Applied Intelligence 13, 187–204, 2000

Two-Loop Real-Coded Genetic Algorithms with Adaptive Control of Mutation Step Sizes

F. HERRERA AND M. LOZANO
Department of Computer Science and A.I., University of Granada, 18071 Granada, Spain
herrera@decsai.ugr.es
lozano@decsai.ugr.es

Abstract. Genetic algorithms are adaptive methods based on natural evolution that may be used for search and optimization problems. They process a population of search space solutions with three operations: selection, crossover, and mutation. Under their initial formulation, the search space solutions are coded using the binary alphabet, however other coding types have been taken into account for the representation issue, such as real coding. The real-coding approach seems particularly natural when tackling optimization problems of parameters with variables in continuous domains.

A problem in the use of genetic algorithms is premature convergence, a premature stagnation of the search caused by the lack of population diversity. The mutation operator is the one responsible for the generation of diversity and therefore may be considered to be an important element in solving this problem. For the case of working under real coding, a solution involves the control, throughout the run, of the strength in which real genes are mutated, i.e., the step size.

This paper presents TRAMSS, a Two-loop Real-coded genetic algorithm with Adaptive control of Mutation Step Sizes. It adjusts the step size of a mutation operator applied during the inner loop, for producing efficient local tuning. It also controls the step size of a mutation operator used by a restart operator performed in the outer loop, for reinitializing the population in order to ensure that different promising search zones are focused by the inner loop throughout the run. Experimental results show that the proposal consistently outperforms other mechanisms presented for controlling mutation step sizes, offering two main advantages simultaneously, better reliability and accuracy.

Keywords: real-coded genetic algorithms, premature convergence, mutation operator

1. Introduction

Genetic algorithms (GAs) are general purpose search algorithms which use principles inspired by natural genetic populations to evolve solutions for problems [1, 2]. The basic idea is to maintain a population of chromosomes, which represent candidate solutions for the specific problem, which evolves over time through a process of competition and controlled variation. The following bibliography may be examined for a more detailed discussion about GAs: [1–5].

Under their initial formulation, the search space solutions are coded using the binary alphabet. However, other coding types have been considered for the representation issue, such as real coding, which would seem particularly natural when tackling optimization problems of parameters with variables on continuous domains. Then a chromosome is a vector of floating point numbers, the size of which is kept the same as the length of the vector, which is the solution to the problem. GAs with this type of coding are called *real-coded GAs* (RCGAs) [6, 7]. There are other types of *evolutionary algorithms*, i.e., implementing the idea of evolution [3], which are based on real coding as well. These are *evolution strategies* [8] and *evolutionary programming* [9]. This paper deals with RCGAs.

Population diversity is crucial to a GA's ability to continue the fruitful exploration of the search space [10]. If the lack of population diversity takes place too early, a premature stagnation of the search is caused. Under these circumstances, the search is likely to be trapped in a local optimum before the global optimum is found. This problem, called *premature convergence*, has long been recognized as a serious failure mode for GAs [11].

The mutation operator may be considered to be an important element for solving the premature convergence problem, since it serves to create random diversity in the population [12]. Different techniques have been suggested for the *control*, during the GA's run, of parameters associated with this operator, depending on either the current state of the search or other GA related parameters [13–15]. They try to offer suitable diversity levels for avoiding premature convergence and improving the results. In the case of working with real coding, a topic of major importance involves the control of the proportion or strength in which real-coded genes are mutated, i.e., the *step size* [16].

The objective of this paper is to formulate a mechanism for the control of mutation step sizes for RCGAs, which should handle and maintain population diversity that in some way helps produce good chromosomes, i.e., *useful diversity* [17]. We present TRAMSS, a *Two-loop RCGA model with Adaptive control of Mutation Step Sizes* that attempts to do this. It consists of two loops, an *inner* loop and an *outer* one:

- The inner loop is designed for processing useful diversity in order to lead the population toward the most promising search areas, producing an effective refinement on them. So, its principal mission is to obtain the best possible *accuracy* levels.

  The inner loop performs the selection process and fires the crossover and mutation operators. Furthermore, for achieving its objective, it controls the step size of the mutation operator.
- The outer loop introduces new population diversity, after the inner loop reaches a stationary point where there are no improvements, that helps the next one to reach better solutions. Therefore, it attempts to induce *reliability* in the search process.

  The outer loop iteratively performs the inner one, and later, it applies a *restart operator* that reinitializes the population by mutating all genes, using a step size that is adapted as well, throughout the runs for this loop.

The paper is set up as follows. In Section 2, we analyze two mutation issues, the ways in which the control of mutation step sizes may be made and the idea of the restart operator. In Section 3, we present TRAMSS. In Section 4, we study TRAMSS from an empirical point of view, by dealing with the following issues: 1) performance improvement, i.e., if its results on a given test suite are better than the ones obtained using other mechanisms for controlling mutation step sizes that were proposed in the GA literature, 2) adaptation itself, i.e., if it adjusts the mutation step size according to the particularities of the problem to be solved, and 3) their relation, i.e., if adaptation is responsible for the performance improvement. Finally, some conclusions are dealt with in Section 5.

2. Mutation Issues

In this section, we explain two issues that will be included as important components in the conceptual foundation of TRAMSS, mutation step size control (Subsection 2.1) and the restart operator (Subsection 2.2).

2.1. Mutation Step Size Control

In general, the mechanisms presented for controlling parameters associated with evolutionary algorithms may be assigned to the following three categories [15]:

- *Deterministic control*. It takes place if the values of the parameters to be controlled are altered by some deterministic rule, without using any feedback from the evolutionary algorithm. Usually, a time-varying schedule is used.
- *Adaptive control*. It takes place if there is some form of feedback from the evolutionary algorithm that is used to determine the direction and/or magnitude of the change to the parameters to be controlled.

  The rules for updating parameters that are used by this type of control and, by the previous one, are termed *absolute adaptive heuristics* [13] and, ideally, capture some lawful operation of the dynamics of the evolutionary algorithm over a broad range of problems.
- *Self-adaptive control*. The parameters to be controlled are encoded onto the chromosomes of the individual and undergo mutation and recombination.

  Self-adaptation was initially proposed for evolution strategies [18] and was extended to GAs in

[19–21]. This type of control exploits the indirect link between favorable control parameter values and objective function values, with the parameters being capable of adapting implicitly, according to the topology of the objective function [20].

Next, we describe mechanisms for the control of mutation step sizes that belong to each one of these categories.

2.1.1. Deterministic Step Size Control. In [5], a mutation operator for RCGAs, called *non-uniform mutation*, was presented, which is based on the absolute adaptive heuristic "*to protect the exploration in the initial stages and the exploitation later*". It implements this idea by decreasing the step size as the GA's execution advances. Let us suppose that this operator is applied on a real-coded gene, $x \in [a, b]$ $(a, b \in \Re)$, at generation t, and that T is the maximum number of generations, then it generates a gene, x', as follows:

$$x' = \begin{cases} x + \Delta(t, b - x) & \text{if } \tau = 0, \\ x - \Delta(t, x - a) & \text{if } \tau = 1, \end{cases}$$

with τ being a random number that may have a value of zero or one, and

$$\Delta(t, y) = y\left(1 - r^{(1-(t/T))^b}\right),$$

where r is a random number from the interval $[0, 1]$ and b is a parameter chosen by the user. This function gives a value in the range $[0, y]$ such that the probability of returning a number close to zero increases as the algorithm advances. The size of the gene generation interval shall be smaller with the passing of generations. This property causes this operator to make an uniform search in the initial space when t is small, and very locally at a later stage, favoring local tuning.

The non-uniform mutation operator has been widely used, reporting good results [22–24]. It is considered to be one of the most suitable mutation operators for RCGAs [6].

2.1.2. Adaptive Step Size Control. The $(1 + 1)$-evolution strategy ((1+1)-ES) [8] is an evolutionary algorithm that uses adaptive step size control. It attempts to adapt its mutation step size to the problem according to the absolute adaptive heuristic: "*expand the step size when making progress*, *shrink it when stuck*". This heuristic will be denoted as *E/S heuristic*.

$(1 + 1)$-ES works using a continuous representation and a mutation operator based on normally distributed modifications with expectation zero and given variance, σ, as the step size. It operates on a vector of variables by applying mutation with identical σ to each variable, so generating a descendant. The better of ancestor and descendant is considered as the new starting point. $(1 + 1)$-ES applies the E/S heuristic for adapting σ by means of the *1/5 success rule*. This rule uses the results obtained by mutation in the last few generations: "*if more than one fifth of the mutation have been successful*, *the step size is increased*, *otherwise it is decreased*".

In [25], a dynamic hill climbing algorithm is presented, which uses the E/S heuristic as well. We would like to point out that the model proposed in this paper, TRAMSS, uses important ideas that are present in this algorithm.

2.1.3. Self-Adaptive Step Size Control. Hinterding et al. [26, 27] apply self-adaptive control of the mutation step sizes for optimizing numeric functions in a real valued GA. Their GA uses Gaussian mutation, which adds Gaussian noise to the gene to be mutated. Gaussian noise is obtained from a normally distributed random variable which has a mean of 0 and standard deviation of σ (step size), where σ is an extra gene added to the front of each chromosome. The values of this gene are allowed to vary from 0.000001 to 0.2, and participate in crossover and mutation. The following steps are followed for mutating genes in a chromosome:

1. Apply Gaussian noise to the σ value using a standard deviation of 0.013 (meta-mutation).
2. Use the σ value as the standard deviation for the Gaussian noise to mutate the other genes in the chromosome.
3. Write the mutated genes (including σ value) back to the chromosome.

