Proceedings of The Fourth International Conference on Informatics & Applications, Takamatsu, Japan, 2015

Adaptive Memory Matrices for Automatic Termination of Evolutionary Algorithms

Abdel-Rahman Hedar*,§
*Dept. of Computer Science in Jamum, Umm Al-Qura Univ., Makkah, Saudi Arabia
ahahmed@uqu.edu.sa
§ Dept. of Computer Science, Assiut Univ., Assiut 71526, Egypt
hedar@aun.edu.eg

Abstract—**Evolutionary Algorithms (EAs) still have no automatic termination criterion. In this paper, we modify a genetic algorithm (GA), as an example of EAs, with new automatic termination criteria and acceleration elements. The proposed method is called the GA with Gene and Landmark Matrices (GAGLM). In the GAGLM method, the *Gene Matrix* (*GM*) and *Landmark Matrix* (*LM*) are constructed to equip the search process with a self-check to judge how much exploration has been done and to maintain the population diversity. Moreover, a special mutation operation called "*Mutagenesis*" is defined to achieve more efficient and faster exploration and exploitation processes. The computational experiments show the efficiency of the GAGLM method, especially its new elements of the mutagenesis operation and the proposed termination criteria.**

Keywords*—*Evolutionary Algorithms; Automatic Termination; Adaptive Memory; Gene Matrix; Landmark Matrix

I. INTRODUCTION

Evolutionary Algorithms (EAs) constitute one of the main tools in the computational intelligence area [5], [18]. Developing practical versions of EAs is highly needed to confront the rapid growth of many applications in science, engineering and economics [1], [18]. Nowadays, there is a great interest in improving the evolutionary operators which go further than modifying their genetics to use the estimation of distribution algorithms in generating the offspring [20], [24]. However, EAs still have no automatic termination criteria. Actually, EAs cannot decide when or where they can terminate and usually a user should pre-specify a maximum number of generations or function evaluations as termination criteria.

There are only a few recent works on termination criteria for EAs [6], [17]. In [6], an empirical study is conducted to detect a maximum number of generations using the problem characteristics. In [17], 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.

Our goal is to construct an intelligent method which seeks optimal or near-optimal solutions of the non-convex optimization problem

$$\min_{x \in X} f(x), \tag{1}$$

where f is a real-valued function defined on the search space $X \subseteq R^n$ with variables $x \in X$. Several versions of EAs have been proposed to deal with this problem, see [8], [11], [16], [20] and references therein. This problem has also been considered by different heuristics such as; tabu search [26], [13], simulated annealing [10], [12], memetic algorithms [25], [30], [29], differential evolution [3], [33], [34], particle swarm optimization [22], [4], ant colony optimization [35], variable neighborhood search [9], [27], scatter search [15], [19] and hybrid approaches [23], [36]. Many applications in different areas of computer science, engineering, and economics can be expressed or reformulated as Problem (1) [1], [7].

The proposed method is composed by modifying GAs with some directing strategies. First, an exploration and exploitation scheme is invoked to equip the search process with accelerated automatic termination criteria. Specifically, matrices called *Gene Matrix* (*GM*) and *Landmark Matrix* (*LM*) are constructed to sample the search space. The role of *GM* and *LM* is to assist the exploration process in two different ways. First, they can provide the search with new diverse solutions by applying a new type of mutation called "*mutagenesis*". The mutagenesis operator alters some survival individuals in order to accelerate the exploration and exploitation processes. In addition, *GM* and *LM* are used to let search process know how far the exploration process has gone in order to judge a termination point. The mutagenesis operation lets GAGLM behave like a so-called "Memetic Algorithm" [28] in order to achieve faster convergence [31], [32]. The numerical results presented later show that mutagenesis is effective and much cheaper than a local search. Then, the final intensification process can be started in order to refine the elite solutions obtained so far. The numerical results shown later indicate that the proposed GAGLM method is competitive with some other versions of GAs.

The rest of this paper is structured as follows. In Section II, we highlight and describe the main components of the proposed method, including the *GM*, *LM* and mutagenesis. In Section III, numerical experiments aiming at analyzing and discussing the performance of the proposed method and its novel operators are presented. Finally, the conclusion makes up Section IV.

