A Study in Function Optimization with the Breeder Genetic Algorithm

Lluís A. Belanche

Secció d'Intel·ligència Artificial.
Dept. de Llenguatges i Sistemes Informàtics.
Universitat Politècnica de Catalunya.
c/Jordi Girona Salgado 1-3
08034 Barcelona, Spain.
belanche@lsi.upc.es

Abstract

Optimization is concerned with the finding of global optima (hence the name) of problems that can be cast in the form of a function of several variables and constraints thereof. Among the searching methods, *Evolutionary Algorithms* have been shown to be adaptable and general tools that have often outperformed traditional *ad hoc* methods. The *Breeder Genetic Algorithm* (BGA) combines a direct representation with a nice conceptual simplicity. This work contains a general description of the algorithm and a detailed study on a collection of function optimization tasks. The results show that the BGA is a powerful and reliable searching algorithm. The main discussion concerns the choice of genetic operators and their parameters, among which the family of Extended Intermediate Recombination (EIR) is shown to stand out. In addition, a simple method to dynamically adjust the operator is outlined and found to greatly improve on the already excellent overall performance of the algorithm.

1 Introduction

A wide range of difficult problems or subproblems in Artificial Intelligence (AI) can be cast in the form of a function optimization problem (a FOP). When cast this way, these problems have been traditionally addressed by the Operations Research community. However, the quest for even better and more general searching algorithms has never stopped. Since the 70's, new and powerful heuristic methods have emerged that are particularly well suited for FOPs —although this was not exactly their original purpose— mainly because of their generality, robustness, and conceptual (though not necessarily analytical) simplicity. In addition, the constant need for general-purpose optimization techniques has widen their horizon and boosted their widespread use. Three of these methods are Simulated Annealing (SA), Tabu Search (TS) and Evolutionary Algorithms (EA). A modern analysis of SA, TS and EA and their possible combinations and applications can be found in [31]. Simulated Annealing was introduced in [26] and, nearly thirty years later, the method was suggested as a feasible FOP solver [25]. Tabu Search is newer [19], and has attracted a lot of interest in the last years. While SA is stochastic, TS is a deterministic procedure. Both methods perform a kind of neighbourhood search and have a means to deal with suboptimal solutions (local optima of the search space). The term Evolutionary Algorithms [6] is very general and includes many methods that have been (and are being) developed independently in the last 30 years. All of them are based on techniques that mimic or are inspired in population genetics, and have the added appeal of being easily parallelazible (both intuitively and physically). Among them, the Breeder Genetic Algorithm (BGA) has been one of the last to emerge [29]. Nevertheless, despite its promising initial results when compared to other methods (evolutionary or not) it has not attracted a great deal of attention, possibly because of the enormous impact of the other —in a sense already classical— Evolutionary Algorithms (mainly Evolution Strategies and Genetic Algorithms).

When trying to assess the goodness of a new algorithm, or when attempting the difficult task of fine-tuning an existing one, it is desirable to have a carefully designed test suite of FOPs, both challenging and diverse. The main interest in devising an artificial FOP relies in that it is much more controllable than real-world ones (in the ideal case, the position and value of the best solution are known; topology is, to some degree, also known; it is scalable to any number of dimensions; there is a known degree of non-linearities, of noise, of symmetries, etc). A collection of such FOPs constitutes a useful test suite for use in benchmarking tasks. Whatever the motivation, once the empirical behaviour of an algorithm among a set of known artificial problems has been assessed, the knowledge and experience gained can be transferred to the solution of real-world problems.

In the remainder of this paper, the possibilities of the Breeder Genetic Algorithm are studied to gain such knowledge, with the following four main purposes in mind:

1. To explore how well the BGA copes with a given test suite. To this end, a set of classical test functions are analyzed and a subset thereof are used to evaluate the algorithm;

2. To ascertain what parameter settings are generally better. A detailed study on the different recombination and mutation settings is presented;

3. To compare the BGA, whenever other results are available, to other optimization techniques. This is performed on a particular function, F8F2, in high dimensions (up to 200);

4. In addition, a new heuristic appliable to recombination operators is proposed and initially tested.

The results, which are presented in detail in Sections 7, 8 and in the Conclusions, show that the BGA is a powerful and robust search algorithm, well suited for continuous optimization. This assertion is based on the light of the extremely good results obtained for all of the FOPs tested, for a variety of population sizes (from 50 to 400 individuals). In particular, the function F8F2 (see Section 5) is solved to satisfaction at all dimensions tested, and the results are shown to be superior to those obtained with other optimization techniques.

Regarding the choice of genetic operators and settings the results show that, first, mutation is highly dependent on its parameters (specially on the *precision* parameter) and that the continuous version of this operator is in general more reliable than the discrete one, although the results are inconclusive on that. Second, there are significative differences among recombination operators, though none can be said to be markedly inferior or superior in *all* situations. We find EIR (Extended Intermediate Recombination) to be the most reliable one, regardless of the value of its single parameter δ. In other words, this operator is always at least as good as the others for some value of δ. To set this parameter, apart from the traditional fixed values (ranging to $\delta = 0$ to $\delta = 0.35$), a method for dynamically setting its value (called $range_\delta$) is proposed, and shown to greatly improve on performance. Nevertheless, it is also found that —even in the limited scope of this study— there are FOPs for which other operators do specially well, as for example DR (Discrete Recombination) on F7 (Schwefel's function).

The report is organized as follows. In Section 2 a Function Optimization Problem (FOP) is defined, compiling a collection of aspects that pose it as a generally difficult problem. In Section 3, the basics of an Evolutionary Algorithm are outlined from a conceptual point of view. In Section 4 the Breeder Genetic Algorithm is introduced as a particular case of EA, with special emphasis on the analysis of the set of genetic operators available for it. The general properties that a suitable artificial FOP should exhibit are surveyed in Section 5, along with a description of the classical test suite used for EA benchmarking and how well it fulfills the properties.

The experiments are centered on mutation and recombination operator settings (the specific operators and the choice of their parameters) to assess to what degree they influence algorithm performance. The experimental description begins at Section 6. The general experimental setup, a set of working hypotheses, and an explanation of how the results are presented are then in order. Section 7 is entirely devoted to mutation whereas Section 8 is dedicated to recombination. Both sections are respectively ended with comments on the results obtained and some guidelines about how to interpret them. The last part of the work —in Section 9— concentrates on a particular function, F8F2, using it as a test bed for optimization in high dimensions. The paper ends with the conclusions along with a summary of the experience gained from the experiments performed and proposals for future work.

2 Function Optimization Problems

Many real-world problems are naturally or can somehow be described as a function, a scalar expression depending of a normally fixed set of *decision variables*, often inside a given *range*, and perhaps in the presence of *constraints* on these variables. This function is then to be minimized or maximized globally, although this distinction is of no importance, since conversion is always possible by realizing that

$$\max_{\vec{x}} F(\vec{x}) = -\min_{\vec{x}} F(-\vec{x})$$

In the following, it is assumed that the function is to be minimized. When the variables are all discrete (for example, integers) the term *combinatorial* optimization is used. A FOP is formulated as:

$$\text{Minimize } F(\vec{x}) : D \longrightarrow \mathbb{R}$$

$$\text{subject to } c_i(\vec{x}) \geq 0, \quad 1 \leq i \leq m.$$

where $\vec{x} = (x_1, x_2, \ldots, x_n)$ is the vector of variables, $D = D_1 \times D_1 \times \ldots \times D_n$ and $x_i \in R_i$. The $R_i \subseteq D_i$ are the domains of definition of variable i, the set of possible values the variable can take. If the variable is continuous, it is a real interval of the form $R_i = [r_i^-, r_i^+] \subseteq \mathbb{R}$, called the range. If it is discrete, then R_i is a finite set, possibly with an order relation defined on it. The specific form of the $c_i(\vec{x})$ defines classes of problems. One of the best known is given by restricting $F(\vec{x})$ and $c_i(\vec{x})$ to be linear, and the variables x_i to be in $\mathbb{Q}$ (the set of rational numbers), traditionally tackled by *linear programming* techniques, such as the Simplex algorithm.

A solution to a FOP requires finding an $\vec{x}^*$ such that,

1. $\forall \vec{x} \in D : F(\vec{x}^*) \leq F(\vec{x})$
2. $\forall i : 1 \leq i \leq m : c_i(\vec{x}^*) \geq 0$

We will call a solution *infeasible* if it fulfills condition 1. but not condition 2. In general, this optimization is a difficult problem, due to the following aspects of $F(\cdot)$:

- A complicated (and most of the times completely unknown) topology, characterized by strong *multimodality*, i.e. the existence of several (may be thousands) local minima $\vec{x}'$ such that

  $$\exists \epsilon > 0 : \forall \vec{x} \in D : ||\vec{x} - \vec{x}'|| < \epsilon \Rightarrow F(\vec{x}') \leq F(\vec{x})$$

  with several difficult characteristics as isolated global optima, many peaks, troughs, flat plateaus, discontinuities, etc.

- The presence of the constraints, very difficult to handle by most algorithms, because the set of possible solutions is further restricted to just a subset of those included in the domain of the variables. That is, the original space D shrinks to

  $$D^* = \{\vec{x} \in D : c_i(\vec{x}) \geq 0, \quad \forall i : 1 \leq i \leq m\}$$

  For many algorithms, including EAs, it is useful to include the constraints and $F(\cdot)$ in a functional form into $F^*(\cdot)$ that gives worse values the more the constraints are violated. The rationale of this is that it is good for the search to temporarily exit the space of feasible solutions to enrich it, because in many cases the best feasible ones are borderline cases, and in this way the search gets more balanced. The other reason is that there are problems for which no feasible solution may exist, and thus it is desirable to have the best non-feasible solution as the result of the search (that is, the one less violating the constraints). This technique has been shown to work very well [24] provided two conditions are fulfilled:

1. The penalty term $\pi(\vec{x})$ incorporated —either in an additive or multiplicative way— into $F(\cdot)$ is a function of the distance from feasibility (e.g. from the closest borderline feasible solution).
2. The best infeasible solution can never be assigned a lower (that is, better) value than the worst feasible one.

There is also a possible parameter λ to control the influence of both terms, such that

$$F^*(\vec{x}) = f\{F(\vec{x}), \lambda, \pi(\vec{x})\}$$

where a typical setting could be

$$F^*(\vec{x}) = \lambda F(\vec{x}) + (1 - \lambda)\pi(\vec{x})$$

where

$$\lambda = \begin{cases} 1 & \text{if } \vec{x} \text{ is feasible} \\ 0 & \text{otherwise} \end{cases}$$

and

$$\pi(\vec{x}) = - \sum_{c_i(\vec{x})<0}^{n} c_i(\vec{x}) = \sum_{c_i(\vec{x})<0}^{n} |c_i(\vec{x})|$$

is the penalty term.

- The presence of noise.
- The large dimensionality of $F(\cdot)$ (high values of n). In general, all algorithms suffer with increasing numbers of variables, especially if they interact non-trivially. Also, it is well possible that some of these variables are irrelevant for the problem.
- The strong non-linearities between variables.
- The (possible) non-differentiability of $F(\cdot)$ due to the presence of discrete variables, or any cause of discontinuity, or even the presence of dimensionless variables.
- The (possible) time-varying behaviour of $F(\cdot)$, characteristic of living systems, although this aspect is rarely touched by FOP solvers. In practical terms, this carries the assumption that the topology of the function does not change during optimization.
- The imprecision or vagueness inherent in all kinds of variables, an issue not addressed by most methods. For example, coding information that is known to be imprecise as an (exact) crisp value, or vague concepts like "tall" as discrete nominal values. This is also an oversimplification that has its consequences in a loss of flexibility and expressive power.

Even if the general FOP is unsolvable, for some practical problems a small improvement can make a real difference because of the potential cost savings.

3 Basics of an Evolutionary Algorithm

The term Evolutionary Algorithms refers to a big family of search methods based on concepts taken from Darwinian evolution of species and natural selection of the fittest. Some concepts from genetics are also present. Given a problem to be solved —usually in the form of a FOP— EAs maintain a population of *individuals* that represent potential solutions to it. Each individual in the population is represented by a *chromosome* consisting of a string of atomic elements called *genes*. Each gene contains (represents) a variable, either for the problem or for the algorithm itself. The possible values of a gene are called *alleles* and the gene's position in the chromosome is called *locus* (pl. *loci*). There is also a distinction between the *genotype*, the genetic material of an individual, and the *phenotype*, the individual result of genotype development (that is, the born living thing). In EAs the genotype coincides with the chromosome, and the phenotype is simulated via a *fitness function*, a scalar value —similar to a reinforcement— expressing how well and individual has come out of a given genotype[1]. However, there are many differences with natural evolution, reviewed in [1].

The search process usually starts with a randomly generated population and evolves over time in a quest for better and better individuals where, from generation to generation, new populations are formed by application of three fundamental kinds of operators to the individuals of a population, forming a characteristic three-step procedure:

1. *Selection* of the fittest individuals, yielding the so-called *gene pool*;
2. *Recombination* of (some of) the previously selected individuals forming the gene pool, giving rise to an offspring of new individuals;
3. *Mutation* of (some of) the newly created individuals.

By iterating this three-step mechanism, it is hoped that increasingly better individuals will be found (that is, will appear in the population). This reasoning is based in the following ideas:

1. The selection of the fittest individuals ensures that only the best ones[2] will be allowed to have offspring, driving the search towards good solutions, mimicking the natural process of selection, in which only the more adapted species are to survive.
2. By recombining the genetic material of these selected individuals, the possibility of obtaining an offspring where *at least* one child is better than any of its parents is high.
3. Mutation is meant to introduce new traits, not present in any of the parents. It is usually performed on freshly obtained individuals by slightly altering some of their genetic material.

