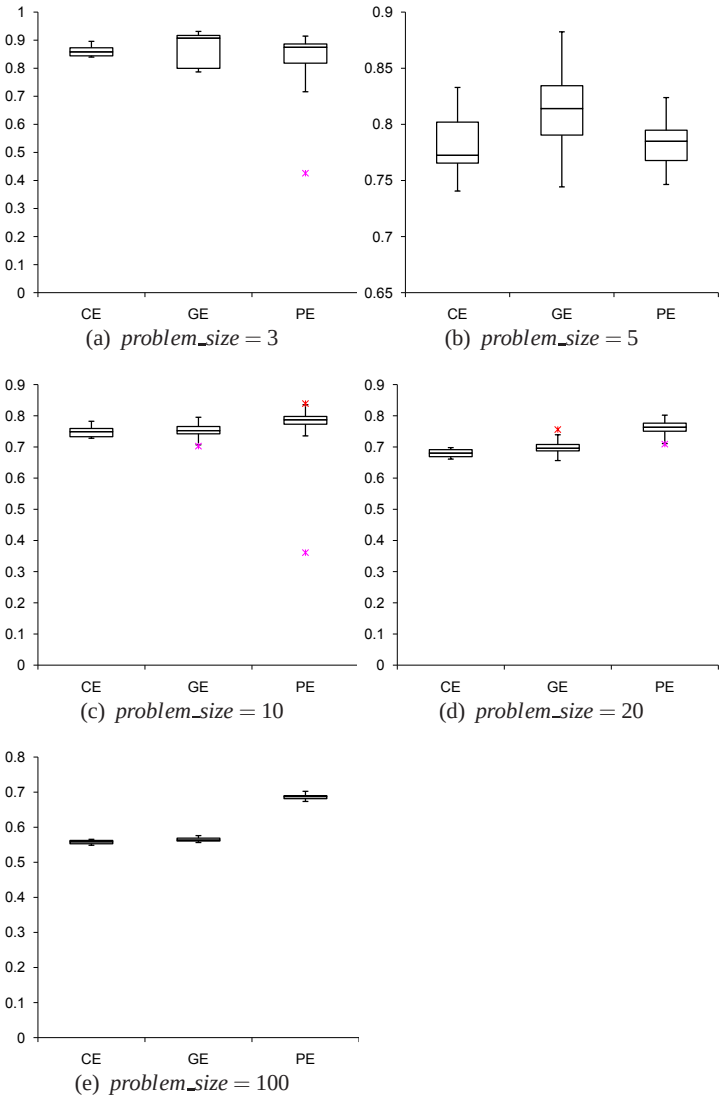
As it is noticeable in Figure 1.11, while the number of lamps evaluated before reaching an acceptable solution grows more than linearly for GE and CE, PE shows a less steep growth. On the other hand, GE presents the lowest overlap for all values of *problem_size* (Figure 1.12).

Problem size	Evolution	Avg. fitness	Std dev	Avg. enlight.	Std dev	Avg. lamps	Std dev	Avg. overlap	Std dev	Avg. lamps before acceptable	Std dev
3	Classical Evolution	0.861	0.0149	0.8933	0.0212	4	0	0.0323	0.0168	313.2	126
	Group Evolution	0.8764	0.0498	0.8963	0.0533	3.75	0.435	0.0198	0.0103	267.32	138.1
	Parisian Evolution	0.8355	0.064	0.8945	0.0439	4.02	0.3344	0.059	0.0502	316.9	262.2
5	Classical Evolution	0.7802	0.023	0.8574	0.04	6.2	0.64	0.0772	0.0278	572.7	215.38
	Group Evolution	0.8136	0.0241	0.8537	0.0349	6.03	0.7372	0.0401	0.0166	741.36	221.86
	Parisian Evolution	0.7825	0.03	0.8803	0.0335	6.96	0.6936	0.0978	0.0395	511.35	373.85
10	Classical Evolution	0.7487	0.0149	0.834	0.0235	11.3	0.62	0.0853	0.0216	1,779.8	407.4
	Group Evolution	0.7532	0.0178	0.8132	0.0255	10.66	0.6336	0.0599	0.0215	1,836.87	412.08
	Parisian Evolution	0.7791	0.0221	0.8847	0.0207	12.84	0.8184	0.1055	0.0274	1,018.47	546.16
20	Classical Evolution	0.6804	0.0117	0.7749	0.0148	20.6	0.72	0.0946	0.0123	3,934.7	702.24
	Group Evolution	0.697	0.0127	0.7837	0.0147	20.49	0.5978	0.0867	0.0142	4602.1	1156.5
	Parisian Evolution	0.7624	0.0147	0.8762	0.0168	23.57	1.053	0.1138	0.0177	2,759.28	965.45
100*	Classical Evolution	0.558	0.0049	0.7334	0.0062	102.9	1.88	0.1755	0.0093	29,567.3	4,782.96
	Group Evolution	0.5647	0.0057	0.7309	0.0073	101.5	1.7	0.1662	0.0072	30,048	8,876.8
	Parisian Evolution	0.6867	0.0073	0.8708	0.0078	117.2	3.2	0.1841	0.0134	5,318.9	332.72