II. GENETIC ALGORITHM WITH GENE AND LANDMARK MATRICES

In this section, a new modified version of GAs called Genetic Algorithm with Gene and Landmark Matrices (GAGLM) is presented. Before presenting the details of the GAGLM steps, we introduce its components including the new features of *GM*, *LM* and mutagenesis.

A. Gene Matrix

The basic idea of gene matrix has been introduced in [14]. Specifically, each individual x in the search space consists of n variables or genes since GAGLM uses the real-coding representation of individuals. The range of each gene is divided into m sub-ranges in order to check the diversity of the gene values. Then, we define a solution counter matrix C of size $n \times m$, in which entry c_{ij} represents the number of generated solutions such that gene i lies in the sub-range j, where $i = 1, \ldots, n$, and $j = 1, \ldots, m$. The "Gene Matrix" (*GM*) is initialized to be an $n \times m$ zero matrix in which each entry of the i-th row refers to a sub-range of the i-th gene. *GM* is a 0-1 matrix and while the search is processing, the entries of *GM* are updated from zeros to ones if new values for genes are generated within the corresponding sub-ranges. After having a *GM* full, i.e., with no zero entry, the search learns that an advanced exploration process has been achieved and can be stopped. Therefore, *GM* is used to equip the search process with practical termination criteria. Moreover, *GM* assists in providing the search with diverse solutions as will be shown in Subsection II-C. We consider two types of *GM* as follows.

1) Simple Gene Matrix (GM^S*):* GM^S does not take into account the number of solutions lying within each sub-range. Indeed, during the search, GAGLM updates the solution counter matrix C. Let x_i be the representation of the i-th gene, $i = 1, \ldots, n$. Once the gene i gets a value corresponding to a non-explored sub-range j, i.e., $c_{ij} > 0$, then *GM* is updated by flipping the zero into one in the corresponding (i, j) entry. Therefore, the updating process for GM^S can be defined as

$$(GM^S)_{ij} = \begin{cases} 0, & \text{if } c_{ij} = 0 \\ 1, & \text{if } c_{ij} > 0, \end{cases} \qquad (2)$$

where $i = 1, \ldots, n$, $j = 1, \ldots, m$, and $(GM^S)_{ij}$ is the (i, j) entry of the gene matrix GM^S.

Figure 1 shows an example of GM^S in two dimensions, i.e., $n = 2$. In this figure, the range of each gene is divided into ten sub-ranges. We can see that for the first gene x_1, no individual has been generated inside the sub-ranges 1, 7 and 10, i.e., $c_{1,1} = c_{1,7} = c_{1,10} = 0$. Consequently, the $(1, 1)$, $(1, 7)$ and $(1, 10)$ entries of GM^S are equal to zero. For the second gene x_2, only the first and the last sub-ranges are unvisited, hence the entries $(2, 1)$ and $(2, 10)$ of GM^S are null.

2) Advanced Gene Matrix (GM^A_α*):* GM^A_α comes along with a ratio α predefined by the user. Unlike GM^S, the GM^A_α is not immediately updated unless the ratio of the number of individuals that have been generated inside a sub-range so far and m (the total number of gene sub-ranges) exceeds or equals α. Therefore, the updating process for GM^A_α can be defined as

$$(GM^A_\alpha)_{ij} = \begin{cases} 0, & \text{if } c_{ij} < \alpha m \\ 1, & \text{if } c_{ij} \geq \alpha m, \end{cases} \qquad (3)$$

where $i = 1, \ldots, n$, $j = 1, \ldots, m$, and $(GM^A_\alpha)_{ij}$ is the (i, j) entry of the gene matrix GM^A_α.

An example of GM^A_α with $\alpha = 0.2$ in two dimensions can be found in Figure 1. Like GM^S, no individual has been generated inside the sub-ranges 1, 7 and 10 for the gene x_1. However, unlike GM^S, entry $(1, 3)$ is equal to 0 in $GM^A_{0.2}$ since there is only one individual lying inside the third sub-range, that is, $c_{1,3} = 1 < \alpha m = 2$. For the same reason, x_2 has six zero-entries corresponding to six sub-ranges in which the number of generated individuals divided by m is less than α. This example refers to the first generation of individuals. In a succeeding generation, if one or more individuals are generated inside the third sub-range for x_1 for example, then entry $(1, 3)$ will be set equal to one.