There is a last operation involved, the *replacement* criterion, that basically says which elements, among those in the current gene pool and their newly generated offspring, are to be given a chance of survival onto the next generation. There are two basic strategies, generically denoted by (μ, λ) (the *comma* strategy) and $(\mu + \lambda)$ (the *plus* strategy). The letter μ denotes the population size and the letter $\lambda > \mu$ the number of offspring to be generated out of the μ elements. In the plus case, both the parents and their (recombined and mutated) offspring will be taken into account to form a new generation of again μ elements. In the comma case the parents, after generating offspring, die off and are not taken into account to form the next generation.

Thus, an EA may be seen as a non-empty sequence of ordered operator applications: fitness evaluation, selection, recombination, mutation and replacement. The entire process iterates until one of the following criteria is fulfilled:

1. *Convergence*: it happens because the individuals are too similar. Fresh and new ideas are needed, but recombination is incapable of providing them because the individuals are very close to one another, and mutation alone is not powerful enough to introduce the desired variability. Convergence can be monitored by on-line (average of the best individuals) and off-line (average of average individuals) throughout the generations;
2. *Problem solved*: the global optimum is found up to a satisfactory accuracy (if optimum known);
3. *End of resources*: the maximum number of function evaluation has been reached.

Evolutionary Algorithms are effective mainly because their search mechanism keeps a well-balanced tradeoff between *exploration* (trying to always drive the search to the discovery of new, more useful, genetic material) and *exploitation* (trying to fine-tune good already-found solutions). Exploration is

[1] In other disciplines, like Artificial Life methods, the phenotype is a real (or simulated) entity that interacts with an environment.

[2] Or the luckiest in some EA instances, like most GAs.

mainly dealt with by the mutation operator. Exploitation is carried out by the selection process and the use of recombination operators, although mutation may also play a role in the fine-tuning of solutions. The fitness function is built out of the function to be optimized (called the *objective function*). All EAs represent the decision variables in the chromosome in one way or another, either directly as real values (like ESs) or resorting to a discrete coding, usually binary (like most GAs). The particular coding scheme is the classical knowledge representation problem in AI, and completely conditions the results. In addition, some algorithms (like ESs) append their own variables to the representation in the form of auxiliary information that evolves with time like the other variables.

According to the representation scheme chosen, there must be a decoding method $\Gamma(\cdot)$ —equivalent to the genotype to phenotype development— to decode the decision variables from their chromosomic representation

$$\Gamma : i \in \Pi_t \rightarrow \vec{x} \in D$$

where Π_t stands for the population at a certain generation t. Once decoded, these variables can readily be used as arguments of the objective function $F(\cdot)$ to yield a fitness value. The fittest individuals are those with a lowest (in case of minimization) fitness value. Thus, the fitness function $\Phi(\cdot)$ of an individual i is composed of a decoding function $\Gamma(\cdot)$ and applies to it the objective FOP $F(\cdot)$ or some variant of it $F^*(\cdot)$ to be solved, to yield the fitness value associated with each individual, $\Phi(i) = F^*(\Gamma(i))$. Some EAs require a form of post-processing such as a global rescaling function, but it is much more convenient to consider it as part of the selection mechanism itself.

An EA can be formally described by the conceptual algorithm in fig. 1, parameterised by a tuple:

$$< \text{EA-Setup} > = < \Pi_0, (\mu, \lambda), \Upsilon, \Omega, \Psi, \Theta, \Phi, \Xi >$$

where $\Pi_t = (i_1^t, i_2^t, \ldots, i_\mu^t)$ is the population at time t and thus Π_0 is the, usually random, initial population, μ the population size, λ the offspring size (out of μ), Υ the selection operator, Ω the recombination operator, Ψ the mutation operator, Θ the termination criterion, Ξ the replacement criterion and Φ the fitness function. In this algorithm, operator sequencing on the population is as follows: Π_t represents the population at time (i.e., generation) t, Π_t^Υ the population after selection, Π_t' after recombination and Π_t'' after mutation, to end in a new population Π_{t+1}.

The three main representatives of EAs are: Genetic Algorithms, proposed by Holland [22], then settled [23], and made popular [20]; Evolution Strategies, developed by Rechenberg [32] and Schwefel [34], during the 60's and more or less settled in the 70's [33], [35]; and Evolutionary Programming (EP), introduced by Fogel [16] and spread by him and his coworkers [17], an approach that resembles ESs although they were developed independently. One of the main references to EAs is [6]; other, good and brief surveys can be found in [1], [9]. An excellent state-of-the-art and review of EAs, and a useful departure point because of its rich set of references is [2]. Modern surveys and introductions to specific algorithms are [3] and [4] for ESs; [28] and [39] for GAs; and [18] for EP. There is also a very complete FAQ with lots of pointers to papers, books, software and the main groups working on EAs all over the world [21].

4 Breeder Genetic Algorithms

The Breeder Genetic Algorithm [29] is in midway between GAs and ESs (see Fig. 3). While in GAs selection is stochastic and meant to mimic —to some degree— Darwinian evolution, BGA selection is named *truncation* selection, a deterministic procedure driven by the so-called *breeding* mechanism[3], an artificial selection method stating that only the best individuals —usually a fixed percentage τ of total population size— are selected and enter the gene pool to be recombined and mutated, as the basis to form a new generation[4]. Recombination/mutation operators are applied by randomly and uniformly selecting two parents until the number of offspring equals $\mu - q$. Then, the former q best elements are re-inserted into the population, forming a new generation of μ individuals that replaces the previous

[3]This method is employed in livestock breeding.

[4]It is interesting to note that Tournament Selection in GAs is an stochastic form of rank-based selection, of which truncation selection is the most used instance.

```
Procedure Evolutionary-Algorithm (<EA-Setup>)
{
        t:=0;
        create Π_t;
        evaluate Φ(i), ∀i ∈ Π_t;
        while not(Θ(Π_t)) do
        {
            /* Create the gene pool Π_t^Υ */
               select: Π_t^Υ := Υ(Π_t);

            /* Apply genetic operators */
               recombine: Π'_t := Ω(Π_t^Υ);
               mutate: Π''_t := Ψ(Π'_t);

            /* Evaluate their effect */
               evaluate Φ(i), ∀i ∈ Π''_t;

            /* Form the new generation */
               replace: Π_{t+1} := Ξ(Π''_t ∪ Π_t^Υ);
               t := t+1
        }
}
```

Figure 1: Evolutionary Algorithm.

one. This guaranteed survival of some of the best individuals is called *elitism* whatever the EA. For the BGA, the typical value is $q = 1$. The BGA selection mechanism is then deterministic (there are no probabilities), extinctive (the best elements are guaranteed to be selected and the worst are guaranteed *not* to be selected) and 1-elitist (the best element is always to survive from generation to generation). Self-mating is always prohibited. This is a form of the comma strategy (μ, λ) employed by ESs because the parents are not included in the replacement process, with the exception of the q previous best. Note that in the BGA only μ needs to be specified, since the number λ of offspring[5] can be calculated as $\lambda = \mu - q$. The BGA procedure is depicted in figure 2, where τ is the truncation percentage for selection.

The other strong resemblance of BGAs to ESs is that, unlike GAs, BGAs use a direct representation, that is, a gene *is* a decision variable (not a way of coding it) and its allele is the value of the variable[6]. An immediate consequence is that, in the absence of other conditionings as constraint handling, the fitness function equals the function to be optimized, $\Phi(\vec{x}) = F(\vec{x})$. In addition, in a BGA chromosome there are no additional variables other than the x_i, that is to say, the algorithm does not self-optimize any of its own parameters, as is done in ESs and in some meta GAs. Chromosomes are thus potential solution vectors $\vec{x}$ of n components, where n is the problem size, the number of free variables of the function to be optimized. This issue is of crucial importance because i) it eliminates the need of chosing a coding function (e.g., binary, Gray, ...) and ii) clears the way to the direct coding of different kinds of variables other than real numbers (e.g., fuzzy quantities, discrete quantities, etc).

The strongest contact point of BGAs with ordinary GAs is the fact that both are mainly driven by recombination, with mutation regarded as an important but background operator intending to reintroduce some of the alleles lost in the population. This view is conceptually right for GAs, because the cardinality of the alphabet used to code variables into the chromosome (the number of alleles per gene) is usually very small (two, in most cases). But in the case of algorithms that make use of real-valued alleles, like the BGA, mutation has to be seen in the double role of solution fine-tuner (for very small mutations) and as the main discovery force (for moderate ones). Increasing voices have raised claiming mutation not

[5] In this case, $\lambda < \mu$ and the BGA mechanism deviates from that of ESs.

[6] Of course, in a digital machine, we still have a coding, namely, that of the floating point representation but the decoding is transparent to the high level treatment of real numbers.

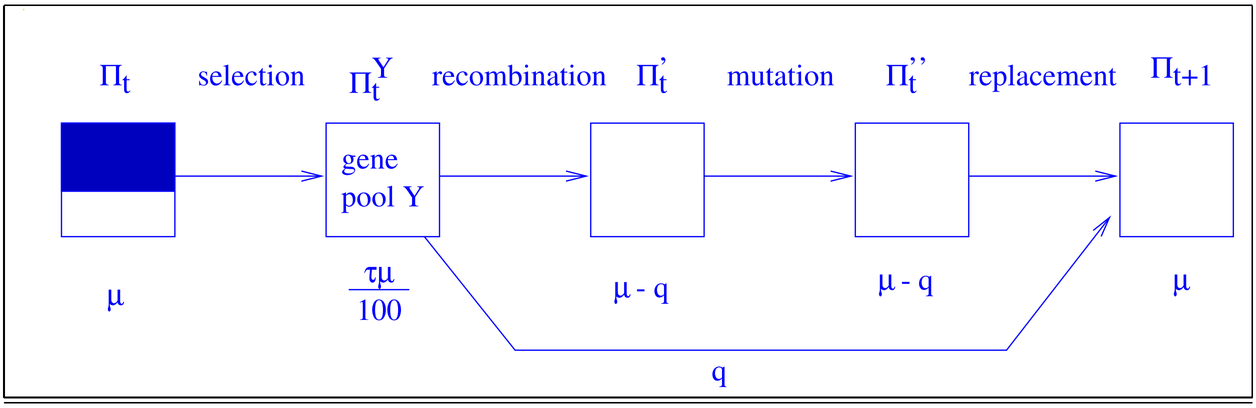

Figure 2: A scheme of the BGA procedure. Each box represents the population at different stages in the process to form a new generation. Notation on top of the boxes names the population at that point (see text) and the label from box to box (above the arrows) denotes operator sequencing (from left to right). The expressions at the bottom of the boxes indicate the population size at each step. Note how the final population size μ is formed by summing its two incoming values.

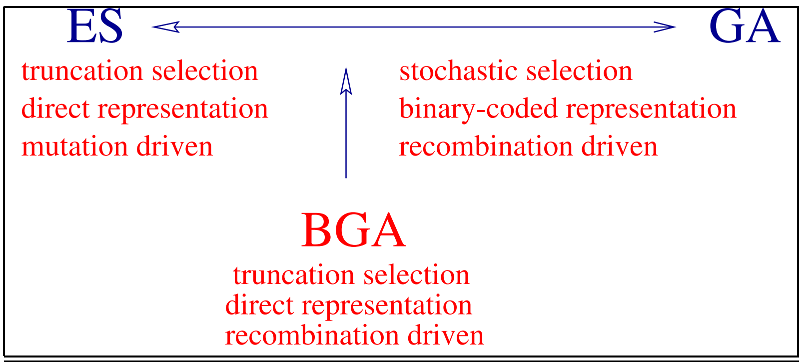

Figure 3: The BGA lies in a midway between ESs and GAs.

only as a necessary operator, but being even more powerful than recombination when either is applied alone [5]. One would say that recombination is still present because is intuitively appealing and because of it being traditionally used in GAs as the main operator. In fact, the initial BGA formulation ([29], p.23) readily acknowledged this superiority and remarked that it is the *synergistic* effect of their combined and iterated application what extracts the most from an EA. What is more, in ESs and EP, the roles are exchanged and mutation is the driving force, in the form of a very powerful self-adapting operator that tries to take the (unknown) relationships between variables into account, such that optimization is performed in several dimensions simultaneously. We will now describe the different possibilities for the operators in more detail. Their influence on BGA performance will be assessed in later parts of the work.

4.1 Recombination

Any operator combining the genetic material of $p \geq 2$ parents is called a recombination operator. The typical value is $p = 2$, although there are some studies claiming the superiority of higher values for p —usually for GAs [13], where the operator is called *crossover* because of the way of combining the alleles, particular of discrete (mostly binary) genes. Many variations have been proposed to date, although just a couple of them are normally used. The other main source of variation in a recombination operator consists in allowing different sets of parents (regardless of p) for every gene in the individual, provided the parents belong to the mating pool. In BGAs, recombination is applied unconditionally, that is, $\Pr(\Omega) = 1$. Let $\vec{x} = (x_1, \ldots, x_n)$, $\vec{y} = (y_1, \ldots, y_n)$ be two selected gene-pool individuals $\vec{x}, \vec{y} \in \Pi_t^{\Upsilon}$ such that $\vec{x} \neq \vec{y}$. Let $\vec{z} = (z_1, \ldots, z_n)$ be the result of recombination and $1 \leq i \leq n$. Elements $\vec{z} \in \Pi_t'$ are formed as follows:

1. Discrete Recombination (DR).
   $$z_i \in \{x_i, y_i\}$$
   chosen with equal probability.
   Geometric effect: Let $H(\vec{x}, \vec{y})$ be the smallest hyperectangle containing both $\vec{x}$ and $\vec{y}$. Then $\vec{z}$ is one of the corners of $H(\vec{x}, \vec{y})$.

