Genetic Algorithm with Automatic Termination and Search Space Rotation

Bun Theang Ong and Masao Fukushima*

Abstract

In the last two decades, numerous evolutionary algorithms (EAs) have been developed for solving optimization problems. However, only a few works have focused on the question of the termination criteria. Indeed, EAs still need termination criteria prespecified by the user. In this paper, we develop a genetic algorithm (GA) with automatic termination and acceleration elements which allow the search to end without resort to predefined conditions. We call this algorithm "Genetic Algorithm with Automatic Termination and Search Space Rotation", abbreviated as GATR. This algorithm utilizes the so-called "Gene Matrix" (GM) to equip the search process with a self-check in order to judge how much exploration has been performed, while maintaining the population diversity. The algorithm also implements a mutation operator called "mutagenesis" to achieve more efficient and faster exploration and exploitation processes. Moreover, GATR fully exploits the structure of the GM by calling a novel search space decomposition mechanism combined with a search space rotation procedure. As a result, the search operates strictly within two-dimensional subspaces irrespective of the dimension of the original problem. The computational experiments and comparisons with some state-of-the-art EAs demonstrate the effectiveness of the automatic termination criteria and the space decomposition mechanism of GATR.

Keywords—Genetic Algorithms, Termination Criteria, Gene Matrix, Mutagenesis, Space Rotation, Space Decomposition

1   Introduction

Evolutionary algorithms (EAs) are population-based stochastic algorithms that draw inspiration from processes of biological evolution for problem solving. As such, they make use of mechanisms such as reproduction, recombination, mutation and competitive selections in order to create solutions known as individuals in the population that would fit better their environment. The population of individuals evolve through repetition of the above mentioned mechanisms in an attempt to mimic the life cycles of living species (Konar, 2005; Back et al., 1997).

*The authors are with the Department of Applied Mathematics and Physics, Graduate School of Informatics, Kyoto University, Kyoto 606-8501, JAPAN (tel: +81-75-753-5519; fax: +81-75-753-4756; email: ong_bt@amp.i.kyoto-u.ac.jp; fuku@amp.i.kyoto-u.ac.jp).

Genetic algorithm (GA) is one of the oldest and most popular evolutionary algorithms (Holland, 1975). Pioneered by Holland (1975), it largely imitates genetic inheritance from parents to children and natural selection procedures until a termination criterion is satisfied. However, an essential difference between natural evolution and problem solving is that in natural evolution, species do not usually seek for termination. In problem solving, on the other hand, at some point and under a given budget, we deliberately need to stop the life cycle process. When to stop is not a trivial question. It is admitted that in many real world applications, saving computational resources is of prime importance. Complex optimization problems for instance endure an intensive function evaluation process. By stopping the search right before unnecessary function evaluations are performed, it is the algorithmic efficiency that is increased.

In this regard though, an undesirable phenomenon is the *premature convergence*. When diversity of the population decreases below a certain level, the population may converge to a suboptimal similar individual. Thus when dealing with stopping criteria, one should also pay meticulous attention to the balance between exploration and exploitation. Hence, good automatic termination criteria should assure that the search avoids premature termination but also indicates the point in time when further computations becomes unnecessary. This feature is of key importance in some real-world applications such as in *evolutionary testing* (O'Sullivan et al., 1998; McMinn, 2004). Indeed, during the development of embedded systems, testing is one of the most important quality assurance measure. A huge amount of effort and budget are allocated for testing. In evolutionary testing, EAs are used for test data generation and to verify the logical and temporal correctness of a system. Most testing methods are specialized in the logical correctness. However, for real-time systems, it is also essential to check the temporal correctness. Evolutionary testing fills this gap by testing the timing constraints where a temporal error occurs when outputs are produced too early or if the computational time is too long. In such situations, it is crucial to have reliable automatic termination criteria for EAs.

Multi-start methods may also benefit from automatic termination criteria. Among the main components of a multi-start method, we note the stopping criterion used within the generation mechanism of candidate solutions. The stopping criterion, in this case also referred to as the restarting criterion, is prespecified by the user and has a big impact on the overall computational cost of the method. Consequently, a reliable automatic termination criterion may have a positive effect on multi-start methods, by reducing the cost of generating candidate solutions, thereby more iterations can be allowed with a fixed budget.

Basically, EAs cannot decide when or where they can terminate the search and usually a user should prespecify the maximum number of generations or function evaluations as termination criteria. There are only a few recent works on termination criteria for EAs (Giggs et al., 2006; Kwok et al., 2007; Jain et al., 2001). In (Giggs et al., 2006), an empirical study is conducted to detect the maximum number of generations using the problem characteristics. In (Kwok et al., 2007), the particle swarm optimization algorithm is stopped using a termination condition based on statistics. The hypothesis testing non-parametric sign-test method is considered as a decision making process using a list of the stored highest fitness values in each iteration. The search stops when the hypothetical test indicates that no significant improvement in terms of solution quality

is going to occur. In (Jain et al., 2001), eight termination criteria have been studied with an interesting idea of using clustering techniques to examine the distribution of individuals in the search space at a given generation.

The most commonly employed termination criteria for EAs can be enumerated as the T_{Fit} *Criterion*, the T_{Pop} *Criterion*, the T_{Bud} *Criterion* and the T_{SFB} *Criterion*. The T_{Fit} *Criterion* uses convergence measures of the best fitness function values over generations. This criterion is used for instance in (Ong et al., 2006; Tsai et al., 2004; Zhong et al., 2004; Hansen and Kern, 2004), where the goal is to get as close as possible to the known global minima. In (Leung and Wang, 2001), the search stops after reaching the maximum number of consecutive generations without improvement. When used alone, however, T_{Fit} *Criterion* may easily lead EAs towards local minima, especially if the algorithm tends to reach in early stages a deep local minimum (Hedar and Fukushima, 2006b; Jain et al., 2001; Safe et al., 2004). The T_{Pop} *Criterion* uses convergence measures of the population over generations. This criterion is not particularly efficient though, since having one individual to reach a global minimum is enough. Moreover making the whole population or a part of it convergent can be expensive. The T_{Bud} *Criterion* uses a prespecified budget, that can be the number of generations or function evaluations (Koumousis and Katsaras, 2006; Lee and Yao, 2004; Ong and Keane, 2004; Ong et al., 2006; Tu and Lu, 2004; Yao et al., 1999; Zhou et al., 2007). The drawback is that it requires prior information about the test problem and is also highly problem dependent. Finally, the T_{SFB} *Criterion* checks the progress of exploration and exploitation processes by using search feedback measures. Unfortunately the use of search feedback may bring a complexity problem due to the need to save and check historical search information that can be huge and is also very sensitive to the dimensionality.

Our work is devoted to the development of a GA that would terminate without *a priori* knowledge of any desirable or available solution range, and of any specific number of iterations or function evaluations. It is desired that the termination instant after completion of adequate exploration and exploitation is determined by the algorithm itself. We propose in this paper an improved method of the Genetic Algorithm with Automatic Accelerated Termination method (G3AT) presented in (Hedar et al., 2007). G3AT is originally a GA with new directing strategies. The key elements of G3AT are the Gene Matrix (GM), the mutagenesis operator and a final intensification process. The GM is a matrix constructed to represent subranges of the possible values of each variable and consequently reflects the distribution of genes over the search range. Its role is to assist the exploration process in two different ways. First, GM can provide the search with new diverse solutions by applying the mutagenesis operator. Mutagenesis operator is a new type of mutation that works in combination with GM. It alters some individuals in order to accelerate the exploration and exploitation processes by guiding the search specifically towards unexplored areas. Also, GM is the key to let G3AT know how far the exploration process has been performed in order to determine an adequate termination instant. By definition, however, although numerical experiments lead to positive results, the GM is a two-dimensional structure and there is no evidence that it is able to represent the distribution of individuals in the multi-dimensional search space accurately, especially in high-dimensional, multi-modal and highly epistatic problems. We thus provide in this paper a response to those considerations, while attempting to improve the performance of the G3AT algorithm. We keep focused however on the

main objective of this work, that is, on the automatic termination. We would like to stress out that our main objective is not to outperform existing results, although we will show that what is proposed in this work is competitive.

The response is a rotation-based version of the G3AT method, designated as GATR, which stands for Genetic Algorithm with Automatic Termination and Search Space Rotation. The main new elements are the Space Decomposition (SD) and the Space Rotation (SR). SD and SR work in combination in order to create a two-dimensional environment for the GM irrespective of the original dimension of the problem to be solved. In this environment, the GM goes through a series of rotations which allow the search to avoid premature convergence and termination due to specificities of the problems. As a hybrid GA, GATR first emphasizes on exploring the whole search space using the GM. Afterward, the exploitation process is invoked through a local search method in order to refine the best candidates obtained so far. GATR thus behaves like a "Memetic Algorithm" (Moscato, 1999) in order to achieve faster convergence (Ong and Keane, 2004; Ong et al., 2006).

The performance of the algorithm is evaluated in 10, 30 and 50 dimensions on the set of 25 test problems of the CEC 2005 real-parameter optimization contest (Suganthan et al., 2005) and compared against a number of existing algorithms such as G3AT, a Real-Coded Memetic Algorithm (RCMA) (Lozano et al., 2005), a version of the Evolution Strategy with Covariance Matrix Adaptation method (which is a well-known state-of-the-art method for adaptive mutation) that is combined with a restart strategy (L-CMA-ES) (Hansen and Kern, 2004; Hansen, 2006) as well as the recent Non-Revisiting Genetic Algorithm with Parameter-less Adaptive Mutation (NrGA) (Yuen and Chow, 2009), a Differential Evolution method using an adaptive local search (DEahcSPX) (Noman and Iba, 2008) and the winner of the CEC 2005 competition, the Restart CMA Evolution Strategy With Increasing Population Size (G-CMA-ES) (Hansen et al., 2005b).

The rest of the paper is organized as follows. Section 2 provides a review of the GM and the mutagenesis operator. Section 3 introduces the new concepts developed to reinforce the GM model. The SD and SR are also described in this section. In Section 4, components and the formal algorithm of GATR are detailed. In Section 5, the methodology adopted for the numerical experiments is explained and results in dimensions 10, 30 and 50 are presented with the benchmark functions from CEC 2005. A comparative study against other methods from the literature is also conducted. A summary with conclusions and future work is provided in Section 6. The influence of the choice of some key parameters as well as the impact of the SD and SR is discussed in the Appendix.

2   Gene Matrix and Mutagenesis

EAs are in general not designed to terminate without some kind of external authority. When some kind of knowledge about the problem or the answer is not available, a downside is that the search may as well obtain an optimal or near-optimal solution in an early stage of the search, but still, will not be able to judge whether it should terminate at that point. Thus one has to consider which termination criterion would
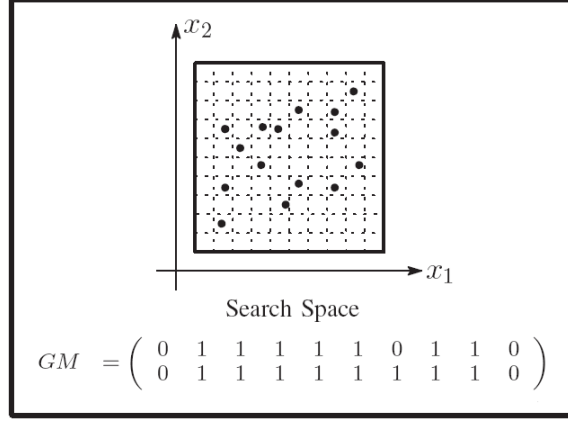
Figure 1: An example of the *Gene Matrix* in R^2.

be appropriate under the given circumstances. The following gives a description of the GM and the mutagenesis operator, both working mutually in order to determine a proper termination instant. The benefits of those two concepts compared with a canonical GA and some other EAs are discussed in (Hedar et al., 2007).

2.1 Gene Matrix and Termination

G3AT adopts the real-coding representation of individuals. Hence in the search space, every individual x consists of n variables or genes. The range of each gene is divided into m subranges in order to check the diversity of the gene values. GM is initialized as the $n \times m$ zero matrix in which each entry of the i-th row refers to a subrange of the i-th gene. While the search is processing, the entries of GM are updated if new values for genes are generated within the corresponding subranges. Those entries are granted with a non-null value. Specifically, during the search, the value of each gene is considered in order to extract the number associated with the subrange where the considered gene is located, say $v \in \{1,\ldots,m\}$. Let x_i be the representation of the i-th gene, $i = 1,\ldots,n$. Once a gene gets a value corresponding to a non-explored subrange, GM is updated by flipping a zero into a one in the corresponding (i,v) entry. Let us note that, with this mechanism, GM is not sensitive to the number of genes lying inside each subrange.

Figure 1 shows an example of GM in two dimensions. In the figure, the range of each gene is divided into ten subranges, thus partitioning the search space into a hundred sectors. For the first gene x_1, no individual has been generated inside the subranges 1, 7 and 10. Consequently, GM's values in the $(1,1)$, $(1,7)$ and $(1,10)$ entries are still equal to zero. For the second gene x_2, only the first and the last subranges are still unexplored, hence GM's values in entries $(2,1)$ and $(2,10)$ are null.