During the initialization process, the value of σ for all chromosomes is set using a Gaussian distributed random variable with mean 0.1 and standard deviation 0.01.

2.2. Restart Operator

Premature convergence causes a drop in the GA's efficiency; the genetic operators do not produce the

feasible diversity to tackle new search space zones and thus the algorithm reiterates over the known zones producing a slowing-down in the search process. Under these circumstances, resources may be wasted by the GA searching an area not containing a solution of sufficient quality, where any possible improvement in the solution quality is not justified by the resources used. Therefore, resources would be better utilized in restarting the search in a new area, with a new population [28]. This is carried out by means of a *restart operator*. Next, we review some different approaches to this operator.

- In [29], it was suggested restarting GAs that have substantially converged, by reinitializing the population using both randomly generated individuals and the best individual from the converged population.
- In [30], upon convergence, the population is reinitialized by using the best individual found so far as a template for creating a new population. Each individual is created by flipping a fixed proportion (35%) of the bits of the template chosen at random without replacement. If several successive reinitializations fail to yield an improvement, the population is completely (100%) randomly reinitialized.
- In [28], a *selectively destructive restart* is proposed that does not completely destroy the converged population; a percentage of the converged genes will survive untouched to begin the next convergence stage. A probability of gene reinitialization, p_r, is used; the higher the rate, the more genes are initialized. Experiments carried out with some p_r values showed that different problems have different optimal reinitialization probabilities. This model seems to provide an improved method for renewing genetic diversity in GA search. Intuitively, the complete reinitialization of the population forgets the previous solutions, therefore it cannot make use of previously discovered building blocks.
- In [31], a similar mechanism, called *partial hypermutation* model, was introduced, which replaces, at each generation, a percentage of the population by randomly generated individuals. The percentage is called *replacement rate*. The intended effect is similar to the one of the previous approach: to maintain a continuous level of exploration of the search space, while trying to minimize disruption for the ongoing search.

Other important GA models based on the restart operator are *ARGOT* [32], *dynamic parameter encoding* [33] and *delta coding* [34].

3. Two-Loop RCGA Model with Adaptive Control of Step Sizes

In this section, we present TRAMSS. It uses:

- an instance of the absolute adaptive E/S heuristic, presented in Subsection 2.1.2, for the adaptive step size control of the mutation operator applied in the inner loop, and
- an instance of its opposite version, denoted here as S/E heuristic, for the adaptive step size control of the mutation operator used by the restart operator that is executed by the outer loop.

Next, in Subsection 3.1, we examine the application of the E/S and S/E heuristics for step size control in RCGAs, and, in Subsections 3.2 and 3.3, we present the TRAMSS inner and outer loops, respectively.

3.1. The E/S and S/E Heuristics

Let's suppose that an RCGA is applying a mutation operator with δ being its step size. If a stationary state is detected (the fitness of the best individual or the average fitness have not been improved during the previous generations), there are two possible causes concerning δ:

1. it is too high, which does not allow the convergence to be produced for obtaining better individuals, or
2. it is too low, which induces a premature convergence, with the search process being trapped in a local optimum.

On the one hand, if we decided to include an adaptive control of δ based on the instance of the E/S heuristic "*increase δ when making progress*, *decrease it when stuck*", a stationary state caused by (1) would be suitably tackled, since δ would become lower, so introducing more convergence. However, this heuristic would not be adequate if the stationary state is caused by (2), because it would complicate the problem even more.

Precisely, this last circumstance will occur as the number of iterations increases. Since the RCGA will find more difficulties for making progress, the natural trend of the instance of the E/S heuristic will be to lead δ to lower values, so producing more convergence. Some authors have claimed the possibility of this problem. For example, in [35], the following was stated about the 1/5 success rule:

"*... the* 1/5 *success rule may cause premature stagnation of the search due to the deterministic decrease of the step size whenever the topological situation does not lead to a sufficiently large success rate*".

For complex problems, this effect will probably become a premature convergence. This explains the following claim, again about the 1/5 success rule [13]:

"*... this heuristic is especially useful in smooth multimodal environments of the type well studied by the evolution strategies community but would be less applicable in discontinuous or extremely rough environments*".

On the other hand, if we are inclined to use the instance of the S/E heuristic "*decrease* δ *when progress is made, increase it when there are no improvements*", a stationary state produced by (2) will be adequately attacked, since δ would be greater and so, more diversity is introduced with the possibility of escaping from the local optimum. However, an important problem may occur: as no improvements are made by the RCGA, higher δ values are tried, so introducing too much diversity and not considering the possibility that convergence may be suffice for improving results.

So, all these facts show that serious problems may arise when the E/S and S/E heuristics are applied separately. However, we think that a mechanism applying both of these heuristics would handle the population diversity suitably to avoid the premature convergence problem and improve the behavior of the search process.

The adaptive RCGA model proposed, TRAMSS, includes this idea: it uses the E/S heuristic for adapting the step size of a mutation operator applied in the inner loop and the S/E heuristic for adapting the step size of a mutation operator used by a restart operator performed in the outer loop.

3.2. TRAMSS Inner Loop

The inner loop performs the usual process (selection, crossover, and mutation) over a number of generations, G, called *time-interval between observations*. Then, depending on the progress of the population mean fitness found throughout these generations, it adjusts the step size of the mutation operator, and calculates a new value for G. Next, we fully describe these steps where a minimization problem is assumed.

Selection, Crossover, and Mutation (Step 2.2). Over the time-interval between observations, G, the following selection mechanism, and crossover and mutation operators are applied.

- The selection probability calculation follows *linear ranking* [36], with $\eta_{\min} = 0.25$, and the sampling algorithm is the *stochastic universal sampling* [37].

  The *elitist strategy* [38] is considered as well. It involves making sure that the best performing chromosome always survives intact from one generation to the next. This is necessary since it is possible that the best chromosome disappears, due to crossover or mutation.
- We have tried different crossover operators, which are described in Appendix A.
- The mutation operator used is denoted as *Mutation*(δ), where δ is the step size ($0 \leq \delta \leq 1$). This operator is defined as follows: If $x \in [a, b]$ is a gene to be mutated, then the gene resulting from the application of this operator, x', will be a random (uniform) number chosen from $[x - \delta \cdot (x - a), x + \delta \cdot (b - x)]$. Clearly, the higher δ is, the greater changes on x are produced.

Adaptive Control of δ (Step 2.3). After G generations, the δ parameter used by the mutation operator is adapted following a particular instance of the E/S heuristic: "*increase* δ *when observing progress on* $\bar{f}$ (*population mean fitness*), *decrease it when stuck*". δ is kept in the interval $[\delta_{\min}, \Delta]$, where Δ is a parameter calculated by the outer loop, as described in Subsection 3.3, and $\delta_{\min}$ ($\delta \geq \delta_{\min}$) is the minimum threshold defined by the user (in experiments we assume a value of 1.0e−100).

The inner loop ends when δ reaches the $\delta_{\min}$ value. By finishing with a fine grained search with small step sizes, we are sure that a local optimum, or the global one, will be located precisely [25]. It will stop as well, when a maximum number of generations is reached.

The update rates for δ depend on the number of previous successive observations that were successful or not successful. Two variables, *yes* and *no*, are used for recording these occurrences, respectively. If progress is made during many successive previous observations ($\bar{f}_{Old} \geq \bar{f}$, $\bar{f}_{Old}$ being the population mean fitness of the previous iteration), then the increasing rate for δ is very high (in particular, δ is multiplied by 2^{yes}), whereas if these observations were not successful, then the decreasing rate is high (δ is divided by 2^{no}). In this way,

TRAMSS INNER LOOP

1. $\delta := \Delta$; $G := G_0$; *yes* $:= 0$; *no* $:= 0$;

2. while $(\delta \geq \delta_{min})$ **and** (**not** termination-condition) **do**

2.1. $\bar{f}_{Old} := \bar{f}$;

2.2. perform over G generations:

2.2.1 *Selection, Crossover, and Mutation*(δ);

2.2.2 *Evaluation*;

2.2.3 *Make sure that the population has the best-so-far chromosome*;

2.3. if $(\bar{f}_{Old} \geq \bar{f})$ **then**

yes:=yes+1; no:=0;

$\delta := \delta \cdot 2^{yes}$; **if** $(\delta > \Delta)$ **then** $\delta := \Delta$;

else

no:=no+1; yes:=0;

$\delta := \delta / 2^{no}$;

2.4. $G := G_0 \cdot \delta / \Delta$; **if** $(G < G_{min})$ **then** $G := G_{min}$;

Figure 1. TRAMSS inner loop structure.

when the search process is located in a local optimum and improvements are still not surely expected by reducing δ, the inner loop duration will not be too long.