2. Extended Line Recombination (ELR).
   $$z_i = x_i + \alpha(y_i - x_i)$$
   with $\alpha \in [-\delta, 1+\delta]$ chosen with uniform probability and $\delta \geq 0$ (typical $\delta = 0.25$).
   Geometric effect: Let $r(\vec{x}, \vec{y})$ be the line containing both $\vec{x}$ and $\vec{y}$. Then $r(\vec{x}, \vec{y})$ contains $\vec{z}$. That is, in general, the resulting point lies in the line passing through $\vec{x}$ and $\vec{y}$. If $\delta = 0$ then the resulting point specifically lies *in between* $\vec{x}$ and $\vec{y}$.

3. Extended Intermediate Recombination (EIR).
   $$z_i = x_i + \alpha_i(y_i - x_i)$$
   with $\alpha_i \in [-\delta, 1+\delta]$ chosen with uniform probability. Same as ELR but a new α is chosen for each gene (that is, for each i).
   Geometric effect: Let $H(\vec{x}, \vec{y})$ be the smallest hyperectangle containing both $\vec{x}$ and $\vec{y}$. Then, for $\delta = 0$, $\vec{z} \in H(\vec{x}, \vec{y})$. For $\delta > 0$ the resulting point can also lie anywhere in the outside vicinity of $H(\vec{x}, \vec{y})$, further away the greater δ is.

   Initially, the EIR and ELR operators were not extended (hence denoted just IR and LR) and were meant to yield the parents's midpoint, that is, they worked with a fixed α or α_i equal to 0.5. The extension given by $\alpha_i \in [-\delta, 1+\delta]$ such that $\delta \geq 0$ has been proven to be very useful, allowing more variety and increasing the variance of the operator. Thus, in both ELR and EIR there are two main issues: the selection of α (or α_i) from a given probability distribution function (*pdf*) and the selection of its range, given by δ. The classical *pdf* is the uniform, although a Gaussian one, centered at the parents's middle point could also be a reasonable choice, seen as a different extension of the initial LR operator with $\alpha = 0.5$. When considered in a range $[-\delta, 1+\delta]$, the expression for the selection of α could for instance be

$$\alpha_i \in N\left(0.5, \frac{\delta + 0.5}{2.5}\right)$$

considering values within five standard deviations. The other issue, the selection of δ, seems much more arguable and empirical. In this work, we will study several values for it (from $\delta = 0$ to the typical $\delta = 0.25$). In addition, a method for dynamically setting its value is initially tested, called $range_\delta$, as follows,

$$z_i = y_i + \alpha_i(x_i - y_i), \qquad \text{with } x_i \geq y_i$$

such that $\alpha_i \in [-\delta_i^-, 1+\delta_i^+]$ with some chosen *pdf* (uniform or Gaussian, as we have seen) and

$$\begin{aligned} \delta_i^- &= \frac{y_i - r_i^-}{r_i^+ - r_i^-} \\ \delta_i^+ &= \frac{r_i^+ - x_i}{r_i^+ - r_i^-} \end{aligned}$$

This procedure assigns different values for the left (δ_i^-) and right (δ_i^+) limits of the interval from which α is to be selected, and does never generate a value outside its range, an aspect not fulfilled by the other methods that otherwise has to be dealt with a posteriori. However, any point within range can in principle be generated, in a sense diluting the influence of parents and making it a more disruptive recombination operator. Its behaviour will be assessed throughout the experiments.

4. Fuzzy Recombination (FR). This operator is more recent than the other three, the classical BGA recombination operators. Introduced in [36], it basically replaces the uniform *pdf* by a bimodal one, where the two modes are located at x_i and y_i, the two parents, that is $\Pr(z_i) \in \{\Pr_{x_i}(z_i), \Pr_{y_i}(z_i)\}$ thus favouring offspring values close to them, and not in any intermediate point with equal probability, as with previous operators. The label "fuzzy" comes from the fact that the two parts $\Pr_{x_i}(t), \Pr_{y_i}(t)$ of the probability distribution resemble fuzzy numbers (triangular in the original formulation) such that they fulfill the general conditions (where $y_i \geq x_i$):

$$x_i - e|y_i - x_i| \leq \; t \; \leq x_i + e|y_i - x_i|$$

$$y_i - e|y_i - x_i| \leq \; t \; \leq y_i + e|y_i - x_i|$$

stating that the offspring t lies in one (or both) of the intervals, being $e > 0$ the fuzzy number's spread, the same for both parts. The distribution is a symmetric bimodal with a median equal to $\frac{x_i + y_i}{2}$. The favour for offspring values near the parents is thus stronger the closer the parents are. This operator is depicted in Fig. 4.

The membership function of a normalized triangular fuzzy number with mode m and symmetric spread s (left-right distance from mode) is

$$\mu(t)_T\{s, m\} = 1 - \frac{2|m - t|}{s}$$

whereas the corresponding unimodal triangular *pdf* is

$$\Pr(t)_T\{s, m\} = \begin{cases} 0 & t < m - s \\ \frac{1}{s^2}(t + s - m) & m - s \leq t \leq m \\ \frac{1}{s^2}(-t + s + m) & m \leq t \leq m + s \\ 0 & t > m + s \end{cases}$$

In the simplest case, assuming $e = 0.5$ (that is, the two parts meet at the median and this point has zero probability, as in Fig. 4(b)), the resultant parameterized bimodal triangular *pdf* is written

$$\Pr(t)_{BT}\{s_1, m_1, s_2, m_2\} = \frac{1}{2}(\Pr(t)_T\{s_1, m_1\} + \Pr(t)_T\{s_2, m_2\})$$

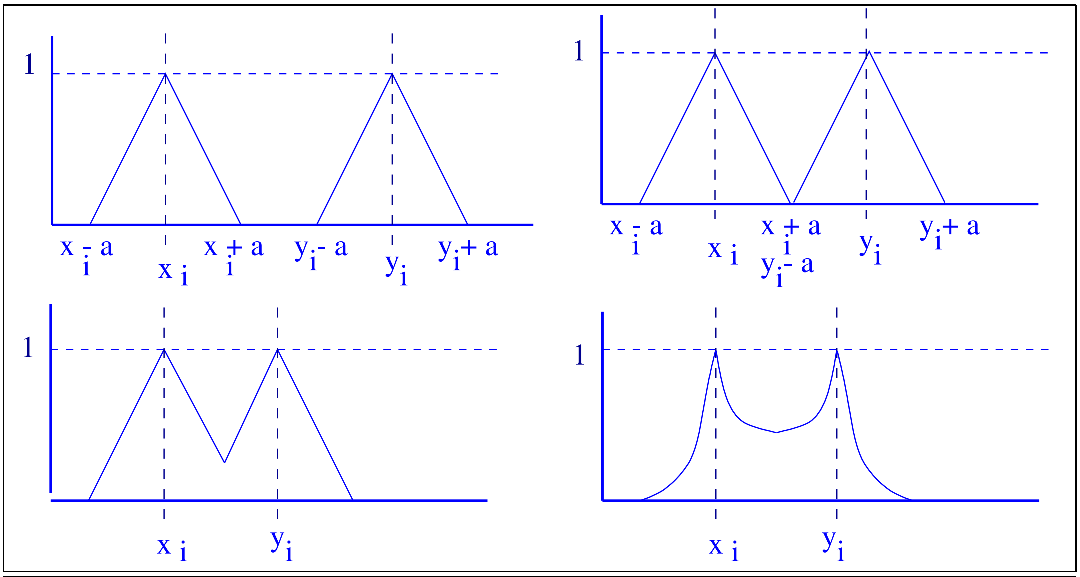

Figure 4: Bimodal probability distribution for the FR operator, where $a = e|y_i - x_i|$. (a) Not all the intermediate values are possible ($e < 0.5$). (b) Original formulation with $e = 0.5$. (c) Overlapped distribution $0.5 < e \leq 1.0$. (d) A general "fuzzy" symmetric probability distribution.

So that, turning to our problem, the offspring z_i is obtained as

$$\Pr(z_i) = \Pr(z_i)_{BT}\{e|y_i - x_i|, x_i, e|y_i - x_i|, y_i\}$$

This idea need not be limited to the original triangular numbers. For example, a bimodal Gaussian should reasonably give similar results. In fact, one could devise a (possibly problem-specific) form for the probability distribution resembling a bimodal fuzzy (Fig. 4 (d)). The geometric effect is in this case clear from the figures.

5. Gene Pool Recombination (GPR). This is an old idea [7] conceptually very nice [30], although rarely used. The z_i are build out of x_i, y_i but this time the parents $\vec{x}, \vec{y}$ are selected for each i from the gene pool (in the case of the BGA, from the best τ%). Either DR, ELR or EIR can be used for each z_i. Geometric effect: the effect of having more than two parents easily generalizes the previous interpretations. In general, for a set of parents $\vec{x}^1, \vec{x}^2, \ldots, \vec{x}^p$ the resulting offspring $\vec{z}$ will lie inside the smaller "container box" that includes all the $\vec{x}^i$, for $\delta = 0$. As usual, $\delta > 0$ allows to exit the scope of the parents.

4.2 Mutation

Mutation is applied to each gene with some probability $\Pr(\Psi) = 1/n$ so that, on average, one gene is mutated for each individual. Let $\vec{z} = (z_1, \ldots, z_n)$ be the result of mutation, $1 \leq i \leq n$, from the previously generated $\vec{x} \in \Pi'_t$. The elements $\vec{z} \in \Pi''_t$ are formed as follows:

1. Discrete Mutation (DM). This is the classical BGA mutation operator.

$$z_i = x_i + sign \cdot range_i \cdot \delta$$

with $sign \in \{-1, +1\}$ chosen with equal probability, $range_i = \rho(r_i^+ - r_i^-)$, $\rho \in [0.1, 0.5]$ and

$$\delta = \sum_{i=0}^{k-1} \varphi_i 2^{-i}$$

where $\varphi_i \in \{0, 1\}$ from a Bernouilli probability distribution where $\Pr(\varphi_i = 1) = 1/k$. In this setting $k \in \mathbb{N}^+$ is a parameter originally related to the *precision* with which the optimum was to be located, a machine-dependent constant. Modern machines, capable of `double` precision, would in principle allow for higher values of k (e.g. 24, 32) than those traditionally used (e.g. 8, 16). In practice, however, the value of k is related to the *expected* value of mutation steps: the higher k is, the more fine-grained is the resultant mutation operator. The expected value of δ for a given k is that of a Binomial (n, p), with $n = k$ and $p = 1/k$:

$$E\{\delta\}_{(k)} = \frac{1}{k}\sum_{i=0}^{k-1} 2^{-i} = \frac{1}{k}(2 - 2^{-k+1}) < \frac{2}{k}$$

It can be proven that $E\{\delta\}_{(k')} < E\{\delta\}_{(k)}$ for $k' > k$ and thus expected mutation steps are lower. The inequality holds for $k \geq 1$. This, as we shall see, has strong consequences on algorithm performance.

The factor ρ is the *range ratio*, related to the *maximum* step that mutation is allowed to produce as a ratio of variable range. All in all, this scheme favours small values but cannot generate all possible representable points[7], but only a discrete amount and prefers small values in an approximately (on average) logarithmic ($\log_2$) scale, always up to a precision of $range_i \cdot 2^{-k+1}$.

2. Continuous Mutation (CM). Same as DM but with

$$\delta = 2^{-k\beta}$$

where $\beta \in [0, 1]$ with uniform probability. The expected value of δ for a given k is now

$$E\{\delta\}_{(k)} = 2^{-\frac{k}{2}} = \sqrt{2^{-k}}$$

Again, $E\{\delta\}_{(k')} < E\{\delta\}_{(k)}$ for $k' > k$ and expected mutation steps are lower.

4.3 Constraint handling

We will not be considering constrained functions in this work and we refer the reader to [27] for a through survey. There are two basic ways of dealing with constraints, which we will call the generative and penalty methods.

1. Generative method: the algorithm does not generate points that violate the constraints, that is, constraints are fulfilled by construction, via a careful and specialised operator redesign; for example, "repairing" unfeasible solutions forcing them to be feasible.

2. Penalty method: the algorithm is allowed to generate points that violate the constraints, and these are taken into account in the fitness function by adding a penalty term for each constraint, usually higher the more the constraints are violated (see §2).

A related and commonly encountered issue is that of methods of keeping variables x_i within their predeclared ranges $R_i = [r_i^-, r_i^+]$, after recombination or mutation. There are various ways to achieve this. First, they can be regarded as linear constraints $c_j(\vec{x})$ where only one x_i is affected. But these constraints are too basic and thus specific and simpler ways of treating them apart are a clear choice. Perhaps the more obvious one is that performed by simply iterating operator application until all variables in the obtained vector fulfill the range limitations. This method can become too costly when working close to the bounds. Other methods keep the illegal value and alter it somehow. We will refer to the method used as the *bounding* method and denote it by $\Lambda(\cdot)$. Let $\vec{z}$ be the vector generated from recombination or mutation.

[7] By this we mean *machine*-representable. We assume that there is a machine-dependent floating point constant ϵ equal to the smallest positive representable number in a chosen precision. For example, in our machines, such number for `double` precision is $\epsilon \approx 2.22 \cdot 10^{-16}$.

1. Clipping. The values are clipped to the bounds.

$$\Lambda(z_i) = \begin{cases} r_i^+ & \text{if } z_i > r_i^+ \\ r_i^- & \text{if } z_i < r_i^- \\ z_i & \text{otherwise} \end{cases}$$

2. Bouncing. The values are bounced against the bounds.

$$\Lambda(z_i) = \begin{cases} r_i^+ - (z_i - r_i^+) & \text{if } z_i > r_i^+ \\ r_i^- + (r_i^- - z_i) & \text{if } z_i < r_i^- \\ z_i & \text{otherwise} \end{cases}$$

Strict inequalities for bounds can be handled by rewriting the ranges R_i to new ranges $\hat{R}_i = [\hat{r}_i^-, \hat{r}_i^+]$ as follows:

$$\begin{aligned} \hat{r}_i^- &= r_i^- + \epsilon \\ \hat{r}_i^+ &= r_i^+ - \epsilon \end{aligned}$$

where ϵ stands for the machine precision (see previous footnote). This way, original bounds of the type (r_i^-, r_i^+) (meaning that $r_i^- < x_i < r_i^+$) are rewritten into the standard form $[\hat{x}_i^-, \hat{x}_i^+]$ with the usual meaning $\hat{x}_i^- \leq \hat{x}_i \leq \hat{x}_i^+$.