After having a GM full, i.e., with no zero entry, the search judges that an advanced exploration process has been achieved and can be stopped. In this way, the principal use of GM is to equip the search process with a practical termination tool. Other ver-

5

sions of GM, sensitive to the number of genes within each subrange, have been investigated. However, our comparative study has revealed that the zero-one GM mechanism presented here (and implemented in GATR) yields the best performances in terms of number of function evaluations versus solution quality. We define the GM completion ratio, referred to as *CP*, as the number of non-null entries divided by the total number of entries of GM.

2.2 Mutagenesis

"Mutagenesis" is a more artificial mutation operation that allows some characteristic children to improve themselves by modifying their genes. It is called after computing the offspring in each generation, using two different types of mutagenesis operation: the *gene matrix mutagenesis* (GM-*Mutagenesis*) and the *best child inspiration mutagenesis* (BCI-*Mutagenesis*). Specifically, GATR sorts the current population of size μ, and then selects the worst N_w ($< \mu$) individuals that will participate in the operations. GM-*Mutagenesis* and BCI-*Mutagenesis* alter N_1 and N_2 out of these N_w (with $N_1 + N_2 \leq N_w$) worst individuals, respectively.

2.2.1 GM-*Mutagenesis*

GM-*Mutagenesis* operates in two ways in combination with the GM. First, in order to keep genetic diversity and accelerate the exploration process, GM-*Mutagenesis* mimics the mutation operation by altering N_1 from the N_w worst individuals that have been selected to survive for the next generation. The alteration is however guided by the status of the GM and thus not completely random. The second feature is to generate new diverse solutions in some hopefully unexplored partitions of the search space, which supports the exploration process by guidance of GM instead of relying solely on the crossover operation. Specifically, a zero-position in GM is randomly chosen, say the position (i, j) (i.e., the variable x_i has not yet taken any value in the j-th partition of its range). Then a random value for x_i is chosen within this partition to alter one of the chosen individuals for mutagenesis. Using this setting for x_i, there is a chance for the crossover operation to explore different combinations of solutions containing this new value of x_i. GM is then updated since a new partition has been visited. If there is no zero-position available in GM, this operation is omitted. The formal procedure for GM-*Mutagenesis* is given in Procedure 2.1.

Procedure 2.1 GM−Mutagenesis(x, GM)

> 1. *If there is no zero-position in GM, then return; otherwise, go to Step 2.*
> 2. *Choose a zero-position (i, j) in GM randomly.*
> 3. *Update x by setting $x_i = l_i + (j - r) \frac{u_i - l_i}{m}$, where r is a random number from $(0, 1)$, and l_i, u_i are the lower and upper bounds of the variable x_i, respectively.*
> 4. *Update GM and return.*

2.2.2 BCI-*Mutagenesis*

BCI-*Mutagenesis* operation is also based on the mutation operation, but instead of using the GM as the basis for guiding the mutations, it uses the best child's gene values in the children pool. From the N_w worst children, another group of N_2 individuals are modified. The idea is that if an elite child confers some of its genetic information to children demonstrating poor performance, they might improve themselves. Thus for each of the N_2 worst children, one gene from the best child is randomly chosen and copied to the same position of the considered bad child as stated formally in Procedure 2.2.

Procedure 2.2 BCI$-$Mutagenesis(x, x^{Best})

    1. Choose a random gene position i from $\{1, 2, \ldots, n\}$.
    2. Alter x by setting $x_i := x_i^{Best}$, and return.

3 Space Rotation and Space Decomposition

In this section, new concepts that reinforce the GM model are presented. G3AT is a method that aims to determine a stopping point without knowledge about the problem. We want GATR to be a method that not only possesses a proper stopping judgement but also exhibits more accurate and versatile performance than G3AT against a wider class of problems.

3.1 High-dimensional Search Space

By definition, GM has a two-dimensional structure, i.e., a matrix of indicator variables for subranges in each dimension of the search space. Consequently it may face some difficulties in representing the distribution of individuals in a high-dimensional search space. Basically, for m subranges and n dimensions, we can count m^n hyperrectangles in the search space. However the GM can become complete with less than $m \times n$ points. Therefore, there is a possibility of having a misguided termination of the exploration process, as depicted in a simple example in Figure 2(a), where the GM is already full although the search space is far from being entirely covered. The crossover operation given in Procedure 4.1, which will be described in the next section, can overcome this drawback as shown in Figure 2(b). This type of crossover has been chosen to support the GATR exploration process. Mutagenesis also cooperates with this type of crossover by combining mutated genes with other existing genes of the current population through recombination. As a matter of fact, numerical simulations show that GM explores the search space widely. However, although it is easy to comprehend in two dimensions, it is difficult to admit whether such mechanism can perform a sufficient exploration of the search space when the dimension of the problem increases.

3.2 Space Rotation

Those considerations on GM prompt us to introduce the space rotation mechanism. As depicted in Figure 3(a), a particular distribution of the individuals may lead to
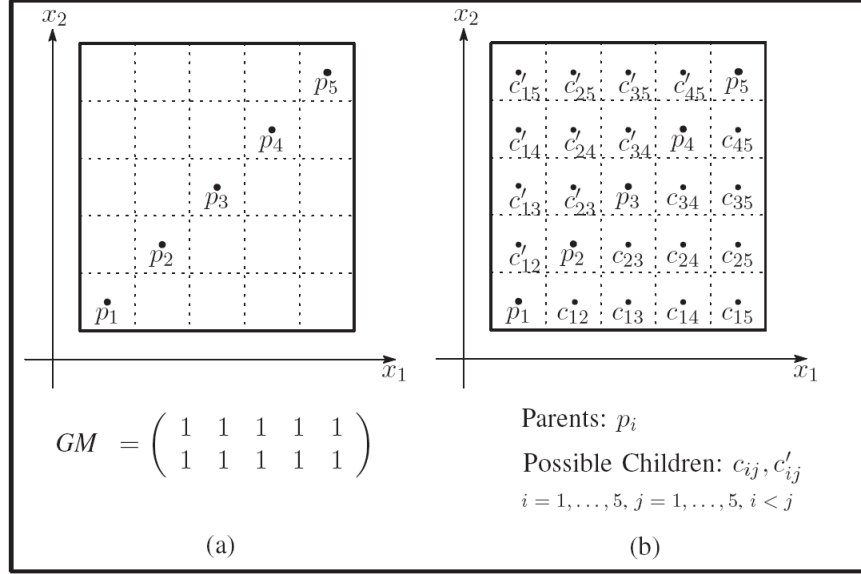
Figure 2: The role of crossover operation and GM.

premature completion of GM. Indeed in this case, it is clear that the entire search space has not been fully covered. In Figure 3(b), after applying to the search space a clockwise 45 degrees rotation, the same distribution of the explored areas is seen by GM from a different angle. Recomputing the GM then reveals unexplored regions (i.e., the matrix is not complete anymore). Thus in order to escape from premature termination of the search, the concept is to "rotate" the search space and attach to the resulting spaces a GM that has not necessarily the same configuration as the original one at a given instant. Note that SR specifically works on the transformation of two-dimensional spaces, which does not mean however that G3AT cannot handle higher dimensional problems.

Let us consider the optimization problem $\min_{x \in X} f(x)$, where f is a real-valued function defined on the search space $X \subseteq R^n$ with variable $x \in X$. The search space X can be seen as an n-dimensional rectangle in R^n. The edges of this rectangle are given by the lower bounds l_i and upper bounds u_i of the variables x_i with $l_i \leq x_i \leq u_i$ ($i = 1,...,n$). In particular when $n = 2$, the search space may be represented as in Figure 4.

Rotation is defined by the rotation angle α, which is a divisor of 360, as well as the number of rotations, say NR. To each rotation is associated a GM, namely GM_{NR}. They are independently fed by their respective rotated spaces. Let us note that, for some angle α and number of rotations NR, repeated rotations of the search space will eventually result in coming back to the initial search space, as seen in Figure 5. Obviously there is no benefit in computing several GMs for the same resulting space.

During the search process, SR applies the rotation operator, named $Rot_\alpha[x]$, to each individual x using angle α such that $x' = Rot_\alpha[x]$, where x' represents the rotated indi-
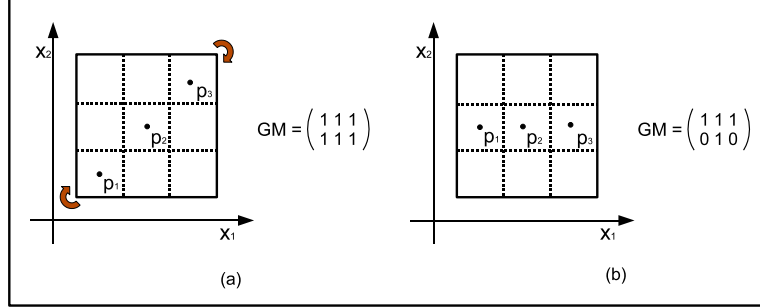
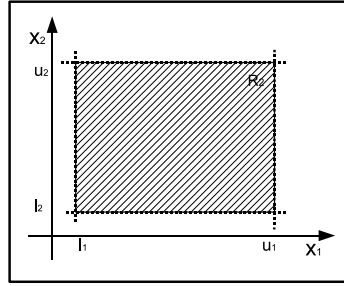Figure 3: Diagonal distribution of individuals before and after rotation and the associated GM.

Figure 4: Search space with $n = 2$.

vidual. Then the original objective function can be recomputed by $f(x) = f(Rot_\alpha^{-1}[x'])$. Formal description of SR is given in Procedure 3.1, where x_i represents the i-th gene of each individual x within the population X, and X' is the population in the rotated subspace associated with its GM that will be denoted as GM'. Each population is of size μ.

Procedure 3.1 *Rotation(X', GM')*

1. Set values of α. Set $Rot_\alpha := \begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix}$.

2. For each individual x in X, let $x_i' := (Rot_\alpha[x])_i$, $i = 1, ..., n$.

3. Update GM' using all individuals x', and return.

3.3 Space Decomposition

GM and SR are by nature two-dimensional mechanisms. For problems in higher dimensions, we invoke the space decomposition mechanism. Its role is to transform a high-dimensional search space of a problem into several two-dimensional subspaces. In order to do so, before the search process, SD selects two variables, say x_a and x_b, among $\{x_1, x_2, \ldots, x_n\}$, which will be referred to as the current "active variables". During the generation process, only x_a and x_b are directly treated as the variables to be
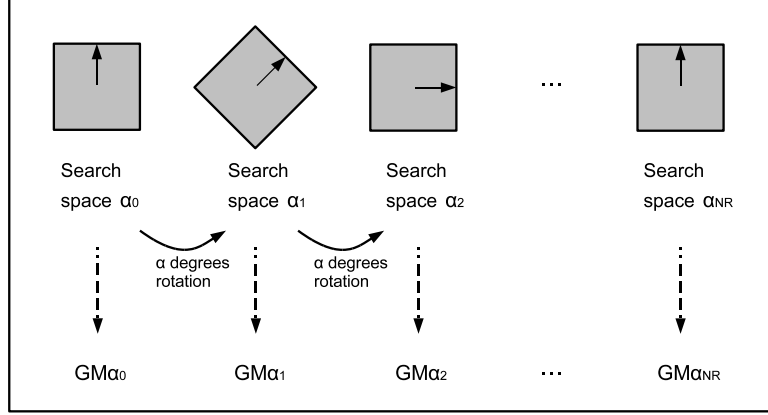
Figure 5: *NR* rotations of the search space by angle α and the associated GM.

manipulated. The other variables, referred to as "passive variables", momentarily hold their values fixed (i.e., they will not participate in the improvement for a moment). The upper and lower bounds of x_a and x_b are used to define the edges of the rectangle that will be considered by SR. At the end of the search, all genes of the best individual found (both active and passive variables) are given a chance to be improved in an intensification phase. The resulting candidate (individual) is referred to as x^{elite}. At this point, we end the search using the GM information, but obviously the obtained x^{elite} does not announce the end of the entire process since we have only considered a portion of variables. Thus, if "enough" genes have not participated yet in the search as active variables, the current ones are set as passives and two other genes are selected as new actives. The search is then resumed, as what we will call a new "era". It is important to understand that the entire process will end when enough active variables have participated in their own eras. However, it is the role of the GM to terminate adequately each era. The efficiency of the whole algorithm is thus under the influence of the performance of the automatic termination. At the beginning of each era, the whole population is randomly generated. However, once the passive variables are designated, the values of all individuals' passive variables (or genes) are transformed in such a way that their values are aligned with those of the previously obtained x^{elite}.

3.4 Space Normalization

Rotation is defined in two dimensions and the search space X can thus be seen as a rectangle. Rotating this rectangle implies that the lower and upper bounds should also be adapted in order to match the structure of the GM. Indeed, generation of new individuals has to be restricted within the inscribed circle of each rotated rectangle in order to avoid inconsistencies in regard to each gene's bounds. Namely, we want to avoid the generation of individuals that would actually lie outside the limits of the original search space. In GATR, the normalization operation sets the bounds of the active variables in such a way that X is always maintained as the square $X = (-1, 1) \times (-1, 1)$. Generation
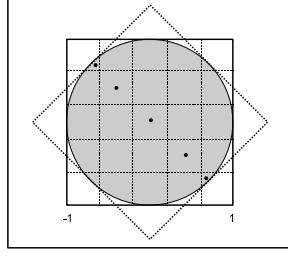
Figure 6: Normalized search space and the inscribed circle.

of new individuals is then limited within the inscribed circle of each square, as shown in Figure 6. Of course this operation is valid as long as we solely consider the search space X as a region wherein we can expect an optimal solution. In case the considered problem is constrained, the area in which the population is generated should be extended to the escribed circle. Note that the search domain is essential for constructing GM and it can be easily determined for many application problems. However, the idea of constructing a GM can be modified if the actual search domain is not specified. The user can start with a sufficiently large initial search domain and then new partitions of GM can be added whenever new regions are reached outside the initial search domain.

