# How an Optimal Observer can Collapse the Search Space

Christophe Philemotte
cphilemo@iridia.ulb.ac.be

Hugues Bersini
bersini@ulb.ac.be

IRIDIA, CoDE, Université Libre de Bruxelles
Université Libre de Bruxelles CP 194/6
50 Avenue F.D. Roosevelt
B-1050 Brussels
Belgium

## ABSTRACT

Many metaheuristics have difficulty exploring their search space comprehensively. Exploration time and efficiency are highly dependent on the size and the ruggedness of the search space. For instance, the Simple Genetic Algorithm (SGA) is not totally suited to traverse very large landscapes, especially deceptive ones. The approach introduced here aims at improving the exploration process of the SGA by adding a second search process through the way the solutions are coded. An "observer" is defined as each possible encoding that aims at reducing the search space. Adequacy of one observer is computed by applying this specific encoding and evaluating how this observer is beneficial for the SGA run. The observers are trained for a specific time by a second evolutionary stage. During the evolution of the observers, the most suitable observer helps the SGA to find a solution to the tackled problem faster. These observers aim at collapsing the search space and smoothing its ruggedness through a simplification of the genotype. A first implementation of this general approach is proposed, tested on the Shuffled Hierarchical IF-and-only-iF (SHIFF) problem. Very good results are obtained and some explanations are provided about why our approach tackles SHIFF so easily.

**Categories and Subject Descriptors:** I.2.8 Problem Solving, Control Methods, and Search: Heuristic methods

**General Terms:** Algorithms, Design, Experimentation, Performance

**Keywords:** Landscape, Observer, Representation, Simple Genetic Algorithm, Speedup Technique

## 1. INTRODUCTION

A key element of research dedicated to evolutionary algorithms and optimisation in general is the discovery of mechanisms to improve the search. Many metaheuristics and their hybridisations are invented and compared on their capacity and efficiency to traverse the search space. Most metaheuristics try to find a good balance between exploitation and exploration to travel the search space in the most efficient way [2]. The exploration efficiency is highly dependent on the size and ruggedness of the search space. A metaheuristic like the Simple Genetic Algorithm (SGA) is not totally suited for traversing very large landscapes, especially deceptive ones [12, 18]. Each metaheuristic proposes its own compromise between exploration and exploitation. For wide search space, the SGA spends most of the time trying to discover a promising area in which to refine the search.

Any clever way to reduce the time needed to discover interesting areas of the search space is beneficial for any optimisation method. The notion of Intrinsic Emergence (IE), which was originally inspired by the developments of Crutchfield and Mitchell, appears to be very helpful here [16, 3, 4, 1, 15, 14, 8, 7]. According to these authors, there are mechanical ways to observe a complex system so as to provide this mechanical observer with added and profitable functionalities. Here an observer, by modifying the coding of the solutions, helps the optimisation process by simplifying and reducing the search space. Another level at which observers are optimised is then added. From a cognitive point of view, a human does not take into consideration all details when performing a searching process, such as finding the shortest driving path for a trip. He first concentrates on the global map and adopts a coarse level of observation. Only then, he focuses on the details to refine the path. In the same vein, our algorithm improves the SGA through the use of high-level observers modifying and simplifying the whole process.

An observer is defined as a mechanism which, by imposing a different view on the problem (a simpler encoding), alters the way the SGA traverses the landscape. The observers compete among each other since some of them provide a better view for the problem under investigation. This competition allows us to rank the observers according to their quality and select the best one. The new approach proposed here selects the observers for a specific time and decides which of them is the most suitable to solve the complete problem. This training can be done sequentially or in parallel, i.e. in a co-evolutionary fashion. In a same vein, other algorithms, such as the Delta Coding algorithm [22], adapt the encoding of the problem.

While previous works aims at a conceptual introduction of the approach and a rough illustration by an evolutionary optimisation of a Cellular Automaton [15, 14], this pa-


*GECCO'06,* July 8–12, 2006, Seattle, Washington, USA.

per attempts a generalisation to all metaheuristics with special focus on evolutionary ones. To test this generalisation, the problem to be tackled is one instance of Shuffled HIFF (SHIFF) [20, 21, 19, 5]. The choice of this problem is justified by its well-known deceptive characteristics and by the fact that it poses many problems for basic GA-like algorithms [21, 6, 17]. Our main goal here is to convince readers that this added optimised observer can lead to a possible improvement of the behaviour of these basic algorithms. We do not aim at an exhaustive experimental comparison with all metaheuristics and all possible improvements or hybrids for many problem instances but just to illustrate the potentiality of this idea on a classical and popular optimisation problem known in the community to be hard for GA. The plan of the paper goes as follows. Section 2 introduces the different ways to add the observers and their selection on the classical sequence of steps of the SGA. Section 3 explains the nature of the observers and the way they considerably reduce the search space of the problem. Sections 4 and 5 present the SHIFF problem and the results we obtained by the addition and the selection of the observers in comparison with the direct SGA.

