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. In the same way, automatic termination criteria may also be used effectively for dynamic EA (Koo et al, 2010) where the convergence is very dependent on the behavior of the dynamic problem.

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, 2006; 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.

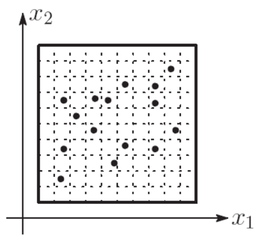
The response is a rotation-based version of the G3AT method, designated as GATR, which stands for Genetic Algorithm with Automatic Termination and Search Space Rotation. The main new elements are the Space Decomposition (SD) and the Space Rotation (SR). SD and SR work in combination in order to create 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; Le et al, 2009) in order to achieve faster convergence (Ong and Keane, 2004; Ong et al, 2006; Kramer, 2010; Jakob, 2010).

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. The effects of the introduction of the new mechanisms are discussed in Section 4. In Section 5, components and the formal algorithm of GATR are detailed. In Section 6, 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 7.

2 Gene Matrix and Mutagenesis

This section 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).



$$GM = \begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \end{pmatrix}$$

Figure 1: An example of the *Gene Matrix* in R^2.

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 a $2 \times m$ zero matrix in which each entry of the i-th row refers to a subrange of the i-th gene. As explained in Subsection 3.3, GM deals solely with two-dimensional subspaces, although the problem being optimized can be of any dimension $n \geq 2$. 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. By keeping track of explored area of the search space in order to concentrate on parts that have not been considered yet, GM yields some similarities with tabu search ideas (Glover, 1986; Ting et al, 2009).

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 versions 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 introduce 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. Specifically, GATR sorts the current population of size μ, and then selects the worst N_w ($< \mu$) individuals that will participate in the operations.

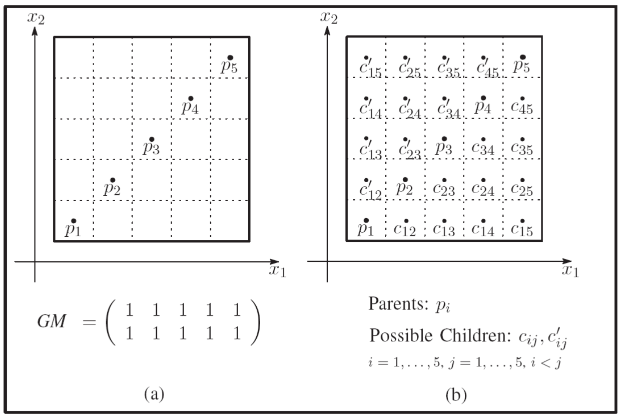
Mutagenesis operates in two ways in combination with the GM. First, in order to keep genetic diversity and accelerate the exploration process, Mutagenesis mimics the mutation operation by altering N_w worst individuals that have been selected to survive for the next generation. The alteration is however 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 Mutagenesis is given in Procedure 2.1. This feature also guarantees that GM will eventually get full (i.e., all subregions of the search space are explored).

Procedure 2.1 *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.*

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.



x_2
x_1
p_1
p_2
p_3
p_4
p_5
$GM = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix}$
(a)
c'_{15} c'_{25} c'_{35} c'_{45} p_5
c'_{14} c'_{24} c'_{34} p_4 c_{45}
c'_{13} c'_{23} p_3 c_{34} c_{35}
c'_{12} p_2 c_{23} c_{24} c_{25}
p_1 c_{12} c_{13} c_{14} c_{15}
Parents: p_i
Possible Children: c_{ij}, c'_{ij}
$i = 1, \dots, 5,\ j = 1, \dots, 5,\ i < j$
(b)

Figure 2: The role of crossover operation and GM.

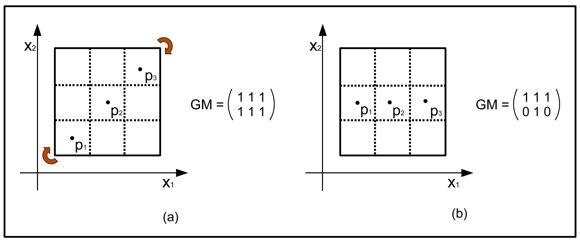
3.1 High-dimensional Search Space

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 can overcome this drawback as shown in Figure 2(b). 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, which is one of the reasons why we introduce the SR and SD mechanisms.

3.2 Space Rotation

Those considerations on GM prompt us to develop the space rotation mechanism. As depicted in Figure 3(a), a particular distribution of the individuals may lead to 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 any more). Thus in order to escape from premature termination of the search, the idea is to "rotate" the search space and attach to the resulting spaces a GM that do not necessarily have the same configuration as the original one at a given instant.