B. Landmark Matrix

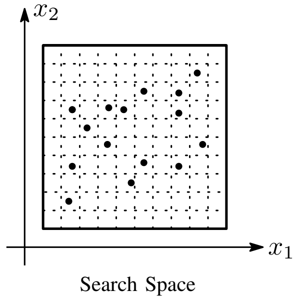
The Landmark Matrix (*LM*) is an indicator set that related to diverse points which are generated at the beginning of the search as well-distributed sample points collected from the whole search space. The size of this set of points is denoted by l, therefore *LM* is a row-matrix of size l and initialized to be full of ones. The *LM* is updated to represent only the points which their surrounding regions have not been discovered yet, i.e., *LM* is updated by converting its ones to zeros whenever the generated solutions are close to the corresponding landmark points. *LM* assists in providing the search with diverse solutions. *LM* is also used to equip the search process with practical termination criteria. Specifically, some diverse solutions can be generated close the unvisited landmark points that represented by ones in *LM*. This diverse generation process will lead to an empty *LM* after several generations. Then, the search process is learned that the exploration process is achieved and the algorithm can be terminated to an exploitation process.

An example of *LM* in two dimensions is shown in Figure 2. First, *LM* is initialized to be full of ones with size 15. In a succeeding generation, some of the generated solutions are close to the landmark points represented by 1, 7, 10 and 15. Therefore, *LM* is updated by converting these positions to be zeros.

C. Mutagenesis

After computing all children in each generation, GAGLM may give a chance to some characteristic children to improve themselves by modifying their genes. This is done by using a more artificial mutation operation called "*mutagenesis*". Specifically, three different types of mutagenesis operation are defined; the *gene matrix mutagenesis* (GM-*Mutagenesis*), the *landmark matrix mutagenesis* (LM-*Mutagenesis*) and the *best child inspiration mutagenesis* (BCI-*Mutagenesis*). These mutagenesis schemes alter some of the worst individuals of the current population. Specifically, GAGLM sorts the current population of size μ, and then selects the worst N_w ($< \mu$) individuals. GM-*Mutagenesis* and BCI-*Mutagenesis* alter N_1 and N_2 out of these N_w worst individuals, respectively, as described below, where $N_1 + N_2 \leq N_w$.

1) GM-Mutagenesis: In order to preserve the diversity in the search process and accelerate the exploration process, GM-*Mutagenesis* is used to alter N_1 from the N_w worst individuals selected for the next generation. Instead of waiting for the crossover operation to generate new diverse solutions in some unexplored search space partitions, GAGLM uses GM-*Mutagenesis* operations to do this by guidance of *GM*. Specifically, a zero-position in *GM* is randomly chosen, say 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

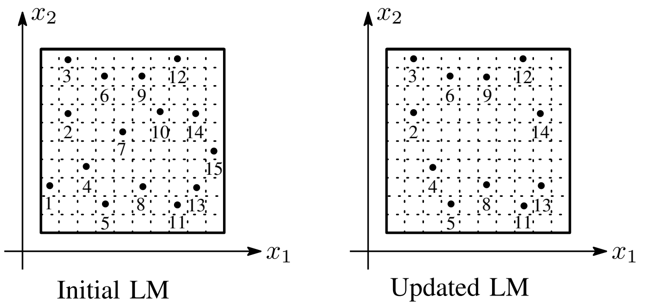


Solution Counter: $$C = \begin{pmatrix} 0 & 3 & 1 & 2 & 2 & 2 & 0 & 3 & 2 & 0 \\ 0 & 1 & 1 & 3 & 2 & 1 & 4 & 2 & 1 & 0 \end{pmatrix}$$

Simple GM: $$GM^S = \begin{pmatrix} 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \end{pmatrix}$$

Advanced GM: $$GM^A_{0.2} = \begin{pmatrix} 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 \end{pmatrix}$$

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



Initial LM

$$LM = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Updated LM

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

Fig. 2: An example of the *Landmark Matrix* in R^2.

chosen individuals for mutagenesis. Using this setting for x_i, there is a chance for crossover operation to explore different combinations of solutions containing this new value of x_i. Moreover, *GM* is updated since a new partition has been visited. The formal procedure for GM-*Mutagenesis* is given as follows.

Procedure 2.1: GM$-$*Mutagenesis*(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.