4 Formal GATR Algorithm

Before stating the formal algorithm of GATR, basic components that compose a GA and that are used by GATR are briefly described in this section.

4.1 GATR components

The parent selection mechanism implemented in GATR is based on the *linear ranking selection mechanism* (Baker, 1985; Hedar and Fukushima, 2003). This mechanism ranks the initial population P according to the fitness function value of each individual. From P, an intermediate population P' is produced by copying repeatedly individuals based on their fitness ranking until P' becomes full. Note that an already chosen individual can be selected more than once.

The crossover operation has an exploration tendency, and therefore is not applied to all parents. The crossover used in GATR can be seen as a ρ-point operator with a random integer $\rho \leq n$, where n is the dimension of the problem. Therefore, a recombined child is calculated to have ρ partitions. The following procedure describes the GATR crossover operation formally, with two parents p_1 and p_2, and two children c_1 and c_2.

Procedure 4.1 Crossover(p_1, p_2, c_1, c_2)

　　1. Choose an integer ρ ($2 \leq \rho \leq n$) randomly.

2. *Partition each parent into ρ partitions at the same positions, i.e., $p_1 = [X_1^1\ X_2^1\ \ldots\ X_\rho^1]$ and $p_2 = [X_1^2\ X_2^2\ \ldots\ X_\rho^2]$.*
3. *Choose a random mask (binary string) s of size ρ, i.e., $s = o_1 o_2 \ldots o_\rho$, where $o_i \in \{0,1\}$, $i = 1, \ldots, \rho$.*
4. *Set $c_1 = p_1$ and $c_2 = p_2$.*
5. *Swap partitions X_i^1 and X_i^2 in c_1 and c_2 if the corresponding digit o_i of mask s is equal to 1, where $i = 1, \ldots, \rho$, and return.*

The mutation operator is designed to take advantage of the information contained in the GM. Indeed, existing zeros from GM are randomly selected, say in the position (i, j), and a randomly chosen individual from the intermediate pool IP_M has its gene x_i modified by a new value lying inside the j-th partition of its range. For each individual in the intermediate population P' and for each gene, a random number from the interval $(0, 1)$ is associated. If the associated number is less than the mutation probability π_m, then the individual is copied to the intermediate pool IP_M. The number of times the associated numbers are less than the mutation probability π_m is counted, and let num_m denote this number. Afterward the mutation operation ensures that the total number num_m of genes to be mutated does not exceed the number of zeros in the GM, denoted num_{zeros}. Otherwise the number of genes is reduced to num_{zeros}. The formal procedure for mutation is analogous to Procedure 2.1, where m represents the number of each gene subranges.

Procedure 4.2 Mutation(x, GM)

1. *If GM is full, then return; otherwise, go to Step 2.*
2. *Choose a zero-position (i, j) in GM randomly.*
3. *Update x by setting $x_i = l_i + (j - r)\frac{u_i - l_i}{m}$, where r is a random number from $(0, 1)$, and l_i, u_i are the lower and upper bounds of the variable x_i, respectively.*
4. *Update GM and return.*

Parents and children then compete together for survival, making GATR a steady-state GA.

4.2 Intensification

Intensification is the process whose purpose is to refine the elite solution obtained at the end of an era. In GATR, intensification calls a local search process based on the Kelley's modification (Kelley, 1999) of the Nelder-Mead (NM) method (Nelder and Mead, 1965). The use of systematic intensification at the end of each era, however, can be expensive in terms of function evaluations, abbreviated as Feval. In GATR, intensification is therefore sparingly called by specific eras since it yields positive effects when called at intermediate stages of the search. We should remark, however, that the final refinement by intensification at the end of the search is most important to achieve higher accuracy. The numbers designating the eras that will go through an intensification step are determined using Procedure 4.3 and stored in the intensification list (IL).

Procedure 4.3 *Intensification list IL(n)*

1. *Initialize $IL = \emptyset$. Compute $w = round(n/2)$ and $z = round(n/10)$. Initialize $IL_{temp} = [1, 2, \ldots, w]$.*
2. *For $i := 1, \ldots, z$, add $IL_{temp}(i)$ and $IL_{temp}(w + 1 - i)$ to IL.*

4.3 Formal Algorithm

The search process in GATR is repeated several times, with different active variables taking part in distinct eras. Each search can be considered to be a G3AT run for a two-dimensional problem, taking advantage of GM and SR to terminate adequately. Preliminaries consist in SD whose role is to transform an *n*-dimensional search space into several two-dimensional subspaces. During an era, GATR starts by generating an initial population of size μ and dimension n. Two genes, x_a and x_b, are chosen as active variables. They will participate in the current era's evolution, while the other genes have their values replicated from x^{elite}'s respective gene values. In the very first era, however, each gene of x^{elite} is pre-initialized as $x_i^{elite} := \left(\frac{u_i + l_i}{2}\right)$, where l_i and u_i are the lower and upper bounds of gene x_i, respectively. The way in which x_a and x_b are selected will be discussed later. For now, we will refer to the list of successive pairs of active variables as the "Combination List" (CL) of length *CL_length*. For any *n*-dimensional problem, *CL_length* eras are carried out with distinct pairs of active variables, and the values of passive genes remain unchanged during each era.

As a GA based method, GATR evaluates the population members, and then generates offsprings by applying parent selection, crossover and mutation operations. SR's role is to manipulate the two-dimensional space in which the active variables are evolved by executing *NR* rotations of angle α. *NR* distinct GMs are updated, each corresponding to one of the *NR* rotated subspaces. Mutagenesis is invoked until the GM completion ratio *CP*, which may be set as different from 1, is reached. An era terminates when all *NR* GMs reach *CP*. It is worthwhile to mention that the main role of GM and SR is to guide the search towards unexplored regions, while exploration by itself is conducted by the crossover and mutagenesis operations. Finally, an era ends with a final intensification phase that uses a local search method in order to refine the best solution found so far. Let us emphasize that each era consecutively improves the best found solution x^{elite} by using the active variables. However, intensification improves all genes, actives and passives. The solution x^{elite} is then used in the starting point for the next era, where a different couple of active variables is selected and the remaining ones become passive. When all the pairs of variables in CL have actively participated in some eras, GATR terminates with the last refined solution. A formal description of GATR is given below.

Algorithm 4.4 *GATR Algorithm*

1. ***Initialization.*** *Set values of m, μ, N_w, NR, α, CP, and (l_i, u_i) for $i = 1, \ldots, n$. Set the crossover and mutation probabilities $\pi_c \in (0, 1)$ and $\pi_m \in (0, 1)$, respectively. Set the Intensification List IL by applying Procedure 4.3. Set the Combination List $CL := \{x_1, x_2, \ldots, x_n\}$. If the number of elements in CL is odd, add x_r to CL, where x_r is an element*

chosen randomly from $\{x_1, x_2, \ldots, x_n\}$. *Set the generation counter* $t :=$ 1.

1.1. *Compute* x^{elite} *by* $x_i^{elite} := \frac{u_i + l_i}{2}$ *for* $i = 1, \ldots, n$.

1.2. *Set all variables to be passive. Select* x_a *and* x_b *from CL. Update CL by* $CL := CL \backslash \{x_a, x_b\}$ *and designate* x_a *and* x_b *as active variables.*

1.3. *Generate an initial population* P_0 *of size* μ, *normalize the search space and update* P_0 *such that* $x_i^m := x_i^{elite}$ *for* $i \in \{1, \ldots, n\} \backslash \{a, b\}$ *and* $m = 1, \ldots, \mu$.

1.4. *Initialize NR GMs,* $GM^1, GM^2, \ldots, GM^{NR}$, *as the* $n \times m$ *zero matrices.*

2. Parent Selection. *Evaluate the fitness function F for all individuals in* P_t. *Select an intermediate population of parents* P_t' *from the current population* P_t. *Let the initial parent pool* SP_t *and children pool* SC_t *be empty.*

3. Crossover. *Associate a random number from* $(0, 1)$ *with each individual in* P_t' *and add this individual to the parent pool* SP_t *if the associated number is less than* π_c. *Repeat the following Steps 3.1 and 3.2 until all chosen parents from* SP_t *are mated:*

3.1. *Choose two parents* p_1 *and* p_2 *from* SP_t. *Mate* p_1 *and* p_2 *using Procedure 4.1 to reproduce children* c_1 *and* c_2.

3.2. *Update the children pool* SC_t *by* $SC_t := SC_t \cup \{c_1, c_2\}$ *and update* SP_t *by* $SP_t := SP_t \setminus \{p_1, p_2\}$.

4. Mutation. *Associate a random number from* $(0, 1)$ *with each gene in each individual in* P_t'. *Let* num_m *be the number of genes whose associated number is less than* π_m, *and let* num_{zeros} *be the number of zero elements in GM. If* $num_m \geq num_{zeros}$, *then set* $num_m := num_{zeros}$. *Mutate* num_m *individuals among those which have an associated number less than* π_m *by applying Procedure 4.2 with* GM^r. *Add the mutated individual to the children pool* SC_t. *Update all the GMs with SR by applying Procedure 3.1 on the rotated subspaces successively NR times.*

5. *If* GM^r *has reached the completion ratio CP, go to Step 8. Otherwise go to Step 6.*

6. Survivor Selection. *Evaluate the fitness function for all generated children* SC_t, *and choose the* μ *best individuals in* $P_t \cup SC_t$ *for the next generation* P_{t+1}.

7. Mutagenesis. *Apply Procedures 2.1 and 2.2 with* GM^r *to alter the* N_w *worst individuals in* P_{t+1}, *set* $t := t + 1$, *update all the GMs with SR by applying Procedure 3.1 on the rotated subspaces successively NR times, and go to Step 2.*

8. *If* $r = NR$, *then go to Step 9. Otherwise, let* $r := r + 1$ *and go to Step 5.*

9. Intensification. *If IL is composed of the current era, then go to Step 9.1. Otherwise go to Step 9.2.*

9.1. *Apply a local search method starting from the best solution obtained in the previous search stage using all variables, and go to Step 9.2.*

9.2. *Let x^{elite} be the best solution obtained so far. If $CL \neq \emptyset$, go to Step 1.2. Otherwise, return with x^{elite}.*

5 Numerical Results

Numerical experiments were carried out to evaluate the performance of GATR. Before presenting the results, we describe the methodology adopted to conduct the numerical study of GATR. The automatic termination of GATR was compared against G3AT (Hedar et al., 2007), a Real-Coded Memetic Algorithm (Lozano et al., 2005), the Evolution Strategy with Covariance Matrix Adaptation method (Hansen et al., 2005a), the Non-Revisiting Genetic Algorithm with Parameter-less Adaptive Mutation (Yuen and Chow, 2009), a Differential Evolution method using an adaptive local search (DEahc-SPX) (Noman and Iba, 2008) and the Restart CMA Evolution Strategy With Increasing Population Size (Hansen et al., 2005b), for which the experimental results are available in the literature.

5.1 Methodology

The numerical study performed in this work is based on a test bed of 25 standard test functions from the special session on real-parameter optimization in the IEEE Congress on Evolutionary Computations, CEC 2005 (Suganthan et al., 2005). The set was carefully built based on classic benchmark functions in an attempt to cover a diverse set of problem properties. The first 5 functions are unimodal functions while the 20 other functions are multimodal, with functions f_{13} to f_{25} being hybrid composition functions. The 25 test problems are listed below and the mathematical form can be found in (Suganthan et al., 2005).

- f_1: Shifted Sphere Function

- f_2: Shifted Schwefels Problem 1.2

- f_3: Shifted Rotated High Conditioned Elliptic Function

- f_4: Shifted Schwefels Problem 1.2 with Noise in Fitness

- f_5: Schwefels Problem 2.6 with Global Optimum on Bounds

- f_6: Shifted Rosenbrocks Function

- f_7: Shifted Rotated Griewanks Function without Bounds

- f_8: Shifted Rotated Ackleys Function with Global Optimum on Bounds

- f_9: Shifted Rastrigins Function

- f_{10}: Shifted Rotated Rastrigins Function

- f_{11}: Shifted Rotated Weierstrass Function

- f_{12}: Schwefels Problem 2.13

- f_{13}: Expanded Extended Griewanks plus Rosenbrocks Function

- f_{14}: Shifted Rotated Expanded Scaffers f_6

- f_{15}: Hybrid Composition Function

- f_{16}: Rotated Hybrid Composition Function

- f_{17}: Rotated Hybrid Composition Function with Noise in Fitness

- f_{18}: Rotated Hybrid Composition Function

- f_{19}: Rotated Hybrid Composition Function with a Narrow Basin for the Global Optimum

- f_{20}: Rotated Hybrid Composition Function with the Global Optimum on the Bounds

- f_{21}: Rotated Hybrid Composition Function

- f_{22}: Rotated Hybrid Composition Function with High Condition Number Matrix

- f_{23}: Non-Continuous Rotated Hybrid Composition Function

- f_{24}: Rotated Hybrid Composition Function

- f_{25}: Rotated Hybrid Composition Function without Bounds

Before proceeding to the description of the test settings and results, some remarks should be stated in order to emphasize the particularity of the proposed GATR method and thus to clarify the dissimilarities in the way we present and compare the data with other methods, which does not exactly follow the evaluation guideline suggested in CEC 2005. When conducting numerical experiments, the CEC 2005 guideline recommends the use of common evaluation criteria such as the initialization scheme, the size of problems, a common termination criterion, etc. The guideline in particular indicates that the search should be stopped when reaching a pre-specified maximum number of function evaluations ($10,000 \times n$) or if the error in the function value becomes lower than a given threshold (10^{-8}). Solution quality is measured through the function error value given by ($f(x) - f(x^*)$), where $f(x)$ represents the best function value obtained by the algorithm and $f(x^*)$ is the known exact global minimum value. Also, when evaluating the amount of function evaluations needed, the search is stopped as soon as a fixed accuracy level is achieved. It is clear that GATR cannot fully comply with those termination criteria, since the point here is to let the search terminate automatically. Consequently, the results of the numerical experiments are not reported exactly as suggested in CEC 2005, but comparison is still possible and realistic, as done in this section, where interesting findings are discussed.