Let us consider the optimization problem $\min_{x \in X} f(x)$, where f is a real-valued



x2
x1
p1
p2
p3
GM = $\begin{pmatrix}1&1&1\\1&1&1\end{pmatrix}$
(a)
x2
x1
p1
p2
p3
GM = $\begin{pmatrix}1&1&1\\0&1&0\end{pmatrix}$
(b)

Figure 3: Diagonal distribution of individuals before and after rotation and the associated GM.

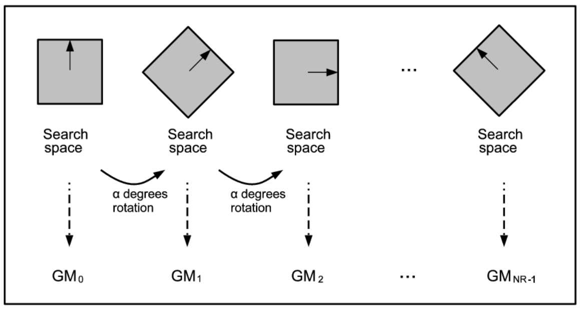


Figure 4: *NR* rotations of the search space by angle α and the associated GM.

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 $l_i \leq x_i \leq u_i$ $(i = 1, ..., n)$.

The rotation of the rectangle is depicted in Figure 4. Rotation is defined by the rotation angle α, which is a divisor of 360, as well as the number of rotations, $NR = 360/\alpha$. To each rotation is associated a GM, namely GM_l, $l = 0, \ldots, NR-1$. They are independently fed by their respective rotated spaces.