Time-Interval Calculation (Step 2.4). The time-interval between observations, G, is calculated depending on the current values of δ with regard to Δ. If δ is similar to Δ, then the time-interval is high ($G_0 = 100$ in the experiments), and if it is lower, the time-interval will become like $G_{\min}$ ($G_{\min} = 5$ in the experiments). This allows δ values similar to Δ to be used for a long time (Δ is considered a good starting point for δ, because it is adapted in the outer loop on the basis thereof, as we will explain in Subsection 3.3). Furthermore, we need to point out that the initial Δ, Δ_0, was assigned to 1 in the experiments, in order to favor exploration during the initial stages of the first inner loop's run.

Figure 1 shows the pseudocode algorithm for the whole TRAMSS inner loop. In short, the objective of this loop is to find and refine local optima (or the global one), in an efficient way.

3.3. TRAMSS Outer Loop

The outer loop randomly initializes the population that will be handled throughout the TRAMSS run. It fires the inner loop, and when this one returns, it applies a restart operator based on a step size that is adaptively controlled throughout its execution. Now, we explain, in depth, the main issues related to this loop.

Restart Operator Application (Step 3.4). The outer loop applies a restart operator, called *Restart*(Δ) $(0 \leq \Delta \leq 1)$ that applies *Mutation*(Δ) to all the genes in the chromosomes stored in the population. The objective of this operator is similar to one of the partial restart operators for binary-coded GAs, described in Section 2.2, i.e., to maintain a continuous level of exploration of the search space, while trying to use the promising zones located as a kind of sketch. It attempts to ensure that new and promising genetic material is available in the population for being handled and treated by the next inner loop.

Adaptive Control of Δ (Step 3.3). The outer loop adapts the Δ parameter, using information obtained

TRAMSS OUTER LOOP

1. $\Delta := \Delta_0$;

2. run *Initialize*;

3. while (**not** Termination-condition) **do**

3.1. $f_{OldBest} := f_{Best}$;

3.2. run *Inner Loop*;

3.3. if $(f_{OldBest} \geq f_{Best})$ **then**

$\Delta := \Delta/2$; **if** $(\Delta < \delta_{min})$ **then** $\Delta := \delta_{min}$;

else

$\Delta := \Delta \cdot 2$; **if** $(\Delta > 1)$ **then** $\Delta := 1$;

3.4. run *Restart*(Δ);

Figure 2. TRAMSS outer loop structure.

after each inner loop run, by means of an instance of the S/E heuristic: "*decrease* Δ *when observing progress on* f_{Best} (*fitness of the best element found so far*), *otherwise increase it*". This is implemented by dividing the previous Δ value by 2 or multiplying it by 2, respectively. The new Δ value will be the first value for the δ parameter used in the next inner loop.

The pseudocode algorithm for the outer loop is depicted in Fig. 2. To sum up, the outer loop attempts to introduce adequate diversity levels for allowing the subsequent inner loop processing to be capable of finding better local optima, or the global one, every time. For this reason, it uses the f_{Best} for the adaptive step size control. When no better local optima are found after the last inner loop runs, the outer loop produces more diversity in order to increase the probability of having access to a better one, which will be refined by the next inner loop. On the other hand, if better solutions are being found by previous inner loop's runs, Δ becomes low, so avoiding, for the moment, great destructive effects of the restart operator.

4. Experiments

Minimization experiments on the test suite, described in Subsection 4.1, were carried out in order to study the behavior of the TRAMSS model. In Subsection 4.2, we describe the algorithms built in order to do this, and, in Subsection 4.3, we show the results and discuss some conclusions about them.

4.1. Test Suite

For the experiments, we have considered six test functions used in the GA literature: *Sphere* model (f_{Sph}) [18, 38], *Generalized Rosenbrock's* function (f_{Ros}) [38], *Schwefel's Problem* 1.2 (f_{Sch}) [18], *Generalized Rastrigin's* function (f_{Ras}) [40, 41], *Griewangk's* function [39], and *Expansion of* f_{10} (ef_{10}) [42]. Figure 3 shows their formulation. The dimension of the search space is 25.

- f_{Sph} is a continuous, strictly convex, and unimodal function.
- f_{Ros} is a continuous and unimodal function, with the optimum located in a steep parabolic valley with a flat bottom. This feature will probably cause slow progress in many algorithms since they must permanently change their search direction to reach the optimum. This function has been considered by some authors to be a real challenge for any continuous function optimization program [43]. A great part of its difficulty lies in the fact that there are nonlinear interactions between the variables, i.e., it is *nonseparable* [44].

f_{Sph}

$$f_{Sph}(\vec{x}) = \sum_{i=1}^{n} x_i^2$$
$$-5.12 \leq x_i \leq 5.12$$
$$f_{Sph}(x^*) = 0$$

f_{Ros}

$$f_{Ros}(\vec{x}) = \sum_{i=1}^{n-1} (100 \cdot (x_{i+1} - x_i^2)^2 + (x_i - 1)^2)$$
$$-5.12 \leq x_i \leq 5.12$$
$$f_{Ros}(x^*) = 0$$

f_{Sch}

$$f_{Sch}(\vec{x}) = \sum_{i=1}^{n} \left(\sum_{j=1}^{i} x_j \right)^2$$
$$-65.536 \leq x_i \leq 65.536$$
$$f_{Sch}(x^*) = 0$$

f_{Gri}

$$f_{Gri}(\vec{x}) = \frac{1}{d} \sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$$
$$d = 4000$$
$$-600.0 \leq x_i \leq 600.0$$
$$f_{Gri}(x^*) = 0$$

f_{Ras}

$$f_{Ras}(\vec{x}) = a \cdot n + \sum_{i=1}^{n} x_i^2 - a \cdot \cos(\omega \cdot x_i)$$
$$a = 10, \omega = 2\pi$$
$$-5.12 \leq x_i \leq 5.12$$
$$f_{Ras}(x^*) = 0$$

ef_{10}

$$ef_{10}(\vec{x}) = f_{10}(x_1, x_2) + \ldots + f_{10}(x_{n-1}, x_n) + f_{10}(x_n, x_1)$$
$$f_{10}(x, y) = (x^2 + y^2)^{0.25} \cdot [\sin^2(50 \cdot (x^2 + y^2)^{0.1}) + 1]$$
$$x, y \in (-100, 100]$$
$$ef_{10}(x^*) = 0$$

Figure 3. Test functions.

f_{Sch} is a continuous and unimodal function. Its difficulty lies in the fact that searching along the coordinate axes only gives a poor rate of convergence, since the gradient of f_{Sch} is not oriented along the axes. It presents similar difficulties to f_{Ros}, but its valley is much narrower.

f_{Ras} is a scalable, continuous, separable, and multimodal function which is produced from f_{Sph} by modulating it with $a \cdot \cos(\omega \cdot x_i)$.

f_{Gri} is a continuous and multimodal function. This function is difficult to optimize because it is nonseparable [45] and the search algorithm has to climb a hill to reach the next valley. Nevertheless, one undesirable property exhibited is that it becomes easier as the dimensionality is increased [44].

f_{10} is a function that has nonlinear interactions between two variables. Its expanded version, **ef_{10}**, is built in such a way that it induces nonlinear interaction across multiple variables. It is nonseparable as well.

4.2. *Algorithms*

We have built five different TRAMSS versions that apply the following crossover operators: linear [46], discrete [47], BLX-α [48], and fuzzy recombination [49] (they are described in Appendix A). The TRAMSS versions are called TRA-LIN, TRA-DIS, TRA-BLX, and TRA-FR, respectively. They are compared with RCGAs that use the same crossover operators and different mutation operators:

- *Random mutation* [5]. If $x \in [a, b]$ is a gene to be mutated, then the gene resulting from the application of this operator, x', is a random (uniform) number from the domain $[a, b]$. The RCGAs based on this operator will be denoted as R-RAN-LIN, R-RAN-DIS, R-RAN-BLX, and R-RAN-FR.
- *BGA mutation* [47]. $x' = x \pm rang \cdot \gamma$, where *rang* defines the mutation range and it is normally set to $0.1 \cdot (b - a)$. The $+$ or $-$ sign is chosen with a probability of 0.5 and $\gamma = \sum_{k=0}^{15} \alpha_k \cdot 2^{-k}$, where $\alpha_k \in \{0, 1\}$ is randomly generated with $p(\alpha_i = 1) = 1/16$.

  This operator returns values in the interval $[x - rang, x + rang]$, with the probability of generating a neighborhood of x being very high. The minimum possible proximity is produced with a precision of $rang \cdot 2^{-15}$. The RCGAs with this operator will be called R-BGA-LIN, R-BGA-DIS, R-BGA-BLX, and R-BGA-FR.

  Random mutation and the BGA mutation are two representatives of non-adaptive mutation operators for RCGAs.

- *Non-uniform mutation.* In this operator the step size is controlled using a deterministic rule (Subsection 2.1.1). We have implemented five RCGAs with it: R-NU-LIN, R-NU-DIS, R-NU-BLX, and R-NU-FR.
- *Self-adaptive mutation.* We have built five RCGAs with the mutation operator described in Subsection 2.1.3, which follows the idea of self-adaptive control of step sizes. These algorithms are R-SELF-LIN, R-SELF-DIS, R-SELF-BLX, and R-SELF-FR.