2) LM-Mutagenesis: LM-*Mutagenesis* can be used as an alternative to GM-*Mutagenesis*. LM-*Mutagenesis* is applied to alter N_1 from the N_w worst individuals selected for the next generation. Specifically, a one-position in *LM* is randomly chosen, say position (i), i.e., the area around the landmark i has not been visited yet. Then, a random point is generated close to the landmark i. Moreover, *LM* is updated since a new landmark has been visited. The formal procedure for LM-*Mutagenesis* is given as follows.

Procedure 2.2: LM$-$*Mutagenesis*(LM)

1. If *LM* is empty, then return; otherwise, go to Step 2.
2. Choose a one-position i in *LM* randomly.
3. Generate a new solution x randomly inside a neighborhood of the landmark i.
4. Update *GM* and return.

3) BCI-Mutagenesis: N_2 individuals from the N_w worst children are modified by considering the best child's gene values in the children pool. 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 the following procedure.

Procedure 2.3: 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.

D. Parent Selection

The parent selection mechanism first produces an intermediate population P' from the initial population P, i.e., $P' \subseteq P$, as in the canonical GA. For each generation, P' has the same size as P but an individual can be present in P' more than once. The individuals in P are ranked with their fitness function values based on the *linear ranking selection mechanism* [2], [11]. Indeed, individuals in P' are copies of individuals in P depending on their fitness ranking; the higher fitness an individual has, the more the probability that it will be copied is. This process is repeated until P' is full while an already chosen individual is not removed from P.

E. Crossover

The crossover operation has an exploration tendency, and therefore it is not applied to all parents. First, for each individual in the intermediate population P', the crossover operation chooses a random number from the interval $(0, 1)$. If the chosen number is less than a pre-specified crossover probability $p_c \in (0, 1)$, the individual is added to the parent pool. After that, two parents from the parent pool are randomly selected and mated to produce two children c_1 and c_2, which are then placed in the children pool. These procedures are repeated until all parents are mated. The crossover operator used in GAGLM method is a ρ-point operator with random $\rho \leq n$. Therefore, a recombined child is calculated to have ρ partitions. Both parents genotypes are cut in ρ partitions and the randomly chosen parts are exchanged to create two children. Let us note that a parent is selected only once, and if the total number of parents inside the parent pool is uneven, then the last parent that was added into the pool is not considered for the mating procedure. As an example, Figure 3 shows two parents p_1 and p_2 partitioned into five partitions at the same positions (after the second, third, fourth and sixth genes). Then, a recombined child c_1 is generated to inherit partitions from the two parents according to a random sequence of zeros and ones, $(0, 0, 1, 0, 1)$ in this example, meaning that its first, second and fourth partitions will be inherited from p_1 and third and fifth partitions from p_2. The other recombined child c_2's partition sequence is the complementary of c_1's sequence, namely $(1, 1, 0, 1, 0)$. The following procedure describes the GAGLM crossover operation precisely.

Procedure 2.4: 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.

Actually, this type of crossover is chosen to support the GAGLM exploration process. Specifically, there is no information related to different sub-ranges of different variables saved in *GM* in order to escape from the complexity of high dimensional problems. Therefore, there is a possibility of having misguided termination of the exploration process as in the example shown in Figure 4(a), where *GM* is already full although genes x_1 and x_2 have not their search spaces entirely covered. However, invoking this type of crossover operation as in Procedure 2.4 can easily overcome the drawback as shown in Figure 4(b). Actually, mutagenesis operation cooperates with this type of crossover operation by combining new generated genes by mutagenesis with other existing genes of the current population individuals.

F. Mutation

The mutation operator uses information contained in *GM*. 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 pre-specified mutation probability p_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 p_m is counted. Let num_m denote this number. Afterward, the mutation operation makes sure 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 to be mutated is reduced to num_{zeros}. Finally, a zero from *GM* is randomly selected, say in position (i, j), and a randomly chosen individual from IP_M has its gene x_i modified by a new value lying inside the j-th partition of its range. The formal procedure for mutation is analogous to Procedure 2.1.

Procedure 2.5: 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.

G. GAGLM Algorithm

The formal algorithm of the GAGLM method is stated below.

