Group evolution: Emerging synergy through a coordinated effort

Group Evolution: Emerging Synergy through a Coordinated Effort

Ernesto Sanchez*, Giovanni Squillero*, Alberto Tonda*
*DAUIN, Politecnico di Torino,
Corso Duca degli Abruzzi 24, Torino, Italy
Email: {ernesto.sanchez,giovanni.squillero,alberto.tonda}@polito.it

***Abstract*—A huge number of optimization problems, in the CAD area as well as in many other fields, require a solution composed by a set of structurally homogeneous elements. Each element tackles a subset of the original task, and they cumulatively solve the whole problem. Sub-tasks, however, have exactly the same structure, and the splitting is completely arbitrary. Even the number of sub-tasks is not known and cannot be determined a-priori. Individual elements are structurally homogeneous, and their contribution to the main solution can be evaluated separately. We propose an evolutionary algorithm able to optimize groups of individuals for solving this class of problems. An individual of the best solution may be sub-optimal when considered alone, but the set of individuals cumulatively represent the optimal group able to completely solve the whole problem. Results of preliminary experiments show that our algorithm performs better than other techniques commonly applied in the CAD field.**

***Index Terms*—Evolutionary Algorithms, Group Evolution, Synergy, Coordinated Effort.**

I. Introduction

In many fields, and especially in the CAD area, many problems have an optimal solution composed by a set of homogeneous elements, whose individual contribution to the main problem solution can be evaluated separately. In this context, by *homogeneous* we mean elements sharing the same structure. A simple example is the placement of a set of lamps to ensure that a certain area is fully brightened with light. The minimum number of lamps required is unknown, and depends on the topology of area. All lamps are alike, and each one may be evaluated separately with respect to the final goal. In the example, the optimal solution requires 4 lamps (Figure 1, top).

Interestingly, when examined independently, all lamps do waste a certain amount of light outside the target field. However, if the first lamp is positioned in a local optimum, it becomes impossible to brighten up the field with the remaining three (Figure 1, bottom).

The example simplifies a common situation: the optimal solution is composed of homogeneous sub-optimal elements. A more mundane example is the code coverage of the description of a electronic device. Such devices are commonly described using hardware description languages (HDL), like Verilog or VHDL. The HDL *describes* the device, and such a description is evaluated by a simulator to determine how the unite behave in presence of certain stimuli.

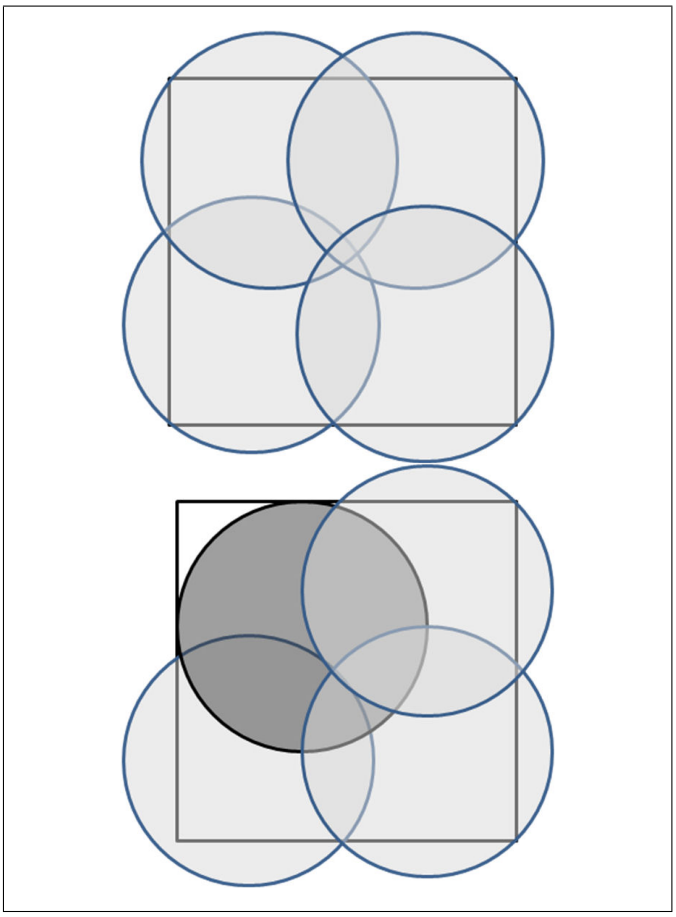

Fig. 1. Placement of a set of lamps. The aim is to enlighten all the square area. It is interesting to notice how a solution where some of the light of each lamp is wasted outside the area (**top**) overall performs better than a solution where the grayed lamp maximizes its own performance (**bottom**).