The crossover probability used by the algorithms is 0.6, the mutation probability 0.005, and the population size 60 chromosomes. The selection probability calculation follows linear ranking ($\eta_{\min} = 0.25$) and the sampling algorithm is the stochastic universal sampling (as the inner loop of the TRAMSS versions). The algorithms were executed 30 times, each one with 10,000 generations, except the ones based on linear crossover, which performed 6,666 generations (each application of this crossover operator needs three evaluations). In this way, the number of fitness function evaluations required by all algorithms are similar.

4.3. Results

Tables 1–4 show the results obtained for each test function. The performance measures used are:

- **A** performance: average of the best fitness function found at the end of each run.
- **SD** performance: standard deviation.

Moreover, a *t-test* (at 0.05 level of significance) was applied in order to ascertain if differences in the *A* performance for the TRAMSS versions are significant when compared against one for the other RCGAs in the respective table. The direction of any significant differences is denoted either by:

- a plus sign $(+)$ for an improvement in the *A* performance for the TRAMSS versions with regards to the other RCGAs, or
- a minus sign $(-)$ for a reduction, or
- an approximate sign $(\sim)$ for non significant differences.

The places in these tables where these signs do not appear correspond with the performance values for TRAMSS versions.

Next, we analyze these results following three ways. First, we compare the behavior of the TRAMSS versions with the one achieved using the mutation operators for RCGAs (Subsection 4.3.1). Then, we ascertain whether the TRAMSS versions achieve a *robust* operation, in the sense that they obtain a significant performance for each one of the test functions (Subsection 4.3.2). Finally, we attempt to determine the type of crossover operator that allow TRAMSS to obtain the best results (Subsection 4.3.3).

4.3.1. TRAMSS and Mutation Operators. For each mutation operator (random, BGA, non-uniform, and self-adaptive), Table 5 shows the percentages for improvements, reductions, and non differences (according to the *t*-test results in Tables 1–4) in the *A* performance for the TRAMSS versions with regards to their corresponding RCGAs based on this mutation operator, on all test functions, and independently from the crossover operator used. The table is useful for comparing the results of TRAMSS against the ones obtained when using these mutation operators.

We may observe in the table the remarkable percentages for improvements and the very low percentages for reductions. Furthermore, the percentages for non differences are significant as well, however, they are lower than the ones for improvements. These results indicate that TRAMSS is the most effective model for controlling mutation step sizes as compared with the mutation operators considered for these experiments.

On the other hand, we need to point out that the non-uniform mutation has been the mutation operator that offered the most significant resistance against the results for the TRAMSS model, in fact, this operator is considered to be one of the most profitable mutation operators for RCGAs (such as it is claimed in 2.1.1).

4.3.2. TRAMSS and Test Functions. For each test function, Table 6 shows the percentages for improvements, reductions, and non differences (according to the *t*-test results in Tables 1–4) in the *A* performance for the TRAMSS versions with regards to the other RCGAs on such function, independently from the crossover operator and mutation operator used. This table may be useful for ascertaining whether the TRAMSS versions achieve a *robust* operation, in the sense that they obtain a significant performance for each one of the test functions, which have different features (modality, complexity, type of interactions between variables, etc.), such as mentioned in Subsection 4.1.

Table 1. Results for algorithms based on linear crossover.

	f_{Sph}		f_{Ros}		f_{Sch}		f_{Ras}		f_{Gri}		ef_{10}	
Algorithms	**A**	**SD**	**A**	**SD**	**A**	**SD**	**A**	**SD**	**A**	**SD**	**A**	**SD**
R-RAN-LIN	6.0e−13 (+)	2.0e−12	2.1e1 (∼)	4.0e−1	5.1e−1 (+)	3.8e−1	3.2e−10 (−)	4.8e−10	2.0e−2 (+)	2.0e−2	2.2e−2 (∼)	1.6e−2
R-BGA-LIN	5.9e−14 (+)	9.8e−14	1.9e1 (+)	1.0e0	6.6e−2 (+)	6.9e−2	9.3e−1 (∼)	2.0e0	2.3e−2 (+)	2.5e−2	1.5e−2 (∼)	1.3e−2
R-NU-LIN	1.9e−20 (+)	4.3e−20	1.9e1 (+)	8.3e−1	4.2e−3 (+)	4.3e−3	0.0e0 (−)	0.0e0	1.9e−2 (+)	2.1e−2	5.6e−5 (∼)	3.6e−5
R-SELF-LIN	4.5e−16 (+)	1.1e−15	1.9e1 (+)	1.7e0	8.5e−4 (+)	1.3e−3	1.2e1 (+)	4.1e0	1.4e−2 (+)	1.3e−2	7.1e1 (+)	7.7e0
TRA-LIN	3.6e−72	2.0e−71	1.8e1	5.5e−1	2.9e−5	7.8e−5	7.6e−1	1.2e0	6.1e−3	1.1e−2	3.7e−2	2.0e−1

Table 2. Results for algorithms based on discrete crossover.

	f_{Sph}		f_{Ros}		f_{Sch}		f_{Ras}		f_{Gri}		ef_{10}	
Algorithms	**A**	**SD**	**A**	**SD**	**A**	**SD**	**A**	**SD**	**A**	**SD**	**A**	**SD**
R-RAN-DIS	2.4e−4 (+)	9.8e−5	5.2e1 (+)	3.5e1	7.4e2 (+)	3.1e2	4.7e−2 (+)	2.0e−2	1.5e−1 (+)	4.6e−2	1.1e1 (+)	1.5e0
R-BGA-DIS	2.0e−9 (+)	3.8e−10	2.3e1 (∼)	2.7e1	6.2e1 (+)	2.6e1	4.0e−7 (∼)	7.9e−8	3.0e−2 (+)	2.6e−2	1.7e0 (∼)	5.8e0
R-NU-DIS	2.4e−17 (+)	2.3e−17	1.9e1 (∼)	2.5e1	1.6e1 (+)	1.1e1	3.5e−14 (∼)	2.8e−14	1.9e−2 (∼)	2.4e−2	1.9e−2 (∼)	2.0e−2
R-SELF-DIS	3.5e−10 (+)	2.8e−10	3.0e1 (∼)	2.3e1	8.1e1 (+)	5.9e1	5.8e1 (+)	1.9e1	3.3e1 (+)	1.4e1	1.3e2 (+)	1.2e1
TRA-DIS	4.8e−78	1.8e−77	1.7e1	2.6e1	3.0e−1	2.7e−1	2.3e−14	2.4e−14	1.5e−2	1.6e−2	3.1e−2	1.6e−1

Table 3. Results for algorithms based on BLX-α crossover.

	f_{Sph}		f_{Ros}		f_{Sch}		f_{Ras}		f_{Gri}		ef_{10}	
Algorithms	**A**	**SD**	**A**	**SD**	**A**	**SD**	**A**	**SD**	**A**	**SD**	**A**	**SD**
R-RAN-BLX	7.7e−24 (+)	2.7e−23	2.0e1 (+)	1.5e1	5.1e−5 (+)	6.3e−5	0.0e0 (−)	0.0e0	8.1e−3 (+)	1.5e−2	2.2e−5 (∼)	1.6e−5
R-BGA-BLX	6.6e−23 (+)	3.5e−22	1.7e1 (∼)	9.7e0	1.7e−6 (+)	1.5e−6	5.4e−1 (−)	1.4e0	4.3e−3 (+)	6.9e−3	6.5e−7 (∼)	3.5e−6
R-NU-BLX	2.0e−31 (+)	5.8e−31	1.8e1 (+)	1.4e1	3.6e−7 (+)	4.2e−7	2.3e−1 (−)	9.5e−1	4.7e−3 (+)	8.4e−3	1.7e−7 (∼)	9.9e−8
R-SELF-BLX	1.4e−26 (+)	4.7e−26	1.6e1 (∼)	1.0e1	7.4e−8 (+)	5.7e−8	1.8e1 (∼)	4.8e0	2.9e−2 (∼)	1.1e−1	9.5e−1 (+)	2.2e0
TRA-BLX	1.4e−200	0.0e0	1.2e1	4.8e0	8.8e−12	1.5e−11	3.3e0	2.2e0	0.0e0	0.0e0	1.8e−4	9.8e−4

Table 4. Results for algorithms based of fuzzy recombination.

	f_{Sph}		f_{Ros}		f_{Sch}		f_{Ras}		f_{Gri}		ef_{10}	
Algorithms	**A**	**SD**	**A**	**SD**	**A**	**SD**	**A**	**SD**	**A**	**SD**	**A**	**SD**
R-RAN-FR	1.3e−26 (+)	2.4e−26	1.8e1 (∼)	1.1e1	1.4e0 (+)	7.7e−1	0.0e0 (∼)	0.0e0	1.2e−2 (+)	1.7e−2	1.3e−6 (+)	1.5e−6
R-BGA-FR	9.8e−19 (+)	4.1e−18	2.0e1 (∼)	1.8e1	2.6e−1 (+)	1.4e−1	0.0e0 (∼)	0.0e0	2.1e−2 (+)	2.3e−2	1.4e−4 (+)	1.9e−4
R-NU-FR	1.6e−32 (+)	3.7e−32	2.3e1 (∼)	2.2e1	1.1e−2 (+)	1.1e−2	0.0e0 (∼)	0.0e0	1.9e−2 (+)	2.2e−2	2.4e−8 (+)	1.9e−8
R-SELF-FR	2.4e−30 (+)	8.0e−30	1.9e1 (∼)	1.8e1	6.7e−4 (+)	7.1e−4	2.3e1 (+)	5.8e0	3.2e−1 (+)	4.9e−1	6.1e0 (+)	5.1e0
TRA-FR	5.5e−189	0.0e0	1.6e1	1.5e1	8.9e−6	1.1e−5	6.6e−2	3.6e−1	9.1e−4	3.4e−3	9.6e−23	5.2e−22

Table 5. Comparison between TRAMSS and mutation operators.