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 individual. 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 $\alpha$. 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.
```

The SR mechanism can be compared to a previous work by Schwefel that first appeared in (Schwefel, 1974) and also later in (Beyer and Schwefel, 2002). In that work, Schwefel introduces the idea of correlated mutation based on rotation of the solution space by using a rotation matrix for a variant of EA, namely, the Evolution Strategies (ES) (Rechenberg, 1965; Beyer and Schwefel, 2002). One of the peculiarities of ES is that individuals not only comprise the decision variables but also a set of strategy parameters. Among them, we can find those of the mutation operator. Hence, the decision variables along with the strategy parameters evolve simultaneously during the evolution process. However, in essence this strategy is different from SR. Indeed, the rotation matrix introduced by Schwefel aims at improving the mutation operation by controlling the area of the search space where are generated individuals. SR, on the other hand, do not aim at composing new individuals but rather ensuring that GM has explored a widespread area of the search space and detecting unexplored regions.

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 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}.

4 Effects of the GATR Mechanisms

Although GATR is used as an optimization tool, its parameters cannot be set to be optimal for all problems at the same time. Since GATR is not a self-adaptive method, in order to observe the guideline proposed by the CEC 2005, we have decided to run preliminary experiments on the 25 test functions presented in Subsection 6.1 and observe which parameter values would give the best results on a large set of test functions. In particular, we here discuss the effects of the introduction of each component of GATR and how the parameter values have been determined and used in this work.

4.1 Number of Subranges of the GM

The parameter m regulates the size of the sub-regions and its suitable value appears to be very problem dependent. The number of function evaluations, abbreviated as Feval, is directly proportional to m in any dimension, but the success rates, referred to as SRate and defined in Subsection 6.1, have unpredictable behaviors. For our numerical experiments, the convergence speed has been given more importance, since the Feval with m varying from 50 to 300 can dramatically increase, while SRate keeps fluctuating and does not necessarily increase significantly. It is thus favorable to keep m as small as possible. In our experiments, $m = 100$ was the value demonstrating the most satisfactory results on a large set of functions in terms of SRate, with moderate Feval.

4.2 Effect of the Space Decomposition

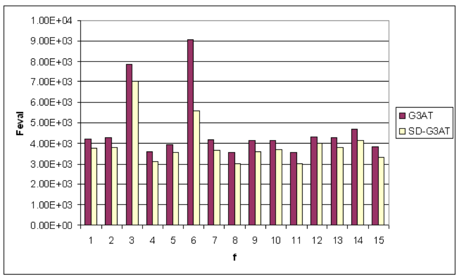
Here we discuss the effects of the introduction of SD into G3AT. To do so, a variant of G3AT, referred to as SD-G3AT, has been implemented. This variant employs SD with G3AT, but using a single GM without rotations, and only one intensification phase during the final era, as in G3AT. SD-G3AT is compared against G3AT in terms of SRate and Feval in 10 and 30 dimensions.

As shown in Figure 5(a) and 6(a) for functions f_1–f_{15} in 10 and 30 dimensions, respectively, the use of SD (through SD-G3AT), i.e., using solely $n = 2$ during the search, tends to slightly decrease the required Feval before termination (except function f_2 in 30 dimensions, for which Feval is slightly higher). Performance in terms of SRate in 10 dimensions, as shown in Figure 5(b), is advantageous for SD-G3AT. Functions that could be solved by G3AT were also solved by SD-G3AT. Moreover, SD-G3AT could perform better for functions f_3 and f_6, and could also solve functions f_7 and f_9. Figure 6(b) depicts the results achieved in 30 dimensions. We can observe that SRate stays around the same order. Functions f_1 and f_2 could be solved by both variants. We also note that function f_7 could not be solved by G3AT, but could be handled (with a small SRate) by SD-G3AT.

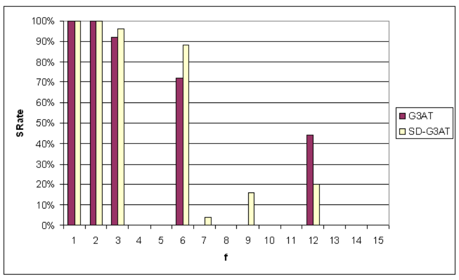
Thus, in terms of Feval and SRate, the introduction of SD could slightly accelerate the convergence of the algorithm, and obtain equivalent or better SRate.

4.3 Number of Space Rotations

In GATR, the role of the rotations is to lower the possibility of premature termination at a non-optimal solution, which is caused by insufficient exploration of the search space due to the structure of the GM. We have tested 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 was measured, as in Subsection 4.2. We could observe that Feval is

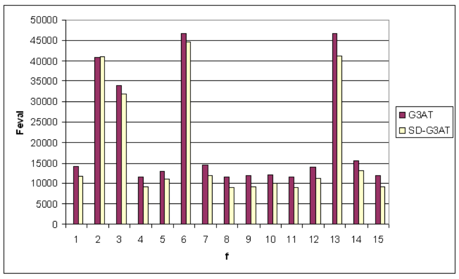


(a)

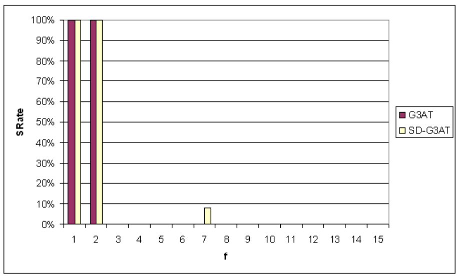


(b)

Figure 5: Comparison of Feval and SRate for G3AT vs SD-G3AT on functions f_1–f_{15} in 10 dimensions.



(a)



(b)

Figure 6: Comparison of Feval and SRate for G3AT vs SD-G3AT on functions f_1–f_{15} in 30 dimensions.