## 2. OBSERVER ALGORITHM

Our algorithm offers a way to assist the exploration process of a given metaheuristic. An observer defines how the metaheuristic can look at the search space: it is like a clever lens used by the metaheuristic to explore the landscape. This additional help includes two main steps (see Algorithm 1): the efficiency evaluation `EvaluateMetaheuristicRunWithObserver(...)` and the building `BuildAnotherObserver(...)`.

**Algorithm 1** Canonical Skeleton of the proposed approach

$O(0) \leftarrow$ `BuildAnotherObserver`()
$t \leftarrow 0$
**while** termination conditions not met **do**
  $SO \leftarrow$ `EvaluateMetaheuristicRunWithObserver`$(O(t))$
  {$SO$ temporarily stores the fitness of the Observer $O(t)$}
  $O(t+1) \leftarrow$ `BuildAnotherObserver`$(O(t), SO)$
  $t \leftarrow t+1$
**end while**
**return** best solution found by observed metaheuristic {i.e. a solution of the treated problem, not an observer}

The first step evaluates the quality of an observer: according to the solutions found, the observer is rewarded as a function of the performance that it helps the metaheuristics to obtain (`EvaluateMetaheuristicRunWithObserver(...)`) and the reward is stored in a temporary variable $SO$. This evaluation is a function of the quality of the solutions obtained when such an observer is applied. The way this quality is defined in the SGA case is discussed below.

The second step builds a new lens i.e. a new observer $O(t+1)$ with respect to the previous one $O(t)$. The next observer construction is directed by the fitness $SO$ (`BuildAnotherObserver(...)`) and depends on the encoding and its effect in the problem search space.

## 3. THE ADDITION OF COMPETITIVE OBSERVERS ON SGA

A second search process is engaged here but, this time, in the space of the observers. Again, any search algorithm could be used for improving the observer solutions. In a preliminary attempt to evolve Cellular Automata, the observers were generated in a random and unguided way [1]. In a successive attempt, an evolutionary search was proposed and tested [15, 14]. Here we follow this second evolutionary search in the space of the observers. It is an iterated process to optimise the way the solutions are coded and therefore to improve the progression of the SGA [11, 12, 18] in this "observed" space (see Algorithm 3). The termination criterion depends on the chosen search algorithm in the space of the observers. The current implementation of this search is again based on a SGA (see Algorithm 2) transforming the whole algorithm as an intertwining of two evolutionary stages (through `Evaluate(...)`). The `Evaluate(...)` function corresponds to the `EvaluateMetaheuristicRunWithObserver(...)` and the observer evaluation function is then defined by [14]:

$$f_{obs}(O(t)) = \sqrt{\overline{f_{sol}} \times \max(f_{sol})}\,, \qquad (1)$$

where $\overline{f_{sol}}$ is the average of the individual fitness over the population and $\max(f_{sol})$ is the maximum of these fitness values. An observer should be as good as it allows a population of average good quality and an excellent best individual. And the `Select(...)`, `ApplyReproductionOperators(...)` and `Replace(...)` functions of the Algorithm 2 compose the `BuildAnotherObserver(...)` from the Algorithm 1.

**Algorithm 2** Evolutionary Implementation of our meta-(metaheuristic)

Generate initial observer population $O(0)$
Generate initial solution population $P(0)$
$t_1 \leftarrow 0$
**while** termination conditions not met **do**
  `Evaluate`$(O(t_1), P(0))$ {Defined in Algorithm 3}
  $O'(t_1) \leftarrow$ `Select`$(O(t_1))$
  $O''(t_1) \leftarrow$ `ApplyReproductionOperators`$(O'(t_1))$
  $O(t_1+1) \leftarrow$ `Replace`$(O(t_1), O''(t_1))$
  $t_1 \leftarrow t_1 + 1$
**end while**
**return** best solution found in $P$ with best found observer $O$

## 4. ENCODING OF THE OBSERVER

The observer aims at improving the SGA by altering the landscape, hence the encoding of the individuals. It aims at sampling the search space and restricting it to the most informative areas. When coding each individual as a chromosome, the observer effect is to structurally group the genes. All genes are linked by the relation attributed to their corresponding group. Currently, all genes in a group are simply assigned with the same value (allele) (see Algorithm 4). So, the search space is projected onto a hyperplane.