Mutation	% Impr. (+)	% Red. (−)	% Non diff. (∼)
Random	70.83	8.33	20.83
BGA	58.33	4.16	37.50
Non-Uniform	58.33	8.33	33.33
Self-Adaptive	79.16	0	20.83

Table 6. Behavior of TRAMSS on each test function.

Function	% Impr. (+)	% Red. (−)	% Non diff. (∼)
f_{Sph}	100	0	0
f_{Ros}	37.5	0	62.5
f_{Sch}	100	0	0
f_{Ras}	25	31.25	43.75
f_{Gri}	87.5	0	12.5
ef_{10}	50	0	50

This table indicates that the TRAMSS versions clearly improve the A performance of all the other algorithms on f_{Sph}, f_{Sch}, and f_{Gri}. For f_{Ros} and ef_{10} the results are similar or superior. f_{Ras} has been the unique function where the TRAMSS versions have shown a sign of performance degradation (31.25% of reduction).

Therefore, we may remark that a high degree of *robustness* is obtained with the TRAMSS model on the test functions considered. In order to explain this suitable behavior, we should focus on the study of the adaptation ability that it provides, which is tackled in Subsection 4.4.

4.3.3. TRAMSS and Crossover Operators. For each crossover operator (linear, discrete, BLX-α, and fuzzy recombination), Table 7 summarizes the percentages for improvements, reductions, and non differences (according to the t-test results in Tables 1–4) in the A performance for the TRAMSS versions with regards to the corresponding RCGAs based on this crossover operator, on all test functions, and independently from the mutation operator used. This table is useful for detecting the crossover operator that couples suitably with the TRAMSS model, obtaining the best results.

Table 7. Behavior of TRAMSS when using different crossover operators.

Crossover	% Impr. (+)	% Red. (−)	% Non diff. (∼)
Linear	70.83	8.33	20.83
Discrete	66.66	0	33.33
BLX-α	58.33	12.5	29.16
Fuzzy Rec.	70.83	0	29.16

In this table, we may see that the improvements of the TRAMSS implementations with regards to the other RCGAs are very notable when using fuzzy recombination, furthermore, no reductions are achieved with this crossover operator. None of the remaining crossover operators allows a better operation to be obtained. Therefore, we conclude that the performance of the TRAMSS model is enhanced using fuzzy recombination. Next, we attempt to explain why. The adaptive step size control performed in the inner loop depends on the changes produced on $\bar{f}$, which are determined by the joint effects of the selection process, and the mutation and crossover operators. Let us consider only the interactions between the last ones. The mutation operator generates diversity and the crossover operator would have to use it for creating better individuals. If the crossover operator achieves this task, then the mutation operator would be generating *useful diversity*, and so evolution is introduced. Therefore, only if the mutation and the crossover operators are being suitably coupled, the success of the inner loop may be accomplished. Results have shown that in the case of using fuzzy recombination, this circumstance is held. Moreover, the associated property of this crossover operator (to fit their action range depending on the diversity of the population) is one of the main responsible aspects for making this union so profitable. It would allow this operator to exploit the diversity generated by the mutation operator.

4.4. Study of the Adaptation in TRAMSS

There are, at least, two ways to study the operation of an adaptive mechanism for GAs [50]. The first is from the point of view of performance (test functions are commonly used to evaluate performance improvement). The second view is quite different in that it ignores performance and concentrates more on the adaptive mechanism itself, i.e., its ability to adjust the GA configuration according to the particularities of the problem to be solved. Once given these two points of view, it is natural to investigate the way in which adaptive behavior is responsible for the performance improvement.

In Subsection 4.3, we studied the first point of view. In this subsection, we consider the point of view of the

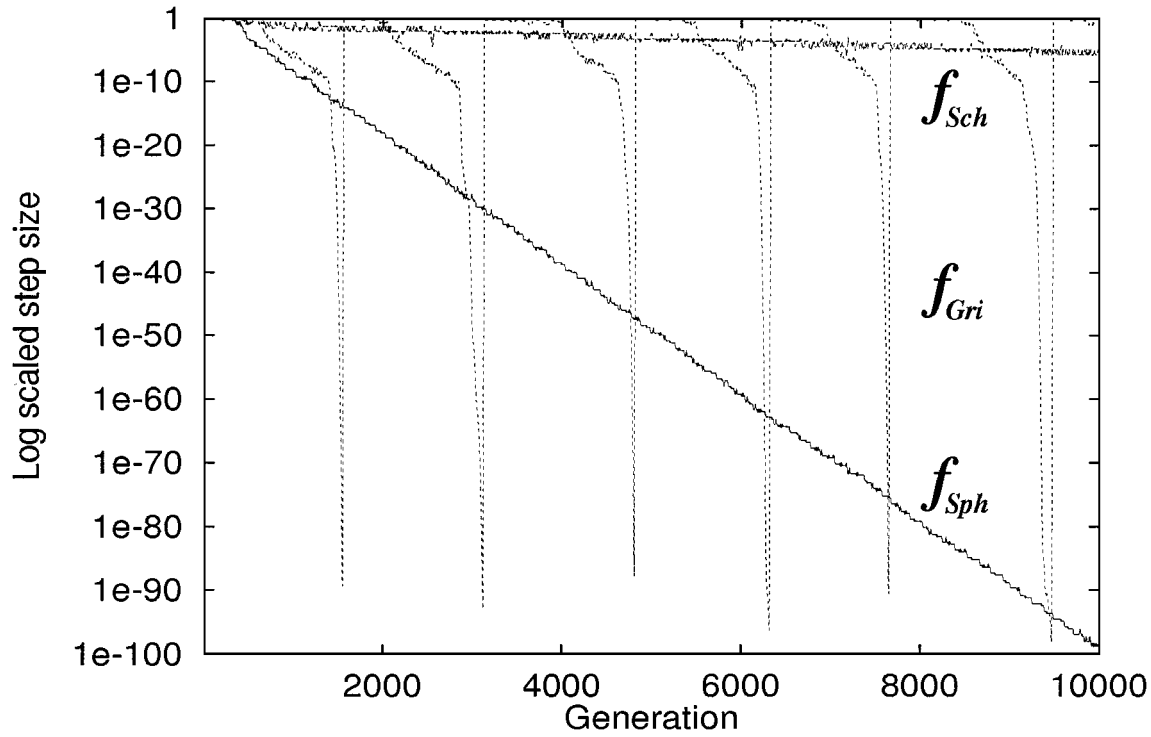

Figure 4. Step sizes used by TRA-FR for f_{Sph}, f_{Sch}, f_{Gri}.

adaptation itself. In particular, we are interested in ascertaining whether TRAMSS adjusts step sizes according to the particularities of the problem to be solved, and if this adaptation ability is responsible for the performance improvement observed in Subsection 4.3. Figure 4 was introduced in order to do this. It outlines the log scaled step size values used by TRA-FR during a run on three test functions, f_{Sph}, f_{Sch}, and f_{Gri}, which have different features. f_{Sph} and f_{Sch} are unimodal functions, with f_{Sch} being more complex than f_{Sph}, whereas f_{Gri} is a complex multimodal function (Subsection 4.1). We may observe that there are notable differences in the way in which TRA-FR controls the step size for each one of these functions:

- For f_{Sph}, TRA-FR decreases continually the step size from 1 to 1e−100 to properly suit the local nature of the landscape in this function. The effects induced on performance may be seen in Fig. 5. It shows the log scaled best fitness value, f_{Best}, for each generation in the run of TRA-FR on this function. We observe that improvements on f_{Best} predominate during the whole run, obtaining very accurate final results. These results suggest that TRA-FR has been generating useful diversity throughout the run for this function.