In order to test all functionalities a large set of stimuli may be required. The problem is strikingly similar to the example of the lamps: each sub-group of input stimuli is able to activate only a part of the device. Verification engineers look for the minimum set able to reach a complete coverage.

The example in Figure 2 shows a simple circuit in VHDL, where three completely distinct test cases are needed in order to excite all the functionalities: the first input greater than the second; the second greater than the first; two identical inputs.

```
library ieee;
use ieee.std_logic_1164.all;

entity SAMPLE is
port(
    X: in std_logic;
    Y: in std_logic;
    F2:out std_logic
);
end SAMPLE;

architecture behv of SAMPLE is
begin
    process(X,Y)
    begin
if(X > Y) then
    F2 <= X or Y; -- behavior #1
elsif(X < Y)
    F2 <= X and Y; -- behavior #2
else
    F2 <= 0; -- behavior #3
endif;
    end process;
end behv;
```

Fig. 2. VHDL description of a simple circuit which needs three distinct test cases in order to be fully covered properly.

Trying to apply an Evolutionary Algorithm (EA) to solve this problem raises a number of significant issues. The push to optimize a single sequence of input stimuli, maximizing the number of *covered* lines, may oppose the global goal of finding a *set of sequences* able to cover the whole design [1]. On the other hand, a certain amount of optimization for each single sequence is usually beneficial. Not being known the number of sequences in the optimal set, defining the individual as the set is also not possible. Furthermore, since all sub-components of the optimal solution are evaluated against the same fitness function, Multi-Objective Evolutionary Algorithms (MOEA) [2] are not applicable as well.

Literature reports different approaches to hardware optimization problems resorting to Evolutionary Computation. The simplest, and probably the most commonly used in practice, consists in iterative runs. Since each individual contribution can be evaluated separately, first the performance of a single individual representing a solution is maximized. Then the part solved by that individual is removed from the original problem. A new problem is obtained, whose solutions does not take into account at all the fraction where the first individual operated: an evolutionary approach can be applied to the second problem and a not complete but complementary solution is found. This process iterates until a set of almost complementary solutions is obtained. Individuals belonging to the final set solve completely the original problem with a small quantity of overlapping features. This approach can lead to good solutions [3], but it usually requires an extensive *a-priori* knowledge of the problem. In the following, we will refer to this procedure as *Multi-Run*.

Another feasible approach is called Parisian evolutionary process: by partitioning the original problem, a series of simpler sub-problems is obtained. Those problems can be solved through the evolution of a single individual. A group of solutions (one from each sub-problem) is aggregated into a composite solution which is then tested against the original problem. New individuals are generated, and the process is repeated until an optimal global solution is found. Interaction between different search spaces is based on adjusting the population fitness values in accordance to the global fitness evaluation of a group. The Parisian evolutionary process has been used in various fields of study [4][5] obtaining good results, especially with regards to the computational time needed to reach an optimal global solution: its implementation, however, requires addressing aspects such as problem decomposition and representation, local and global fitness integration, as well as diversity preservation mechanisms. Again, in order to apply the algorithm effectively, a great theoretical knowledge of the problem and a preliminary study are needed to model problem decomposition correctly: approaching the evolution step without an accurate model would lead to inconsistent results.

On the other side, Orthogonal Evolution of Teams (OET) algorithms have been used with a significant degree of success to evolve the behavior of multi-agent systems [6]. OET proves to be particularly useful where the aim is to obtain teams of heterogeneous individuals with pre-determined roles and very specific competences that are known before the evolution starts. OET algorithms apply pressure on both teams and individuals during selection and replacement, because they alternate between two orthogonal views of the population: as a single population of teams of size N and as a set of N independent populations of individuals, where each population is associated to a specific individual role in the team.

Evolving sets of solutions able to collectively solve a given problem is an interesting and thriving research line in the Evolutionary Computation field [7] [8], with real applications that range from robotics [9] to the study of animal behavior [10].