The observer defines a function that maps a gene location onto a group location. Each gene is assigned to a specific group. The genes are divided into groups (see Figure

**Algorithm 3** `Evaluate`($O$: an Observer, $P$: a solution population) implementation for an EA

$Q(0) \leftarrow$ `ObservePopulation`$(P, O)$
$t_2 \leftarrow 0$
**while** termination conditions not met **do**
  `Evaluate`$(Q(t_2))$
  $Q'(t_2) \leftarrow$ `Select`$(Q(t_2))$
  $Q''(t_2) \leftarrow$
  `ApplyReproductionOpsWithObservation`$(Q'(t_2), O)$
  $Q(t_2 + 1) \leftarrow$ `Replace`$(Q(t_2), Q''(t_2))$
  $t_2 \leftarrow t_2 + 1$
**end while**
**return** efficiency of observation and best solution, i.e. $f_{obs}(O)$

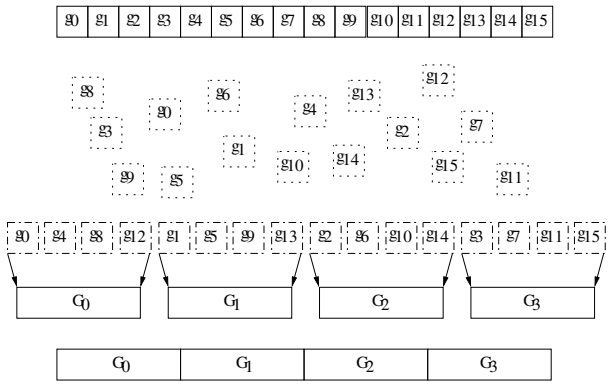


**Figure 1: This figure presents how genes are grouped. First, the genes are assembled into groups in a way defined by the oberver. The genes are linked by the relation attributed to their corresponding group.**

1). This assignment and mapping functions partially reconstruct the encoding of the treated problem. The function works as described in Algorithm 4 and 5. The `map(...)` function computed the boolean *AND* (noted by $\wedge$) of a gene locus $i$ and a given observer $O$, both encoded as binary number. The `BuildGroup(...)` computes the genes group, i.e. the observer effect on problem encoding. Theses functions compose the `ObservePopulation(...)` from Algorithm 3. Therefore, an observer is encoded in a bits string of length $l_O = \lceil \log_2(l_C) \rceil$ and defines the number of groups as $2^{N_1}$, where $N_1$ is the number of **1** bit in the observer chromosome. In the case of an observer of $N_1$ of **1** bits, the reduction in size of the search space is from $2^{l_C}$ to $2^{2^{N_1}}$. Since $N_1$ goes from 0 to $l_O$, depending on the observer, the size of the search space goes from 1 to $l_C$. The greater the number of **1** in the observer coding the less this observer reduces the precision of the coding and consequently the size of the search space.

**Algorithm 4** $l =$ `map`$(i, O)$

$i$ {a given binary number for a gene locus}
$O$ {a given binary number for a given observer}
$l \leftarrow i \wedge O$ {$l$ is a binary number for the group locus}
**return** $l$

**Algorithm 5** `BuildGroup`($C$: a chromosome of length $l_C$, $O$: an observer)

**for all** $i$ such that $0 \leq i \leq l_C$ **do**
  $l \leftarrow$ `map`$(i, O)$
  $C(i) \in G(l)$
**end for**

For sake of clarity, let's explain what happens in the simplest case of individuals coded on four binary genes and an observer coded on two binary genes. Four observers are possible which all correspond to a different way to group the four genes. The four genes have locus: **00**, **01**, **10** and **11**. For instance, the **11** observer means that all genes belong to four different groups. Indeed, by performing the logical *AND* between the respective locus and the observer, the locus remains unchanged. Again, as a result of the *AND* operation, the **01** observer now distributes the four genes in two groups: a first group contains the gene located in **00** and **10**, and the second group contains the genes located in **01** and **11**. Finally the observer **00** makes all genes to belong in one only group.

## 5. SHUFFLED HIFF

Our algorithm is tested on the Shuffled Hierarchical IF-and-only-iF (SHIFF) problem which is a maximisation problem [20, 21]. HIFF is the canonical version of a specific class of problems modelling the interdependency between building blocks [10, 21, 5]. This kind of problems turns out to be very interesting for different reasons. They are tools to better understand the compositional and accretive mechanism of a SGA [21, 19]. They are also representative of hierarchies that we could identify in natural systems. Above all, HIFF is very difficult for the SGA, especially the shuffled version (SHIFF). SGA does not efficiently explore the search space and looses itself in the fractal landscape of HIFF (see Figure 2): SGA does not succeed in traversing through multiple local optima [21]. Different specific algorithms have been designed to tackle HIFF problem and its shuffled version like SEAM [21, 19], Hierarchical-GA [5, 6] or Compact Genetic Codes [17]. We use the fitness function $f_{hiff}$ of HIFF to compute the fitness of a given binary string [21]:

$$f_{hiff}(B) =$$

$$\begin{cases} 1 & if\ |B| = 1 \\ |B| + f_{hiff}(B_L) + f_{hiff}(B_R) & if\ |B| > 1 \\ & and\ \forall i\ b_i = 0\ or\ b_i = 1 \\ f_{hiff}(B_L) + f_{hiff}(B_R) & otherwise \end{cases}, \quad (2)$$

where $B$ is a block of bits $\{b_1, \ldots, b_n\}$, $|B|$ is the size of the block B i.e. $n$, $b_i$ is the $i^{th}$ element of $B$, $B_L = \{b_1, \ldots, b_{n/2}\}$ and $B_R = \{b_{1+n/2}, \ldots, b_n\}$ are the left and right halves of $B$ and $n = l_C$ is the size of the binary string and must be an integer power of 2. The HIFF problem and its shuffled version can be summarised as the maximisation of this fitness function $f_{hiff}$.