GATR was programmed using MATLAB and the code for each test function was run 25 times in 10, 30 and 50 dimensions, in accordance with the guideline proposed in CEC 2005. Also, a run is considered successful when the fixed accuracy level is

Table 1: GATR Parameter Setting

Parameter	Definition	Value
μ	Population size	30
p_c	Crossover probability	0.6
p_m	Mutation probability	0.1
m	No. of GM columns	100
N_1	No. of individuals used by GM-*Mutagenesis*	2
N_2	No. of individuals used by BCI-*Mutagenesis*	2
NR	Number of space rotations	3
α	Space rotation angle	45°
CP	GM Completion ratio	90%

achieved within the pre-specified maximum number of function evaluations (before 100,000 Fevals in 10 dimensions, 300,000 Fevals in 30 dimensions and 500,000 Fevals in 50 dimensions). The success rate (SRate) is defined as the number of successful runs divided by the total number of runs.

In this study, the Wilcoxon matched-pairs signed-ranks non-parametric test is used for directly comparing the results of two methods. This test, as described in (García et al., 2009), does not assume that the data are sampled from a normal distribution. It assumes however that the data are symmetrically distributed around the median. We employ the Wilcoxon non-parametric test because, for the methods that follow the framework suggested by the CEC 2005, the authors report their achieved average results using the same conditions for each algorithm and test problem. In particular, we consider in this work the function error values obtained by each method.

5.2   Initial Parameters

Table 1 summarizes all GATR parameters with their assigned values. These values are based on the common setting in the literature or determined through preliminary numerical experiments presented in the Appendix.

The initial population P_0 is generated using the scatter search diversification generation method (Laguna and Marti, 2003; Hedar and Fukushima, 2006a). In this method, the interval $[l_i, u_i]$ of each variable is divided into four sub-intervals of equal size. New points are generated and added to P_0 as follows:

1. Choose one sub-interval for each variable randomly with a probability inversely proportional to the number of solutions previously chosen in this sub-interval.

2. Choose a random value for each variable that lies in the corresponding selected sub-interval.

All other parameters receive the default values as specified in Table 1. The construction by default of the final intensification process is stated in Subsection 4.2. The effects of various parameters introduced or influenced by SD and SR are also discussed in the Appendix. We examine the effect of the choice of the population size μ, the number of GM subranges m, the combination list CL, as well as the impact of the SD and

17

Table 2: Wilcoxon's test for GATR against G3AT (at level 0.05)

n	GATR (R–)	G3AT (R+)	Significant Method
10	147	178	–
30	236	89	GATR
50	261	64	GATR

SR through the number of space rotations *NR* in Appendices A, B, C, D and E, respectively. The study reveals that the population size μ affects the convergence speed of the search, with larger value of μ leading to faster convergence. However it is often harmful in terms of the success rate. The study also shows that the effect of the number of GM columns *m* is problem dependent, with larger value of *m* leading to more function evaluations. The Combination List *CL* should also preferably not be constructed randomly, since forcing each variable to participate only once during the series of eras for instance can lead to better results. Direct effects of SD and SR are also investigated. Using SD required additional Fevals. However, its introduction allowed the search to solve successfully more test functions. SR also demonstrated beneficial results in terms of SRates, with moderate additional costs in Feval.

5.3 Comparison with G3AT

With the intention of improving the G3AT method, the SD and SR mechanisms have been developed and implemented within the GATR method. The aim of this section is to study the impact of those new mechanisms within GATR against G3AT by comparing their results.

We first try to determine whether there is any significant difference between both methods using the Wilcoxon non-parametric test over the 25 benchmark functions of the CEC 2005. Table 2 summarizes the results of the test in 10, 30 and 50 dimensions. The signed ranked statistics are reported (R– for GATR and R+ for the compared method, here G3AT) along with the indication of the significantly superior method at significance level 0.05 (for this test bed, the Wilcoxon critical value is equal to 89). The highest values of the test statistics represent the best results.

From Table 2, we note that in dimension 10, GATR and G3AT are not significantly different. In higher dimensions however, for dimensions 30 and 50, GATR significantly outperforms G3AT, the gap getting bigger as the dimension increases. The introduction of the SD and SR mechanisms thus seems particularly effective in higher dimensions.

Since both G3AT and GATR are under automatic termination, it is possible to empirically compare their results using the framework proposed in CEC 2005 and get a better insight of the performance of GATR on a *function-level* rather than on a *method-level*. We support our findings by the paired Student's t-test (Montgomery and Runger, 2003). Tables 3, 4, Tables 5, 6 and Tables 7, 8 report the results of G3AT and GATR, respectively in 10, 30 and 50 dimensions using the same test bed as for the Wilcoxon's test. The Feval, as reported in Table 3, indicates that GATR is on average around 1.4 times more expensive in terms of function evaluations in 10 dimensions. In 30 and 50 dimensions (Tables 5 and 7), the computational cost in Feval approximately doubles

Table 3: Solution Costs for GATR and G3AT with $n = 10$

	Solution Costs			
	GATR		G3AT	
f	Feval Mean	SD	Feval Mean	SD
f_1	6.24e+03	1.36e+02	4.22e+03	4.17e+01
f_2	6.31e+03	1.29e+02	4.28e+03	7.49e+01
f_3	1.26e+04	1.51e+03	7.87e+03	1.30e+03
f_4	4.63e+03	1.54e+02	3.60e+03	6.08e+01
f_5	5.72e+03	1.10e+02	3.91e+03	1.04e+02
f_6	1.17e+04	2.02e+03	9.05e+03	2.06e+03
f_7	6.21e+03	1.70e+02	4.18e+03	4.42e+01
f_8	4.01e+03	1.47e+02	3.52e+03	5.42e+01
f_9	5.41e+03	2.49e+02	4.14e+03	1.34e+02
f_{10}	5.61e+03	3.23e+02	4.16e+03	1.80e+02
f_{11}	4.51e+03	1.77e+02	3.55e+03	5.50e+01
f_{12}	6.39e+03	2.19e+02	4.30e+03	9.78e+01
f_{13}	6.01e+03	2.87e+02	4.28e+03	1.74e+02
f_{14}	6.13e+03	6.59e+02	4.67e+03	4.93e+02
f_{15}	5.12e+03	1.82e+02	3.85e+03	8.56e+01
f_{16}	5.34e+03	1.74e+02	3.87e+03	8.39e+01
f_{17}	5.02e+03	1.48e+02	3.64e+03	3.73e+01
f_{18}	6.12e+03	3.66e+02	4.10e+03	1.56e+02
f_{19}	6.25e+03	3.80e+02	4.24e+03	1.29e+02
f_{20}	6.26e+03	4.07e+02	4.23e+03	3.04e+02
f_{21}	6.94e+03	6.00e+02	4.10e+03	4.80e+02
f_{22}	5.15e+03	1.77e+02	3.77e+03	8.03e+01
f_{23}	5.24e+03	1.53e+02	3.72e+03	1.33e+02
f_{24}	5.27e+03	1.64e+02	4.65e+03	4.45e+02
f_{25}	4.83e+03	1.61e+02	3.67e+03	5.08e+01

on average. The introduction of SD and SR mechanisms is however not the principal cause of the raise of Feval. It is the increase of frequency of the local searches during the intensification phase that has a more predominant weight in the total Feval cost of GATR.

In 10 dimensions, as reported in Table 4, t-tests indicate that GATR gives significantly better solutions than G3AT in terms of solution quality on 9 functions, while G3AT is significantly better on 5 functions. The gap gets bigger in 30 and 50 dimensions (Tables 6 and 8). In 30 dimensions, GATR performs significantly better on 10 functions while G3AT performs better on 3 functions. In 50 dimensions, out of the 25 problems, G3AT achieves significantly better results for only 3 problems, while GATR outperforms G3AT in 14 problems. Those improved results, especially in higher dimensions, demonstrate the efficiency of the SD and SR mechanisms, especially when the problem complexity increases.

5.4 Comparison with a Real-Coded Memetic Algorithm

Performance of GATR is now compared against a Real-Coded Memetic Algorithm (RCMA) introduced in (Lozano et al., 2005). As Memetic Algorithms (MA) are GAs combined with a local search (LS) process to refine individuals (Moscato, 1999), RCMA is a Real-Coded GA combined with LS techniques. *Real-Coded* means that the structure of an individual is based on the real number representation (in opposition with the binary coding for example), as it is the case for GATR. This representation seems particularly natural when dealing with variables in a continuous domain. In (Lozano

Table 4: Solution Qualities for GATR and G3AT with $n = 10$

	Solution Qualities				t-test
	GATR		G3AT		Significant Method
f	Error Mean	SD	Error Mean	SD	at level 0.05
f_1	6.84e–13	3.09e–13	6.21e-13	3.10e-13	–
f_2	2.54e–12	2.58e–12	2.23e-12	2.47e-12	–
f_3	4.73e–12	5.31e–12	1.20e-03	4.28e-03	–
f_4	1.02e+04	2.28e+03	1.54e+03	1.49e+03	G3AT
f_5	2.28e+02	1.53e+02	1.38e+02	1.62e+02	–
f_6	3.19e–01	1.10e+00	1.50e+00	2.71e+00	GATR
f_7	9.72e–02	1.01e–01	2.25e-01	6.24e-02	GATR
f_8	2.00e+01	2.74e–04	2.00e+01	1.97e-04	–
f_9	3.98e–13	4.70e–01	2.83e+00	1.21e+00	GATR
f_{10}	4.63e+01	2.15e+01	1.83e+01	7.48e+00	G3AT
f_{11}	1.12e+01	1.50e+00	7.59e+00	1.01e+00	G3AT
f_{12}	9.48e–12	2.79e–11	3.09e+02	5.83e+02	GATR
f_{13}	4.92e–01	1.62e–01	7.34e-01	2.72e-01	GATR
f_{14}	4.01e+00	1.72e–01	3.94e+00	2.55e-01	–
f_{15}	3.79e+01	3.35e+01	1.48e+02	1.28e+02	GATR
f_{16}	1.46e+02	2.83e+01	1.34e+02	1.62e+01	–
f_{17}	2.10e+02	3.78e+01	1.89e+02	3.72e+01	–
f_{18}	9.00e+02	2.02e–06	8.71e+02	7.38e+01	–
f_{19}	9.00e+02	1.72e–09	8.84e+02	3.73e+01	G3AT
f_{20}	9.00e+02	4.00e–04	8.43e+02	7.06e+01	G3AT
f_{21}	4.78e+02	1.68e+02	8.86e+02	3.08e+02	GATR
f_{22}	8.38e+02	5.38e+01	8.13e+02	4.00e+01	–
f_{23}	7.86e+02	2.93e+02	9.99e+02	2.72e+02	GATR
f_{24}	2.91e+02	2.64e+01	2.48e+02	1.14e+02	–
f_{25}	4.37e+02	1.52e+02	6.54e+02	2.79e+02	GATR

Table 5: Solution Costs for GATR and G3AT with $n = 30$

	Solution Costs			
	GATR		G3AT	
f	Feval Mean	SD	Feval Mean	SD
f_1	3.05e+04	2.72e+02	1.41e+04	1.04e+02
f_2	1.41e+05	1.15e+04	4.08e+04	8.02e+03
f_3	1.50e+05	5.10e+03	3.40e+04	2.63e+03
f_4	1.46e+04	2.43e+02	1.16e+04	8.78e+01
f_5	1.99e+04	5.65e+02	1.30e+04	4.09e+02
f_6	2.03e+05	1.62e+04	4.66e+04	1.58e+03
f_7	3.41e+04	2.46e+03	1.44e+04	1.53e+03
f_8	1.30e+04	1.91e+02	1.16e+04	1.04e+02
f_9	1.57e+04	3.27e+02	1.19e+04	2.03e+02
f_{10}	1.79e+04	4.41e+02	1.21e+04	2.71e+02
f_{11}	1.47e+04	3.58e+02	1.16e+04	7.25e+01
f_{12}	3.16e+04	4.51e+03	1.39e+04	1.67e+03
f_{13}	1.67e+05	2.74e+04	4.66e+04	4.07e+03
f_{14}	3.89e+04	3.74e+03	1.56e+04	1.45e+03
f_{15}	1.60e+04	2.42e+02	1.21e+04	3.57e+02
f_{16}	1.89e+04	8.02e+02	1.24e+04	2.47e+02
f_{17}	1.66e+04	3.35e+02	1.18e+04	8.93e+01
f_{18}	2.25e+04	1.60e+03	1.22e+04	1.02e+02
f_{19}	2.41e+04	1.07e+03	1.32e+04	1.01e+02
f_{20}	2.24e+04	1.55e+03	1.22e+04	1.11e+02
f_{21}	2.16e+04	3.48e+02	1.45e+04	1.14e+03
f_{22}	1.99e+04	7.47e+02	1.22e+04	3.82e+02
f_{23}	2.00e+04	6.19e+02	1.24e+04	2.22e+02
f_{24}	1.99e+04	4.02e+02	1.25e+04	9.69e+02
f_{25}	1.80e+04	4.57e+02	1.23e+04	2.23e+02