II. Proposed approach

We propose *Group Evolution*, a general approach to evolve a set of solutions facing problems requiring a solution composed by a set of homogeneous elements. The approach uses a population of partial solutions, and exploits non-fixed sets of individuals called *groups*. Group Evolution acts on individuals and groups, managing both in parallel. During the evolution, individuals are optimized as in a common EA, but concurrently groups are turned into *teams*.

A group in itself does not necessarily constitute a team: teams have members with complementary skills and generate synergy through a coordinated effort which allows each member to maximize his strengths and minimize his weaknesses. A team comprises a group of entities, partial solutions in our case, linked in a common purpose. Teams are especially appropriate for conducting tasks that are high in complexity and have many interdependent subtasks. Remarkably, in our groups of individuals, there are no fixed roles: this feature is

particularly functional with regards to non-separable problems. Pseudo-code for the algorithm is given in Figure 3.

```
1. create initial population (individuals and groups);
2. evaluate initial population (individuals and groups);
3. while (stop condition has not been reached)
4. {
5.   choose group genetic operators;
6.   choose parent groups;
7.   apply group genetic operators to groups;
8.   add offspring to group population;
9.
10.   choose individual genetic operators;
11.   choose parent individuals;
12.   apply individual genetic operators to individuals;
13.
14.   for each new individual
15.   {
16.     create new group;
17.   }
18.
19.   evaluate(new groups);
20.   delete worst groups;
21.   delete individuals not associated to groups;
22. }
```

Fig. 3. Pseudo Code for Group Evolution algorithm.

A. Individuals and groups

The algorithm proposed is population-based: we generate and keep track of a set of distinct individuals which share the same structure. In parallel we manage a set of groups: a groups is defined as a set of individuals in the population. An individual, at any step, is part of at least one group.

At the beginning of the evolutionary process (Figure 3, line 1) the 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 μ is the maximum number of individuals in the population, and it is set by the user before the evolution starts. Minimum and maximum size of the groups are set by the user as well. Figure 4 shows a sample population where minimum group size is 2, and maximum group size is 4.

1) Generation of new individuals and groups: We choose to exploit 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:

1) Application of *group genetic operators* (Figure 3, line 5);
2) Application of *individual genetic operators* (Figure 3, line 10).

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 of genetic operators to apply at each step can be set by the user.

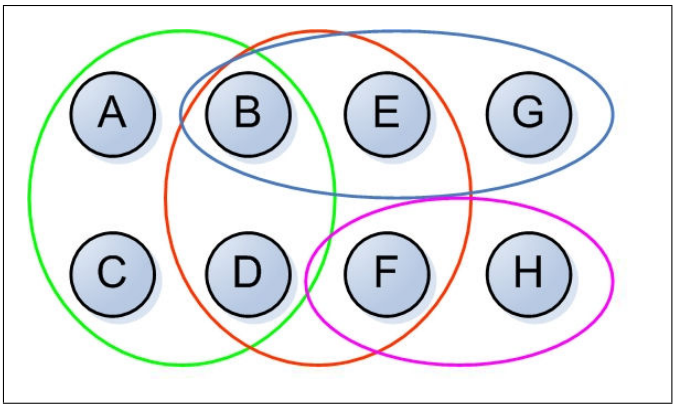

Fig. 4. Individuals and Groups in a sample population of 8 individuals. While individual A is part of only one group, Individual B is part of 3 different groups.

2) 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) **union**: generates offspring by selecting two parents groups. One new group is created with individuals that were in either of the original groups;
3) **separation**: generates offspring by selecting one parent group. The parent group is divided into two (or more), creating new groups;
4) **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;
5) **removal-mutation**: generates offspring by selecting a group and one or more individuals inside it. Individuals are removed from the parent group.

Parent groups are chosen via tournament selection [11].

3) Individual genetic operators: Individual Genetic Operators (IGOs) operate on the population of individuals, very much like they are exploited in usual GA. The novelty we propose 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, we choose to generate a copy of it with the offspring taking the place of the parent.

This approach, however, could lead to an exponential in-

crease 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 5: an IGO, ScanMutation, selects individual C as a parent. The chosen individual is part of only one group, Group 1. ScanMutation 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.