directly proportional to *NR*. 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%. Also high *NR* tends to slow down the algorithm. Starting with no rotation, after 3 rotations, the functions that could not be solved before came with positive results. Thus, to keep an acceptable convergence speed, we have decided to adopt $NR = 3$ as the default value in our numerical experiments.

4.4 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. One of the easiest and most intuitive way to construct the CL is to select randomly x_a and x_b from the set of all possible variables $\{x_1, x_2, \ldots, x_n\}$ with the only condition that x_a is different from x_b. However, experiments show that selecting the active variables in a sequential manner (i.e., starting with x_1 and x_2, then x_3 and x_4, etc.) could provide equal or better results for all functions. Although there is no theoretical proof to support this finding, we decided to adopt the "sequential" way of creating the CL (as opposed to the random construction) in GATR.

Now, we are concerned about 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. According to our preliminary experiments, it is without surprise that the cost in Feval increases proportionally with the length of the CL. Intuitively, one can expect better SRate with a longer CL, since more eras would be allowed to refine a solution. However, the results show that SRate increases slowly or fluctuates with *CL_length* for many functions. Moreover, a length of 5 and 15 in 10 and 30 dimensions, respectively, provided in general good performance in terms of SRate on a large set of problems. Increasing those values may improve slightly the SRate, but the Feval will considerably increase at the same time. According to those results, in GATR, *CL_length* has been set equal to $round(n/2)$, where the operation *round* rounds towards the nearest integer. This value aims at good SRate with acceptable Feval. Note that setting $CL\_length = round(n/2)$ guarantees that each variable is given a chance to participate in the search as an active variable at least once.

4.5 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 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. Indeed, GM combined with the SD and SR can be seen as the components that will explore the search space widely in order to find promising individuals. Those individuals will be given a chance to improve further through an exploitation phase handled by intensification. The numbers designating the eras that will go through an intensification step are stored in the intensification list (IL). Preliminary experiments, using the terminology introduced in Subsection 6.1 concerning the test functions and methodology, show that the best results are obtained when the first and last eras are subject to intensification. Various patterns to select the remaining eras have been studied. However, no significative differences can be observed when compared to a random selection of the intermediate intensified eras. The total number of eras that go through intensification (i.e., the length of IL) has also been studied. Intensification calls a local search method at the end of an era. Consequently, it relatively requires a considerable amount of additional computational resources. As a matter of fact, experiments show that a longer IL considerably increase Feval. However, SRate does not necessarily increase for all functions. Thus, in order to keep a good balance between exploration and exploitation, the length of IL in GATR has been set equal to $n/5$.

It should also be noted that although GATR relies on intensification when it comes to the exploitation phase, intensification alone cannot reach the results obtained by GATR. Indeed, experiments on some functions, such as f_9 for instance, revealed that the success rate for that function could reach 36% using GATR over the 25 trials. On the same function, the use of intensification only and using random starting points led to a success rate of 0%. From this simple experiment, we may observe that intensification has an important part to play in GATR. However, if used alone, the results are not satisfactory compared to GATR. It is thus in combination with GATR that superior results could be achieved.

4.6 Termination

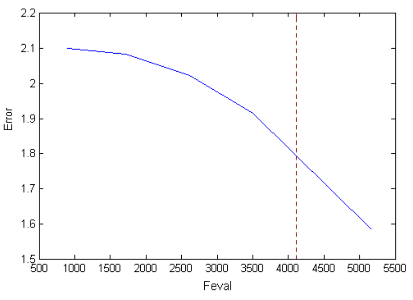
In order to estimate whether the GMs are able to provide the search with a correct termination point, the automatic termination has been compared for each function with the same run in the same conditions, as described in Subsection 6.1, but with an artificially maintained higher number of generations and only one final intensification occurring during the last era. For function f_{10} in 10 dimensions for instance, Figure 7(a) represents the evolution of the error with Feval using GATR and only one final intensification, referred to as Short-GATR. We can observe that around 5100 Fevals were required to reach an error level of around 1.6. The dotted vertical line represents the instant at which the algorithm decided to stop and launch the final intensification phase. Intensification could improve the results by reducing the error from roughly 1.8 to 1.6. Figure 7(b) depicts the evolution of the error over Feval obtained by a modified version of GATR, called Long-GATR, whose termination instant was artificially delayed. To