Table 1.2 Average results for 100 runs, for each evolutionary approach. *Due to the times involved, data for problem size 100 is computed on 10 runs.

1.7 Conclusion and future work

The Lamps benchmark has the major advantage to provide a set of toy problems that are simple, and for which the scale can be characterised with a single real value (the surface ratio between room and lamp sizes). This formulation is very convenient to get some insight on the behaviour of algorithms with respect to scale.



(a) *problem_size* = 3 (b) *problem_size* = 5

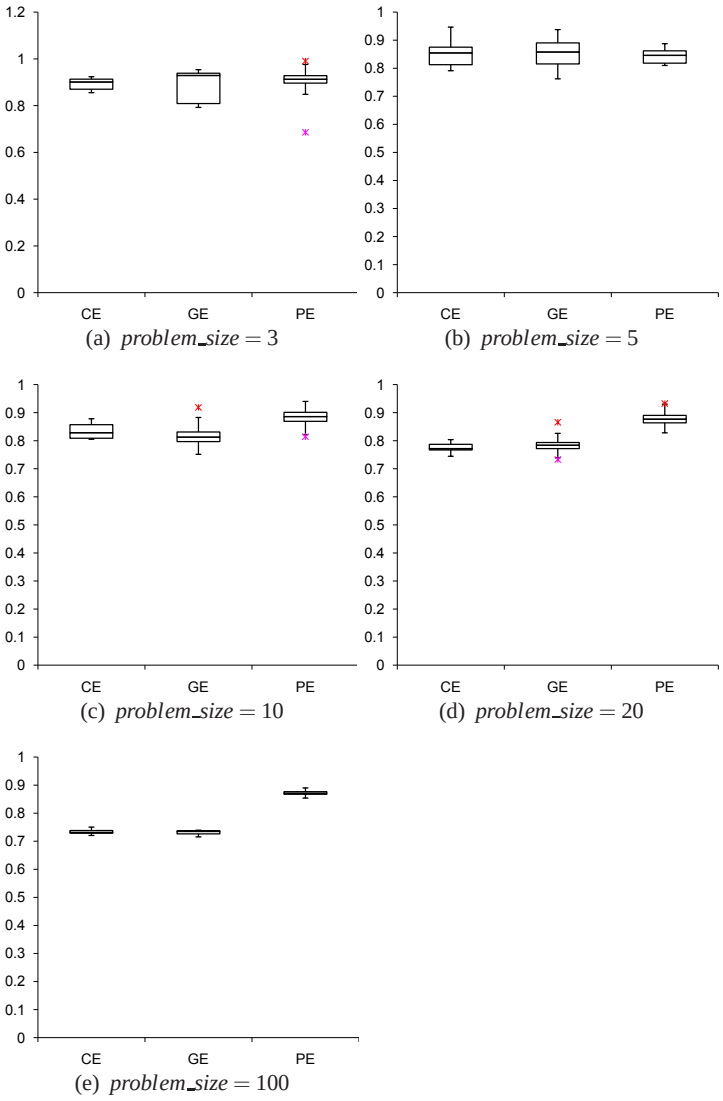
(c) *problem_size* = 10 (d) *problem_size* = 20

(e) *problem_size* = 100

Fig. 1.5 Box plots of fitness values for each *problem_size*.