Our aim 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.

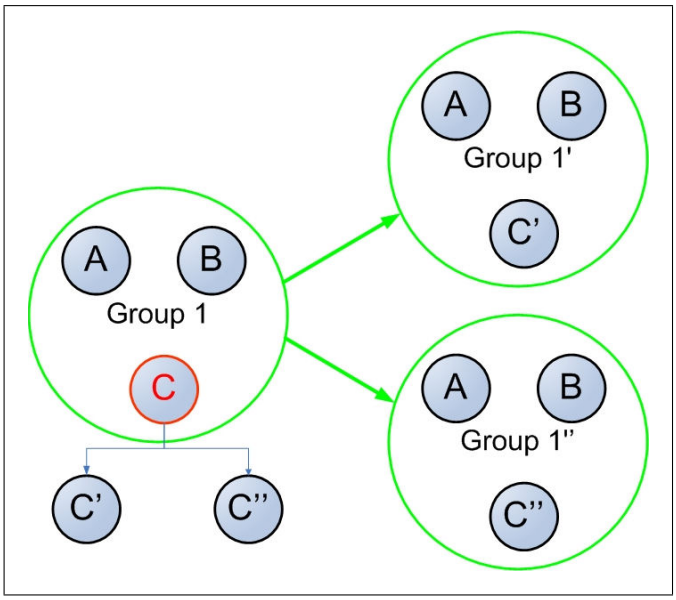

Fig. 5. Effects of ScanMutation, an IGO, applied to individual C. Since individual C is part of Group 1, two groups are created and added to the population.

B. 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, we also assign a fitness value 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 our approach resides in the evaluation step: if we already have a fitness value for an individual that is part of a new group, we can choose 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.

C. Slaughtering

After each generation step, the group population is resized. The groups are ordered fitness-wise and the worst one is deleted until we reach the desired population size. 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.

III. Experimental results

In order to perform the preliminary experiments on our algorithm, we chose to expand an existing EA, μGP [12]. μGP is a versatile and easily expandable tool [13], which has been developed by the CAD Group of Politecnico di Torino in C++ and is available as a GPL software [14]. 32 new classes have been added to the 110 of the original project, to manage and evolve groups of individuals. μGP evolves a population of graphs. Each node in the graph encodes a specific macro with a defined number of variable parameters described in a constraints file. Individuals are translated to text files before being evaluated by an external application.

We tested the proposed algorithm against a classical EA on two different sets of problems. To perform a comparison, the EA is run several times, each time evolving a single individual, following the well-known strategy that we label *Multi-Run*: once a first individual is evolved, we zero the fitness value of the part of the solution space covered by the individual before we start the evolution of the second individual. For the third individual, we exclude the parts of the solution space covered by the first and the second, and so on. The individuals obtained in a Multi-Run approach show a slight decrease of the individual fitness values, but a great improvement in the group fitness values. Determining the specific moment to start the generation of a new individual, however, is not a trivial task, and requires an extensive knowledge of the problem under evaluation: a `steady state` parameter of μGP is exploited to stop evolutionary runs that do not improve their best individual after a certain number of generation steps.

For the sake of comparison, in each experiment the main parameters of *Group Evolution* and *Multi-Run* were selected trying to keep the initial conditions as fair as possible.

A. Lamp placement

As described in the introduction, the placement of a set of lamps to ensure that a certain area is fully brightened with light is a typical problem where a Group Evolution approach could be beneficial, especially if the minimum number of lamps required is unknown, and it is strongly dependent on the topology of area.

In all the subsequent experiments all lamps have a radius of light of 172 m and the objective is to enlighten a square field with a side of 467 m. A lamp can be placed at any point in space between (0,0) and (467,467), thus the search space comprehends $2 \cdot 10^5$ different positions for each lamp. The optimal solution is a set of several position that covers exactly all the field. To mimic a situation often found in real problems, both the Multi-Run and Group Evolution EAs have a limited number of evaluations to find a solution to the problem, orders of magnitude smaller than the search space dimension. We set this limit to $2 \cdot 10^3$.