do so, CL_length has been set equal to 15 instead of 5 (i.e., $round(n/2)$) in the original setting. More eras were thus allowed to explore the search space. Also, CR was set equal to 100% instead of 90% in order to obtain longer eras and let the search refine the obtained solutions further. In Long-GATR, around 16500 Fevals were necessary before termination, which is slightly more than 3 times the short version. We can see that after around 4000 Fevals, which is the instant at which Short-GATR decided to stop, the improvement in terms of error begins to stagnate at around 1.8. After 10000 function evaluations, the error improvement is still not significative. Finally, intensification could reduce the error from around 1.75 to 1.5. This experiment reveals that letting the search run after the original termination instant does not necessarily improve the best solution obtained, considering the substantial amount of additional Feval required to improve a solution further.

5 Formal GATR Algorithm

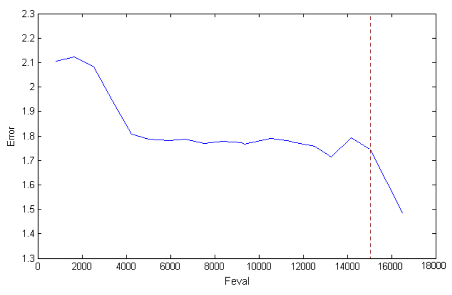
Before stating the formal algorithm of GATR, basic components that compose a GA and that are used by GATR are briefly described in this section.

5.1 GA Operators

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. In GATR, since $n = 2$ during the search, the



(a)



(b)

Figure 7: Comparison of termination instant after intensification (vertical dotted line) for function f_{10} using Short-GATR (a) and Long-GATR (b).

crossover operation is straightforward. The procedure consists in swapping the active variables of two parents p_1 and p_2, and create two children c_1 and c_2. 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 variable 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 a prespecified 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's subranges. Parents and children then compete together for survival, making GATR a steady-state GA.

Procedure 5.1 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.*

5.2 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 variables, 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 $x_i^{elite} := (\frac{u_i+l_i}{2})$, where l_i and u_i are the lower and upper bounds of gene x_i, respectively. 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*. Eventually, it is guaranteed that the GMs will reach *CP* since mutagenesis makes sure that new individuals are generated within unvisited areas. 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. The termination of the algorithm is thus guaranteed. 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 current best 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 5.2 *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 as described in Subsection 4.5. Set the Combination List $CL := \{x_1,x_2,\ldots,x_n\}$. If the number of elements in CL is odd, add a randomly chosen element from $\{x_1,x_2,\ldots,x_n\}$. Set the generation counter $t := 1$. Compute x^{elite} by $x_i^{elite} := \frac{u_i+l_i}{2}$ for $i = 1,\ldots,n$.*

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

2.1. Generate an initial population P_0 of size μ and update each individual x^k in P_0 so that $x_i^k := x_i^{elite}$ for $i \in \{1,\ldots,n\}\setminus\{a,b\}$.

2.2. Initialize Gene Matrices GM_l, $l = 1,...,NR$, as $2 \times m$ zero matrices, where GM_l corresponds to the l-th space rotation.

3. ***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.*

4. 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 4.1 and 4.2 until all chosen parents from* SP_t *are mated:*

4.1. *Choose two parents p_1 and p_2 from* SP_t*. Mate p_1 and p_2 by swapping their active variables to create children c_1 and c_2.*

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

5. 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 5.1. Add the mutated individual to the children pool* SC_t*. Update GM_l, $l = 1, \ldots, NR$ with SR by applying Procedure 3.1.*

6. *If all GMs have reached the completion ratio CP, go to Step 9. Otherwise go to Step 7.*

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

8. Mutagenesis. *Apply Procedure 2.1 to alter the N_w worst individuals in P_{t+1}, set $t := t+1$, update GM_l, $l = 1, \ldots, NR$ with SR by applying Procedure 3.1, and go to Step 3.*

9. Intensification. *If the current era is in IL, 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 2. Otherwise, terminate with x^{elite}.*

6 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.

6.1 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 mathematical form of the 25 test problems can be found in (Suganthan et al, 2005).

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. This is 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.

Table 1 summarizes the GATR parameters with their assigned values. 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 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.

6.2 Discussion from the Benchmark Results

The 25 test functions of the CEC 2005 can be classified according to their characteristics. Hence, we will discuss here the performance of GATR by taking into account the nature of the functions. Results are available in Table 2 for GATR, where the mean