5 The Test Suite and its Properties

The use of an artificial test suite, at least for EAs, can be said to be formally open by De Jong [12] with the first five problems (hereafter named F1 to F5). Since then, several other researchers have appended their own developed problems (Schwefel, Ackley, Michalewicz and others), conforming a more or less standard order up to F9. Recent work, though, showed that some of these functions either were not as difficult as it was thought (at least for an EA) or were not particularly well suited as test beds. Arguably, the most important issue in the design of a test suite is the possibility of having control on the variability about the type of the function and the properties it shows, which reflects in how challenging the function is. Taking into account their mathematical properties is a way of having extra knowledge on the problem and thus helps interpreting and correctly assessing the obtained results. Most important, some of these properties can also be known (or estimated) for real problems though, in this case, one usually has little or no control on them. Specifically, the following are the main aspects to be considered about the problems:

Separability A function is said to be separable if there are no interactions between different variables. This means, in practice, that they can be solved by simply optimizing independently in each variable. In EAs, the interactions between genes in a chromosome is called *epistasis* (a term borrowed and adapted from genetics). Epistasis is undesirable in practice but useful when developing difficult test functions. It is favoured by the discrete codings used in most GAs (like binary or Gray codes) because the originally atomic real value is coded into many genes that are thus expected to interact strongly. Hence, epistasis depends on representation: a separable function will show null epistasis only when using a direct coding for the variables.

Symmetry In $n = 2$ dimensions, symmetry means that

$$F(x_1, x_2) = F(x_2, x_1) \qquad \forall x_1, x_2$$

In general, in $n = n_0$ dimensions, symmetry can go up to $n_0!$ possibilities, eventually making the problem easier, since up to $n_0!$ equivalent solutions may exist.

Scalability The problem with some of the classical test functions is that their complexity changes with dimension. For instance, there are some that become easier as dimension increases (e.g. F8, known

Index	Name	n	Expression	Range of x_i	Sep.
(F1)	De Jong-1	$n=3$	$\sum_{i=1}^{n} x_i^2$	$[-5.12, 5.12]$	Y
(F2)	De Jong-2	–	$100(x_1^2 - x_2)^2 + (1 - x_1)^2$	$[-2.048, 2.048]$	N
(F3)	De Jong-3	$n=5$	$\sum_{i=1}^{n} [x_i]$	$[-5.12, 5.12]$	Y
(F4)	De Jong-4	$n=30$	$\sum_{i=1}^{n} i x_i^4 + N(0,1)$ where $N(0,1)$ is a normal distribution	$[-1.28, 1.28]$	Y
(F5)	De Jong-5	– $m=25$	$\{0.002 + \sum_{j=1}^{m} [j + (x_1 - a_j)^6 + (x_2 - b_j)^6]\}^{-1}$ $a_j = 16(j \ \mathbf{mod}\ 5 - 2)$ $b_j = 16(j \ \mathbf{div}\ 5 - 2)$	$[-2^{16}, 2^{16}]$	Y
(F6)	Rastrigin	$n=20$	$nA + \sum_{i=1}^{n} x_i^2 - Acos(2\pi x_i)$ where $A = 10.0$	$[-5.12, 5.12]$	Y
(F7)	Schwefel	$n=10$	$V - \sum_{i=1}^{n} x_i sin(\sqrt{\vert x_i \vert})$ where $V = 418.9829n$	$[-512, 512]$	Y
(F8)	Griewangk	$n=10$	$\sum_{i=1}^{n} \frac{x_i^2}{400n} - \prod_{i=1}^{n} cos(\frac{x_i}{\sqrt{i}}) + 1$	$[-512, 512]$	N
(F9)	Ackley	$n=30$	$V - 20e^{-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}} - e^{\frac{1}{n}\sum_{i=1}^{n} cos(2\pi x_i)}$ where $V = 20 + e$	$[-30, 30]$	N
(F10)	Schaffer-1	–	$\frac{1}{2} + \frac{sin^2(\sqrt{x_1^2 + x_2^2}) - \frac{1}{2}}{[1 + 0.001(x_1^2 + x_2^2)]^2}$	$[-100, 100]$	N
(F11)	Schaffer-2	–	$(x_1^2 + x_2^2)^{\frac{1}{4}} \left(sin^2[50(x_1^2 + x_2^2)^{\frac{1}{10}}] + 1 \right)$	$[-100, 100]$	N

Table 1: The basic test suite: Index, Name (non-standard), typical dimension, expression, typical ranges (equal for all dimensions) and separability property (Y=yes, N=no).

as Griewangk's function [37]). The need for scalable functions arises from the fact that the results obtained in some tiny test functions may say little about real performance in complex problems, with possibly dozens of variables interacting in unknown ways. The availability of high-dimensional controllable functions is thus of great practical concern.

Stochasticity Real (in the sense of physical) functions represent by their nature an stochastic process: all we see is a possible realization of the function. In some contexts, like neural network training processes, the function to be minimized (typically, some measure of error) is deterministic but relies on noisy samples of an unknown function. It is interesting thus to devise artificial FOPs to resemble an stochastic process by, for instance, adding a noise component to a deterministic function, as in F4 (see Table 1).

Non-linearity Problems that lack strong non-linear dependencies between variables may be used as initial testers or as benchmarkers but will not reflect the complexity of many real-world problems, and, as in the case of non-scalable functions, will be of little practical use.

Relevance In many real applications, there is no guarantee that the set of variables being used to characterize the problem at hand are those actually relevant. That is to say, on the one hand, there are no important variables missing. On the other hand, the degree of redundancy is suficciently small to be beneficial. In addition, the chosen variables should represent the most convenient degree of abstraction.

Note that separability has little to do with linearity. A function can be highly non-linear and still be separable. Also, separable functions can be solved by exact methods like line search in $O(n)$ time, where n is the number of variables (although the constants involved can be very high). All this means that non-linear, non-symmetric, non-separable, scalable functions should be present in a test suite. These arguments were introduced in [37], along with a nice method to obtain such functions from some of those already present in the shown standard test suite. There are two basic procedures to achieve this:

5.1 Expansion

1. Depart from an existing function defined in two dimensions $F(x_1, x_2)$.
2. Design a weight matrix $W_{n\times n} = (w_{ij})$, where n is the desired dimension of the new function.
3. Then, the *expansion* of F in n is defined as the function

$$E_F(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij} F(x_i, x_j)$$

As an example, we can devise a "wrap" matrix $W_{n\times n}$ such that

$$w_{ij} = \begin{cases} 1 & \text{if } j = i+1 \wedge i < n \\ 1 & \text{if } j = 1 \wedge i = n \\ 0 & \text{otherwise} \end{cases}$$

which gives rise to an expression of the form

$$E_F(x_1, x_2, \ldots, x_n) = F(x_1, x_2) + F(x_2, x_3) + \ldots + F(x_{n-1}, x_n) + F(x_n, x_1)$$

5.2 Composition

1. Depart from an existing function defined in two dimensions $F(x_1, x_2)$, and another one-dimensional $G(z)$ such that $Im(F(x_1, x_2)) \subseteq Dom(G)$, where $Dom(\cdot)$ stands for the domain of definition of a function and $Im(\cdot)$ for its image, defined as

$$Im(f) = \{y \in \mathbb{R} \mid \exists \vec{x} \in Dom(f) : y = f(\vec{x})\}$$

2. Then, the *composition* of both functions is defined in the usual way

$$C_{G,F}(x_1, x_2) = G(F(x_1, x_2))$$

The interesting news is that the two methods can be combined. That is,

$$EC_{G,F}(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij} G(F(x_i, x_j))$$

As an example, combining the previously mentioned Griewangk's function F8 and De Jong's F2 (also known as Rosenbrock's function) yields the function (using wrap expansion):

$$EC_{F8,F2}(x_1, x_2, \ldots, x_n) = F8(F2(x_1, x_2)) + F8(F2(x_2, x_3)) + \ldots + F8(F2(x_n, x_1))$$

This function is non-symmetric (because F2 is), non-linear (because both F8, F2 are), non-separable (because of the common terms) and scalable in n, that is, its complexity grows with n always at the same rate, although F8 was *not* scalable [37] (actually it gets much simpler, tending to a unimodal macrostructure), and this is because F8 is now always used only in its one-dimensional form. Care must be taken, however, not to choose symmetric functions as a base for expansion because, in that case:

$$E_F(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij} F(x_i, x_j) = 2\sum_{i=1}^{n}\sum_{j>i}^{n} w_{ij} F(x_i, x_j) + \sum_{j=1}^{n} w_{jj} F(x_j, x_j)$$

and the number of interacting terms shrinks from n^2 to $\frac{n^2+n}{2}$, thus simplifying the function. In general, thus, these two methods have the effect of retaining (and enhancing) scalability and forcing non-separability.

5.3 The used Test Suite

The most classical test suite (up to our knowledge) is summarized in Table 1, ordered in a more or less standard way and christened with a name to reflect their introducer as a FOP in the context of EAs. It ranges from functions (supposedly) easy to (supposedly) hard; this is not a minor issue: a good FOP solver is such that performs reasonably well for a wide range of difficulties. In particular, functions F1 to F5 are easy (at least to BGAs, see [10]). All of them are separable or of very low dimension, or both, and will not be considered here in deep. Functions F6 to F11 are all multimodal and, although smooth, are very difficult to an algorithm relying on derivatives information. F6 has many suboptimal peaks whose values increase as the distance from the global optimum increases. However, it shows a *unimodal macrostructure* in high dimensions: for large n, the quadratic term dominates and the function is like a quadratic perturbed by sinusoidal deviations. This makes it difficult but non-scalable. F8 and F9 are also highly difficult because they are non-separable due to the interdependencies between variables. They also exhibit a unimodal macrostructure to some degree in high dimensions. F10 and F11 are functions with a extremely large number of local optima distributed uniformly, with hardly any structure other than a regularly rugged landscape. Their use in only two dimensions makes them affordable. F7 is a very difficult function for which the best minima are far away from each other. There is also a "second best" point located in the opposite corner of the space w.r.t. the global best. This function does not show any unimodal macrostructure. The global minimum is at $\approx -418.9829n$ and hence the use of the term V. Note that all the functions F6 to F11 have been adapted to have a global best optima as a minimization problem of solution 0.

6 Experimental Design

6.1 General Considerations

In this section we proceed to describe the experiments performed and their design procedure. They are split in two parts: experiments with mutation and experiments with recombination. For each part, different operators and parameters will be consistently explored, varying the population size. For each configuration, a number of independent runs will be performed, keeping track of the mean and best solutions found. This experimental design is suboptimal because not all the possible combinations of mutation with recombination are tested. As we shall see, this number is too high to allow a full study to be performed and, furthermore, it is our belief that many of the combinations can readily be discarded *a priori* by a smaller but more effective experimental design. Also, even with the full results, the conclusions could not in any case be general ones since there is probably no configuration that is optimal for every conceivable (even reasonably) FOP, not even if, as in this work, the study is circumscribed to a specific search algorithm.

6.2 Working Hypotheses

Rather, experiments with mutation are to be performed with a fixed recombination setting, hoping it to be as neutral a choice as possible w.r.t. mutation. The same applies to experiments with recombination. We believe that these aspects are very important and should be clear (and made clear) in any practical study. For this reason, we have collected the following set of hypotheses:

1. *The truncation threshold τ is not going to affect the experiments.* With this we mean that choosing a fixed and reasonable value for it will not affect the particular configurations *differently*.

2. *The replacement criterion Ξ is not going to affect the experiments.* With this we mean that using the explained BGA criterion for all the experiments will not affect the particular configurations differently. This is important to say because different criterions could be better (or worse) suited for some operators. However, it is not an issue here because changing Ξ would be a departure from the BGA.

Parameter	Value	Comment
τ	15	Truncation threshold
Ξ	BGA	BGA Replacement criterion
Λ	Clipping	Bounding method
`NRuns`	30	Number of runs/experiment
`FFEvals`	40000	Number of function evaluations/run
ϵ	10^{-16}	Less than ϵ is considered as 0
μ	400, 200, 100, 50	Population sizes

Table 2: BGA Setup kept constant for all of the experiments.

3. *The bounding method Λ is not going to affect the experiments.* The same explanation applies.

4. *A good mutation operator is robust w.r.t. recombination.* This and the next one are the main hypotheses. Although it is known that strong interactions take place between recombination and mutation, it is reasonable to assume that a good mutation operator is good in itself and would improve on any general, non-specialized recombination operator, such as the ones presented.

5. *A good recombination operator is robust w.r.t. mutation.* Idem.

6. *Hypotheses 1 to 5 can be applied simultaneously.*

We also believe that this constitutes a more reasonable approach than the other possibility, namely, performing experiments with varying mutation but *no recombination* and viceversa, because in this case the a posteriori integration of both operators will definitely make a difference. In conclusion, the search is not for the best recombination/mutation setting —because this surely does not exist— but constitutes instead a heuristic way of finding generally good configurations (or generally bad ones, also useful to know).