The tackled SHIFF is $n = l_C = 128$ bits long and generated according to the Watson's implementation[1]. This problem size gives a maximum fitness of 1024 for both optima, i.e. binary strings of only **1** or **0**. As explained

[1]The watson's source code is available on `http://www.cs.brandeis.edu/ richardw/hiff.html`.

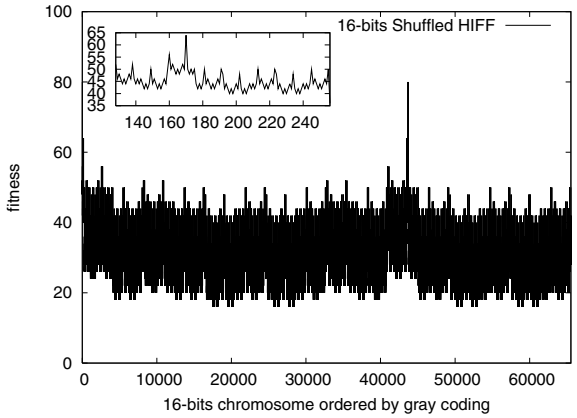


**Figure 2: The fitness landscape is fractal. This results from the recursive definition of modules and sub-modules. It is clearly visible on this. The figure shows the landscape for the 16-bits one. They give a good idea of the landscape ruggedness and the number of local maxima [21].**

above, the observers are encoded by a bits string of length $l_O = \lceil \log_2(128) \rceil = 7$. We compare our approach with two other algorithms: a single SGA and a SGA with randomly generated observers. The comparison is made through the following performance evaluation, i.e. what best solution can be found following a given number of fitness function evaluations. This is the most logical way to compare SGA without and with the addition of observers, both random and evolved.

# 6. RESULTS

In this section, we experimentally answer to the following questions:

1. Does the addition of observers improve the performance of SGA?
2. Can we use macroscopical observation[2] to collect useful information?
3. Which level of details is needed to observe efficiently?
4. How can we find good observers?
5. Does this generic method help SGA to tackle harder problems and classical combinatorial problems?

We compare the basic SGA with RO-SGA (in this case the observers are randomly generated) and EO-SGA (in this case the observers are being evolved by a second SGA). The main criterion of interest when comparing algorithms is their scalability, i.e. the impact of the problem size on the computation time. For difficult problems, the cost in computation time is mainly determined by the evaluation of fitness values. This is why progresses in the GA community often aim at reducing this number. We therefore measure the best solution fitness $s*$, which is computed by $f_{hiff}$ (see Equation (2)), as a function of the number of evaluations $n_f$. The fitness of the best observer $o*$, which is evaluated by $f_{obs}$ (see Equation (1)), is also measured in order to investigate the second question above. For each parameters set (to be described below), the obtained results are summarised by drawing two statistics as a function of the number of evaluations:

- a measure of central tendency, i.e. the center or middle of a frequency distribution, by the calculation of the median which is marked by a small dark and bold horizontal bar,
- and a measure of statistical dispersion[3] by the calculation of the range and of the interquartile range. The range is represented by a box. The interquartile range is represented by two vertical lines. The first line is plotted between the maximum and the third quartile and the second one between the minimum and the first quartile.

For each experiment, we run the algorithms 30 times on 50 different instances of SHIFF: the data set contains 1500 samples. These measures provide a fair comparison between the algorithms performances and give answers to most of the questions raised above except the third one. This question will be investigated later through the study of the average number of **1** bits in the best observer chromosome. The more **1**s we have, the more detailed is the observation. This number provides some indications on the capacity of the algorithm to autonomously tune the observation level, i.e. the information compression of the search space still sufficient to efficiently travel it.

SGA uses non-overlapping populations, roulette wheel selection, one point crossover, flip mutation and the elitist heuristic. Both populations of the solutions and the observers are initialised in an uniformly distributed way. The probabilities of crossover and mutation are 0.7 and 1/128 respectively. The SGA needs two parameters: the size of the solutions population $p_{sol}$ and the maximum number of generations $g_{sol}$. The EO-SGA is composed of two intertwined evolutionary stages and needs two additional parameters: the size of the observers population $p_{obs}$ and the maximum number of observer generations $g_{obs}$. However, in the case of the RO-SGA, only one more parameter is needed: the maximum number of random generation of observer $g_{obs}$. The used parameters sets are listed in Table 1.

As shown in [21], the SGA can not exceed a fitness around 400, even for higher evaluation numbers. By adding some heuristics like "deterministic crowding diversity maintenance", Watson has experimentally shown that SGA is unable to escape from local optima and hardly reaches a fitness around 700. We obtain similar results for the basic SGA (see Figure 3). The small results dispersion indicates how hard it is to travel through the SHIFF fractal landscape for a basic SGA.