  We have included in Fig. 5 the log scaled f_{Best} for the RCGA based on non-uniform mutation (R-NU-FR), which has arisen as the most suitable reference point for comparing the results of the TRAMSS versions (Subsection 4.3.1). In this case, we notice that f_{Best} advances through descending steps (such as the ones near generations 4000, 4500, and 6000). Non-uniform mutation does not take into account whether the diversity being generated is useful or not. It only decreases the step size depending on the time without observing if improvements are made, or not. This fact causes f_{Best} to be stagnated during long time intervals, up until a suitable step size value is reached, which will introduce a new descending step.
- For f_{Sch}, TRA-FR has controlled the step size following a different way (Fig. 4). Now, the step sizes used are between 1 and 1e−5. The optimum of the function is located at the end of a very steep and curved valley. In order to reach the optimum, a GA must permanently change their search direction. TRA-FR has attempted to achieve this behavior by considering such high step size values. In this

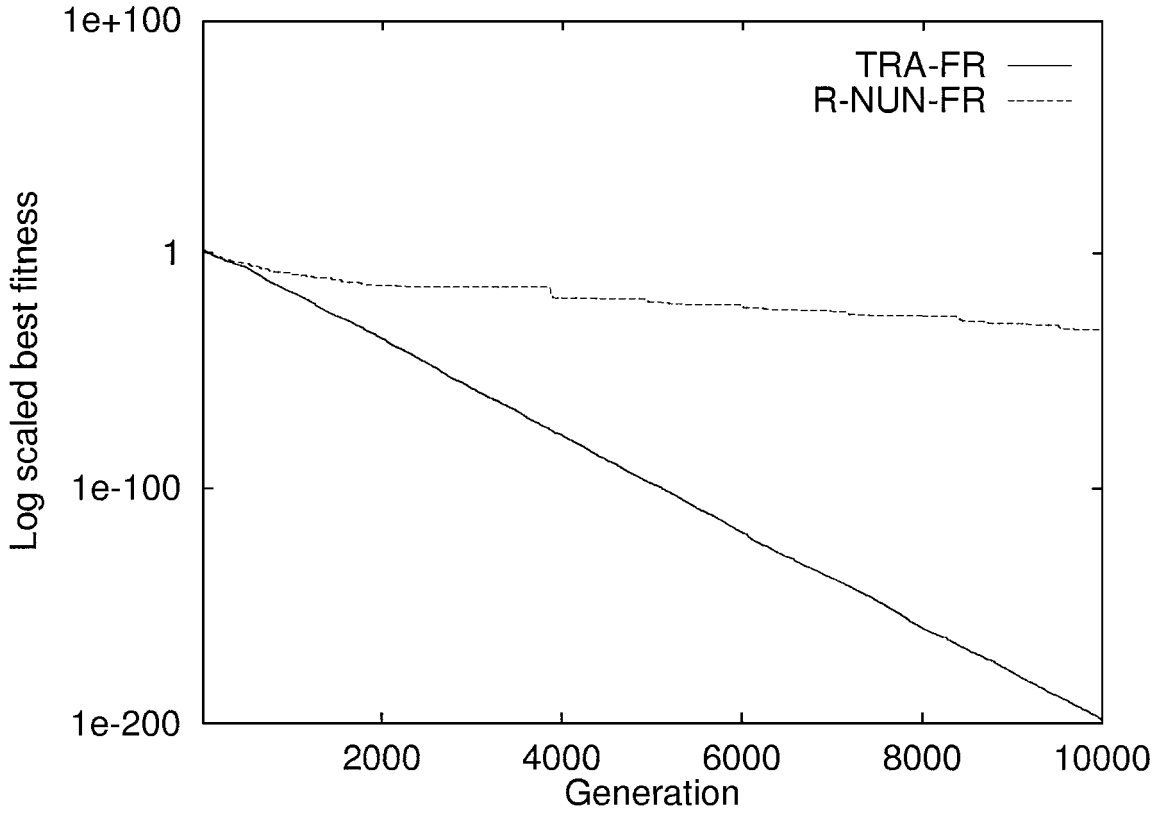

Figure 5. Evolution of TRA-FR and R-NU-FR for f_{Sph}.

way, it might produce improvements on f_{Best} continually, such as is shown in Fig. 6. This suitable progress may be compared with the one made by R-NU-FR. Until approximately generation 1200, this algorithm achieves results that are similar to the ones of TRA-FR, which is due to the high step size values used throughout these generations. However, the deterministic step size descent produced during the following generations induced a drop in performance.

Another interesting conclusion from Fig. 4 is that TRA-FR performed its inner loop only once when dealing with f_{Sph} and f_{Sch}, which seems reasonable since they are unimodal. In fact, this has been observed during the runs of the TRAMSS instances on all unimodal functions. This did not occur with the multimodal functions, as we will see for the case of f_{Gri}.

From these results we may say that the implementation of the E/S heuristic used for controlling step sizes is highly suitable for dealing with unimodal functions. In fact, we should point out that this heuristic has already been used for designing efficient local search procedures [25].

- The plot for f_{Gri} (Fig. 4) shows many peaks, while the ones for f_{Sph} and f_{Sch} none. Each peak corresponds to the location and refinement of a local optimum (or the global one) by the inner loop. After this is accomplished, the outer loop fires the restart operator and sets the step size to an initial value. The influence of this step size control on performance may be seen in Fig. 7. The peaks in the plot for TRA-FR are produced by the application of the restart operator, and each one is associated with a peak in Fig. 4. The most interesting remark is that TRA-FR might find the global optimum after the third restart operator call, produced at generation 4800 (and after the following one as well). On the other hand, we may observe the low performance of R-NU-FR on this function. The descent of the step size performed by the non-uniform mutation does not allow the search direction to escape from a possible stagnation in a local optimum, when working on multimodal functions.

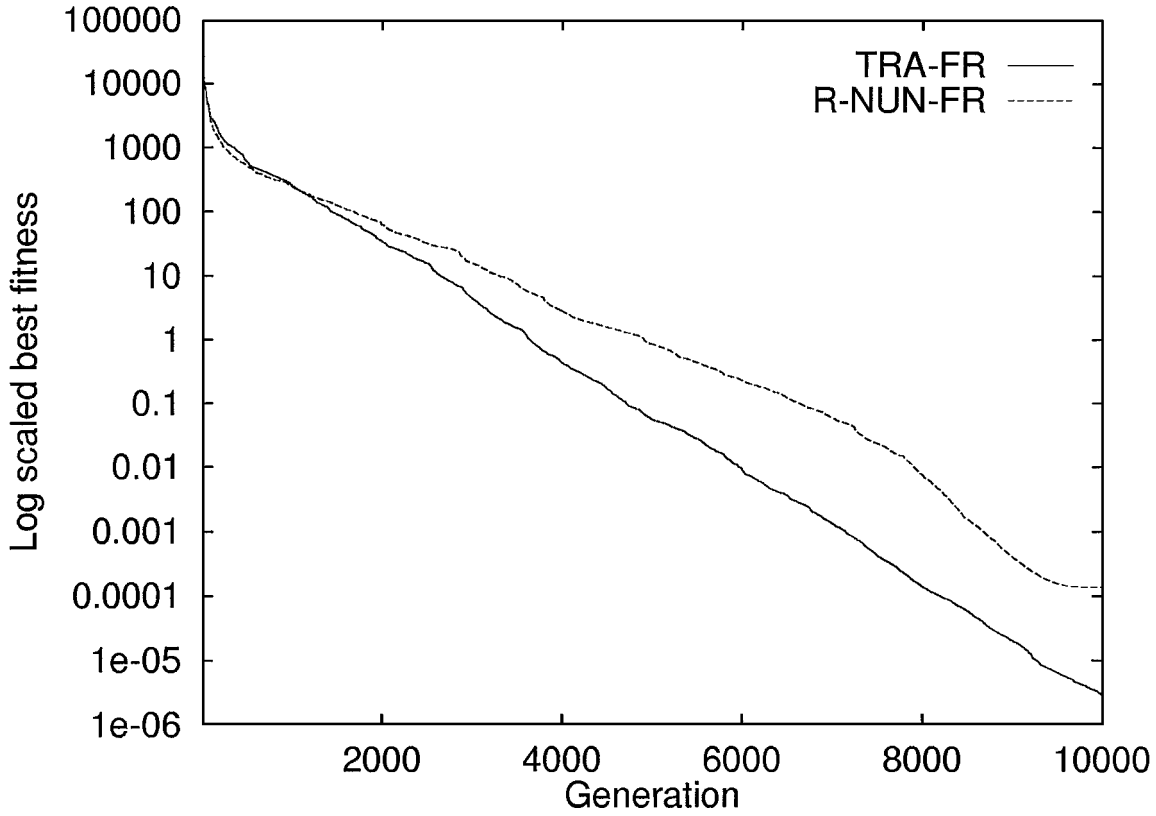

Figure 6. Evolution of TRA-FR and R-NU-FR for f_{Sch}.

Therefore, we conclude that the participation of the restart operator in the outer loop allowed reliability to be improved on the multimodal functions, with regard to the RCGAS based on non-uniform mutation.

To sum up, this study shows that the adaptation ability of the TRAMSS model allows the population diversity to be controlled according to the particularities of the search space, allowing significant performance to be achieved for problems with different difficulties.

5. Conclusions

This paper presented TRAMSS, a two-loop RCGA model that adjusts the step size of a mutation operator applied during the inner loop, for producing an efficient local tuning, and controls the step size of a mutation operator used by the outer loop, for reinitializing the population in order to ensure that different promising search zones are focused by the inner loop throughout the run. An instance of the E/S heuristic was used for implementing the adaptive mechanism in the inner loop whereas an instance of its opposite, the S/E heuristic, was considered for the outer loop.

Four TRAMSS algorithms were built using four crossover operators for RCGAs, linear, discrete, BLX-α, and fuzzy recombination, which represent different ways in which randomness may be used for generating real-coded genes.