The intuition that guided the development of Group Evolution and Parisian Evolution has been confronted to experimental analysis, to yield the following conclusions: Parisian Evolution is the most efficient approach in terms of computational expense, and scalability; Group Evolution yields better and more precise results, in a computational time that is similar to Classical Evolution.

These differences can be explained by the nature of *a priori* information that has been injected into the algorithms. Parisian Evolution relies on a deterministic



(a) *problem_size* = 3 (b) *problem_size* = 5

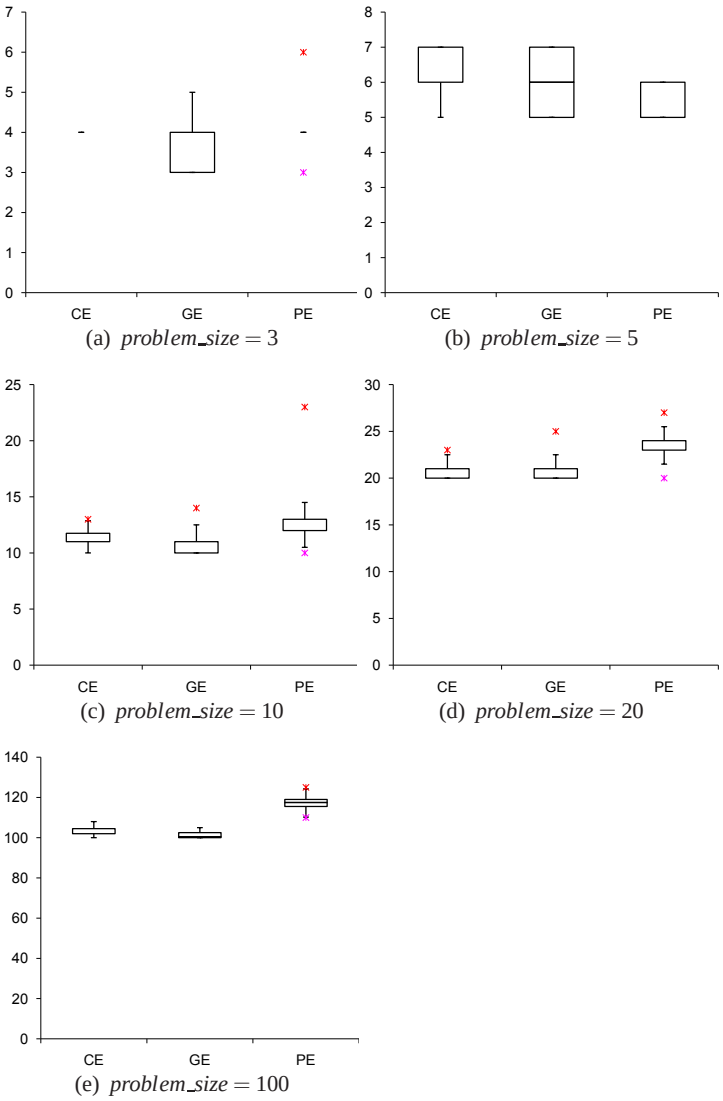
(c) *problem_size* = 10 (d) *problem_size* = 20

(e) *problem_size* = 100

Fig. 1.6 Box plots of enlightenment percentage for each *problem_size*.

algorithm for selecting the group of lamps that are used as the current global solution at each generation, while Group Evolution does not make any assumption on it and let evolution decide which lamps can be grouped to build various solutions.

In some sense Parisian Evolution is making a strong algorithmic choice for the grouping stage, that acts as a constraint on the evolution of the population of lamps. It has the major advantage to reduce the complexity of the problem by providing solutions from the evolution of simpler structures (lamps instead of groups of lamps



(a) *problem_size* = 3 (b) *problem_size* = 5

(c) *problem_size* = 10 (d) *problem_size* = 20

(e) *problem_size* = 100

Fig. 1.7 Box plots of number of lamps used for each *problem_size*.

for Group Evolution or for Classical Evolution). It may be considered as a "quick and dirty" approach.