The RO-SGA provides a wide fitness range (see Figure 4.b). The observers are randomly generated and they do not adapt their zoom properly. Due to the uniform distribution, each observer resolution is generated with the same frequency. The observer fitness $o*$ provides information on

[2]By macro observation, we mean a measure of a global property.

[3]Statistical dispersion is quantifiable variation of measurements of differing members of a population within the scale on which they are measured.

| | SGA | RO-SGA | EO-SGA |
|---|---|---|---|
| $p_{sol}$ | $\{20, 200, 2000\}$ | $\{20, 50, 200\}$ | $\{1, 5, 10, 20, 200, 2000\}$ |
| $g_{sol}$ | $\{5, 10\}$ | $\{1, 10, 100\}$ | $\{1, 2, 5, 10\}$ |
| $p_{obs}$ | - | $\{1\}$ | $\{1, 2, 5, 10\}$ |
| $g_{obs}$ | - | $\{1, 2, 5\}$ | $\{1, 2, 5, 10\}$ |
| $\max(n_f)$ | $p_{sol}g_{sol}$ | $p_{sol}g_{sol}g_{obs}$ | $p_{sol}g_{sol}p_{obs}g_{obs}$ |
| $\overline{n_f}$ | $P_{cross}P_{mut}\max(n_f)$ | $\sim \max(n_f)$ | $P^2_{cross}P^2_{mut}\max(n_f)$ |
| $s*$ | Fig. 3 | Fig. 4.b | Fig. 5.b |
| $o*$ | — | Fig. 4.a | Fig. 5.a |
| $\overline{\mathbf{1}}$ | — | Fig. 7 | Fig. 6 |

**Table 1: Parameters sets for each algorithm (SGA, EO-SGA and RO-SGA): solution population $p_{sol}$, solution generations $g_{sol}$, observer population $p_{obs}$ and observer generations $g_{obs}$. With respect to these parameters, we also calculate the maximum number of evaluations $\max(n)$ and its probabilistically weighted estimation $\overline{n}$. The crossover probability is $P_{cross} = 0.7$ and the mutation probability is $P_{mut} = 1/128$. Both are used for each SGA. The bottom rows summarised the content of the figure.**

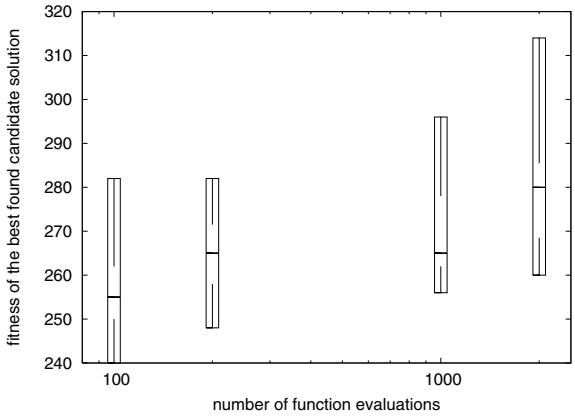


**Figure 3: Results for basic SGA. The statistical measurements of best obtained solution fitness $s*$ are plotted with respect to the number $n_f$ of fitness evaluations. The SGA cannot exceed a fitness of around 300 to 400 even for higher evaluation numbers. By adding some heuristics like "diversity maintenance", Watson has experimentally shown that the SGA is unable to escape from local optima and hardly reaches a fitness of around 700 [21].**

how good the observed population is with respect to $f_{obs}$ definition (see Figure 4.a). For higher evaluation numbers, the median is bigger: SGA spends more time to explore the observer search space and increases the chance to find a good one.

Beyond around 500 evaluations, the EO-SGA efficiently computes the maximum solution fitness (see Figure 5.b). Around 1000 evaluations, no more dispersion is observed (see Figure 5.b). By examining the Figures 5.a and 5.b, we notice that the EO-SGA is capable of finding the maxima following 10 evaluations. The way the observers evolve depends on their quality evaluated by the evaluation function $f_{obs}$ given above (see Equation (1)). After 1000 evaluations, the EO-SGA finds an adequate observer and the resulting observed population which contains the two optima. The convergence of the dispersion of $s*$ and $o*$ seems to support the original intuition that observer could really help the SGA to efficiently travel through the SHIFF rugged landscape. Compared to the random version, we also show the utility of evolving the observers as a function of how well they progress and how well they adapt to the search space of the problem. So, we have answered the questions above, apart from the third one:

- Our EO-SGA drastically improves the basic SGA.
- The observer provides an adapted way to "observe" the landscape so as to better traverse it. With observer, only useful information is retained to explore the SHIFF search space.
- Our evaluation function $f_{obs}$ rightly determines the quality of an observer. The evolution of the observers by means of this second SGA is an adequate and faster way to find them.
- The whole algorithm seems indeed to be well adapted to hard combinatorial problems.