Table 6: Solution Qualities for GATR and G3AT with $n = 30$

	Solution Qualities				t-test
	GATR		G3AT		Significant Method
f	Error Mean	SD	Error Mean	SD	at level 0.05
f_1	1.12e-12	1.28e-12	2.01e-12	2.32e-12	–
f_2	1.45e-09	2.01e-09	4.32e-06	2.03e-05	–
f_3	2.11e+02	9.81e+01	6.89e+04	3.91e+04	GATR
f_4	1.03e+05	1.09e+04	4.27e+04	1.41e+04	G3AT
f_5	2.29e+03	5.08e+02	5.79e+03	1.45e+03	GATR
f_6	5.91e-01	1.87e+00	9.57e+01	1.16e+02	GATR
f_7	3.22e-04	1.48e-03	1.12e-02	6.22e-03	GATR
f_8	2.00e+01	1.33e-02	2.00e+01	1.34e-02	–
f_9	2.61e-12	1.15e+00	2.17e+01	6.37e+00	GATR
f_{10}	5.53e+01	1.27e+01	1.20e+02	3.27e+01	GATR
f_{11}	3.10e+01	3.09e+00	2.98e+01	2.08e+00	–
f_{12}	3.94e+01	7.94e-05	2.06e+03	5.89e+03	–
f_{13}	1.73e+00	2.72e-01	4.66e+00	1.09e+00	GATR
f_{14}	1.32e+01	3.12e-01	1.30e+01	4.34e-01	–
f_{15}	3.67e+02	2.33e+01	3.88e+02	1.33e+02	–
f_{16}	7.47e+01	1.26e+01	1.83e+02	1.31e+02	GATR
f_{17}	2.77e+02	8.07e+01	3.31e+02	1.12e+02	–
f_{18}	9.00e+02	1.21e-02	9.00e+02	0.00e+00	–
f_{19}	9.00e+02	4.56e-04	9.00e+02	0.00e+00	–
f_{20}	9.00e+02	1.22e-03	9.00e+02	0.00e+00	G3AT
f_{21}	8.82e+02	2.32e+00	5.70e+02	1.45e+02	G3AT
f_{22}	5.30e+02	9.72e+00	7.92e+02	1.85e+02	GATR
f_{23}	7.40e+02	5.01e+00	6.77e+02	1.64e+02	–
f_{24}	2.24e+02	5.24e+00	2.96e+02	2.05e+02	–
f_{25}	2.22e+02	3.63e+00	2.98e+02	8.32e+01	GATR

Table 7: Solution Costs for GATR and G3AT with $n = 50$

	Solution Costs			
	GATR		G3AT	
f	Feval Mean	SD	Feval Mean	SD
f_1	7.347e+04	4.647e+02	2.836e+04	2.865e+02
f_2	5.258e+05	2.172e+04	7.841e+04	3.883e+03
f_3	5.535e+05	1.718e+04	7.888e+04	2.972e+03
f_4	2.571e+04	3.785e+02	2.357e+04	1.580e+02
f_5	3.721e+04	9.506e+02	2.562e+04	7.376e+02
f_6	4.437e+05	4.577e+04	7.988e+04	2.392e+02
f_7	4.537e+04	5.745e+03	2.657e+04	2.230e+03
f_8	2.245e+04	2.907e+02	2.346e+04	1.389e+02
f_9	3.066e+04	7.822e+02	2.365e+04	2.283e+02
f_{10}	3.135e+04	9.239e+02	2.402e+04	2.220e+02
f_{11}	2.634e+04	5.122e+02	2.345e+04	1.970e+02
f_{12}	3.121e+04	6.408e+02	2.367e+04	3.715e+02
f_{13}	1.686e+05	8.575e+04	7.910e+04	4.445e+03
f_{14}	8.994e+04	9.003e+03	2.927e+04	3.223e+03
f_{15}	2.842e+04	4.462e+02	2.440e+04	7.433e+02
f_{16}	3.206e+04	9.042e+02	2.472e+04	6.562e+02
f_{17}	2.916e+04	3.989e+02	2.379e+04	1.452e+02
f_{18}	4.674e+04	2.036e+03	2.492e+04	1.630e+02
f_{19}	5.067e+04	3.847e+03	2.511e+04	1.934e+02
f_{20}	4.854e+04	1.686e+03	2.578e+04	1.316e+02
f_{21}	4.417e+04	6.322e+02	2.527e+04	5.034e+02
f_{22}	5.154e+04	2.964e+03	2.847e+04	1.602e+03
f_{23}	4.041e+04	3.729e+02	2.497e+04	3.049e+02
f_{24}	3.499e+04	6.674e+02	2.410e+04	2.928e+02
f_{25}	3.195e+04	6.500e+02	2.415e+04	2.755e+02

Table 8: Solution Qualities for GATR and G3AT with $n = 50$

	Solution Qualities				t-test
	GATR		G3AT		Significant Method
f	Error Mean	SD	Error Mean	SD	at level 0.05
f_1	6.539e-12	2.537e-12	9.779e-12	1.003e-11	–
f_2	4.676e-08	7.812e-08	3.844e-03	1.287e-02	–
f_3	6.790e+02	3.070e+02	1.395e+05	6.765e+04	GATR
f_4	5.066e+05	6.124e+04	1.227e+05	2.745e+04	G3AT
f_5	5.473e+03	9.523e+02	1.614e+04	3.313e+03	GATR
f_6	2.718e+00	5.216e+00	4.677e+02	1.026e+03	GATR
f_7	1.102e-03	1.665e-03	8.689e-03	9.239e-03	GATR
f_8	2.000e+01	3.750e-03	2.001e+01	8.574e-03	GATR
f_9	4.975e+00	1.459e+00	7.053e+01	1.372e+01	GATR
f_{10}	1.830e+02	2.924e+01	3.161e+02	6.143e+01	GATR
f_{11}	7.572e+01	5.855e+00	5.704e+01	3.903e+00	G3AT
f_{12}	3.535e+03	1.335e-03	9.016e+03	8.420e+03	GATR
f_{13}	3.092e+00	2.901e-01	1.813e+01	6.218e+00	GATR
f_{14}	2.257e+01	5.229e-01	2.254e+01	5.024e-01	–
f_{15}	2.726e+02	1.501e+01	4.470e+02	5.372e+01	GATR
f_{16}	8.559e+01	1.816e+01	2.147e+02	1.232e+02	GATR
f_{17}	3.528e+02	7.024e+01	4.219e+02	7.820e+01	GATR
f_{18}	9.000e+02	8.154e-03	9.000e+02	0.000e+00	–
f_{19}	9.000e+02	1.291e-03	9.000e+02	0.000e+00	G3AT
f_{20}	9.000e+02	2.662e-03	9.000e+02	0.000e+00	–
f_{21}	7.439e+02	3.567e+00	7.446e+02	2.229e+01	–
f_{22}	5.007e+02	5.984e-01	5.144e+02	6.692e+01	–
f_{23}	7.529e+02	2.521e+00	7.522e+02	3.267e+01	–
f_{24}	3.189e+02	2.587e+01	8.712e+02	3.212e+02	GATR
f_{25}	3.230e+02	2.167e+01	8.259e+02	3.339e+02	GATR

Table 9: Wilcoxon's test for GATR against RCMA (at level 0.05)

n	GATR ($R-$)	RCMA ($R+$)	Significant Method
10	131	194	–
30	162	163	–

Table 10: Wilcoxon's test for GATR against L-CMA-ES (at level 0.05)

n	GATR (R–)	L-CMA-ES (R+)	Significant Method
10	154	171	–
30	191	134	–
50	199	126	–

et al., 2005), the authors propose an RCMA that uses an adaptive LS process with a high diversity global exploration. The LS process depends on the individual fitness, which is used to compute a probability to decide whether LS is applied or not and also to determine the LS intensity.

Numerical experiments described in (Lozano et al., 2005) follow the guideline of CEC 2005 (consequently it should be reminded that the required optimal number of function evaluations is defined under some specific termination conditions, i.e., when the maximum number of Feval is reached or when the optimal accuracy is obtained).

Results of the Wilcoxon's test in 10 and 30 dimensions are presented in Table 9 (RCMA was not tested in 50 dimensions by (Lozano et al., 2005)). It reveals that the two methods are not significantly different in any dimension, although RCMA seems to have a slight advantage in lower dimensions. Let us note however that in 10 dimensions, the precision obtained by RCMA was mostly achieved after 10^5 function evaluations, while GATR decided to stop before 10^4 function evaluations. In 30 dimensions, similar performance was obtained after around 10 times less function evaluations for GATR, thus showing the effectiveness of the automatic termination of GATR.

5.5 Comparison with CMA-ES

In this subsection, GATR is compared against the Evolution Strategy with Covariance Matrix Adaptation (CMA-ES) method (Hansen and Kern, 2004; Hansen, 2006), which is a state-of-the-art method recommended by experts for adaptive mutation (Lobo et al., 2007). CMA-ES has been designed to improve the local search performance of ES (Hansen and Ostermeier, 1996). An important property of CMA-ES is its invariance under linear transformations of the search space. The CMA-ES version considered for our comparisons also implements the so-called rank-μ-update, abbreviated as L-CMA-ES (Hansen et al., 2005a) which reduces significantly the needed number of generations to reach a certain function value (Hansen et al., 2003).

Results of the Wilcoxon's statistical test are shown in Table 10. In 10, 30 and 50 dimensions, the test does not reveal any statistically significant difference at level 0.05. Nevertheless, we may notice that L-CMA-ES shows slightly higher quality solution in 10 dimensions, but the situation is opposite in 30 and 50 dimensions. Thus, we may conclude that the SR and SD mechanisms seem to enhance the performance of GATR particularly in higher dimensions.

5.6 Comparison with State-of-the-Art GA

In this section, GATR is compared against a recent GA that is equipped with a memory and designed in such a way that positions that have already been searched are

never revisited. This version of GA is named Non-Revisiting Genetic Algorithm with Parameter-less Adaptive Mutation (NrGA) (Yuen and Chow, 2009) and makes use of a binary space partitioning (BSP) tree which is dynamically constructed and designed to reflect the evolution history of the search by remembering the positions already explored and avoid the reevaluation of the fitness of those positions. The BSP search history is also employed to guide the search towards unvisited positions through a new adaptive mutation mechanism (Yuen and Chow, 2009).

In order to compare GATR with NrGA, we use the numerical results reported in (Yuen and Chow, 2009) for NrGA and the paired Student's t-test. The authors of the NrGA method (Yuen and Chow, 2009) tested the performance of their algorithm by using a set of 19 well-known benchmark functions. Among those 19 functions, five are taken from the CEC 2005 contest. They were however slightly modified by removing the shift of the global optima positions. For this reason, although the difference is minor, the numerical results reported here for GATR might be slightly different from those reported elsewhere in this paper.

The functions used for the comparison are the following (Suganthan et al., 2005):

- Rotated high conditioned elliptic function (f_3)

- Rotated Griewanks function (f_7)

- Rotated Rastrigins function (f_9)

- Rotated Weierstrasss function (f_{11})

- Hybrid Composition function (f_{15})

Simulation settings for G3AT have been slightly adjusted in order to correspond with NrgA's and allow a fair competition. In NrGA, the search has been set to stop after exactly 40100 of function evaluations in 10 and 30 dimensions. GATR stopped according to its own automatic termination criteria. Except for function f_3 in 30 dimensions, the Fevals for all functions in 10 and 30 dimensions were less than 40100 for GATR. Function f_3 in 30 dimensions required more Fevals and therefore has been artificially stopped when Feval reached 40100.

The results are reported in Tables 11 and 12, where NrGA's data for those five common benchmark functions in 10 and 30 dimensions are taken from (Yuen and Chow, 2009). It is worthwhile to note that, except for function f_3 in 30 dimensions, GATR stopped automatically with notably less Fevals than NrGA. Comparisons based on 100 independent runs for each test function indicate that GATR performs quite well, compared to NrGA. In particular, it significantly outperformed NrGA on 3 out of 5 functions, namely functions f_3, f_9 and f_{15}.