As stated, both sets of experiments will be performed with varying μ, the population size, in a basic attempt to assess its influence. To this end, the stopping criterion Ξ will be based on the number of FOP evaluations permitted (given by the variable `FFEvals`). In particular, given a finite number of `FFEvals`, the algorithm will stop each run whenever $\lfloor \frac{\texttt{FFEvals}}{\mu} \rfloor$ generations are reached. This stopping criterion allows to compare different general settings in a fair way, since, for example, a smaller population will be allotted more generations, but always keeping the number of FOP evaluations in similar values. This scheme would also allow comparison with other search methods[8]. The parameters kept constant for all of the experiments are shown in Table 2.

6.3 Test Functions

As discussed in §5.3, among the functions from Table 1 the decision has been to use only a subset of them, mainly because functions F1 and F3 to F5 are separable and thus easy targets for an evolutionary searching algorithm. On the contrary, the functions F6 and F8 to F11 are non-separable and show a reasonable variety of non-linearities and dimensions. In addition, the function F7 has also been considered because, although being separable, has been shown to be a difficult one, for which suboptimal extrema are very far away from each other and far away from the global optimum (see [37]). The other non-separable function shown is F2 but this one is too simple to be a choice. It will be used instead in conjunction with F8 to form a new F8F2 function, as explained in §5.2.

[8] Indeed, one could ask a method: "You have this function and up to `FFEvals` evaluations of it. Use them and give your best answer back".

6.4 Presentation of Results

Due to the fact that relatively different values can be reached as solutions, the *geometric* mean will be used sometimes to average the results, instead of the arithmetic mean, because it works better for a set of values of different orders of magnitude. To see this, suppose that two different runs give as results the values 10^{-3} and 10^{-9}. Denoting by $\overline{\{a,b\}} = \frac{a+b}{2}$ the arithmetic mean, we get $\overline{\{10^{-3}, 10^{-9}\}} \cong \frac{10^{-3}}{2}$, that is, whenever the two numbers greatly differ in orders of magnitude, this averaging measure gives as a result one of the two —the biggest in absolute value— halved. In contrast, the geometric mean in this case gives the middle point in orders of magnitude, 10^{-6}. Denoting by $\widehat{\{a,b\}} = \sqrt{ab}$ the geometric mean, it holds that

1. The numbers $a, \overline{\{a,b\}}, b$ form an arithmetic progression;
2. The numbers $a, \widehat{\{a,b\}}, b$ form a geometric progression.

In general, for a set N of n real numbers $N = \{x_1, x_2, \ldots, x_n\}$ such that $x_i > 0, \forall i$ (if any $x_i = 0$, the result is zero), the geometric mean is defined as:

$$\widehat{N} = \sqrt[n]{\prod_{i=1}^{n} x_i}$$

This averaging measure fulfills two desirable properties, typical of the arithmetic mean:

1. Scalability. The deviations from the arithmetic mean scale linearly with orders of magnitude. For example, let $A = \{32, 33, 34\}$ and $B = \{32 \cdot 10^2, 33 \cdot 10^2, 34 \cdot 10^2\}$. Then, $\widehat{A} = 33\varepsilon$ (with $\varepsilon \approx 1$) and $\widehat{B} = 33 \cdot 10^2 \varepsilon$.
2. Incrementality. Let $P_k(N)$ any partition of N in k equally sized sets, being $k > 0$ a divisor of n. Let
$$\mathcal{X}(N) = \{\widehat{X} \mid X \in P_k(N)\}$$
Then it follows that $\widehat{\mathcal{X}}(N) = \widehat{N}$. That is, the geometric mean computation of a set of numbers can be decomposed in two steps: selecting a partition, computing the set of geometric means of this partition and finally computing their geometric means. This will not change the final result, no matter what partition we consider. This property will be used in the experiments, where many quantities will be averaged.

The relation between the two averaging methods is the following: given N, it holds that $\overline{N} \geq \widehat{N}$. Both measures will be used, depending on the quantities to be averaged.

7 Experimental results (I): Mutation

The results are presented as follows. Four tables are given, one for each population size μ, separated in continuous (CM) and discrete (DM) mutation for each FOP. For each configuration (ρ, k) the best solution found throughout the `NRuns` is kept. Instead of giving a separate entry for each such configuration, additional computations are performed to compact the information and increment the level of abstraction. For example, the entries in the column for ρ are obtained averaging out (using the geometric mean) the results **forall** k **in** $\{8, 16, 24, 32\}$. Similarly, the entries in the column for k are those obtained averaging out **forall** ρ **in** $\{0.1, 0.3, 0.5\}$. By proceeding this way, one has to deal with less information and the obtained values are more representative. Additionally, the last two columns express the overall average (**forall** ρ, k) and the overall best solution found.

There is a special case to cope with: whenever a solution is generated beyond machine precision (in our case study, a value below 10^{-16}), the problem will be considered solved and, as all of the problems

```
Procedure Mutation-Test ()
{
        Ω := EIR (δ = 0.25);
        forall μ in {400, 200, 100, 50}
            forall F_i in {F6, ..., F11}
                forall k in {8, 16, 24, 32}
                    forall ρ in {0.1, 0.3, 0.5}
                        forall Ψ in {CM, DM}
                            BGA (μ, F_i, k, ρ, Ψ, NRuns, FFEvals, Ω, τ, Λ, Ξ);
}
```

Figure 5: `Mutation-Test` Algorithm pseudocode.

have solution 0, this situation will be indicated as 0 (%), where % is the (rounded) percentage of times this happened, so as to make comparison still possible. The only exception to this rule is F11, for which the obtained values, although very small, are still meaningful and will be the ones shown. All mutation variations are tested using a fixed standard recombination setting Ω = EIR ($\delta = 0.25$). The BGA procedure in the algorithm of Fig. 5 is run 4320 times (720 for each FOP) for every value of μ, making a grand total of 17280 runs. The results are shown in tables 3, 4, 5 and 6.

Index	Ψ	ρ =0.1	ρ =0.3	ρ =0.5	k =8	k =16	k =24	k =32	Avg.	Best
F6	CM	**1.1e-7**	1.8e-7	1.1e-6	4.5e-5	1.8e-7	3.4e-8	**2.0e-8**	2.8e-7	1.1e-9
	DM	**3.0e-7**	9.3e-4	1.0e-3	1.1e-4	1.0e-4	**1.6e-5**	2.1e-3	6.5e-5	9.9e-9
F7	CM	1.05	4.1e-5	**4.0e-5**	**4.1e-7**	2.1e-4	0.16	0.16	1.2e-3	2.8e-7
	DM	1.14	0.10	**2.0e-3**	31	**1.3e-5**	0.19	0.19	6.1e-2	2.7e-7
F8	CM	1.3e-15	**8.2e-16**	2.3e-15	1.9e-12	1.6e-16	0 (27)	**0 (66)**	1.3e-15	0 (62)
	DM	0 (62)	0 (56)	**0 (69)**	0 (49)	0 (54)	0 (66)	**0 (80)**	0	0
F9	CM	**3.7e-4**	5.9e-4	6.0e-4	1.1e-2	2.8e-4	1.9e-4	**1.1e-4**	5.1e-4	7.8e-5
	DM	**3.8e-4**	5.6e-4	9.0e-4	6.0e-3	3.7e-4	**2.1e-4**	2.4e-4	5.8e-4	1.6e-4
F10	CM	**0 (25)**	0 (21)	0 (18)	0 (14)	0 (20)	0 (22)	**0 (29)**	0 (21)	0
	DM	0 (18)	0 (14)	**0 (21)**	0 (13)	0 (19)	0 (14)	**0 (24)**	0 (18)	0
F11	CM	**6.2e-12**	8.5e-12	7.6e-12	9.1e-11	2.0e-11	3.2e-12	**4.9e-13**	7.4e-12	3.6e-13
	DM	**1.2e-19**	1.8e-19	2.0e-19	4.9e-19	1.9e-19	1.5e-19	**5.0e-20**	1.6e-19	3.8e-20

Table 3: Results for {F6, ..., F11} (μ = 400) for CM and DM. Each value in a column of ρ is the average of the best values found in executions for all the studied values of k, of `NRuns` runs each. The same relation holds for the k labeled columns w.r.t. ρ. The best values for each column are boldfaced. The column labeled Avg. is the average performance for each row. The column labeled Best is the best value found in any of the runs. All averages are obtained with the geometric mean.

Index	Ψ	ρ =0.1	ρ =0.3	ρ =0.5	k =8	k =16	k =24	k =32	Avg.	Best
F6	CM	**1.2e-8**	8.8e-8	6.6e-8	2.3e-4	8.2e-8	**1.6e-10**	9.9e-10	4.1e-8	4.8e-11
	DM	**8.7e-8**	0.37	0.36	6.8e-3	2.6e-3	**1.1e-3**	1.4e-3	2.3e-3	6.1e-10
F7	CM	219.5	**0.80**	1.06	146	0.23	**0.22**	145	5.66	2.7e-7
	DM	**1.29**	136	7.8	3.19	184	167	**0.15**	11.10	2.8e-7
F8	CM	**1.9e-11**	3.5e-11	6.3e-11	4.0e-6	4.5e-9	3.6-13	**0 (16)**	3.5e-11	0
	DM	2.8e-11	**2.3e-11**	1.2e-10	1.7e-5	2.9e-9	4.2-13	**1.7e-16**	4.3e-11	0
F9	CM	**1.8e-4**	3.4e-4	9.2e-4	3.7e-2	1.7e-4	**3.1e-5**	5.3e-5	3.8e-4	2.3e-5
	DM	**5.5e-4**	1.1e-3	1.2e-3	6.7e-2	3.4e-4	1.7e-4	**1.6e-4**	9.0e-4	1.2e-4
F10	CM	0 (21)	**0 (28)**	0 (22)	0 (18)	0 (26)	0 (26)	**0 (27)**	0 (24)	0
	DM	0 (27)	0 (22)	**0 (32)**	0 (12)	**0 (39)**	0 (34)	0 (24)	0 (27)	0
F11	CM	8.5e-22	7.0e-22	**3.1e-22**	5.4e-21	1.9e-21	2.5e-22	**4.1e-23**	5.7e-22	1.2e-23
	DM	**1.2e-37**	2.5e-37	1.5e-37	8.4e-37	2.7e-37	1.4e-37	**2.4e-38**	1.7e-37	1.5e-38

Table 4: Results for {F6, ..., F11} ($\mu = 200$) for CM and DM. See Table 3 for an explanation of entries.

Index	Ψ	ρ =0.1	ρ =0.3	ρ =0.5	k =8	k =16	k =24	k =32	Avg.	Best
F6	CM	**3.0e-9**	3.3e-9	2.0e-8	1.1e-3	1.3e-7	7.5e-12	**1.0e-12**	5.8e-9	2.8e-14
	DM	**4.7e-9**	1.1e-3	1.8e-2	1.1e-2	2.7e-5	**9.4e-8**	1.6e-4	4.5e-5	7.4e-13
F7	CM	490	162	**1.24**	222	**0.29**	245	289	46.2	3.9e-7
	DM	343	**135**	136	**145**	146	302	182	185	114.8
F8	CM	1.7e-6	**1.7e-8**	5.9e-5	4.8e-3	1.4e-5	9.6e-9	**2.4e-12**	1.2e-6	0
	DM	**2.2e-6**	9.5e-5	9.4e-4	1.3e-2	1.4e-5	**1.5e-8**	4.2e-3	5.8e-5	8.1e-12
F9	CM	**4.8e-5**	8.3e-5	1.4e-4	4.2e-2	1.6e-4	**1.6e-6**	4.6e-6	8.2e-5	9.0e-7
	DM	**1.2e-4**	2.9e-4	4.3e-4	7.7e-2	2.7e-4	**1.4e-5**	2.5e-5	2.5e-4	1.1e-5
F10	CM	**0 (12)**	1.6e-13	3.4e-16	3.2e-12	0 (8)	3.0e-16	**0 (10)**	1.8e-15	0
	DM	0 (8)	3.1e-13	**0 (10)**	4.6e-12	0 (9)	**0 (12)**	0 (9)	1.5e-15	0
F11	CM	3.2e-10	**9.1e-13**	7.5e-12	1.8e-7	2.9e-10	8.6e-11	**6.3e-18**	1.3e-11	1.0e-22
	DM	**3.8e-31**	8.6e-24	4.2e-23	2.8e-18	1.9e-19	5.6e-33	**2.3e-33**	5.2e-26	7.4e-52

Table 5: Results for {F6, ..., F11} ($\mu = 100$) for CM and DM. See Table 3 for an explanation of entries.

Index	Ψ	ρ =0.1	ρ =0.3	ρ =0.5	k =8	k =16	k =24	k =32	Avg.	Best
F6	CM	**1.9e-10**	9.2e-10	8.9e-8	4.2e-3	3.4e-7	2.7e-10	**0 (4)**	2.5e-9	0
	DM	**1.4e-9**	0.6	0.84	1.2e-2	3.9e-3	1.5e-4	**8.4e-5**	8.9e-4	2.8e-14
F7	CM	620	**162**	193	311	**196**	314	273	269	114.7
	DM	451	136	**37**	**20**	291	183	277	131	0.305
F8	CM	8.8e-4	**4.2e-9**	1.7e-3	1.7e-2	1.0e-6	1.7e-5	**3.6e-7**	1.8e-5	1.1e-16
	DM	1.8e-2	**2.0e-3**	3.1e-3	2.7e-2	**6.8e-5**	1.5e-2	2.0e-2	4.8e-3	3.9e-6
F9	CM	**1.2e-5**	3.2e-5	4.8e-5	4.2e-2	1.4e-4	7.9e-7	**1.1e-7**	2.6e-5	7.9e-8
	DM	**5.3e-5**	1.1e-4	1.7e-4	7.7e-2	2.4e-4	2.7e-6	**2.0e-6**	1.0e-4	1.4e-6
F10	CM	5.4e-14	3.4e-13	**5.1e-15**	2.1e-7	6.4e-16	3.1e-16	**0 (9)**	4.5e-14	0
	DM	6.6e-11	2.1e-6	**6.5e-11**	3.9e-5	5.3e-9	**4.2e-16**	2.1e-7	2.1e-9	6.0e-15
F11	CM	**1.3e-5**	1.9e-5	1.4e-5	1.8e-3	7.0e-5	2.7e-6	**1.6e-7**	1.5e-5	2.4e-8
	DM	1.5e-5	**1.1e-5**	1.7e-5	9.3e-4	2.8e-4	5.9e-7	**2.7e-7**	1.4e-5	5.8e-8

Table 6: Results for {F6, ..., F11} ($\mu = 50$) for CM and DM. See Table 3 for an explanation of entries.