The principal conclusions derived from the results of experiments carried out are the following:

- The TRAMSS model allows the control of useful population diversity to be accomplished (thanks to its adaptation ability) for improving accuracy in the case of unimodal functions, and, both reliability and accuracy for the multimodal ones, with regard to RCGAs based on other mutation operators presented in the GA literature.
- The adaptive control of step size performed by TRAMSS couples suitably with fuzzy recombination. Its interactions allow TRAMSS to manage useful diversity, so inducing an effective behavior on all test functions.

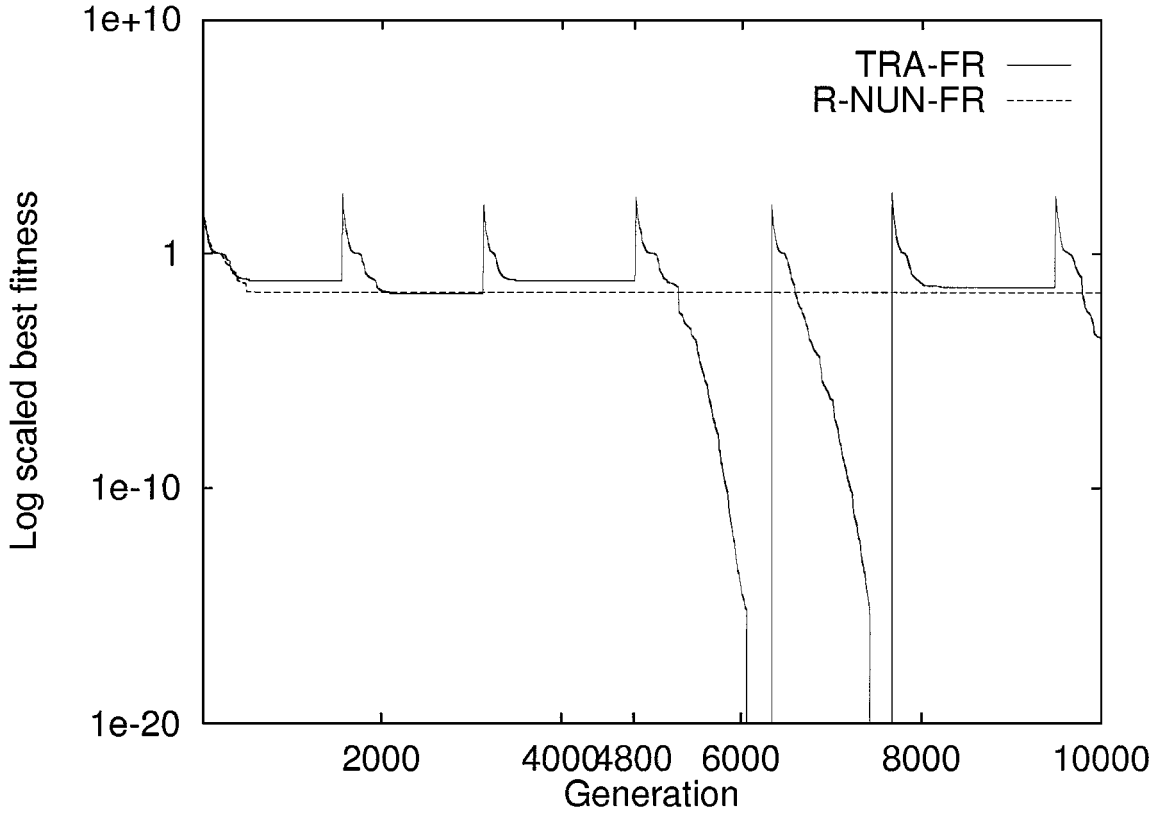

Figure 7. Evolution of TRA-FR and R-NU-FR for f_{Gri}.

Finally, we should point out that TRAMSS extensions may be followed in three ways: 1) control the parameter associated with the fuzzy recombination in order to exploit, even more, the profitable combination between TRAMSS and this crossover operator, 2) study the possible application of dynamic crossover operators, such as the *dynamic FCB-crossovers* [22] and *dynamic heuristic FCB-crossovers* [51], which are based on the same absolute adaptive heuristics as the non-uniform mutation operator, and 3) investigate the impact of the mutation probability into TRAMSS performance, and, on the basis of this future study, to design an extended TRAMSS model that controls the mutation probability together with the mutation step size.

Appendix A. Crossover Operators for RCGAs

Let us assume that $X = (x_1 \ldots x_n)$ and $Y = (y_1 \ldots y_n)$ are two real-coded chromosomes that have been selected to apply the crossover operator to them. Below, the effects of the four crossover operators used in the paper are shown.

- *Linear crossover* [46]. It returns three offspring:

$$Z_1 = (z_1^1, \ldots, z_i^1, \ldots, z_n^1) \text{ with } z_i^1 = \frac{1}{2} \cdot x_i + \frac{1}{2} \cdot y_i,$$
$$Z_2 = (z_1^2, \ldots, z_i^2, \ldots, z_n^2) \text{ with } z_i^2 = \frac{3}{2} \cdot x_i - \frac{1}{2} \cdot y_i,$$
$$Z_3 = (z_1^3, \ldots, z_i^3, \ldots, z_n^3) \text{ with } z_i^3 = -\frac{1}{2} \cdot x_i + \frac{3}{2} \cdot y_i.$$

  The resulting descendents are the two best of these three offspring.
- *Discrete crossover* [47]. z_i is a randomly (uniformly) chosen value from the set $\{x_i, y_i\}$.
- *BLX-α crossover* [48]. z_i is a randomly (uniformly) chosen number from the interval $[Min - I \cdot \alpha, Max + I \cdot \alpha]$, where $Max = \max\{x_i, y_i\}$, $Min = \min\{x_i, y_i\}$, and $I = Max - Min$. We have assumed $\alpha = 0.5$.
- *Fuzzy recombination* [49]. The probability that the i-th gene in the offspring has the value z_i is given by the distribution $p(z_i) \in \{\phi_{x_i}, \phi_{y_i}\}$, where ϕ_{x_i} and ϕ_{y_i} are triangular probability distributions with the following features ($d \geq 0.5$):

Triangular prob. dist.	Minimum value	Modal value	Maximum value
ϕ_{x_i}	$x_i - d \cdot \lvert y_i - x_i \rvert$	x_i	$x_i + d \cdot \lvert y_i - x_i \rvert$
ϕ_{y_i}	$y_i - d \cdot \lvert y_i - x_i \rvert$	y_i	$y_i + d \cdot \lvert y_i - x_i \rvert$

d has been set to 0.5 in the experiments.

All these crossover operators may be ordered with regard to the way randomness is used for generating the genes of the offspring: 1) linear crossover do not use it, 2) discrete crossover considers discrete probability distributions, where there are only two possibilities (x_i or y_i), 3) BLX-α introduces uniform continuous probability distributions, and 4) fuzzy recombination applies triangular continuous probability distributions, and therefore, it may be considered as a hybrid between discrete crossover and BLX-α.

Probability distributions used by BLX-α and fuzzy recombination are calculated according to the distance between the genes in the parents (x_i and y_i), and the α and d values, respectively. So, they fit their action range depending on the diversity of the population using specific information held by the parents [48].

Acknowledgments

This research has been supported by CICYT TIC96-0778.