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 Subranges	100
N_w	No. of individuals used by Mutagenesis	2
NR	Number of space rotations	3
α	Space rotation angle	45^o
CP	GM Completion ratio	90%

error values are reported for each functions as well the SRate, in parentheses, when not null. To begin with, the first five functions are unimodal functions and the others are multimodal. Functions f_1 and f_2 are relatively easy to solve, in 10, 30 and 50 dimensions. The increase in dimensionality did not affect substantially the performance of the algorithm in terms of mean error value. Function f_3 is the shifted rotated high conditioned elliptic function and the optima could be found only in 10 dimensions. Function f_4 is similar to f_2 but shifted with noise. The addition of noise seems to have affected the performance. Function f_6 could be solved in all dimensions with decent success rates, due to the local search capability of GATR through intensification. Function f_7 is a shifted rotated version of Griewank's function and does not have predefined bounds. It was easy to solve even in high dimensions. Function f_8 has a "needle in a haystack" profile and is thus very challenging to solve for most methods. GATR got stuck at the same mean error value order in all dimensions. Functions f_9 and f_{10} are shifted and rotated shifted versions of the Rastrigin function, respectively, and f_{11} is also a shifted rotated function. The global optimum was missed in all runs for f_{10} and f_{11}. Function f_{12} is Schwefel's problem and it could be solved in 10 dimensions. Function f_{13} and f_{14} are expanded functions and could not be solved. Functions f_{15}–f_{25} are hybrid composition functions based on basic functions. In particular, comparing functions f_{21} and f_{23} that have the same error order, we see that the algorithm did not seem affected when discontinuity is introduced. In general, functions f_{15}–f_{25} are designed to be very hard to solve for any algorithm. Except f_{15} in 10 dimensions with low success rate, GATR as well could not locate a global optimum for them under the requirements set by the CEC 2005 guideline.

6.3 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 the two methods using the Wilcoxon non-parametric test over the 25 benchmark functions of the CEC 2005. Table 3 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

Table 2: Solution Qualities for GATR and G3AT with significant method in bold and SRate in parentheses

	$n = 10$		$n = 30$		$n = 50$	
	GATR	G3AT	GATR	G3AT	GATR	G3AT
f	Error Mean	Error Mean	Error Mean	Error Mean	Error Mean	Error Mean
f_1	6.84e–13 (100%)	6.21e-13 (100%)	1.12e-12 (100%)	2.01e-12 (100%)	6.539e-12 (100%)	9.779e-12 (100%)
f_2	2.54e–12 (100%)	2.23e-12 (100%)	1.45e-09 (100%)	4.32e-06 (100%)	4.676e-08 (100%)	3.844e-03 (96%)
f_3	4.73e–12 (100%)	1.20e-03 (92%)	**2.11e+02**	6.89e+04	**6.790e+02**	1.395e+05
f_4	1.02e+04	**1.54e+03**	1.03e+05	**4.27e+04**	5.066e+05	**1.227e+05**
f_5	2.28e+02	1.38e+02	**2.29e+03**	5.79e+03	**5.473e+03**	1.614e+04
f_6	**3.19e–01** (92%)	1.50e+00 (72%)	**5.91e-01** (96%)	9.57e+01	**2.718e+00** (56%)	4.677e+02
f_7	**9.72e–02** (12%)	2.25e-01	**3.22e-04** (100%)	1.12e-02	**1.102e-03** (100%)	8.689e-03 (60%)
f_8	2.00e+01	2.00e+01	2.00e+01	2.00e+01	**2.000e+01**	2.001e+01
f_9	**3.98e–13** (88%)	2.83e+00	**2.61e-12** (36%)	2.17e+01	**4.975e+00**	7.053e+01
f_{10}	4.63e+01	**1.83e+01**	**5.53e+01**	1.20e+02	**1.830e+02**	3.161e+02
f_{11}	1.12e+01	**7.59e+00**	3.10e+01	2.98e+01	7.572e+01	**5.704e+01**
f_{12}	**9.48e–12** (100%)	3.09e+02 (44%)	3.94e+01	2.06e+03	**3.535e+03**	9.016e+03
f_{13}	**4.92e–01**	7.34e-01	**1.73e+00**	4.66e+00	**3.092e+00**	1.813e+01
f_{14}	4.01e+00	3.94e+00	1.32e+01	1.30e+01	2.257e+01	2.254e+01
f_{15}	**3.79e+01** (8%)	1.48e+02	3.67e+02	3.88e+02	**2.726e+02**	4.470e+02
f_{16}	1.46e+02	1.34e+02	**7.47e+01**	1.83e+02	**8.559e+01**	2.147e+02
f_{17}	2.10e+02	1.89e+02	2.77e+02	3.31e+02	**3.528e+02**	4.219e+02
f_{18}	9.00e+02	8.71e+02	9.00e+02	9.00e+02	9.000e+02	9.000e+02
f_{19}	9.00e+02	**8.84e+02**	9.00e+02	9.00e+02	9.000e+02	**9.000e+02**
f_{20}	9.00e+02	**8.43e+02**	9.00e+02	**9.00e+02**	9.000e+02	9.000e+02
f_{21}	**4.78e+02**	8.86e+02	8.82e+02	**5.70e+02**	7.439e+02	7.446e+02
f_{22}	8.38e+02	8.13e+02	**5.30e+02**	7.92e+02	5.007e+02	5.144e+02
f_{23}	**7.86e+02**	9.99e+02	7.40e+02	6.77e+02	7.529e+02	7.522e+02
f_{24}	2.91e+02	2.48e+02	2.24e+02	2.96e+02	**3.189e+02**	8.712e+02
f_{25}	**4.37e+02**	6.54e+02	**2.22e+02**	2.98e+02	**3.230e+02**	8.259e+02

Table 3: 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

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 3, 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) using the mean error as well as the error standard deviation (not reported). Table 2 reports the results of G3AT and GATR in 10, 30 and 50 dimensions using the same test bed as for the Wilcoxon's test. The results of a significantly superior method at level 0.05 for each function are reported in bold font. The Feval, reported in Table 4, 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, the computational cost in Feval approximately doubles 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, which has a more predominant