7.1 Discussion for F6 to F11

Two main types of conclusions can be drawn for these mutation experiments: how the parameters k and ρ exert an influence on CM and DM, and what is the relative performance of CM and DM, regardless of the parameter setting. We comment on them separately.

7.1.1 The influence of parameters k and ρ

By inspection of the best results for k and ρ in Tables 3, 4, 5 and 6 (shown in boldface), it seems that there is a tendency according to which better results are generally obtained for lower ρ and (independently) for higher k, towards $\rho = 0.1$ and $k = 24, 32$. This is very clear for F6, F9 and F11. For F8, the value $\rho = 0.3$ seems more appropiate. In contrast, F10 seems to be the easiest function, consistently solved for $\mu \geq 100$ with varying values for k and ρ. Thus, a combination of low ρ with high k seems to be the best choice.

The function that deviates from this tendency is F7, that seems to be by far the harder one for the BGA. Remember that F7 is the only function that does not show any unimodal macrostructure in high dimensions, thus needing a wider search setting, with bigger mutation steps and ranges. This is confirmed by the good results achieved by $\rho = 0.5$ and $k = 8, 16$, a parameter setting completely opposite to $\rho = 0.1$ and $k = 24, 32$, and one that favours big mutation steps. These overall behaviours are also more defined for CM than for DM, and seem to be valid for all of the studied population sizes. However, despite the high numbers of experiments performed and the chosen level of abstraction, these results are by no means concluding, showing a strong dependency (or independency, in the case of F10) on the particular function being optimized.

7.1.2 Relative performance of CM and DM

Here an additional effort of abstraction has to be made. The chosen procedure is the following: we will construct a table for every μ and every function F_i in $\{F6, \ldots, F11\}$ as follows: for each combination (μ, F_i) we look at the overall average performance as shown in Tables 3, 4, 5 and 6 (column labeled Avg.) and we assign a label $L(\mu, F_i)$ from the set {`CM, DM, T, T(CM), T(DM)`}, indicating which of the two operators yields clearly better results. The label `T` stands for a "tie", and a slight superiority of operator `X` is indicated by `T(X)`. More precisely, the conditions for a label to be assigned are the following:

1. A label `CM` is assigned whenever the value for CM is better than that for DM by more than an order of magnitude. Similarly for `DM`.
2. A label `T(CM)` is assigned whenever both numbers are within one order of magnitude, but that for CM is better than that for DM. Similarly for `T(DM)`.
3. A label `T` is assigned whenever both numbers are within one order of magnitude, and approximately equal.

This procedure is of course arguable because is based on a subjective rather than precise numerical computation but has the advantage of being easy and illustrates the information in a very compact way. The result of this analysis is shown in Table 7.

μ	F6	F7	F8	F9	F10	F11
400	C	C	D	T(C)	T	D
200	C	T(C)	T	C	T	D
100	C	C	C	C	T	D
50	C	D	C	T(C)	C	T

Table 7: Relative performance of CM and DM. See text for an explanation of entries.

The table shows a general superiority of CM over DM (number of `C` and `T(C)` over number of `D` and `T(D)`), as exemplified by F6, F7, F8 and F9. In addition, DM works better for higher population sizes, as exemplified by F8, F10 and F11, this last function being the only one for which DM is markedly superior.

7.2 Discussion for F8F2

The function F8F2 is also tested, for $n = 10, 20$ and 50. In this case, a fixed population size $\mu = 100$ is used and up to **FFEvals**= 200,000 evaluations. Also, the arithmetic mean is the averaging method employed because the results are always in the same order of magnitude. The outcome of this experiment is shown in Table 8, where all the results are collectively shown for CM and DM.

n	Ψ	ρ =0.1	ρ =0.3	ρ =0.5	k =8	k =16	k =24	k =32	Avg.	Best
10	CM	0.185	0.186	**0.165**	0.200	**0.155**	0.174	0.185	0.179	0.128
	DM	**0.217**	0.230	0.268	0.245	**0.180**	0.287	0.241	0.238	0.158
20	CM	0.582	**0.517**	**0.525**	**0.497**	0.547	0.546	0.576	0.541	0.427
	DM	**0.608**	0.720	0.765	0.717	0.696	**0.601**	0.776	0.698	0.502
50	CM	1.833	1.852	**1.789**	1.868	**1.778**	1.883	**1.769**	1.825	1.675
	DM	**2.442**	2.467	2.601	2.384	**2.307**	2.581	2.742	2.503	2.234

Table 8: Results for F8F2 ($\mu = 100$), for CM and DM. Each entry is calculated similarly as for the functions {F6, ..., F11}, but using the arithmetic mean. The best results are those boldfaced.

By looking at Table 8 several aspects are noteworthy.

- First, the differences between operator settings are very small and get smaller with decreasing n.
- Again, the difficulty of the function shows in the fact that parameters involving small mutation steps ($k = 24, 32$) seem to work much worse. With regard to the value of ρ, the relative differences in performance are even smaller and makes it risky to draw conclusions. Notwithstanding, there are two parameter settings consistently yielding superior performance: $\rho = 0.1, k = 16$ for DM and $\rho = 0.5, k = 16$ for CM, for all n. Incidentally, the value $\rho = 0.3$ seems to be a bad choice, but the reasons are not clear; possibly it is too big for DM and too small for CM to work properly. For

```
Procedure Recombination-Test ()
{
    Ψ := CM (ρ = 0.5, k = 16);
    forall μ in {400, 200, 100, 50}
      forall F_i in {F6, ..., F11}
        forall Ω in {DR, LR (α =0.5), EIR (δ =0, 0.15, 0.25, 0.35), EIR (range_δ), FR (e =0.5)}
          BGA (μ, F_i, k, ρ, Ψ, NRuns, FFEvals, Ω, τ, Λ, Ξ);
}
```

Figure 6: `Recombination-Test` Algorithm pseudocode.

$k = 8$ the mutation step is too big. That is, once the best precision is correctly set (in this case, $k = 16$), the best results are obtained by quite different range ratios for DM and CM. We suggest that the reason for this different behaviour has to be found in the way both mutation operators work. Recall their expected values for δ were:

$$E\{\delta\}_{(k)} = \frac{1}{k}(2 - 2^{-k+1}) \qquad \text{(DM)}$$

$$E\{\delta\}_{(k)} = 2^{-\frac{k}{2}} \qquad \text{(CM)}$$

It holds that

$$2^{-\frac{k}{2}} < \frac{1}{k}(2 - 2^{-k+1}) \qquad \forall k \geq 4$$

and hence, given k, CM yields on average values much smaller than DM, thus needing a higher ρ to compensate for it and achieve the correct expected mutation rate, as shown in F8F2, for all n.

- A general superiority of CM over DM is also displayed, both on average and in terms of the best value found.
- Last, but not least, the absolute best values found are extremely good for this function, especially for higher n (20 and 50). This issue will be explored in more detail in §9.

8 Experimental results (II): Recombination

In the case of recombination operators, up to eight different settings are tested, as follows: DR, LR with $\alpha = 0.5$, EIR with $\delta = 0, 0.15, 0.25$ and 0.35, EIR with $range_\delta$ and FR with $e = 0.5$. The mutation operator is fixed to CM with $\rho = 0.5, k = 16$. The BGA procedure in the algorithm of Fig. 6 is run 1440 times (240 for each FOP) for every value of μ, making a grand total of 5760 runs.

The results are presented similarly as for mutation. Four tables are given, one for each population size μ, one row per each FOP and one column per each recombination operator, with an additional row showing the average (again using the geometric mean) solution per operator, useful to compare their relative performance. For each configuration, the average and best (in parentheses) solutions found throughout the `NRuns` are shown.

Incidentally, note how the average values —obtained with the geometric mean for each column— are meaningful and express our intuitive idea of an "average". If we look, for example, at the first column (that of DR) of Table 9 we can easily check how the geometric mean is in fact performing the arithmetic mean of the exponents (the actual indicators of the magnitude of a quantity). Not considering the mantissas, these means are roughly 2.7 and 5.9, corresponding to the values 3 and 6 displayed.

F_i	DR	LR ($\alpha = 0.5$)	EIR ($\delta = 0$)	EIR ($\delta = 0.15$)	EIR ($\delta = 0.25$)	EIR ($\delta = 0.35$)	EIR ($range_\delta$)	FR ($e = 0.5$)
F6	2.2e-6	2.61	1.22	1.1e-6	9.0e-8	7.6e-11	0	1.22
	(1.4e-6)	(6.5e-4)	(1.4e-6)	(2.7e-7)	(9.4e-10)	0 (16)	0 (30)	(1.5e-6)
F7	3.82	1031	350	298	334	478	443	445
	(2.0e-6)	(495)	(5.4e-6)	(2.3e-6)	(2.8e-7)	(229.5)	(119.5)	(114.7)
F8	2.6e-2	1.9e-2	4.1e-3	2.9e-3	2.6e-3	1.1e-3	2.5e-3	3.5e-3
	(1.8e-6)	(3.8e-6)	(3.9e-7)	(1.6e-10)	(1.0e-15)	0 (23)	0 (21)	(2.2e-14)
F9	5.6e-4	2.29	5.5e-4	5.1e-4	3.6e-4	9.3e-6	4.0e-15	2.5e-4
	(4.8e-4)	(1.37)	(4.6e-4)	(4.2e-4)	(2.8e-4)	(1.5e-7)	(4.0e-15)	(1.6e-4)
F10	5.8e-3	1.7e-4	3.8e-3	5.2e-3	2.0e-3	3.6e-3	2.6e-3	3.0e-3
	(7.1e-10)	(1.7e-16)	(1.6e-6)	(4.9e-5)	(1.8e-5)	0 (1)	0 (1)	0 (2)
F11	1.7e-2	3.3e-3	2.9e-3	3.3e-4	1.3e-4	2.3e-4	1.6e-4	1.1e-4
	(6.1e-3)	(1.2e-5)	0 (2)	0 (8)	0 (23)	0 (22)	0 (23)	(2.1e-9)
Avg.	**4.8e-3**	**0.20**	**4.7e-2**	**3.1e-3**	**1.4e-3**	**2.6e-4**	**7.5e-7**	**2.3e-2**
	(4.7e-6)	**(3.9e-5)**	**(7.8e-8)**	**(2.4e-8)**	**(7.1e-10)**	**(3.9e-12)**	**(1.9e-13)**	**(2.2e-8)**

Table 9: Results for {F6, ..., F11} ($\mu = 400$) and the eight recombination operators. For each function and recombination operator, two numbers are shown: the average result and the best result (in parentheses) along the runs. The bottom rows (boldfaced) show the average behaviour per operator throughout all the functions.

F_i	DR	LR ($\alpha = 0.5$)	EIR ($\delta = 0$)	EIR ($\delta = 0.15$)	EIR ($\delta = 0.25$)	EIR ($\delta = 0.35$)	EIR ($range_\delta$)	FR ($e = 0.5$)
F6	0.10	4.98	2.13	0.33	0.64	0.90	0.61	0.50
	(1.6e-6)	(1.00)	(7.6e-6)	(2.6e-7)	(1.7e-7)	(3.0e-8)	(2.2e-12)	(8.4e-12)
F7	57.4	573	389	421	394	444	436	497
	(4.3e-6)	(229.5)	(114.7)	(114.7)	(114.7)	(6.1e-7)	(232)	(229.5)
F8	4.5e-2	6.3e-2	2.7e-2	2.0e-2	1.7e-2	9.4e-3	9.1e-3	1.8e-2
	(2.8e-6)	(1.2e-2)	(2.1e-6)	(1.1e-6)	(3.6e-8)	(1.3e-13)	0 (13)	0 (1)
F9	7.6e-4	2.39	7.1e-3	7.6e-4	4.8e-4	2.5e-4	2.6e-6	6.2e-5
	(5.9e-4)	(1.79)	(1.9e-3)	(5.4e-4)	(3.8e-4)	(1.3e-4)	(5.5e-7)	(2.6e-6)
F10	7.1e-3	2.1e-3	6.5e-3	5.2e-3	5.6e-3	5.2e-3	5.2e-3	5.2e-3
	(1.7e-8)	(1.8e-13)	0 (3)	0 (7)	0 (3)	0 (5)	0 (8)	0 (10)
F11	2.0e-2	6.1e-3	3.4e-3	7.3e-4	4.4e-4	1.2e-4	1.9e-4	8.9e-4
	(5.8e-3)	(7.5e-4)	(6.4e-5)	0 (3)	0 (7)	0 (19)	0 (15)	0 (18)
Avg.	**5.5e-2**	**0.42**	**0.12**	**4.5e-2**	**4.1e-2**	**2.9e-2**	**1.4e-2**	**3.3e-2**
	(1.0e-5)	**(2.9e-3)**	**(5.3e-6)**	**(2.4e-8)**	**(1.2e-8)**	**(3.8e-11)**	**(2.6e-11)**	**(4.1e-11)**

Table 10: Results for {F6, ..., F11} ($\mu = 200$) and the eight recombination operators. See Table 9 for an explanation of entries.