5.7   Comparison with State-of-the-Art MA

Here we confront GATR with one of the current best MA for continuous optimization: the DEahcSPX method (Noman and Iba, 2008). It combines differential evolution (DE) with a crossover-based local search (XLS) that adaptively determines the length of the LS using a hill-climbing heuristic and feedback from the search. The crossover

Table 11: Comparison of GATR and NrGA on the Best Fitness Values Found and Feval in 10 and 30 Dimensions: f_3, f_7 and f_9

Method		f_3		f_7		f_9	
		$n = 10$	$n = 30$	$n = 10$	$n = 30$	$n = 10$	$n = 30$
NrGA	mean	2.56e+06	1.15e+08	0.00e+00	0.00e+00	3.98e+00	2.89e+01
	std. dev.	1.50e+06	5.51e+07	0.00e+00	0.00e+00	6.49e+00	1.36e+01
	Feval	40100	40100	40100	40100	40100	40100
GATR	mean	2.23e-12	2.31e+03	9.72e-02	2.88e-04	1.59e-01	1.89e+00
	std. dev.	4.13e-12	4.90e+02	1.01e-01	5.63e-04	4.70e-01	1.75e+00
	Feval	12620.6	39448.5	6208.6	34123.7	5410.1	15657.3
Significant Method (at level 0.05)		GATR	GATR	NrGA	NrGA	GATR	GATR

Table 12: Comparison of GATR and NrGA on the Best Fitness Values Found and Feval in 10 and 30 Dimensions: f_{11} and f_{15}

Method		f_{11}		f_{15}	
		$n = 10$	$n = 30$	$n = 10$	$n = 30$
NrGA	mean	9.00e-02	5.40e-01	3.83e+03	4.41e+03
	std. dev.	8.00e-02	4.10e-01	6.34e+02	2.79e+02
	Feval	40100	40100	40100	40100
GATR	mean	1.07e+01	4.25e+01	3.79e+01	4.09e+02
	std. dev.	1.57e+00	3.09e+00	3.35e+01	1.73e+01
	Feval	4513.6	14725.7	5116	15959.2
Significant Method (at level 0.05)		NrGA	NrGA	GATR	GATR

Table 13: Wilcoxon's test for GATR against DEahcSPX (at level 0.05)

n	GATR ($R-$)	DEahcSPX ($R+$)	Significant Method
10	95	115	–
30	109	101	–
50	127	83	–

25

Table 14: Wilcoxon's test for GATR against G-CMA-ES (at level 0.05)

n	GATR ($R-$)	G-CMA-ES ($R+$)	Significant Method
10	61	264	G-CMA-ES
30	131	194	–
50	227	98	GATR

operator in DEahcSPX is based on the simplex multi-parent crossover, described in (Tsutsui et al., 1999). According to the authors, DEahcSPX is superior or at least comparable to other well-known MAs.

The DE algorithm has been proposed by Storn and Price (Storn and Price, 1997). It is one of the most recent EAs and is known to be a relatively simple but powerful population-based stochastic search technique. DE uses a limited number of parameters but they may greatly influence the performance of the algorithm. In particular, we note the population size, the scaling factor and the crossover rate.

Numerical results of DEahcSPX for the CEC 2005 test suite functions f_6 to f_{25} are directly available in the literature (Noman and Iba, 2008). The performance comparison with GATR was carried out by means of the Wilcoxon's test. The results are reported in Table 13. We can see that in all dimensions, GATR achieves similar performance to DEahcSPX, the differences being not statistically significant. We can however point out a slight tendency of GATR to obtain better solution quality compared to DEahcSPX as the dimension increases. Specifically, in 10 dimensions, DEahcSPX is slightly superior, but it is outperformed by GATR in 50 dimensions.

5.8 Comparison with the Winner of the CEC 2005 Contest

G-CMA-ES (Hansen et al., 2005b) is the winner of the CEC 2005 competition (Langdon and Poli, 2007) and recognized as a very powerful algorithm for continuous optimization problems (Lunacek and Whitley, 2006; Langdon and Poli, 2007). G-CMA-ES is a restart CMA-ES that stops whenever some prespecified stopping conditions are met and repeats the search with the population size being doubled on each restart. Compared to the pure CMA-ES, G-CMA-ES is significantly superior in particular on multimodal functions.

The performance comparison was carried out by means of the Wilcoxon's test. The results in 10, 30 and 50 dimensions are reported in Table 14. We observe that G-CMA-ES clearly outperforms GATR in 10 dimensions. In 30 dimensions, G-CMA-ES still shows better results but the difference is not statistically significant. In 50 dimensions, it is now the turn of GATR to exhibit significantly superior performance.

These results allow us to conclude that the SD and SR mechanisms improve the search process of GATR particularly in higher dimensions. Also, the use of the automatic termination did not have negative effect on the quality of the solutions achieved by GATR, even outperforming G-CMA-ES in higher dimensions.

6   Conclusion

In this paper, we have introduced a new EA, called GATR, that can terminate the search without conventional predefined criteria such as the maximum number of generations or function evaluations. It is based on the GM and implements new strategies equipped with the SD and SR mechanisms. The SD and SR mechanisms provide an innovative way of handling problems by creating a two-dimensional environment in which the GMs evolve irrespective of the dimension of the original problem. In this environment, the GM goes through a series of rotations which allow the search to avoid premature convergence and termination.

Numerical experiments were carried out on a set of 25 test functions presented at the CEC 2005 in 10, 30 and 50 dimensions. GATR was compared against G3AT, RCMA, a version of CMA-ES, the state-of-the-art NrGA and DEahcSPX as well as the CEC 2005 competition winner G-CMA-ES. The results indicate that GATR is competitive with state-of-the-art EAs, especially in higher dimensions where the performance of GATR is comparable or superior to many of the well-known EAs. The competitive overall performance of GATR demonstrates that the quality of the solutions obtained did not suffer from premature termination. The proposed automatic termination thus enabled us to stop the search without undue objective function evaluations and without negative impact on the quality of the solution obtained.

We believe that GATR, with its SD and SR, raises an interesting feature that deserves further investigations. There is also a scope for future work that addresses the potential of GM, SD and SR mechanisms for other EAs.

Appendix

A   Effect of the Population Size

Here we discuss the effect of μ on the convergence speed and success rate for a sample of functions in 10 and 30 dimensions. The sample consists of functions f_2, f_6 and f_9 from the CEC 2005, which are modified versions of the Schwefel, Rosenbrock and Rastigrin functions, respectively. The success rate, abbreviated as SRate, and Feval required before termination are plotted over different values of μ ranging from 10 to 100 with a step of 10.

It is observed in Figure 7(a) that for function f_2 in 10 dimensions, the convergence speed decreases with increasing μ. By inspecting the curve displayed in Figure 7(b) for function f_6, it can be presumed that an optimum value of μ in terms of convergence speed may lie at around $\mu = 60$ for that function. Its SRate behaves similarly though and shows lower performance for $\mu = 50$ and $\mu = 60$. In Figure 7(c), it can be observed that the SRate and convergence curves have similar behaviors: both SRate and Feval are inversely proportional to μ. For the same functions in 30 dimensions, Figures 8(a)-(c) show that the trendlines are to some extent similar to what can be presumed in 10 dimensions. Function f_6 in Figure 8(b) however demonstrates no correlations between μ and the evolution of SRate and Feval. Moreover the SRate does not seem to be affected by the number of function evaluations in this case.

As a general observation, increasing the population size makes the algorithm converge faster. On the other hand, it may be harmful to the performance in terms of SRate. This behavior is expected because the GMs are easily filled in early stages of the search with large population, but exploration operators such as crossover or mutagenesis do not have enough room to inject
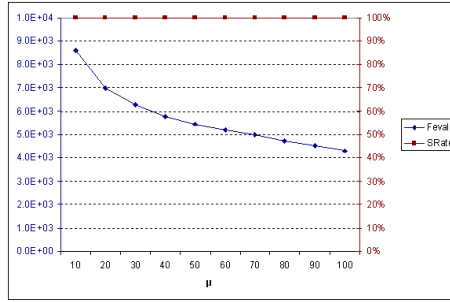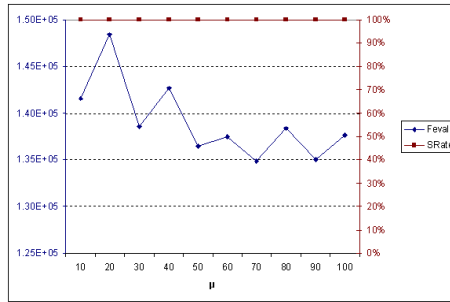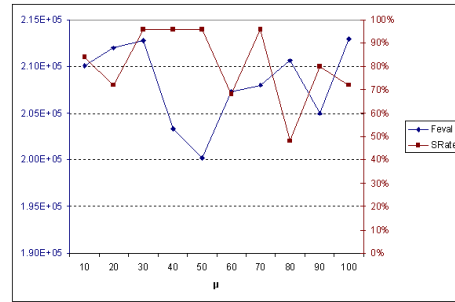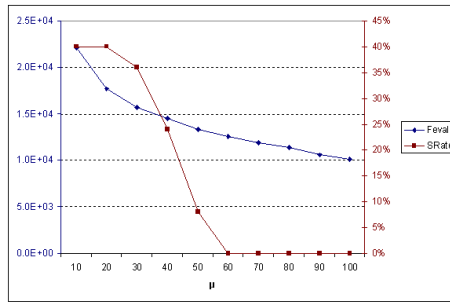
(a) f_2
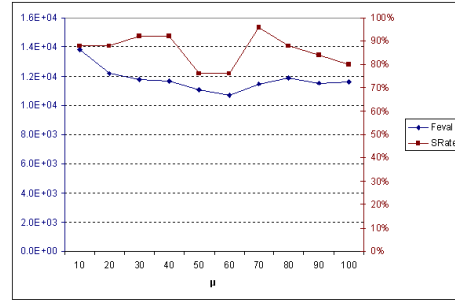
(b) f_6
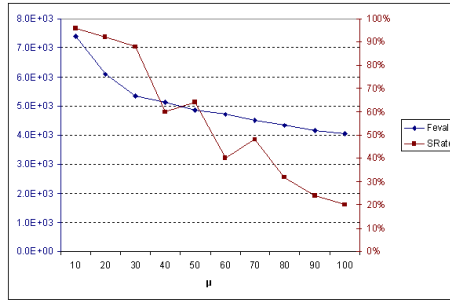
(c) f_9

Figure 7: Functions f_2, f_6 and f_9. SRate and Feval for different μ values in 10 dimensions. $\Diamond$ is for the Feval. $\Box$ is for the SRate.

(a) f_2

(b) f_6

(c) f_9

Figure 8: Functions f_2, f_6 and f_9. SRate and Feval for different μ values in 30 dimensions. $\Diamond$ is for the Feval. $\square$ is for the SRate.

diversity within the population. Consequently, it has been decided to set μ equal to 30, in order to maintain enough generations during the search to promote diversity among the population.

B   Number of Subranges

In relation to the crossover or mutation probabilities, the number of subranges m, which is the number of columns of GM, is a parameter that also needs some tuning before getting the best results. The parameter m regulates the size of the sub-regions and its suitable value appears to be problem dependent. However the methodology adopted in this work advocates the use of a unique set of parameter values for all functions during the numerical experiments. Thus we present a study on the effect of m on the convergence speed and SRate for the same sample of functions presented in Appendix A, in 10 and 30 dimensions. By plotting the SRates and total number of function evaluations required before termination over different values of m, it is possible to investigate the effect of m and determine its suitable value if it exists.

Although the trendline slope of Feval might be dependent on the function, it is easily observed in Figure 9 and Figure 10 that Feval is directly proportional to m, for every function in 10 and 30 dimensions. SRate however has various behaviors and what is observed in 10 dimensions is not necessarily true in 30 dimensions. For instance, SRate for function f_9 in 10 dimensions depicted in Figure 9(c) shows a gradually increasing curve with m, reaching and keeping the SRate of 100% from $m = 175$ on. SRate for the same function in 30 dimensions exhibits the same trends for lower values of m, but tends to fluctuate as m increases, see Figure 10(c). Function f_6 is more regular but displays sharp drop-offs, with $m = 300$ or $m = 250$ in 10 and 30 dimensions, respectively.

As expected, within the range of tested problem sizes, m is very problem dependent. For our numerical experiments, the convergence speed has been given particular attention since the Feval with m varying from 50 to 300 can dramatically increase while SRate keeps fluctuating. In many cases, $m = 50$ does not demonstrate satisfactory results but a larger value $m = 100$ is acceptable. For this reason, it has been decided to set m equal to 100 in order to get sufficient convergence speed with moderate compromise on the SRate performance.

C   Combination List

The CL is a list that keeps track of the pairs of variables that are designated to participate during the eras. Let us consider the pair $\{x_a, x_b\}$. Obviously x_a and x_b should not be identical. But the same pair could as well be active in another era. Also depending on the problem, some variables in particular might have higher weights for the obtention of a good solution. Determining rules and studying which and how variables should be given more importance to optimize the performance are however out of the scope of this paper. What is discussed in this subsection is the empirical construction of the CL *by default* that was employed throughout our numerical experiments. We show how we have set the length of the CL (*CL_length*) (i.e., the number of eras), which has a direct impact on the convergence speed of the search.