Table 4: Solution Costs for GATR and G3AT

	$n = 10$		$n = 30$		$n = 50$	
f	GATR Feval Mean	G3AT Feval Mean	GATR Feval Mean	G3AT Feval Mean	GATR Feval Mean	G3AT Feval Mean
f_1	6.24e+03	4.22e+03	3.05e+04	1.41e+04	7.347e+04	2.836e+04
f_2	6.31e+03	4.28e+03	1.41e+05	4.08e+04	5.258e+05	7.841e+04
f_3	1.26e+04	7.87e+03	1.50e+05	3.40e+04	5.535e+05	7.888e+04
f_4	4.63e+03	3.60e+03	1.46e+04	1.16e+04	2.571e+04	2.357e+04
f_5	5.72e+03	3.91e+03	1.99e+04	1.30e+04	3.721e+04	2.562e+04
f_6	1.17e+04	9.05e+03	2.03e+05	4.66e+04	4.437e+05	7.988e+04
f_7	6.21e+03	4.18e+03	3.41e+04	1.44e+04	4.537e+04	2.657e+04
f_8	4.01e+03	3.52e+03	1.30e+04	1.16e+04	2.245e+04	2.346e+04
f_9	5.41e+03	4.14e+03	1.57e+04	1.19e+04	3.066e+04	2.365e+04
f_{10}	5.61e+03	4.16e+03	1.79e+04	1.21e+04	3.135e+04	2.402e+04
f_{11}	4.51e+03	3.55e+03	1.47e+04	1.16e+04	2.634e+04	2.345e+04
f_{12}	6.39e+03	4.30e+03	3.16e+04	1.39e+04	3.121e+04	2.367e+04
f_{13}	6.01e+03	4.28e+03	1.67e+05	4.66e+04	1.686e+05	7.910e+04
f_{14}	6.13e+03	4.67e+03	3.89e+04	1.56e+04	8.994e+04	2.927e+04
f_{15}	5.12e+03	3.85e+03	1.60e+04	1.21e+04	2.842e+04	2.440e+04
f_{16}	5.34e+03	3.87e+03	1.89e+04	1.24e+04	3.206e+04	2.472e+04
f_{17}	5.02e+03	3.64e+03	1.66e+04	1.18e+04	2.916e+04	2.379e+04
f_{18}	6.12e+03	4.10e+03	2.25e+04	1.22e+04	4.674e+04	2.492e+04
f_{19}	6.25e+03	4.24e+03	2.41e+04	1.32e+04	5.067e+04	2.511e+04
f_{20}	6.26e+03	4.23e+03	2.24e+04	1.22e+04	4.854e+04	2.578e+04
f_{21}	6.94e+03	4.10e+03	2.16e+04	1.45e+04	4.417e+04	2.527e+04
f_{22}	5.15e+03	3.77e+03	1.99e+04	1.22e+04	5.154e+04	2.847e+04
f_{23}	5.24e+03	3.72e+03	2.00e+04	1.24e+04	4.041e+04	2.497e+04
f_{24}	5.27e+03	4.65e+03	1.99e+04	1.25e+04	3.499e+04	2.410e+04
f_{25}	4.83e+03	3.67e+03	1.80e+04	1.23e+04	3.195e+04	2.415e+04

Table 5: Wilcoxon's test for GATR against RCMA (at level 0.05)

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

weight in the total Feval cost of GATR. In 10 dimensions, 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. 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. In addition, it is worth pointing out that the SD and SR mechanisms themselves do not increase the running time of the algorithm. The additional running time in GATR when compared to G3AT comes from the intensification steps. Roughly speaking, the running time is proportional to Fevals in both GATR and G3AT. Experiments carried out using a computer with 3 GHz Intel Core 2 Duo processor and 3 GB of RAM reveals that for function f_1 in 50 dimensions for instance, the measured CPU time was around 37 seconds for G3AT and 95 seconds for GATR.

6.4 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 struc-

Table 6: 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	–

ture 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 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 5 (RCMA was not tested for 50 dimensions in (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.

6.5 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 6. 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.

6.6 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 positions of global optima. 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 Rotated high conditioned elliptic function (f_3), the Rotated Griewanks function (f_7), the Rotated Rastrigins function (f_9), the Rotated Weierstrasss function (f_{11}) and the Hybrid Composition function (f_{15}) (Suganthan et al, 2005). 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 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 7 and 8, 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}.

6.7 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 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.

Table 7: 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

		f_3		f_7		f_9	
Method		$n = 10$	$n = 30$	$n = 10$	$n = 30$	$n = 10$	$n = 30$
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

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 9. 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.

6.8 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.

Table 8: Comparison of GATR and NrGA on the Best Fitness Values Found and Feval in 10 and 30 Dimensions: f_{11} and f_{15}

		f_{11}		f_{15}	
Method		$n = 10$	$n = 30$	$n = 10$	$n = 30$
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 9: 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	–

Table 10: 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

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 10. 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 again 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.

7 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, 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 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.

References

Back T, Fogel DB, Michalewicz Z (eds) (1997) Handbook of Evolutionary Computation. IOP Publishing Ltd., Bristol, UK

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

Beyer HG, Schwefel HP (2002) Evolution strategies - A comprehensive introduction. Natural Computing 1:3–52