Future work on this topic will investigate hybridisation between Parisian and Group Evolution, i.e. running a population of elements in the Parisian mode to rapidly get a good rough solution, and using the Group scheme to refine it. A more realistic modelisation of the lamp's light could also be used, taking into account the gradual fading from the source; this approach would shift the problem from enlight-

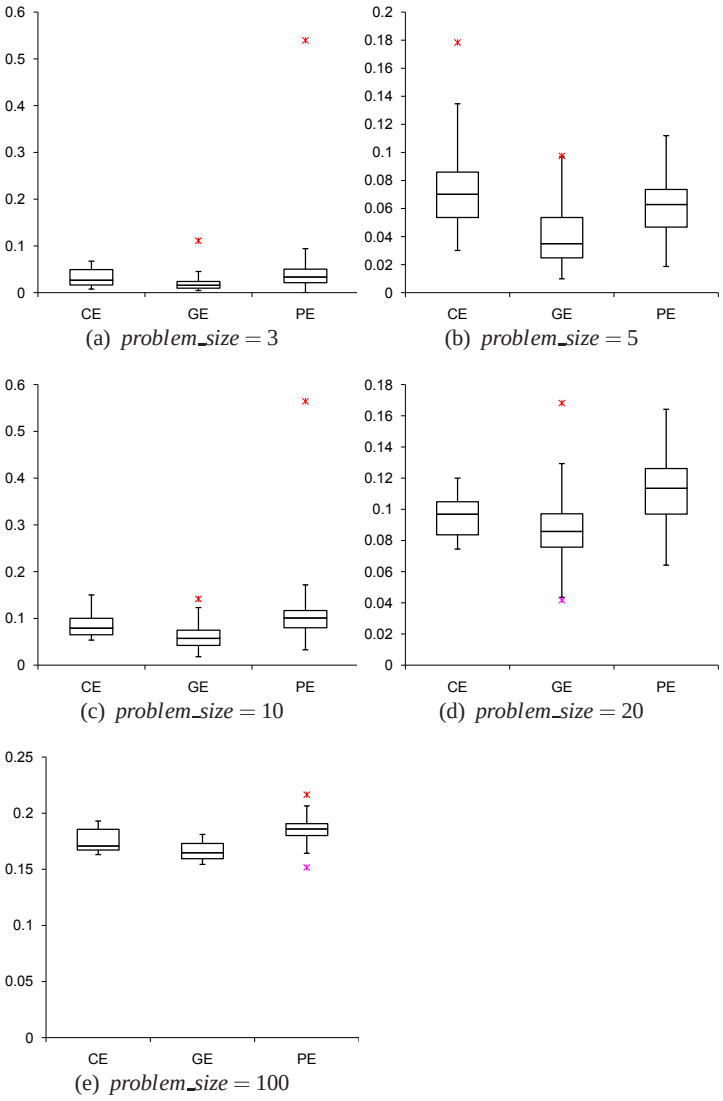
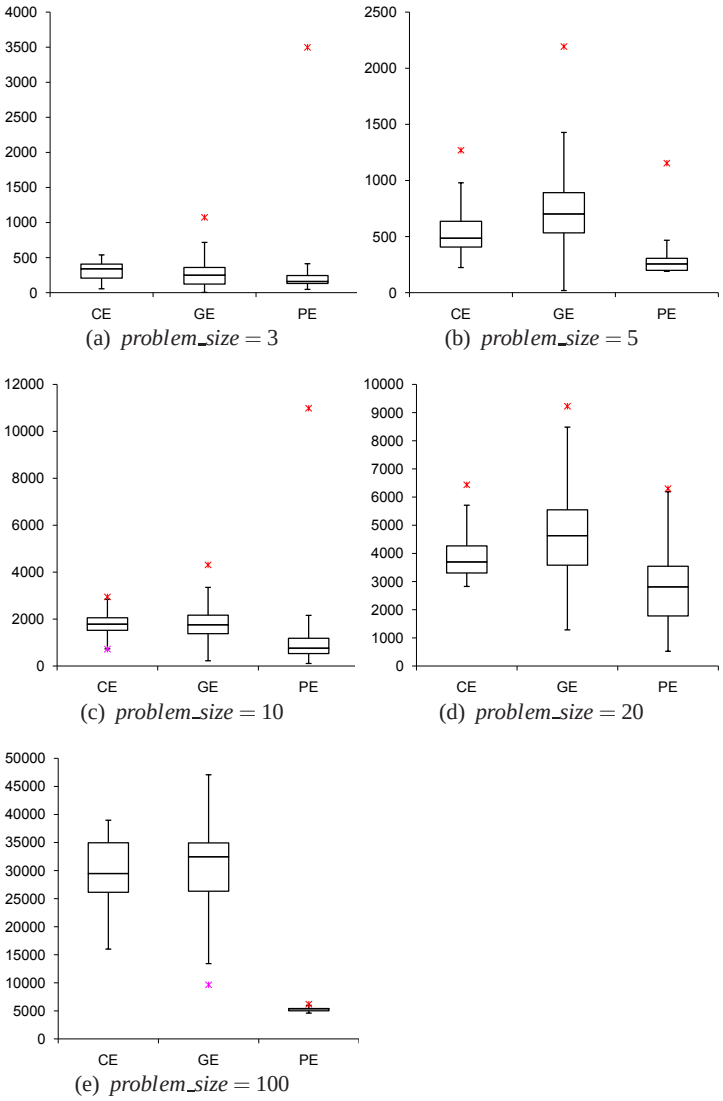