We let *CL_length* range between 10 and 20 for the tests in 30 dimensions. In 10 dimensions, we let it range between 3 and 10. Two patterns were considered: a deterministic pattern (DCL) and a random pattern (RCL). For any n-dimensional problem, the DCL is constructed by choosing the pairs $\{x_{a_i}, x_{b_i}\}$ according to some predetermined rules. The construction of RCL is straightforward. Variables associated to x_{a_i} and x_{b_i} are selected randomly among $\{x_1, x_2, \ldots, x_n\}$, with the only condition that x_a should be different from x_b.
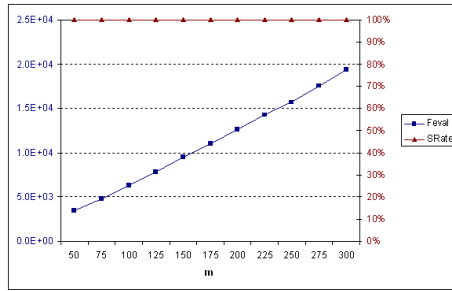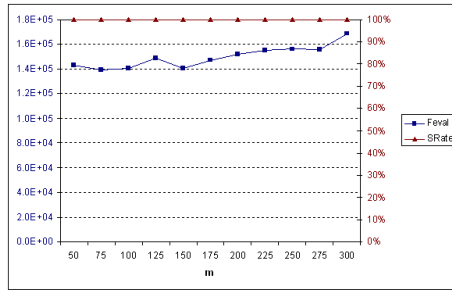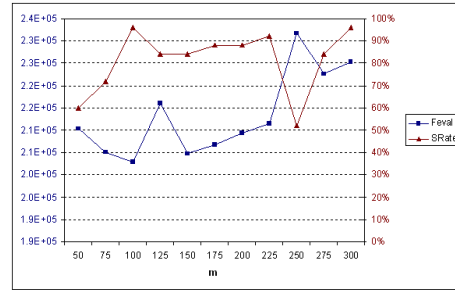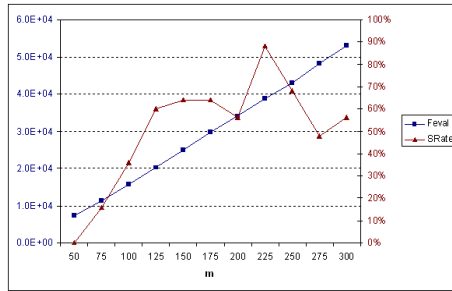
(a) f_2
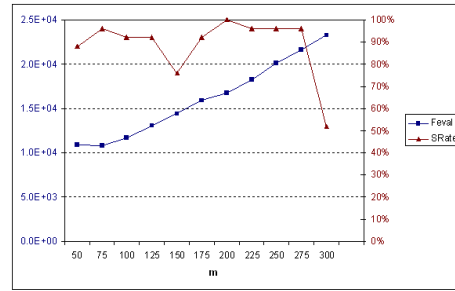
(b) f_6
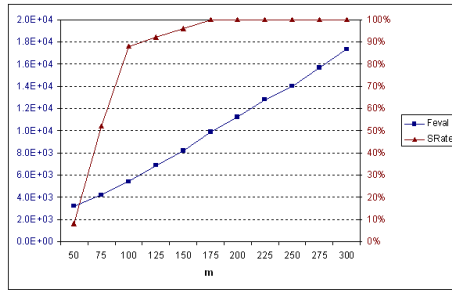
(c) f_9

Figure 9: Functions f_2, f_6 and f_9. SRate and Feval for different m values in 10 dimensions. $\triangle$ is for the Feval. $\square$ is for the SRate.

(a) f_2

(b) f_6

(c) f_9

Figure 10: Functions f_2, f_6 and f_9. SRate and Feval for different m values in 30 dimensions. $\triangle$ is for the Feval. $\square$ is for the SRate.

For RCL, the results are reported with $CL\_length = 15$ in 30 dimensions (referred to as "CL 15 rand") and $CL\_length = 5$ in 10 dimensions ("CL 5 rand") in Figure 12 and Figure 11, respectively. Figure 11(a) shows the convergence speed with respect to the value of CL_length ranging between 3 and 10 for functions f_1 to f_{15} in 10 dimensions. The differences between DCL and RCL can be observed with $CL\_length = 5$ by comparing the values for "CL 5" and "CL 5 rand". As expected, the cost in function evaluations is not significantly different between both patterns. However when it comes to the success rate, as depicted in Figure 11(b), DCL demonstrates constantly better performances. Moreover, functions that could be solved using DCL such as f_9 and f_{15} could not be solved successfully any more using RCL. It is worth noting that preliminary tests showed essentially no difference in Feval and SRate between RCL and a modified version of RCL where variables were restricted to participate only once (unless n is odd, in which case one randomly chosen variable is allowed to participate twice). Figure 12 reports the results in 30 dimensions with values of CL_length ranging between 10 and 20. For $CL\_length = 15$, the comparison of DCL against RCL shows again no significant difference in terms of Feval. As to SRate, DCL is constantly superior to RCL. Moreover, we observe that function f_9 in particular could not be solved by RCL. On the other hand, function f_{12} could be handled by RCL but not by DCL. In general however, DCL proved to be more effective. Consequently we decided to choose DCL as the default pattern for CL.

As for the length of the list, Figures 11(a) and 12(a) for problems in 10 and 30 dimensions, respectively, clearly illustrate without surprise that the cost in function evaluations is proportional to the length of the CL. Consequently, we have decided to set the value of CL_length to be 15 *by default*, in order to aim for good SRate with acceptable Feval. It can be noted that the particularity of DCL by default lies in the fact that the value of CL_length is equal to $round(n/2)$, where the operation *round* rounds towards the nearest integer. Thus each variable appears only once, in an ascendant fashion from x_1 to x_n, except for x_1 that will participate twice when n is odd. In this paper, the designation CL refers to the deterministic CL construction.

D   Effect of the Space Decomposition

In this subsection we are concerned with the SD mechanism. The idea of dealing with high-dimensional problems through a two-dimensional process is, to the best of the authors' knowledge, unusual in EAs. Here we discuss the effect of the introduction of SD into G3AT. To do so, a variant of G3AT, referred to as SD-G3AT, has been implemented. This variant implements SD within G3AT, but with a single GM without rotations. It is compared against G3AT in terms of SRate and Feval. The use of SD in SD-G3AT also requires the CL. It has been given the default construction as described in Appendix C.

As shown in Figure 13(a) for functions f_1 to f_{15} in 30 dimensions, the use of SD tends to increase the Feval. It can be observed that 10 functions out of 15 are more expensive to solve by using SD-G3AT. The increase in Feval can be more than triple for f_6 for instance. There are functions however for which the Feval cost remains on the same order. The raise of the Feval cost observed for functions f_2, f_3, f_6 and f_{13} is attributed to the final intensification process which occurs at the end of each era, as described in Subsection 4.2. However the performance in terms of SRate, as shown in Figure 13(b), also increases with the SD. For instance, functions f_6 and f_7, which could not be solved by G3AT, could be handled with a non-null SRate by SD-G3AT.

In the case of 10 dimensions, as depicted in Figure 14, the behavior is similar but the growth of Feval is less significant. SRate for SD-G3AT is constantly equal or higher than SRate for G3AT. In addition, SD-G3AT could also solve functions f_7 and f_9.
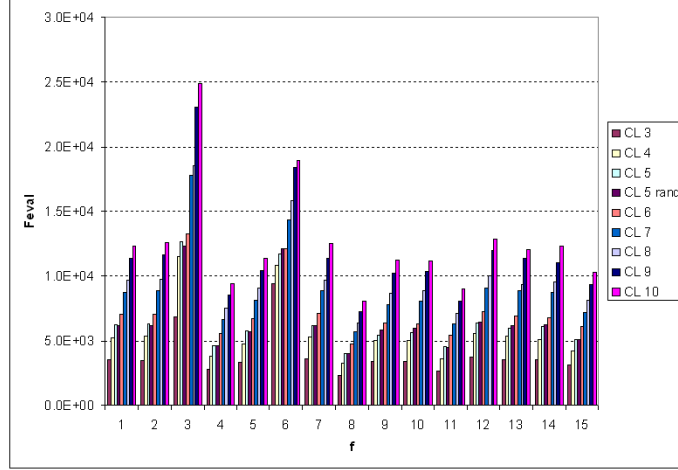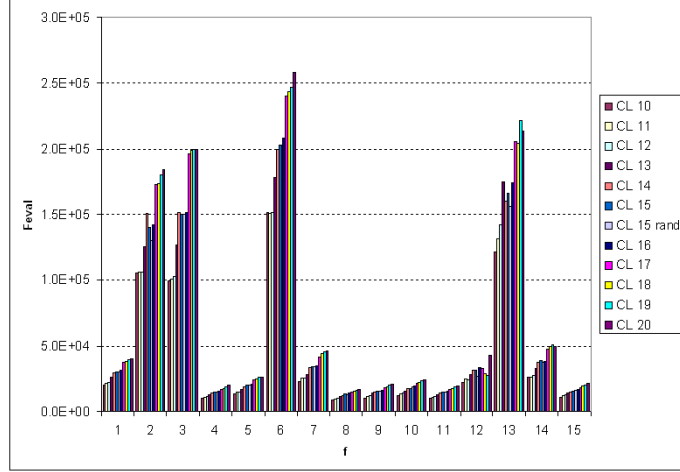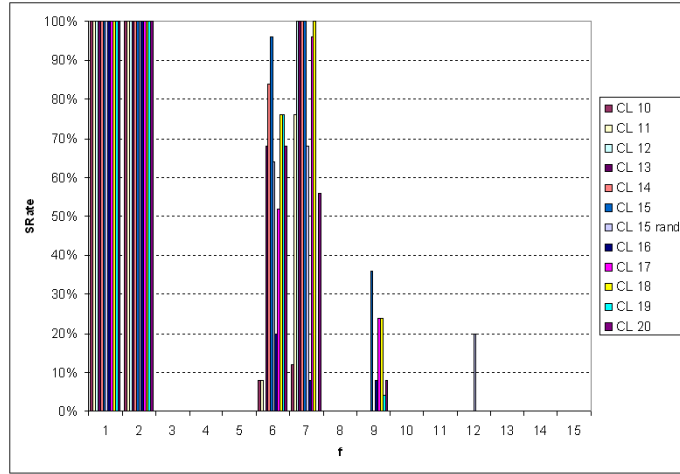
(a)
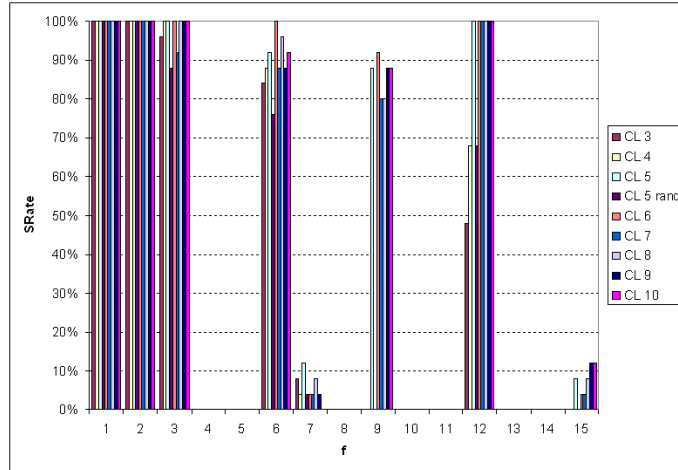
(b)

Figure 11: Feval and SRate for values of *CL_length* ranging from 3 to 10 in 10 dimensions for functions f_1 to f_{15}.

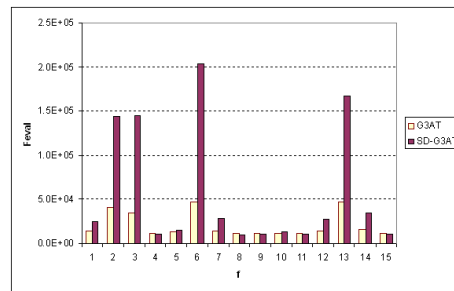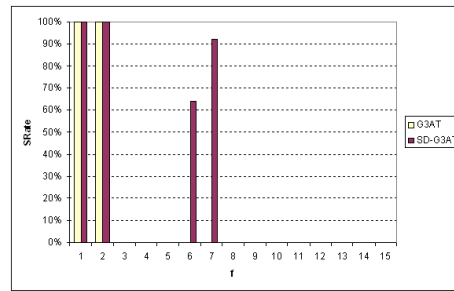(a)

(b)

Figure 12: Feval and SRate for values of *CL_length* ranging from 10 to 20 in 30 dimensions for functions f_1 to f_{15}.
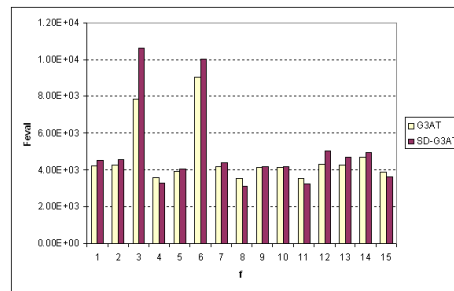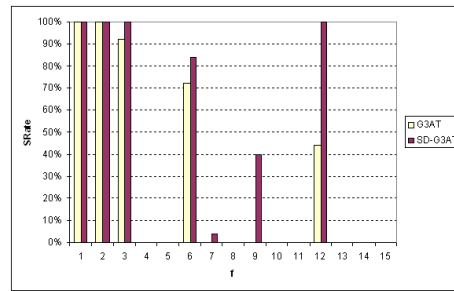
(a)

(b)

Figure 13: Comparison of Feval and SRate for G3AT vs SD-G3AT on functions f_1 to f_{15} in 30 dimensions.

(a)

(b)

Figure 14: Comparison of Feval and SRate for G3AT vs SD-G3AT on functions f_1 to f_{15} in 10 dimensions.

E   Number of Space Rotations

In GATR, the role of the rotations is to lower the possibility of premature termination at a non-optimal solution as a result of insufficient exploration of the search space due to the structure of the GM. In this subsection, we test GATR with different numbers of space rotations *NR* ranging between 0 and 4, where $NR = 0$ means no rotation, thus only the impact of SD is measured, as in Appendix D.