In a first test, the Multi-Run EA and the Group Evolution EA optimize the placement of exactly 4 lamps. The Multi-Run EA uses a (μ, λ) strategy with parameters μ = 20, λ = 15, σ = 0.9. The `steady state` is set to 10, 20 and 50 generations in different runs, marked as SS 10, SS 20 and SS 50 in I: this means that the evolutionary process of a single individual will stop after that number of generations have elapsed. The Group Evolution algorithm has $\mu_{individuals}$ = 20, μ_{groups} = 20, λ= 15, σ = 0.9 . The possible IGO for each algorithm are `mutation` and `crossover`. The Group Evolution EA also has also all GGOs previously described. For both the algorithms, the fitness value is expressed as a percentage of the field covered by light. Table I shows the comparative results for 100 runs of each algorithm.

	Best Individual Fitness (Average)	Best Individual Fitness (St. Dev.)	Best Group Fitness (Average)	Best Group Fitness (St. Dev.)	Average of Individuals Fitness in Best Group
Multi-Run SS 10	0.43	0	0.92	$1.11 \cdot 10^{-3}$	0.35
Multi-Run SS 20	0.43	0	0.91	$5.54 \cdot 10^{-4}$	0.37
Multi-Run SS 50*	0.43*	0*	0.8*	$3.17 \cdot 10^{-4}$*	0.38*
Group Evolution	0.37	0.01	0.99	$9.85 \cdot 10^{-4}$	0.32

TABLE I
AVERAGE RESULTS FOR 100 RUNS OF MULTI-RUN AND GROUP EVOLUTION, EVOLVING A GROUP OF 4 LAMPS. *WHEN THE `STEADY STATE` PARAMETER IS SET TO 50, THE MULTI-RUN APPROACH OFTEN CONSUMES ALL INDIVIDUALS EVALUATION AT ITS DISPOSAL IN THE THIRD INDIVIDUAL GENERATION, SO THE GLOBAL FITNESS IS LOW.

Interestingly, the Multi-Run approach always finds the best possible individual as the first individual generated (the maximum coverage for a single individual is about 43% of the field), but the groups obtained lack behind in field enlightenment when compared to Group Evolution ones. Vice versa, the best individual of a Group Evolution run is on average less performing, but the group as a whole is able to reach high coverage values on a solid basis: each individual forsakes part of its potential fitness to benefit the whole group. The average of individuals' fitness in the best group is similar, but in the Multi-Run approach there is a greater difference between the best and the worst. The performance of the Multi-Run approach is also strongly influenced by the number of steady state generations after which the algorithm stops, that we set tentatively to 10, 20 and 50. Even in such a simple problem, setting the `steady state` parameter is not trivial and it can be performed mainly tentatively.

In a second experiment, the only limit given to both the Multi-Run and the Group Evolution EA is the maximum number of evaluations, set to $1 \cdot 10^4$. The Multi-Run algorithm is free to generate as many individuals as it can, and it stops once the maximum possible fitness is obtained or when the maximum number of evaluations is reached: the parameters are μ = 20, λ = 15, σ = 0.9 and `steady state` is set to 10, 20 and 50 in different runs, marked as SS 10, SS 20 and SS 50 respectively in II. The Group Evolution is set to work with groups ranging from 2 to 8 individuals and parameters $\mu_{individuals}$ = 20, μ_{groups} = 20, λ= 15, σ = 0.9 . Groups obtained by the Group Evolution are also rewarded if the number of individuals is low. The results of 100 runs of this second experiment are reported in Table II.

	Average Individual Fitness (Average)	Average Individual Fitness (St. Dev.)	Best Group Fitness (Average)	Best Group Fitness (St. Dev.)	Best Group Size (Average)	Best Group Size (St. Dev.)
Multi-Run SS 10	0.31	0.14	1	0	7.67	0.33
Multi-Run SS 20	0.31	0.03	0.99	$7.42 \cdot 10^{-6}$	7.32	0.67
Multi-Run SS 50	0.37	0.02	0.93	$1.52 \cdot 10^{-3}$	4	0
Group Evolution	0.28	0.06	1	0	6.11	0.42

TABLE II
AVERAGE RESULTS FOR 100 RUNS OF MULTI-RUN AND GROUP EVOLUTION, EVOLVING A GROUP WHERE THE NUMBER OF LAMPS IS NOT FIXED.