Fig. 1.8 Box plots of overlap percentage for each *problem_size*.

ening the room to ensure that each point of it receives at least a certain amount of light.



(a) *problem_size* $= 3$ (b) *problem_size* $= 5$

(c) *problem_size* $= 10$ (d) *problem_size* $= 20$

(e) *problem_size* $= 100$

Fig. 1.9 Box plots of number of lamps evaluated before reaching an acceptable solution, for each *problem_size*.

References

[1] Amaya JE, Cotta C, Fernndez AJ (2010) A memetic cooperative optimization schema and its application to the tool switching problem. In: PPSN 2010, 11th International Conference on Parallel Problem Solving From Nature, Springer Verlag, Series: Lecture Notes in Computer Science, Vol. 6238 and 6239

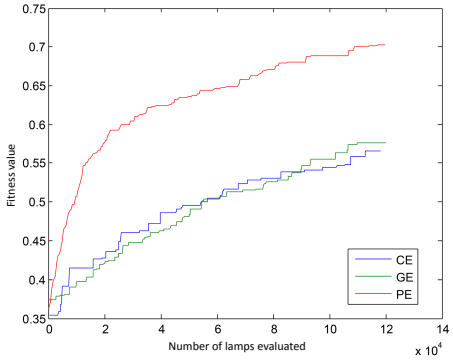


Fig. 1.10 Profile of the best run for each evolutionary algorithm at $problem\_size = 100$.

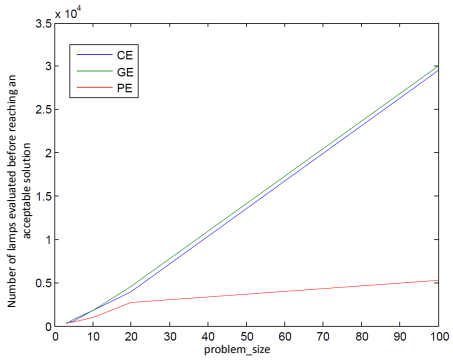


Fig. 1.11 Average number of lamps evaluated before reaching an acceptable solution, for different values of *problem_size*.

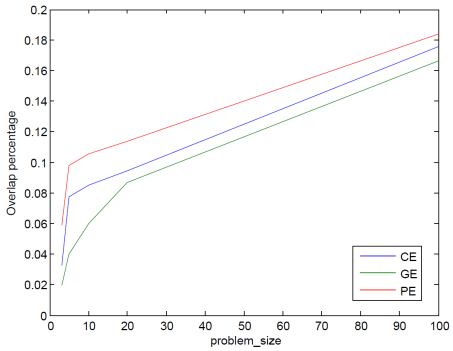


Fig. 1.12 Average overlap, for different values of *problem_size*.