Several specific algorithms have been implemented to tackle hierarchical problems such as de Jong's HGA [5, 6] or Watson's SEAM [21, 19]. Is our algorithm a competitive approach? We compared our algorithm performances with the HGA performance. Considering a 256-bits SHIFF, our EO-SGA is about ten times faster than HGA (see Table 2). Over 1000 different instances (and only one run), the EO-SGA finds the two optima in at most 5000 iterations and

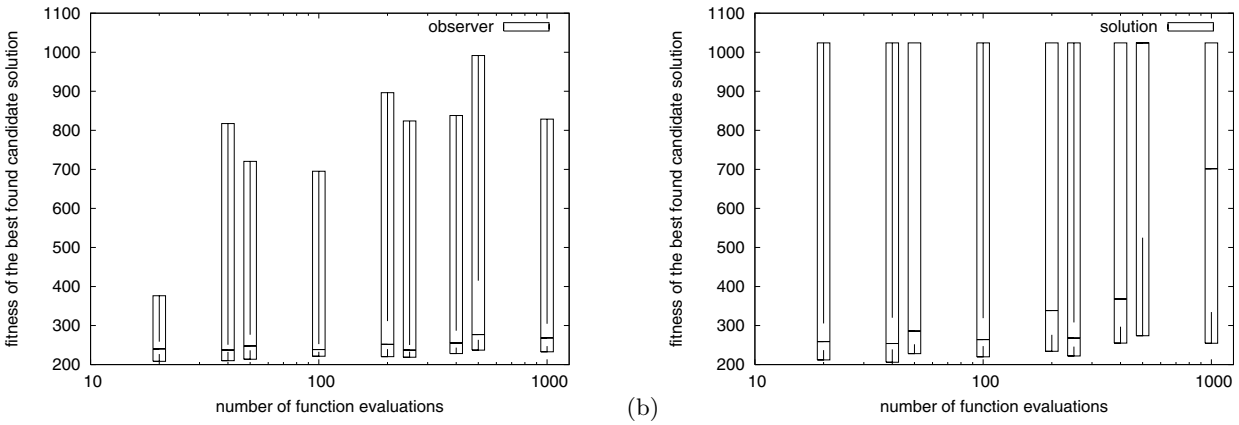


**Figure 4: Results for RO-SGA. The statistical measurements of the best obtained observer fitness $o*$ (a) and of the best obtained solution fitness $s*$ (b) are plotted in function of the number $n_f$ of fitness evaluations.**

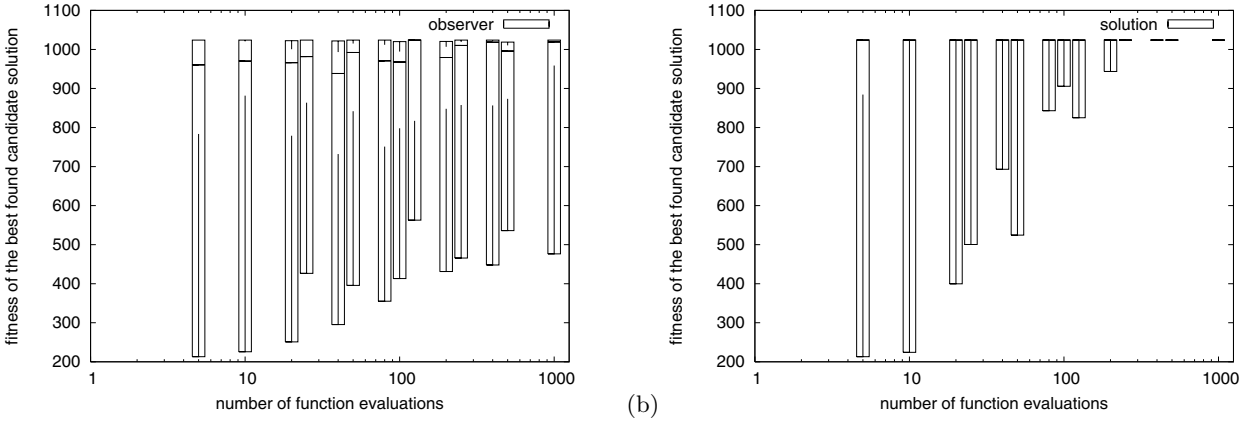


**Figure 5: Results of EO-SGA. The statistical measurements of the best obtained observer fitness $o*$ (a) and of the best obtained solution fitness $s*$ (b) are plotted in function of the number $n_f$ of fitness evaluations. Beyond $n_f \sim 500$, the EO-SGA found the maximum solution fitness with a very small dispersion, which is reduced to zero at $n_f \sim 10^3$.**

| | HGA | EO-SGA |
|---|---|---|
| evaluations number $n_f$ | $\sim 10^5$ | $\sim 5.10^3$ |
| whole processing time | $\sim$ 60s | $\sim$ 5s |