In Figure 15 and Figure 16, it is clearly observed that Feval is directly proportional to *NR* in the cases of both 10 and 30 dimensions. It can be estimated that the introduction of one rotation in 30 dimensions increases the Feval by around 5.4% on average, while in 10 dimensions it increases the Feval by around 10%. Starting with no rotation, it is after 3 rotations that functions that could not be solved before, namely function f_9 in 30 dimensions and function f_{15} in 10 dimensions, show positive results. The SRate doubles for function f_9 in 10 dimensions. Since high *NR* tends to slow down the algorithm, we have decided to adopt $NR = 3$ as the default value in our numerical experiments.
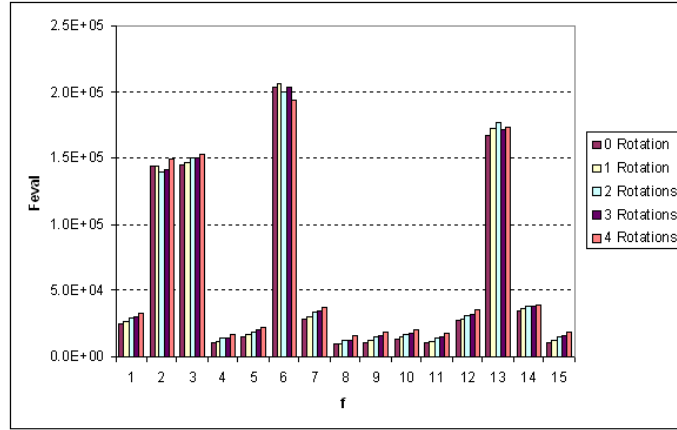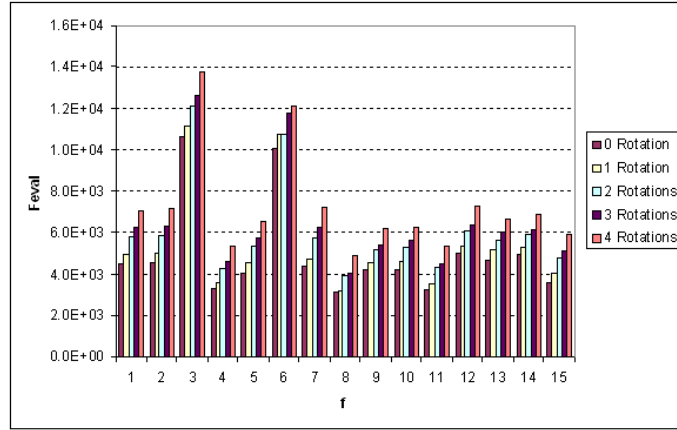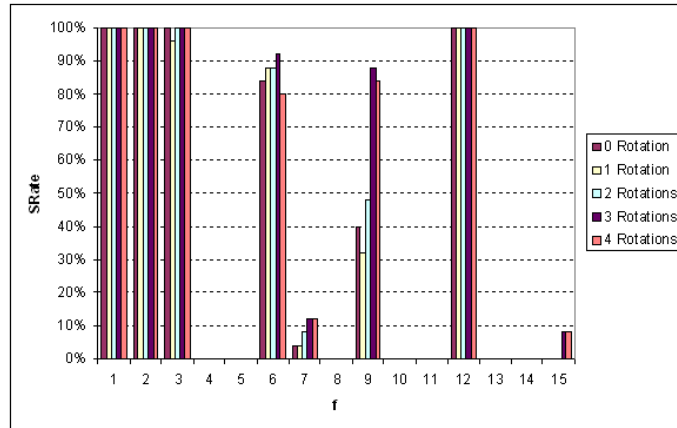
(a)
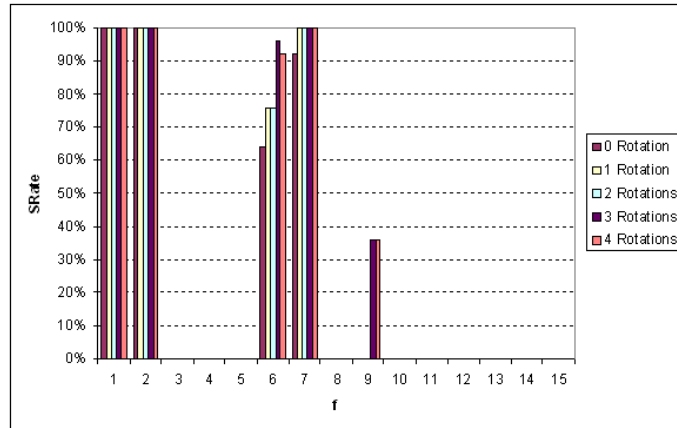
(b)

Figure 15: Comparison of Feval and SRate for different values of *NR* on functions f_1 to f_{15} in 30 dimensions.

(a)

(b)

Figure 16: Comparison of Feval and SRate for different values of *NR* on functions f_1 to f_{15} in 10 dimensions.

References

Back, T., Fogel, D. B., and Michalewicz, Z., editors (1997). *Handbook of Evolutionary Computation*. IOP Publishing Ltd., Bristol, UK.

Baker, J. E. (1985). Adaptive selection methods for genetic algorithms. In *Proceedings of the 1st International Conference on Genetic Algorithms*, pages 101–111, Hillsdale, NJ, USA. L. Erlbaum Associates Inc.

García, S., Molina, D., Lozano, M., and Herrera, F. (2009). A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: a case study on the cec'2005 special session on real parameter optimization. *Journal of Heuristics*, 15(6):617–644.

Giggs, M. S., Maier, H. R., Dandy, G. C., and Nixon, J. B. (2006). Minimum number of generations required for convergence of genetic algorithms. In *Proceedings of 2006 IEEE Congress on Evolutionary Computation*, pages 2580–2587, Vancouver, BC, Canada.

Hansen, N. (2006). The CMA evolution strategy: a comparing review. In Lozano, J., Larranaga, P., Inza, I., and Bengoetxea, E., editors, *Towards a new evolutionary computation. Advances on estimation of distribution algorithms*, pages 75–102. Springer.

Hansen, N., Auger, A., and Kern, S. (2005a). Performance evaluation of an advanced local search evolutionary algorithm. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 1777–1784. IEEE Press.

Hansen, N., Auger, A., and Kern, S. (2005b). A restart cma evolution strategy with increasing population size. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 1769–1776. IEEE Press.

Hansen, N. and Kern, S. (2004). Evaluating the cma evolution strategy on multimodal test functions. In *Proceedings of Eighth International Conference on Parallel Problem Solving from Nature PPSN VIII*, pages 82–291.

Hansen, N., Müller, S. D., and Koumoutsakos, P. (2003). Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (cma-es). *Evol. Comput.*, 11(1):1–18.

Hansen, N. and Ostermeier, A. (1996). Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation. In *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation*, pages 312–317. Morgan Kaufmann.

Hedar, A.-R. and Fukushima, M. (2003). Minimizing multimodal functions by simplex coding genetic algorithm. *Optimization Methods and Software*, 18:265–282.

Hedar, A.-R. and Fukushima, M. (2006a). Derivative-free filter simulated annealing method for constrained continuous global optimization. *J. of Global Optimization*, 35(4):521–549.

Hedar, A.-R. and Fukushima, M. (2006b). Directed evolutionary programming: Towards an improved performance of evolutionary programming. In *Proceedings of Congress on Evolutionary Computation, IEEE World Congress on Computational Intelligence*, pages 1521–1528, Vancouver, Canada.

Hedar, A.-R., Ong, B. T., and Fukushima, M. (2007). Genetic algorithms with automatic accelerated termination. Technical report, Dept. of Applied Mathematics and Physics, Kyoto University.

Holland, J. H. (1975). *Adaptation in natural and artificial systems*. The University of Michigan Press.

Jain, B. J., Pohlheim, H., and Wegener, J. (2001). On termination criteria of evolutionary algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference*, page 768. Morgan Kaufmann Publishers.

Kelley, C. T. (1999). Detection and remediation of stagnation in the nelder–mead algorithm using a sufficient decrease condition. *SIAM J. on Optimization*, 10(1):43–55.

Konar, A. (2005). *Computational Intelligence: Principles, Techniques and Applications*. Springer-Verlag, Berlin.

Koumousis, V. K. and Katsaras, C. P. (2006). A saw-tooth genetic algorithm combining the effects of variable popultion size and reinitialization to enhance performance. *IEEE Transactions on Evolutionary Computation*, 10(1):19–28.

Kwok, N. M., Ha, Q. P., Liu, D. K., Fang, G., and Tan, K. C. (2007). Efficient particle swarm optimization: a termination condition based on the decision-making approach. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 25–28, Singapore.

Laguna, M. and Marti, R. (2003). *Scatter Search: Methodology and Implementations in C*. Kluwer Academic Publishers, Boston.

Langdon, W. B. and Poli, R. (2007). Evolving problems to learn about particle swarm optimizers and other search algorithms. *IEEE Transactions on Evolutionary Computation*, 11(5):561–578.

Lee, C. and Yao, X. (2004). Evolutionary programming using the mutations based on the lévy probability distribution. *IEEE Transactions on Evolutionary Computation*, 8:1–13.

Leung, Y.-W. and Wang, Y. (2001). An orthogonal genetic algorithm with quantization for numerical optimization. *IEEE Transactions on Evolutionary Computation*, 5:41–53.

Lobo, F. G., Lima, C. F., and Michalewicz, Z. (2007). *Parameter Setting in Evolutionary Algorithms*. Springer Publishing Company, Incorporated.

Lozano, M., Herrera, F., and Molina, D. (2005). Adaptive local search parameters for real-coded memetic algorithms. pages 888–895.

Lunacek, M. and Whitley, D. (2006). The dispersion metric and the cma evolution strategy. In *GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 477–484, New York, NY, USA. ACM.

McMinn, P. (2004). Search-based software test data generation: A survey. *Softw. Test. Verif. Reliab.*, 14(2):105–156.

Montgomery, D. and Runger, G. (2003). *Applied Statistics and Probability for Engineers*. John Wiley & Sons Inc.

Moscato, P. (1999). Memetic algorithms: An introduction. In Corne, D., Dorigo, M., Glover, F., Dasgupta, D., Moscato, P., Poli, R., and Price, K. V., editors, *New ideas in optimization*. McGraw-Hill Ltd., UK, Maidenhead, UK, England.

Nelder, J. and Mead, R. (1965). A simplex method for function minimization. *Computer Journal*, 7:308–313.

Noman, N. and Iba, H. (2008). Accelerating differential evolution using an adaptive local search. *IEEE Transactions on Evolutionary Computation*, 12(1):107–125.

Ong, Y.-S. and Keane, A. (2004). Meta-lamarckian learning in memetic algorithms. *IEEE Transactions on Evolutionary Computation*, 8(2):99–110.

Ong, Y.-S., Lim, M.-H., Zhu, N., and Wong, K. (2006). Classification of adaptive memetic algorithms: A comparative study. *IEEE Transactions on Systems, Man, and Cybernetics–Part B*, 36(1):141–152.

O'Sullivan, M., Vssner, S., and Wegener, J. (1998). Testing temporal correctness of real-time systems. In *EuroSTAR'98: Proceedings of the Sixth International Conference on Software Testing Analysis and Review*, Munich, Germany.

Safe, M., Carballido, J., Ponzoni, I., and Brignole, N. (2004). On stopping criteria for genetic algorithms. *Lecture Notes in Computer Science*, 3171:405–413.

Storn, R. and Price, K. (1997). Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *J. of Global Optimization*, 11(4):341–359.

Suganthan, P. N., Hansen, N., Liang, J. J., Deb, K., Chen, Y. P., Auger, A., and Tiwari, S. (2005). Problem definitions and evaluation criteria for the cec-2005 special session on real-parameter optimization. Technical report, Singapore: Nanyang Technol. Univ.

Tsai, J.-T., Liu, T.-K., and Chou, J.-H. (2004). Hybrid taguchi-genetic algorithm for global numerical optimization. *IEEE Transactions on Evolutionary Computation*, 8(2):365–377.

Tsutsui, S., Yamamura, M., and Higuchi, T. (1999). Multi-parent recombination with simplex crossover in real-coded genetic algorithms. In *GECCO '99: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 657–664.

Tu, Z. and Lu, Y. (2004). A robust stochastic genetic algorithm (stga) for global numerical optimization. *IEEE Transactions on Evolutionary Computation*, 8(5):456–470.

Yao, X., Liu, Y., and Lin, G. (1999). Evolutionary programming made faster. *IEEE Transactions on Evolutionary Computation*, 3(2):82–102.

Yuen, S. Y. and Chow, C. K. (2009). A genetic algorithm that adaptively mutates and never revisits. *IEEE Transactions on Evolutionary Computation*, 13(2):454–472.

Zhong, W., Liu, J., Xue, M., and Jiao, L. (2004). A multiagent genetic algorithm for global numerical optimization. *IEEE Transactions on Systems, Man, and Cybernetics–Part B*, 34(2):1128–1141.

Zhou, Z., Ong, Y.-S., Nair, P., Keane, A., and Lum, K. (2007). Combining global and local surrogate models to accelerate evolutionary optimization. *IEEE Transactions on Systems, Man, and Cybernetics–Part B*, 37(1):66–76.