The Multi-Run approach is able to always reach an complete enlightenment of the field if the `steady state` parameter is suitably selected: for `steady state` 20, however, about 11% of the runs do not reach the optimal coverage, because the algorithm terminates by meeting the maximum number of evaluations condition. The same happens with `steady state` 50: this time, the algorithm seeks small improvements in the single individuals, obtaining sub-optimal groups with a limited size in each run. The results obtained by Group Evolution show a remarkable consistency in achieving the optimal solution with respect to the enlightenment of the field, and the average number of individuals in the best group is also lower. Again, the average fitness of the individuals in the best

group is lower, with individuals forsaking part of their fitness to improve the overall performance of the group.

B. Arena coverage

Another simple optimization problem consists in creating a team of robots capable of efficiently exploring a circumscribed space. Initially we have a 10 m x 10 m empty arena, divided into 100 squares of 1 m x 1 m: each individual represents a small robot able to wander inside the arena, executing a block of instructions, one instruction per instant of time. There are only two different kinds of instructions:

1) **Move**: the robot moves 1 square along the direction it is facing;
2) **R+/R-**: the robot performs a rotation of +45 (R+) or -45 (R-) around its vertical axis (thus, a robot can only face N, NE, E, SE, S, SW, W, NW).

Each robot starts from a specific square in the arena, facing North. A group of robots is given only a limited amount of time to explore the arena.

Our objective is to evolve a set of individuals which can collectively visit the maximum possible numbers of squares in the arena in a given time. The fitness of an individual is the number of squares it touched during its path. Group fitness, on the other side, is computed by adding up the number of squares visited by only one individual composing the group; in this way, an implicit penalty is provided to groups containing individuals that intersect their trajectories. It is interesting to highlight that information cannot be shared between individuals.

In this experiment, we evolve a group of 4 individuals, that together try to maximize the arena coverage. For the Multi-Run strategy, we use μ = 150, λ = 100, `steady state` = 5 and a generational approach with an elitist strategy that preserves the best 4 individuals. Table III summarizes the results obtained.

For the Group Evolution we have $\mu_{individuals}$ = 150, μ_{groups} = 150, λ = 100, and groups ranging from 2 to 4 individuals. Table IV shows the results for the Group Evolution.

Number of Evaluations	Avg. Fitness (Individual)	Avg. Fitness (Group)
10,000	30.72	91.84
20,000	30.15	92.61
30,000	31.53	92.90
60,000	31.92	93.8

TABLE III
INDIVIDUALS OBTAINED THROUGH A MULTI-RUN APPROACH, FORCING A MAXIMUM OF FOUR INDIVIDUALS.

Groups produced by Group Evolution outperform those obtained by Multi-Run, this time evolving even fitter individuals, due to the problem's characteristics: unlike what happens in the lamp placement case study, an individual can significantly improve the fitness of its group without sacrificing its own performance. The evolved groups' behavior show that individuals cooperate to explore the arena, filling the gaps of the other members of their team.

Number of Evaluations	Avg. Fitness (Individual)	Avg. Fitness (Group)
10,000	32.15	95.41
20,000	32.47	97.39
30,000	33.72	99.25
60,000	34.02	99.46

TABLE IV
INDIVIDUALS OBTAINED THROUGH A GROUP EVOLUTION APPROACH, WITH GROUPS RANGING FROM 2 TO 4 INDIVIDUALS.

IV. CONCLUSIONS

Problems that have an optimal solution composed by a set of homogeneous elements are becoming more and more common in CAD and in other fields of study.

We propose an EA capable of performing *Group Evolution*, thus obtaining a collection of individuals that together represent a viable solution to a given problem. Unlike other approaches that rely on an extensive a-priori knowledge both of the problem and of the role each individual should play in the global solution, our algorithm can obtain good results starting only with the individual description.

Preliminary experiments show that our algorithm, with respect to the objective of creating sets of solutions that together reach a goal, performs better than other techniques currently used in the CAD field, such as Multi-Run testing. Future works will exploit the Group Evolution algorithm to solve real-world testing-related problems.

ACKNOWLEDGMENT

The authors would like to thank Sonia Drappero, Danilo Ravotto and Massimiliano Schillaci for their useful ideas and invaluable advices.

REFERENCES

[1] J. H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. The University of Michigan Press, 1975.