**Table 2: Our EO-SGA is compared with SEAM and HGA performances on the 256-bits SHIFF problem.**

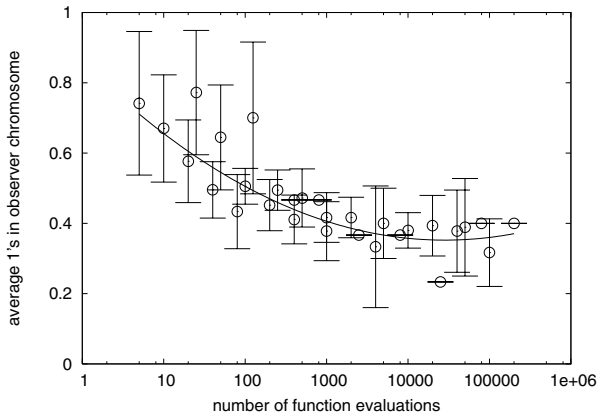


**Figure 6: Average of 1s in the best observer chromosome. For both parameter sets of EO-SGA.**

run in 5s on average[4]. Why is SHIFF such an easy problem for EO-SGA?

We propose an first explanation through the study of the mean of **1**s of the best found observer chromosome. If the observer is encoded by only **0**s, the 128 bits which represent the solution chromosome, compose one single group. Only two solutions are possible and the search space falls from $2^{128}$ down to $2^1$. With one **1**, the bits are distributed in two genes groups and only four candidate solutions are possible. Examining the Figures 6, the best observers contain around 0.5 bit on average. Because of the genes are assigned to the same value of the group they belong, and because of the simple mapping function (see Algorithm 4), our first simple implementation of the Observer Algorithm is very adequate to tackle the SHIFF problem. We have to improve its design to make more general the way how the groups are bought and the genes are linked in a group. The more time the EO-SGA has, the better and coarser the observers are. Figure 7 shows that by randomly generating the observers, the number of **1**s i.e. the precision of the observer is equal to the mean over all uniformly distributed 7-bits string, $3.5 = 7/2$. These results explain why and how the SHIFF problem turns out to be such an easy one for our approach. We also better understand the role of the precision of the observer, the third question raised above.

[4]The computer is a 1GHz AMD processor. The SGA implementation is not especially optimised. For instance, dynamic libraries are used.

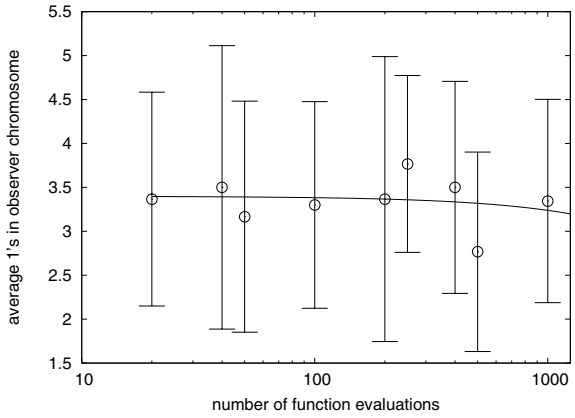


**Figure 7: Average of 1s in the best observer chromosome for RO-SGA.**

# 7. CONCLUSION

Starting from original works discussed in [16, 3, 4, 1, 15, 14], we propose a new approach called EO-SGA. Its main goal consists in improving the basic SGA exploration process. The principle is based on the evolution of observers which alter the way the SGA looks at the landscape and travels through it [15, 14]. This alteration changes the encoding by reducing and smoothing the search space to more informative subsets. We have shown the benefits gained by using our algorithm on the SHIFF problem experimentally [20]. The results confirm the improvement of the SGA and EO-SGA outperforms specific algorithm such as the de Jong's HGA [6]. The addition of these adapted observers allows us to find the best hierarchical level, i.e. the toppest hierarchical level is the best level which "solves" the strong genetic interdependency. So, using these observers, the SGA restricts its search at this level and quickly finds the best solutions due to the collapse of the search space size.

There are still many studies to complete the proposed approach that could be summarised in three main axes of research: experiments, new designs, and theoretical study. New experiments should tackle more realistic problems such as the Travelling Salesman Problem or other kinds of hierarchical problems such as the HXOR problem. These experiments will provide the necessary information to gain a deeper understanding of our approach and improve its design (i.e. the building of group, the relation between genes in a same group, the collaboration between each observed population, ...). For instance, new design should incorporate the coevolution of observers by allowing interaction among the population of solutions obtained by each observer. Finally, the observers can be treated as genetic operators just like crossover and mutation in Vose's theory [18] since it reduces the set of accessible candidate solutions. We would try to provide a theoretical treatment of our Observer Algorithm. Some issues are remain largely opened such as the generalisation of the approach and how well it extends to the machine learning framework in general as well as to the whole family of metaheuristics.

## Acknowledgments

The authors would like to thank Tom Lenaerts for his advices and Iridians for their help.

## 8. REFERENCES

[1] H. Bersini. Whatever emerges should be intrinsically useful. In *Artificial life 9*, pages 226–231. The MIT Press, 2004.