F_i	DR	LR ($\alpha = 0.5$)	EIR ($\delta = 0$)	EIR ($\delta = 0.15$)	EIR ($\delta = 0.25$)	EIR ($\delta = 0.35$)	EIR ($range_\delta$)	FR ($e = 0.5$)
F6	3.3e-2 (1.6e-6)	0.56 (1.7e-6)	0.30 (1.1e-6)	0.10 (5.0e-7)	0.13 (1.7e-7)	0.23 (9.8e-8)	0.23 (1.7e-8)	0.10 (1.9e-7)
F7	161 (4.1e-6)	509 (229.5)	486 (114.7)	486 (114.7)	493 (229.5)	509 (114.7)	493 (114.7)	486 (114.7)
F8	3.6e-2 (7.4e-3)	5.7e-2 (1.4e-5)	5.3e-2 (1.2e-2)	4.7e-2 (1.5e-2)	4.2e-2 (7.4e-3)	3.3e-2 (2.2e-7)	2.9e-2 (7.3e-10)	4.3e-2 (7.4e-3)
F9	6.5e-4 (5.2e-4)	3.7e-2 (1.8e-3)	5.3e-4 (4.0e-4)	4.5e-4 (3.5e-4)	4.3e-4 (3.3e-4)	3.8e-4 (2.4e-4)	2.3e-4 (1.1e-4)	3.4e-4 (2.0e-4)
F10	8.4e-3 (1.4e-7)	5.5e-3 (4.7e-15)	7.4e-3 0 (2)	7.5e-3 0 (2)	6.8e-3 0 (4)	6.8e-3 0 (5)	8.1e-3 0 (3)	6.2e-3 0 (5)
F11	2.7e-2 (7.8e-3)	9.1e-3 (6.3e-4)	1.0e-2 (9.2e-4)	8.3e-3 (4.7e-7)	7.5e-3 (4.7e-10)	3.3e-3 (1.1e-15)	7.1e-4 0 (5)	2.6e-3 (6.5e-10)
Avg.	**5.5e-2 (5.5e-5)**	**0.18 (1.7e-5)**	**8.2e-2 (1.9e-5)**	**6.3e-2 (4.9e-6)**	**6.2e-2 (1.3e-6)**	**5.7e-2 (2.0e-8)**	**4.0e-2 (3.4e-9)**	**4.7e-2 (1.1e-6)**

Table 11: Results for {F6, ..., F11} ($\mu = 100$) and the eight recombination operators. See Table 9 for an explanation of entries.

F_i	DR	LR ($\alpha = 0.5$)	EIR ($\delta = 0$)	EIR ($\delta = 0.15$)	EIR ($\delta = 0.25$)	EIR ($\delta = 0.35$)	EIR ($range_\delta$)	FR ($e = 0.5$)
F6	8.0e-2 (2.1e-6)	0.14 (4.1e-7)	6.6e-2 (6.4e-7)	1.4e-2 (5.4e-7)	0.10 (1.9e-7)	0.13 (3.7e-7)	0.10 (3.0e-7)	5.0e-4 (5.6e-7)
F7	340 (1.0e-5)	497 (229.5)	486 (229.5)	440 (114.7)	497 (114.7)	532 (114.7)	501 (4.5e-6)	520 (114.7)
F8	5.0e-2 (1.2e-2)	4.6e-2 (3.5e-6)	4.6e-2 (4.6e-6)	4.3e-2 (1.7e-2)	4.2e-2 (9.9e-3)	4.6e-2 (7.4e-3)	4.3e-2 (9.9e-3)	5.4e-2 (1.2e-2)
F9	6.4e-4 (5.8e-4)	5.0e-4 (4.2e-4)	4.4e-4 (2.8e-4)	4.2e-4 (3.1e-4)	4.1e-4 (3.1e-4)	3.9e-4 (2.9e-4)	3.8e-4 (2.5e-4)	4.1e-4 (3.0e-4)
F10	9.4e-3 (9.8e-7)	8.1e-3 (7.8e-10)	9.1e-3 (1.3e-11)	8.4e-3 (1.0e-13)	8.4e-3 (5.2e-10)	9.4e-3 (1.4e-9)	8.4e-3 0 (1)	8.1e-3 (1.0e-12)
F11	3.0e-2 (1.5e-2)	1.5e-2 (1.9e-3)	1.4e-2 (1.2e-3)	1.5e-2 (2.5e-3)	8.7e-3 (6.7e-5)	1.4e-2 (8.5e-5)	1.2e-2 (5.0e-7)	1.7e-2 (3.3e-4)
Avg.	**7.9e-2 (1.1e-4)**	**7.6e-2 (7.7e-5)**	**6.6e-2 (3.8e-5)**	**4.9e-2 (6.6e-5)**	**6.3e-2 (1.2e-4)**	**7.4e-2 (1.5e-4)**	**6.6e-2 (2.3e-7)**	**3.0e-2 (6.5e-5)**

Table 12: Results for {F6, ..., F11} ($\mu = 50$) and the eight recombination operators. See Table 9 for an explanation of entries.

8.1 Comments on Recombination

The most interesting comparison is the relative performance of the different operators. To this end, two numbers are shown for each entry in the tables: the average performance (geometric mean) and the best value found, over the `NRuns` runs. This last number is the one shown in parentheses. Again, if a function is solved, a 0 is displayed along with the *number of times* (not the percentage) it is solved. In addition, for each operator, its average performance (geometric mean of average and best values throughout all of the functions) is presented at the bottom row. The use of the geometric mean here is due, as usual, to the integration of very dissimilar quantities (in orders of magnitude). Several conclusions are noteworthy:

1. It seems clear that LR ($\alpha = 0.5$) is by far the worst operator, and for all population sizes. The only exception is for $\mu = 50$ where it is second worst, after DR. This operator is effective only for F10, but all other operators are. Its offspring generation mechanism, the "exact middle point" with probability one, seems too rigid to be of general application, at least for the configurations tested. This said, it seems to be the less affected by population size.

2. Conversely, DR is the second worst for all population sizes after LR ($\alpha = 0.5$) and the worst for $\mu = 50$. Also, its performance decreases as μ does to the point that, for $\mu = 50$, even LR ($\alpha = 0.5$) is better. On the other hand, this operator is very appropiate for solving F7. We suggest that this is because, once mutation has found the correct value for a dimension (which, incidentally, is the same for all dimensions and equal to 420.969) the offspring inherits it with very high probability (0.5). On the other hand, the other operators perform a kind of linear combination between the parents's values that is no good for this function. In short, this operator is the less disruptive.

3. In what regards EIR ($\delta \in \{0, 0.15, 0.25, 0.35\}$), it can be seen that this operator exhibits a neat overall behaviour, First, it is clearly superior to DR and LR, regardless of δ. Moreover, performance is consistently better (both on average and best values) for increasing values of δ, for $\mu = 400, 200, 100$, without exception. This a nobler and more robust operator than the other two (with the exception of F7). Interestingly enough, for $\mu = 50$, the odds turn and we see an inversion in behaviour, being now $\delta = 0$ the best value and $\delta = 0.35$ the worst (although differences are small). The reason may be behind the fact that, for this population size, there is a narrower margin for exploration, which is what greater values of δ favour.

4. The method $range_\delta$ for dynamically calculating the δ seems the best recombination operator when applied to EIR. It is so for all population sizes, both on average (except for $\mu = 50$) and in the best solution found, which is very remarkable. Our suggestion is that this is the "wilder" setting because it less restricts the offspring range. This is a tactics that do seems to work. Second and third best are, consistently, EIR with $\delta = 0.35$ and FR ($e = 0.5$). It is interesting to note that these two operators favour, as EIR with $range_\delta$ does, a wider offspring range than the rest (wider in the sense of leaving the parents's scope). In addition, the behaviour of these three operators decreases strongly with μ. It seems as if these operators need a certain population size to work, falling rather short (although still competitive) at $\mu = 50$ and doing increasingly well as μ grows. Note also that F7 is the only function resistant to these operators: they find acceptable solutions now and then: EIR ($\delta \in \{0, 0.15, 0.25\}$) find it at $\mu = 400$, EIR ($\delta = 0.35$) at $\mu = 200$, EIR ($range_\delta$) at $\mu = 50$ and FR and LR never. In contrast, DR finds some for all population sizes.

The function F8F2 is also tested with the different recombination operators. The parameter setting is the same as for mutation on this function: $\mu = 100$ and `FFEvals`= 200,000. The rest of the parameters are the same as those used for the other functions. The operator CM with $\rho = 0.5$ and $k = 16$ is again fixed for the experiment. The function is tested in n=10, 20 and 50 dimensions and the results —obtained in exactly the same way than for the previous experiments on this function— are displayed in Table 13.

We first note that the results are excellent and even better solutions are found, for the three dimensions considered, when compared to Table 8. Also, again the differences between operators are quite narrow although —as is reasonable to expect— greater at higher dimensions.

n	DR	LR ($\alpha = 0.5$)	EIR ($\delta = 0$)	EIR ($\delta = 0.15$)	EIR ($\delta = 0.25$)	EIR ($\delta = 0.35$)	EIR ($range_\delta$)	FR ($e = 0.5$)
10	0.332	0.317	0.340	0.351	0.327	0.328	0.320	0.321
	(0.090)	(0.158)	(0.099)	(0.213)	(0.170)	(0.121)	(0.170)	(0.089)
20	0.784	0.717	0.746	0.739	0.759	0.805	0.738	0.807
	(0.437)	(0.491)	(0.484)	(0.396)	(0.533)	(0.489)	(0.437)	(0.540)
50	2.539	2.284	2.215	2.104	2.368	2.453	2.016	2.455
	(1.787)	(1.542)	(1.796)	(1.623)	(1.559)	(1.796)	(1.463)	(1.568)

Table 13: Results for F8F2 ($\mu = 100$) and the eight recombination operators. Each entry shows the average (arithmetic mean) and best results (in parentheses) for a given operator and dimension.

The best operators seem to be LR ($\alpha = 0.5$) and EIR with ($range_\delta$). Other good settings are EIR ($\delta \in \{0, 0.15\}$). The operator EIR with ($range_\delta$) is specially good at $n = 50$. This shows that EIR is more than often the best operator, changing only the way the δ is computed. Other operators, like DR and FR are only competitive at low dimensions, where they yield good best values. A word of caution is in order here, because EIR ($\delta = 0.35$) appears as the overall worst.

9 A Further Study on F8F2

In order to draw more useful conclusions and to make comparison to other techniques possible, a more complete experiment setup on this function is performed. Keeping the parameter setup of $\mu = 100$ and `FFEvals`= 200,000, with the usual operator CM with $\rho = 0.5$ and $k = 16$, this time `NRuns` = 50 runs per experiment are executed, and the function is tested in a higher number of dimensions (up to 200). The remaining parameters are those kept constant for all the work (Table 2).

Besides, since this experiment deals with the same function in many different dimensions, a more informative and independent estimate of overall performance than the crude (arithmetic) mean for all n is used. Let $m(\Omega, n)$ denote a given measure (either average or best) for operator Ω at dimension n, as shown in the table. We define the *score* of a measure as its average along all dimensions, inversely weighted by the dimension value. Let N bet the set of dimensions for which the function is tested. Formally,

$$score(\Omega) = \frac{1}{|N|} \sum_{n \in N} \frac{m(\Omega, n)}{n}$$

The score yields the average value per dimension obtained along the entire column. This score can be computed for the average and for the best values found by each operator. The lowest the score, the better the operator. As for Table 14, $N = \{10, 20, 50, 100, 200\}$.

The results are in general accordance with those in Table 13. The most remarkable one is that EIR ($range_\delta$) is definitely the best setting, yielding the lowest score on average and on the best results found. It is also the more robust, as can be checked from the table looking at the variances (score for them not shown). At a distance come EIR ($\delta \in \{0, 0.15, 0.25\}$). This means EIR ($\delta = 0.25$) has finally shown up to be superior to LR ($\alpha = 0.5$). The setting EIR ($\delta = 0.35$) and the operators DR and FR are the worst, although these last two, as before, do well at low dimensions.

A comparison follows to results using other search techniques. The main source of data for this function is [11], which usefully reproduces the original results in [37]. From the tables in these references, we reproduce (Table 15) the most relevant results: the mean and variance for this function in $N = \{10, 20, 50, 100\}$ dimensions as found by Eshelman's adaptive CHC [14], the Random Bit Climber by Davis (RBC) [8], a fairly standard GA –although with the incorporation of elitism and tournament selection (ESGAT) [20], Whitley's Genitor GA [38] and Line Search Algorithm [37] and the aforementioned

n	DR	LR ($\alpha = 0.5$)	EIR ($\delta = 0$)	EIR ($\delta = 0.15$)	EIR ($\delta = 0.25$)	EIR ($\delta = 0.35$)	EIR ($range_\delta$)	FR ($e = 0.5$)
10	0.327	0.331	0.346	0.356	0.328	0.311	0.320	0.334
	0.016	0.013	0.012	0.010	0.006	0.064	0.007	0.013
	(0.090)	(0.129)	(0.099)	(0.213)	(0.170)	(0.121)	(0.170)	(0.089)
20	0.796	0.718	0.723	0.693	0.722	0.776	0.746	0.795
	0.062	0.016	0.028	0.035	0.024	0.039	0.027	0.049
	(0.437)	(0.491)	(0.420)	(0.396)	(0.415)	(0.489)	(0.437)	(0.415)
50	2.467	2.298	2.215	2.110	2.263	2.434	2.048	2.445
	0.165	0.091	0.078	0.099	0.163	0.140	0.073	0.209
	(1.787)	(1.542)	(1.693)	(1.623)	(1.559)	(1.796)	(1.361)	(1.568)
100	6.222	6.075	5.566	5.517	5.668	6.200	5.191	6.924
	0.775	0.422	0.448	0.400	0.491	0.614	0.269	0.940
	(4.386)	(4.809)	(4.198)	(4.444)	(3.754)	(4.111)	(3.830)	(4.842)
200	16.937	17.840	14.988	14.918	15.675	17.386	13.098	19.602
	2.747	3.013	0.962	1.262	2.435	2.117	1.073	3.981
	(12.067)	(15.147)	(12.886)	(12.866)	(12.769)	(14.427)	(10.598)	(15.565)
Score	**0.054**	**0.053**	**0.049**	**0.048**	**0.050**	**0.054**	**0.046**	**0.058**
	(0.034)	**(0.038)**	**(0.034)**	**(0.036)**	**(0.034)**	**(0.037)**	**(0.031)**	**(0.037)**

Table 14: Results for F8F2 ($\mu = 100$) and the eight recombination operators. Each entry shows the *average* (top, using arithmetic mean), *variance* and *best* results (in parentheses) for a given operator and dimension. The scores (boldfaced) are computed for the average and best results along each column.