[2] J. D. Schaffer, "Multiple objective optimization with vector evaluated genetic algorithms," in *Proceedings of the 1st International Conference on Genetic Algorithms*. Hillsdale, NJ, USA: L. Erlbaum Associates Inc., 1985, pp. 93–100. [Online]. Available: http://portal.acm.org/citation.cfm?id=645511.657079

[3] E. Sanchez, M. Sonza Reorda, and G. Squillero, "Test program generation from high-level microprocessor descriptions," in *System-level Test and Validation of Hardware/Software Systems*, ser. Springer Series in Advanced Microelectronics, M. Sonza Reorda, Z. Peng, and M. Violante, Eds. Springer London, 2005, vol. 17, pp. 83–106, 10.1007/1-84628-145-8_6. [Online]. Available: http://dx.doi.org/10.1007/1-84628-145-8_6

[4] E. Dunn, G. Olague, and E. Lutton, "Parisian camera placement for vision metrology," *Pattern Recognition Letters*, vol. 27, no. 11, pp. 1209 – 1219, 2006, evolutionary Computer Vision and Image Understanding. [Online]. Available: http://www.sciencedirect.com/science/article/B6V15-4HX477K-2/2/e82b5b25f9a7a82607ac4b30c9fb9c45

[5] ——, "Automated photogrammetric network design using the parisian approach," in *Applications on Evolutionary Computing*, ser. Lecture Notes in Computer Science, F. Rothlauf, J. Branke, S. Cagnoni, D. W. Corne, R. Drechsler, Y. Jin, P. Machado, E. Marchiori, J. Romero, G. D. Smith, and G. Squillero, Eds. Springer Berlin / Heidelberg, 2005, vol. 3449, pp. 356–365, 10.1007/978-3-540-32003-6_36. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-32003-6_36

[6] R. Thomason, R. Heckendorn, and T. Soule, "Training time and team composition robustness in evolved multi-agent systems," in *Genetic Programming*, ser. Lecture Notes in Computer Science, M. O'Neill, L. Vanneschi, S. Gustafson, A. Esparcia Alcazar, I. De Falco, A. Della Cioppa, and E. Tarantino, Eds. Springer Berlin / Heidelberg, 2008, vol. 4971, pp. 1–12, 10.1007/978-3-540-78671-9_1. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-78671-9_1

[7] M. Waibel, L. Keller, and D. Floreano, "Genetic Team Composition and Level of Selection in the Evolution of Cooperation," *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 3, pp. 648–660, 2009.

[8] M. Potter and K. De Jong, "A cooperative coevolutionary approach to function optimization," in *Parallel Problem Solving from Nature PPSN III*, ser. Lecture Notes in Computer Science, Y. Davidor, H.-P. Schwefel, and R. Mnner, Eds. Springer Berlin / Heidelberg, 1994, vol. 866, pp. 249–257, 10.1007/3-540-58484-6_269. [Online]. Available: http://dx.doi.org/10.1007/3-540-58484-6_269

[9] M. Quinn, L. Smith, G. Mayley, and P. Husbands, "Evolving teamwork and role-allocation with real robots," in *Proceedings of the eighth international conference on Artificial life*. Cambridge, MA, USA: MIT Press, 2003, pp. 302–311. [Online]. Available: http://portal.acm.org/citation.cfm?id=860295.860344

[10] G. M. Werner and M. G. Dyer, "Evolution of herding behavior in artificial animals," in *Proceedings of the second international conference on From animals to animats 2 : simulation of adaptive behavior: simulation of adaptive behavior*. Cambridge, MA, USA: MIT Press, 1993, pp. 393–399. [Online]. Available: http://portal.acm.org/citation.cfm?id=171174.171220

[11] C. Oei, D. Goldberg, and S. Chang, "Tournament selection. niching and the preservation of diversity," *lliGAL Report*, vol. No. 91011, 1991.

[12] E. Sanchez, M. Schillaci, and G. Squillero, *Evolutionary Optimization: the μGP toolkit*, 1st ed. Springer, to be published in July 2011.

[13] F. Corno, E. Sanchez, and G. Squillero, "Evolving assembly programs: how games help microprocessor validation," *Evolutionary Computation, IEEE Transactions on*, vol. 9, no. 6, pp. 695–706, Dec. 2005.

[14] SourceForge. Host of μgp3. http://sourceforge.net/projects/ugp3.