[2] Barrire O, Lutton E, Wuillemin PH (2009) Bayesian network structure learning using cooperative coevolution. In: Genetic and Evolutionary Computation Conference (GECCO 2009)

[3] Bongard J, Lipson H (2005) Active coevolutionary learning of deterministic finite automata. Journal of Machine Learning Research 6:1651–1678
[4] Boumaza AM, Louchet J (2001) Dynamic flies: Using real-time parisian evolution in robotics. In: Boers EJ et al. (eds) Applications of Evolutionary Computing. EvoWorkshops2001, Proceedings, Springer-Verlag, Como, Italy, LNCS, vol 2037, pp 288–297
[5] Bucci A, Pollacj JB (2005) On identifying global optima in cooperative coevolution. In: Proceedings of the 2005 conference on Genetic and evolutionary, Washington DC, USA
[6] Chen W, Weise T, Yang Z, Tang K (2010) Large-scale global optimization using cooperative coevolution with variable interaction learning. In: PPSN 2010, 11th International Conference on Parallel Problem Solving From Nature, Springer Verlag, Series: Lecture Notes in Computer Science, Vol. 6238 and 6239
[7] Collet P, Lutton E, Raynal F, Schoenauer M (2000) Polar ifs + parisian genetic programming = efficient ifs inverse problem solving. Genetic Programming and Evolvable Machines Journal 1(4):339–361, october
[8] De Jong ED, Stanley KO, Wiegand RP (2007) Introductory tutorial on coevolution. In: Proceedings of the 2007 GECCO conference companion on Genetic and evolutionary computation, London, United Kingdom
[9] Dunn E, Olague G, Lutton E (2005) Automated photogrammetric network design using the parisian approach. In: EvoIASP 2005, Lausanne, nominated for the best paper Award
[10] Landrin-Schweitzer Y, Collet P, Lutton E (2006) Introducing lateral thinking in search engines. GPEM, Genetic Programming an Evolvable Hardware Journal, W Banzhaf et al Eds 1(7):9–31
[11] Lutton E, Olague G (2006) Parisian camera placement for vision metrology. Pattern Recognition Letters 27(11):1209–1219
[12] Ochoa G, Lutton E, Burke EK (2007) Cooperative royal road functions. In: Evolution Artificielle, Tours, France, October 29-31
[13] Panait L, Luke S, Harrison JF (2006) Archive-based cooperative coevolutionary algorithms. In: Proceedings of the 8th annual conference on Genetic and evolutionary computation, Seattle, Washington, USA
[14] Popovici E, De Jong K (2006) The effects of interaction frequency on the optimization performance of cooperative coevolution. In: Proceedings of the 8th annual conference on Genetic and evolutionary computation, Seattle, Washington, USA
[15] Potter MA, Couldrey C (2010) A cooperative coevolutionary approachto partitional clustering. In: PPSN 2010, 11th International Conference on Parallel Problem Solving From Nature, Springer Verlag, Series: Lecture Notes in Computer Science, Vol. 6238 and 6239
[16] Sanchez E, Schillaci M, Squillero G (2011) Evolutionary Optimization: the μGP toolkit, 1st edn. Springer

[17] Sanchez E, Squillero G, Tonda A (2011) Group evolution: Emerging synergy through a coordinated effort. In: Proceedings of the 2011 IEEE Congress of Evolutionary Computation (CEC)

[18] Vidal FP, Louchet J, Rocchisani JM, Lutton E (2010) New genetic operators in the fly algorithm: application to medical PET image reconstruction. In: Evolutionary Computation in Image Analysis and Signal Processing, EvoApplications 2010, Part I, LNCS 6024,C. Di Chio et al. (Eds.)

[19] Wiegand RP, Potter MA (2006) Robustness in cooperative coevolution. In: Proceedings of the 8th annual conference on Genetic and evolutionary computation, Seattle, Washington, USA