References

1. D.E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley: New York, 1989.
2. J.H. Holland, *Adaptation in Natural and Artificial Systems*, The MIT Press: London, 1992.
3. T. Bäck, *Evolutionary Algorithms in Theory and Practice*, Oxford University Press: Oxford, 1996.
4. T. Bäck, D. Fogel, and Z. Michalewicz, *Handbook of Evolutionary Computation*, Oxford University Press: Oxford, 1997.
5. Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, Springer-Verlag: New York, 1992.
6. F. Herrera, M. Lozano, and J.L. Verdegay, "Tackling real-coded genetic algorithms: Operators and tools for the behavioural analysis," *Artificial Intelligence Reviews*, vol. 12, no. 4, pp. 265–319, 1998.
7. P.D. Surry and N.J. Radcliffe, "Real representations," in *Foundations of Genetic Algorithms IV*, Morgan Kaufmann: San Mateo, pp. 343–363, 1996.
8. H.-P. Schwefel, *Evolution and Optimum Seeking*. Sixth-Generation Computer Technology Series, Wiley: New York, 1995.
9. D.B. Fogel, *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*, IEEE Press: Piscataway, 1995.
10. T.-H. Li, C.B. Lucasius, and G. Kateman, "Optimization of calibration data with the dynamic genetic algorithm," *Analytica Chimica Acta*, vol. 2768, pp. 123–134, 1992.
11. L.J. Eshelman and J.D. Schaffer, "Preventing premature convergence in genetic algorithms by preventing incest," in *Proc. of the Fourth Int. Conf. on Genetic Algorithms*, edited by R. Belew and L.B. Booker, Morgan Kaufmann: San Mateo, 1991, pp. 115–122.
12. W.M. Spears, "Crossover or mutation?" in *Foundations of Genetic Algorithms 2*, edited by L.D. Whitley, Morgan Kaufmann Publishers: San Mateo, pp. 221–238, 1993.
13. P.J. Angeline, "Adaptive and self-adaptive evolutionary computations," in *Computational Intelligence: A Dynamic Systems Perspective*, edited by M. Palaniswami, Y. Attikiouzel, R. Markc, D. Fogel, and T. Fukuda, IEEE Press: Piscataway, NJ, pp. 152–163, 1995.
14. F. Herrera and M. Lozano, "Adaptation of genetic algorithm parameters based on fuzzy logic controllers," in *Genetic Algorithms and Soft Computing*, edited by F. Herrera and J.L. Verdegay, Physica-Verlag: Wurzburg, pp. 95–125, 1996.
15. R. Hinterding, Z. Michalewicz, and A.E. Eiben, "Adaptation in evolutionary computation: A survey," in *Proc. of the 4th IEEE Conf. on Evolutionary Computation*, IEEE Service Center, 1997, pp. 65–69.
16. T. Bäck, M. Schütz, and S. Khuri, "Evolution strategies: An alternative evolution computation method," in *Artificial Evolution*, edited by J.M. Alliot, E. Lutton, E. Ronald, M. Schoenauer, and D. Snyers, Springer: Berlin, pp. 3–20, 1996.
17. S.W. Mahfoud, "Niching methods for genetic algorithms," Illinois Genetic Algorithms Laboratory, University of Illinois, Illigal Report No. 95001, 1995.
18. H.-P. Schwefel, *Numerical Optimization of Computer Models*, Wiley: Chichester, 1981.
19. T. Bäck, "The interaction of mutation rate, selection, and self-adaptation within genetic algorithm," in *Parallel Problem Solving from Nature*, vol. 2, edited by R. Männer and B. Manderick, Elsevier Science Publishers: Amsterdam, pp. 85–94, 1992.
20. T. Bäck and M. Schütz, "Intelligent mutation rate control in canonical genetic algorithms," in *Foundation of Intelligent Systems 9th Int. Symposium*, edited by Z.W. Ras and M. Michalewicz, Springer: Berlin, 1996, pp. 158–167.
21. A.L. Tuson and P. Ross, "Adapting operator settings in genetic algorithms," *Evolutionary Computation*, vol. 6, no. 2, pp. 161–184, 1998.
22. F. Herrera, M. Lozano, and J.L. Verdegay, "Dynamic and heuristic fuzzy connectives-based crossover operators for controlling the diversity and convengence of real coded genetic algorithms," *Int. Journal of Intelligent*, vol. 11, pp. 1013–1041, 1996.
23. J. Périaux, M. Sefrioui, B. Stoufflet, B. Mantel, and E. Laporte, "Robust genetic algorithms for optimization problems in aerodynamic design," in *Genetic Algorithms in Engineering and Computer Science*, edited by G. Winter, J. Périaux, M. Galán, and P. Cuesta, John Wiley & Sons: New York, pp. 371–396, 1995.
24. M. Sefrioui and J. Périaux, "Fast convergence thanks to diversity," in *Proc. of the Fifth Annual Conference on Evolutionary Programming*, 1996, pp. 313–321.
25. M. De La Maza and D. Yuret, "Dynamic hillclimbing," *AI Expert*, vol. 9, no. 3, 1994.
26. R. Hinterding, "Gaussian mutation and self-adaptation for numeric genetic algorithms," in *Proc. of IEEE International*

Conference on Evolutionary Computation, IEEE Press: New York, 1995, pp. 384–389.
27. R. Hinterding, Z. Michalewicz, and T.C. Peachey, "Self-adaptive genetic algorithm for numeric functions," in *4th International Conference on Parallel Problem Solving from Nature*, edited by H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, Springer-Verlag: Berlin, Germany, 1996, pp. 420–429.
28. J. Maresky, Y. Davidor, D. Gitler, G. Aharoni, and A. Barak, "Selectively destructive re-start," in *Proc. of the Sixth Int. Conf. on Genetic Algorithms*, edited by L. Eshelman, Morgan Kaufmann Publishers: San Francisco, 1995, pp. 144–150.
29. D.E. Goldberg, "Sizing populations for serial and parallel genetic algorithms," in *Proc. of the Third Int. Conf. on Genetic Algorithms*, edited by J.D. Schaffer, Morgan Kaufmann Publishers: San Mateo, 1989, pp. 70–79.
30. L.J. Eshelman, "The CHC adaptive search algorithm: How to have safe search when engaging in nontraditional genetic recombination," in *Foundations of Genetic Algorithms 1*, edited by G.J.E Rawlin, Morgan Kaufmann: San Mateo, pp. 265–283, 1991.
31. J.J. Grefenstette, "Genetic algorithms for changing environments," in *Parallel Problem Solving from Nature*, vol. 2, edited by R. Männer and B. Manderick, Elsevier Science Publishers: Amsterdam, pp. 137–144, 1992.
32. C.G. Shaefer, "The ARGOT strategy: Adaptive representation genetic optimizer technique," in *Proc. Second Int. Conf. on Genetic Algorithms*, L. Erlbaum Associates: Hillsdale, MA, 1987.
33. N.N Schraudolph and R.K. Belew, "Dynamic parameter encoding for genetic algorithms," *Machine Learning*, vol. 9, pp. 9–21, 1992.
34. D. Whitley, K. Mathias, and P. Fitzhorn, "Delta coding: An iterative search strategy for genetic algorithms," in *Proc. of the Fourth Int. Conf. on Genetic Algorithms*, edited by R. Belew and L.B. Booker, Morgan Kaufmann: San Mateo, 1991, pp. 77–84.
35. T. Bäck and H.-P. Schwefel, "Evolution strategies I: Variants and their computational implementation," in *Genetic Algorithms in Engineering and Computer Science*, edited by G. Winter, J. Périaux, M. Galán, and P. Cuesta, John Wiley & Sons: New York, pp. 111–126, 1995.
36. J.E. Baker, "Adaptive selection methods for genetic algorithms," in *Proc. First Int. Conf. on Genetic Algorithms*, L. Erlbaum Associates: Hillsdale, MA, 1985, pp. 101–111.
37. J.E. Baker, "Reducing bias and inefficiency in the selection algorithm," in *Proc. Second Int. Conf. on Genetic Algorithms*, L. Erlbaum Associates: Hillsdale, MA, 1987, pp. 14–21.
38. K.A. De Jong, "An analysis of the behavior of a class of genetic adaptive systems," Doctoral Dissertation, University of Michigan, 1975.
39. A.O. Griewangk, "Generalized descent of global optimization," *Journal of Optimization Theory and Applications*, vol. 34, pp. 11–39, 1981.
40. T. Bäck, "Self-Adaptation in genetic algorithms," in *Proc. of the First European Conference on Artificial Life*, edited by F.J. Varela and P. Bourgine, The MIT Press: Cambridge, MA, 1992, pp. 263–271.
41. A. Törn and Ž. Antanas, *Global Optimization, Lecture Notes in Computer Science*, vol. 350, Springer: Berlin, 1989.
42. D. Whitley, R. Beveridge, C. Graves, and K. Mathias, "Test driving three 1995 genetic algorithms: New test functions and geometric matching," *Journal of Heuristics*, vol. 1, pp. 77–104, 1995.
43. D. Schlierkamp-Voosen and H. Mühlenbein, "Strategy adaptation by competing subpopulations," in *Parallel Problem Solving from Nature*, vol. III, edited by Y. Davidor, H.-P. Schwefel, and R. Maenner, Springer-Verlag: Berlin, Germany, pp. 199–208, 1994.
44. D. Whitley, D. Rana, J. Dzubera, and E. Mathias, "Evaluating evolutionary algorithms," *Artificial Intelligence*, vol. 85, pp. 245–276, 1996.
45. H. Mühlenbein, M. Schomisch, and J. Born, "The parallel genetic algorithm as function optimizer," in *Fourth Int. Conf. on Genetic Algorithms*, edited by R. Belew and L.B. Booker, Morgan Kaufmann: San Mateo, 1991, pp. 271–278.
46. A. Wright, "Genetic algorithms for real parameter optimization," in *Foundations of Genetic Algorithms*, vol. 1, edited by G.J.E. Rawlin, Morgan Kaufmann: San Mateo, pp. 205–218, 1991.
47. H. Mühlenbein and D. Schlierkamp-Voosen, "Predictive models for the breeder genetic algorithm I. continuous parameter optimization," *Evolutionary Computation*, vol. 1, pp. 25–49, 1993.
48. L.J. Eshelman and J.D. Schaffer, "Real-coded genetic algorithms and interval-schemata," in *Foundation of Genetic Algorithms* vol. 2, edited by L.D. Whitley, Morgan Kaufmann Publishers: San Mateo, pp. 187–202, 1993.
49. H.M. Voigt, H. Mühlenbein, and D. Cvetković, "Fuzzy recombination for the breeder genetic algorithm," in *Proc. of the Sixth Int. Conf. on Genetic Algorithms*, edited by L. Eshelman, Morgan Kaufmann Publishers: San Francisco, 1995, pp. 104–111.
50. W.M. Spears, "Adapting crossover in evolutionary algorithms," in *Proc. of the Evolutionary Programming Conference 1995*, 1995, pp. 367–384.
51. F. Herrera and M. Lozano, "Heuristic crossover for real-coded genetic algorithms based on fuzzy connectives," in *4th International Conference on Parallel Problem Solving from Nature*, Lecture Notes on Computer Science, vol. 1141, edited by H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, Springer: Berlin, Germany, 1996, pp. 336–345.