[2] C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Survey*, 35(3):268–308, 2003.

[3] J. Crutchfield. Is anything ever new? considering emergence. In D. P. G. Cowan and D. Melzner, editors, *Integrative Themes*, volume XIX of *Santa Fe Institute Studies in the Sciences of Complexity*. Addison-Wesley Publishing Company, Reading, Massachusetts, 1994.

[4] J. Crutchfield and M. Mitchell. The evolution of emergent computation. In *Proceedings of the National Academy of Science*, volume 23, page 103, 1995.

[5] E. D. de Jong, D. Thierens, and R. A. Watson. Hierarchical genetic algorithms. In X. Yao, E. Burke, J. A. Lozano, J. Smith, J. J. Merelo-Guervós, J. A. Bullinaria, J. Rowe, P. T. A. Kabán, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature - PPSN VIII*, volume 3242 of *LNCS*, pages 232–241, Birmingham, UK, 2004. Springer-Verlag.

[6] E. D. de Jong, R. A. Watson, and D. Thierens. On the complexity of hierarchical problem solving. In *GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 1201–1208, New York, NY, USA, 2005. ACM Press.

[7] A. Defaweux, T. Lenaerts, and J. I. van Hemert. Evolutionary transitions as a metaphor for evolutionary optimisation. In *Advances in Artificial Life, 8th European Conference, ECAL 2005, Canterbury, UK, September 5-9, 2005, Proceedings*, volume 3630 of *Lecture Notes in Computer Science*, pages 342–352. Springer, 2005.

[8] A. Defaweux, T. Lenaerts, J. I. van Hemert, and J. Parent. Transition models as an incremental approach for problem solving in evolutionary algorithms. In *Genetic and Evolutionary Computation Conference, GECCO 2005, Proceedings, Washington DC, USA, June 25-29, 2005*, pages 599–606. ACM, 2005.

[9] S. Forrest and M. Mitchell. Relative building-block fitness and the building block hypothesis. In L. D. Whitley, editor, *FOGA*, pages 109–126. Morgan Kaufmann, 1992.

[10] S. Forrest and M. Mitchell. What makes a problem hard for a genetic algorithm? some anomalous results and their explanation. *Machine Learning*, 13:285–319, 1993.

[11] D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Professional, 1989.

[12] M. Mitchell. *An introduction to genetic algorithms*. MIT Press, Cambridge, MA, USA, 1996.

[13] M. Mitchell, S. Forrest, and J. H. Holland. The royal road for genetic algorithms: Fitness landscapes and ga performance. In F. J. Varela and P. Bourgine, editors, *Proceedings of the First European Conference on Artificial Life*, Cambridge, MA, 1992. MIT Press.

[14] C. Philemotte and H. Bersini. Coevolution of effective observers and observed multi-agents system. In *Advances in Artificial Life, 8th European Conference, ECAL 2005, Canterbury, UK, September 5-9, 2005, Proceedings*, volume 3630 of *Lecture Notes in Computer Science*, pages 785–794. Springer, 2005.

[15] C. Philemotte and H. Bersini. Intrinsic emergence boosts adaptive capacity. In *GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 559–560, New York, NY, USA, 2005. ACM Press.

[16] L. Steels. Towards a theory of emergent functionality. In *Proceedings of the first international conference on simulation of adaptive behavior on From animals to animats*, pages 451–461, Cambridge, MA, USA, 1990. MIT Press.

[17] M. Toussaint. Compact genetic codes as a search strategy of evolutionary processes. In A. H. Wright, M. D. Vose, K. A. D. Jong, and L. M. Schmitt, editors, *Foundations of Genetic Algorithms, 8th International Workshop, FOGA 2005, Aizu-Wakamatsu City, Japan, January 5-9, 2005, Revised Selected Papers*, volume 3469 of *Lecture Notes in Computer Science*, pages 75–94. Springer, 2005.

[18] M. D. Vose. *The Simple Genetic Algorithm: Foundations and Theory*. MIT Press, Cambridge, MA, USA, 1998.

[19] R. A. Watson. A computational model of symbiotic composition in evolutionary transitions. *Biosystems*, 69(2–3):187–209, 2003.

[20] R. A. Watson, G. Hornby, and J. B. Pollack. Modeling building-block interdependency. In *PPSN V: Proceedings of the 5th International Conference on Parallel Problem Solving from Nature*, pages 97–108, London, UK, 1998. Springer-Verlag.

[21] R. A. Watson and J. B. Pollack. *Compositional evolution: interdisciplinary investigations in evolvability, modularity, and symbiosis*. PhD thesis, Brandeis University, 2002. Adviser-Jordan B. Pollack.

[22] L. Whitley, K. Mathias, and P. Fitzhorn. Delta coding: An iterative search strategy for genetic algorithms. In *Proc. of the Fourth International Conference on Genetic Algorithms*, pages 77–84, San Diego, CA, 1991.