Evolution Strategies (here named ES-1 and ES-2, the latter being a setting tuned for this function). These algorithms show a variety of search strategies that, in spite of the fact that they make comparison a difficult task, their collective behaviour give a more complete picture of the difficulty of a function.

n	CHC	RBC	ESGAT	Genitor	Line	ES-1	ES-2
10	1.344	0.139	4.077	4.365	3.029	2.117	0.986
	0.921	0.422	2.742	2.741	5.153	1.417	0.000
20	5.630	7.243	47.998	21.452	2.502	8.852	3.968
	2.862	11.289	32.615	19.459	3.280	21.698	0.021
50	75.099	301.561	527.100	398.120	2.652	141.679	32.339
	49.644	72.745	176.988	220.284	2.558	155.052	2.574
100	670.223	1655.557	2991.890	2844.389	1606.400	1579.264	840.151
	377.590	605.268	596.470	655.159	1583.500	631.193	355.346

Table 15: Results for F8F2 for several other search methods. Each entry shows the *average* (top) and *standard deviation* (bottom). The results have been rounded by the author to three decimals.

As can be checked from the table, only RBC in ten dimensions gives mean better results. In general, Line is the only algorithm that keeps pace, though always behind the BGA. For $n = 100$, the limited number of `FFEvals` is finally not sufficient for the methodical scheme of Line, and its performance dramatically drops.

10 Conclusions

We have performed a detailed study on function optimization with the Breeder Genetic Algorithm (BGA), a member of the big family of Evolutionary Algorithms (EAs). It has been shown that the BGA can effectively cope with a variety of test-bed functions to satisfaction, achieving results that are superior to those obtained with other EAs. The rich variation on genetic operators –parameterized forms of Mutation and Recombination– has been analyzed and put to a comparison. Their relative performance and dependency on the parameters have been the main topics addressed. In addition, a simple method for estimating the main Recombination parameter has been introduced.

As for **mutation**, it seems that more difficult functions require bigger average mutation steps, controlled by the values of ρ and k. This is clearly seen for F7 and confirmed for F8F2, neither of these two functions showing a unimodal macrostructure. The problem is that we may not know this feature a priori. Hence, different problems may well require radically divergent settings. The only reasonably good news is that small differences between values of ρ do not result in significatively different performances, so the search for good settings for a particular problem can be done with a fixed value for this parameter (e.g. $\rho = 0.1$ or 0.5; we recommend discarding 0.3). These findings tell us that the most appropiate choice of operators and parameters depends on the *exact form* of the objective function; this form, as we have seen, may change with the number of dimensions.

On the other hand, the influence of k has been found to be more profound than that of ρ, for which results are much more balanced on average. These two variables together are best regarded as *mutation step* controllers, rather than *precision* controllers. We have shown how they can lead for certain functions to quite different solutions (even in several orders of magnitude) exerting a strong influence in algorithm performance, apparently regardless of the population size. In this sense, high values for k, (e.g. 24, 32) and low values of ρ (e.g. 0.1) seem to be specially suited for those problems for which roughly good solutions are found with no difficulty —so that this configuration is likely to find some— and once these near-solutions have been found, a fine-tuning process can greatly improve on them. This fine-tuning would be performed by the small mutation steps that this setting favours. Nevertheless, for other problems —apparently more difficult like, for example, F7 and F8F2— better results are attained with "opposite" mutation settings –low values of k and high values of ρ. All this suggests a dynamic procedure to find the most appropiate values, possibly within a predeclared range, in which initially wide mutation steps are allowed and progressively reduced, as the search concentrates in some parts of the space. This could be implemented via a kind of annealing schedule.

Regarding the choice between the two mutation operators –Continuous Mutation (CM) and Discrete Mutation (DM), it is difficult to draw any general conclusions. The continuous operator seems to work better or at least as good as the discrete one in the majority of the functions tested, a difference in performance that decreases with increasing μ. Surely one is better suited for some FOPs than for others, but the precise situations for which this could hold true are by no means clear. In any case, both operators seem to be similarly affected by their parameters (ρ, k) so that, once they are correctly set, both operators can be given a try.

The results on **recombination** can be synthesized as follows. The operator EIR has been found to be generally superior to the other operators tested (DR, LR and FR). All of them show the advantage, with respect to mutation operators, of depending only on one parameter. In the case of EIR, the δ parameter has been drawn from the set $\{0, 0.15, 0.25, 0.35\}$ and also dynamically computed with the method $range_\delta$, temptatively introduced in this work. The results show that this last schedule consistently outperforms all of the others, both on average and on the overall best results found. Additionally, the experiments devoted specifically to the function F8F2 in several dimensions show how the BGA is a powerful and robust search algorithm, superior to all previous results (up to our knowledge) for this function. The robustness stems from the fact that all of the operators perform in a relatively narrow margin of solutions. Also, the variance of the solutions is found to be small. However, further studies are needed to explore the scalability of performance versus number of dimensions for this function, as well as for others found via the method of expansion-composition.

Although an empirical study, the high numbers of experiments performed for each function permits to draw conclusions and gain an insight that can be put in practice when solving real-world problems. The only limitation is the small number of FOPs; although chosen to show a variety of features, clearly more difficult problems need to be tackled in carefully controlled experiments. However, the results for {F6, ..., F11} and F8F2 are very good and, although direct comparisons are complicated due to the particularities of different algorithms, the obtained solutions would surely have been considered as satisfactory in real applications. This more through study with a richer set of artificially created and controlled FOPs, possibly in the presence of constraints and noise should pave the way to new benchmarks and conclusions.

References

[1] Bäck, Th., Schwefel, H.P. Evolutionary Computation: An Overview. In Procs. of the 3rd IEEE Conference on Evolutionary Computation, pp. 20-29, IEEE Press, Piscataway NJ, 1996.

[2] Bäck, Th., Schwefel, H.P. An Overview of Evolutionary Algorithms for Parameter Optimization. *Evolutionary Computation*, 1 (1): 1-23, 1993.

[3] Bäck, Th., Schwefel, H.P. A Survey of Evolution Strategies. In Procs. of the 4th Intl. Conf. on Genetic Algorithms, (eds.) Booker and Belew, Morgan Kaufmann: San Mateo, pp. 2-9, 1991.

[4] Bäck, Th. Evolution Strategies: An alternative Evolutionary Algorithm. Technical Report. Informatik Centrum Dortmund.

[5] Bäck, Th. Optimal mutation rates in genetic search. In Procs. of the 5th Intl. Conf. on Genetic Algorithms, (ed.) Forrest, S., Morgan Kaufmann, pp. 2-8, 1993.

[6] Bäck, Th. Evolutionary Algorithms in Theory and Practice. Oxford Univ. Press, New York, 1996.

[7] Bremermann, H.J., Rogson, J., Salaff, S. Global properties of evolution processes. In Pattee, H.H. (Ed.) Natural Automata and Useful Simulations, pp. 3-42, 1966.

[8] Davis, L. Representational Bias and Test Suite Design. In Procs. of the 4th Intl. Conf. on Genetic Algorithms, (eds.) Booker and Belew, Morgan Kaufmann: San Mateo, pp. 2-9, 1991.

[9] De Falco, I. An introduction to Evolutionary Algorithms and their application to the Aerofoil Design Problem - Part I: the Algorithms. Invited paper at the von Karman Lecture Series 1997, Bruxelles.

[10] De Falco, I., Del Balio, R., Della Cioppa, A., Tarantino, E. A Comparative Analysis of Evolutionary Algorithms for Function Optimisation. In Procs. of the Second Workshop on Evolutionary Computing (WEC2), Nagoya, JAPAN.

[11] De Falco, I., Del Balio, R., Della Cioppa, A., Tarantino, E. Evolution Strategies and Composed Test Functions. In Procs. of the First Workshop on Soft Computing (WSC1), Nagoya, JAPAN.

[12] De Jong, K.A. An analysis of the behaviour of a class of genetic adaptive systems. Ph. D. Thesis, Univ. of Michigan, 1975.

[13] Eiben, A.E. van Kemenade, C.H.M. Diagonal Crossover in Genetic Algorithms for Numerical Optimization. *Journal of Control and Cybernetics*, 26(3):447-465, 1997.

[14] L. Eshelman. The CHC Adaptive Search Algorithm. Morgan Kaufmann, 1991.

[15] Fogarty, T.C. Varying the probability of mutation in the genetic algorithm. In Procs. of the 3rd Intl. Conf. on Genetic Algorithms, (ed.) Schaffer, J.D., Morgan Kaufmann, pp. 9-14, 1989.

[16] Fogel, L.J. Toward inductive inference automata. In Procs. of the Intl. Federation for Information Processing Congress, pp. 395-399, Munich, 1962.

[17] Fogel, L.J., Owens, A.J., Walsh, M.J. Artificial Intelligence through Simulated Evolution. Wiley, NY, 1966.

[18] Fogel, L.J. An analysis of evolutionary programming. In Fogel and Atmar (Eds.) Procs. of the 1st annual conf. on evolutionary programming. La Jolla, CA: Evolutionary Programming Society, 1992.

[19] Glover, F. Future Paths for Integer Programming and Links to Artificial Intelligence. *Computers and Ops. Res.*, 5: 533-549 (1986).

[20] Goldberg, D.E. Genetic Algorithms for Search, Optimization & Machine Learning. Addison-Wesley, 1989.

[21] Heitkötter, J., Beasley, D. (Eds.). The Hitch-Hiker's Guide to Evolutionary Computation: A list of Frequently Asked Questions (FAQ). USENET: comp.ai.genetic. Available via anonymous FTP from **rtfm.mit.edu:/pub/usenet/news.answers/ai-faq/genetic**.

[22] Holland, J.H. Outline for a logical theory of adaptive systems. J. of the ACM, 3: 297-314, 1962.

[23] Holland, J.H. Adaptation in natural and artificial systems. The University of Michigan Press. Ann Arbor, MI, 1975.

[24] Khuri, S., Bäck, Th., Heitkötter, J. An Evolutionary Approach to Combinatorial Optimization Problems. In Procs. of the 22nd Annual ACM Computer Science Conf. Cizmar, D. (Ed.) ACM Press, pp. 66-73, New York, 1994.

[25] Kirkpatrick, S., Gellat C.D., Vecchi, M.P. Optimization by Simulated Annealing. *Science*, 220: 671:680 (1983).

[26] Metropolis, N., Rosenbluth, A.W., Teller, A.H., Teller, E., Equation of State Calculation by Fast Computing Machines. *J. of Chem. Phys.*, 21: 1087-1091 (1953).

[27] Michalewicz, Z. Genetic Algorithms, Numerical Optimization and Constraints. In Proc. of the 6th Intl. Conf. on Genetic Algorithms, (ed.) L. Eshelman, Morgan Kaufmann, pp. 151-158, 1995.

[28] Michalewicz, Z. Genetic Algorithms + data structures = evolution programs. Artificial Intelligence. Springer, Berlin, 1992.

[29] Mühlenbein, H., Schlierkamp-Voosen, D. Predictive Models for the Breeder Genetic Algorithm. *Evolutionary Computation*, 1 (1): 25-49, 1993.

[30] Mühlenbein, H., Voigt, H.M. Gene Pool Recombination in Genetic Algorithms. In Proc. of the Metaheuristics Intl. Conf., (eds.) Osman, I.H., Kelly, J.P., Kluwer Academic Publishers, 1995.

[31] Rayward-Smith, V.J., Osman, I.H., Reeves, C.R., Smith, G.D. (Editors). Modern Heuristic Search Methods. Wiley, 1996.

[32] Rechenberg, I. Cybernetic solution path of an experimental problem. Royal Aircraft Establishment, Library Translation No. 1122, Farnborough, Hants., UK, August 1965.

[33] Rechenberg, I. Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution. Frommann-Holzboog, Stuttgart, 1973.

[34] Schwefel, H.P. Kybernetische Evolution als Strategie der experimentellen Forschung in der Strömungstechnik. Diplomarbeit, Technische Universität Berlin, 1965.

[35] Schwefel, H.P. Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie. Vol. 26 of Interdisciplinary Systems Research. Birkhäuser, Basel, 1977.

[36] Voigt, H.M., Mühlenbein, H., Cvetkovic, D. Fuzzy recombination for the continuous Breeder Genetic Algorithm. In Proc. of the 6th Intl. Conf. on Genetic Algorithms, (ed.) L. Eshelman, Morgan Kaufmann: San Mateo, 104-113, 1995.

[37] Whitley, D., K. Mathias, S. Rana, J. Dzubera. Evaluating Evolutionary Algorithms. *Artificial Intelligence* Volume 85, pp. 245-2761, 1996.

[38] Whitley, D. The GENITOR algorithm and Selective Pressure: why rank based allocation of Reproductive Trials is best. In Procs. of the 3rd Intl. Conf. on Genetic Algorithms, (ed.) Schaffer, J.D., Morgan Kaufmann, 1989.

[39] Whitley, D. A Genetic Algorithm Tutorial. *Statistics and Computing*, 4: 65-85, 1994.