García S, Molina D, Lozano M, 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 MS, Maier HR, Dandy GC, Nixon JB (2006) Minimum number of generations required for convergence of genetic algorithms. In: Proceedings of 2006 IEEE Congress on Evolutionary Computation, Vancouver, BC, Canada, pp 2580–2587

Glover F (1986) Future paths for integer programming and links to artificial intelligence. Computers & Operations Research 13:533–549

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

Hansen N, 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, pp 82–291

Hansen N, 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, Morgan Kaufmann, pp 312–317

Hansen N, Müller SD, Koumoutsakos P (2003) Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). Evolutionary Computation 11(1):1–18

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

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

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

Hedar AR, Fukushima M (2006) Directed evolutionary programming: Towards an improved performance of evolutionary programming. In: Proceedings of Congress on Evolutionary Computation, IEEE World Congress on Computational Intelligence, Vancouver, Canada, pp 1521–1528

Hedar AR, Ong BT, Fukushima M (2007) Genetic algorithms with automatic accelerated termination. Tech. rep., Dept. of Applied Mathematics and Physics, Kyoto University

Holland JH (1975) Adaptation in Natural and Artificial Systems. The University of Michigan Press

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

Jakob W (2010) A general cost-benefit-based adaptation framework for multimeme algorithms. Memetic Computing 2:201–218

Kelley CT (1999) Detection and remediation of stagnation in the Nelder-Mead algorithm using a sufficient decrease condition. SIAM Journal on Optimization 10(1):43–55

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

Koo W, Goh C, Tan K (2010) A predictive gradient strategy for multiobjective evolutionary algorithms in a fast changing environment. Memetic Computing 2:87–110

Koumousis VK, Katsaras CP (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

Kramer O (2010) Iterated local search with Powell's method: A memetic algorithm for continuous global optimization. Memetic Computing 2:69–83

Kwok NM, Ha QP, Liu DK, Fang G, Tan KC (2007) Efficient particle swarm optimization: A termination condition based on the decision-making approach. In: Proceedings of the IEEE Congress on Evolutionary Computation, Singapore, pp 25–28

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

Le M, Ong YS, Jin Y, Sendhoff B (2009) Lamarckian memetic algorithms: local optimum and connectivity structure analysis. Memetic Computing 1:175–190

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

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

Lobo FG, Lima CF, Michalewicz Z (2007) Parameter Setting in Evolutionary Algorithms. Springer Publishing Company, Incorporated

Lozano M, Herrera F, Molina D (2005) Adaptive local search parameters for real-coded memetic algorithms. In: Proceedings of the 2005 IEEE Congress on Evolutionary Computation, pp 888–895

Lunacek M, 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, ACM, New York, NY, USA, pp 477–484

McMinn P (2004) Search-based software test data generation: A survey. Software Testing Verification and Reliability 14(2):105–156

Montgomery D, 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, Price KV (eds) New ideas in optimization, McGraw-Hill Ltd., UK, Maidenhead, UK, England

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

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

Ong YS, Keane A (2004) Meta-Lamarckian learning in memetic algorithms. IEEE Transactions on Evolutionary Computation 8(2):99–110

Ong YS, Lim MH, Zhu N, 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, 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

Rechenberg I (1965) Cybernetic solution path of an experimental problem. Tech. rep., Royal Air Force Establishment

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

Schwefel HP (1974) Adaptive mechanismen in der biologischen evolution und ihr einfluss auf die evolutionsgeschwindigkeit (abschlussbericht zum dfg-vorhaben re 215/2). Tech. rep., Technical University of Berlin, Berlin

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

Suganthan PN, Hansen N, Liang JJ, Deb K, Chen YP, Auger A, Tiwari S (2005) Problem definitions and evaluation criteria for the CEC-2005 special session on real-parameter optimization. Tech. rep., Singapore: Nanyang Technol. Univ.

Ting CK, Ko CF, Huang CH (2009) Selecting survivors in genetic algorithm using tabu search strategies. Memetic Computing 1:191–203

Tsai JT, Liu TK, Chou JH (2004) Hybrid Taguchi-genetic algorithm for global numerical optimization. IEEE Transactions on Evolutionary Computation 8(2):365–377

Tsutsui S, Yamamura M, 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, pp 657–664

Tu Z, 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, Lin G (1999) Evolutionary programming made faster. IEEE Transactions on Evolutionary Computation 3(2):82–102

Yuen SY, Chow CK (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, 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 YS, Nair P, Keane A, 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