Algorithm 2.6: GAGLM Algorithm

1. **Initialization.** Set values of m, l, μ, η, and (l_i, u_i), for $i = 1, \ldots, n$. Set the crossover and mutation probabilities $p_c \in (0, 1)$ and $p_m \in (0, 1)$, respectively. Set the generation counter $t := 0$. Initialize *GM* as an $n \times m$ zero-matrix and *LM* as an $1 \times l$ one-matrix, and generate an initial population P_0 of size μ.

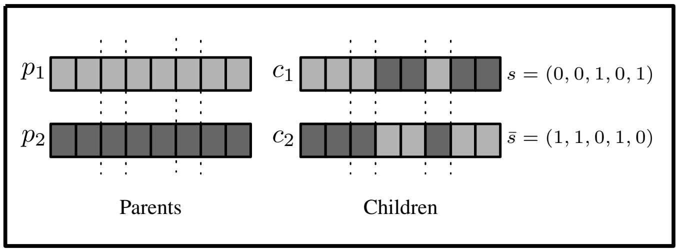
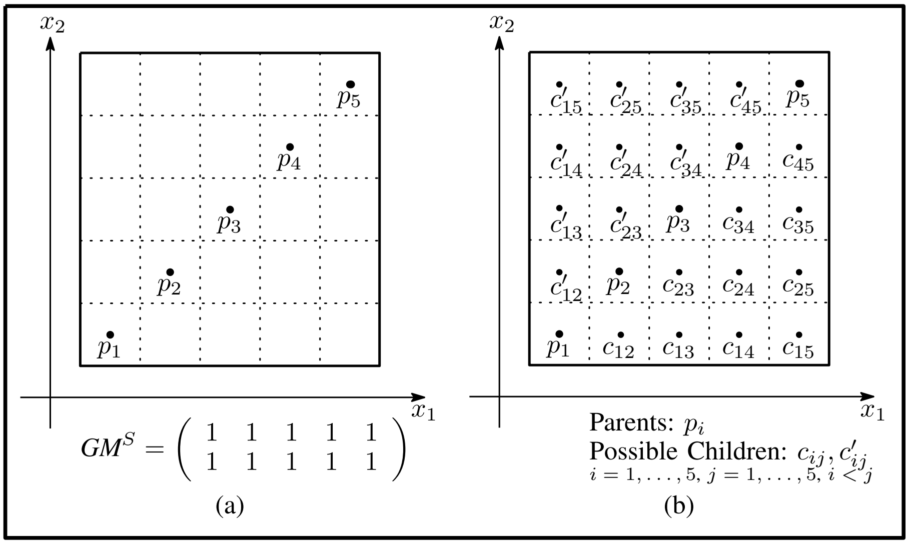


Fig. 3: An example of the crossover operation.


x_2
x_1
p_1
p_2
p_3
p_4
p_5
$GM^S = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix}$
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}
$i = 1, \ldots, 5,\ j = 1, \ldots, 5,\ i < j$

(a) (b)

Fig. 4: The role of crossover operation and *GM*.

2. Parent Selection. Evaluate the fitness function F of all individuals in P_t. Select an intermediate population P'_t from the current population P_t.

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 p_c. Repeat 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 2.4 to reproduce children c_1 and c_2.

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

4. Mutation. Associate a random number from $(0, 1)$ with each gene for each individual in P'_t. Mutate the individuals which have an associated number less than p_m by applying Procedure 2.5. Add the mutated individual to the children pool SC_t, and update *GM*.

5. Local Search. Apply a local search method to improve the best child (if exists), and then update *GM*.

6. Stopping Condition. If η generations have passed after getting a full *GM* or an empty *LM*, then stop. Otherwise, go to Step 7.

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

8. Mutagenesis. Apply Procedures 2.1 or 2.2, and 2.3 to alter the N_w worst individuals in P_{t+1}, set $t := t + 1$, update *GM* or *LM*, and go to Step 2.

III. NUMERICAL RESULTS

Algorithm 2.6 (GAGLM) was programmed in MATLAB and applied to 13 well-known test functions [37], [21], listed in the Appendix. Before discussing the GAGLM results, we summarize the setting of the GAGLM parameters and performance.

A. Parameter Setting

First, the search space for Problem (1) is defined as

$$[L, U] = \{x \in R^n : l_i \le x_i \le u_i,\ i = 1, \ldots, n\}.$$

In Table I, we summarize all GAGLM parameters with their assigned values. These values are based on the common setting in the literature or determined through our preliminary numerical experiments.

TABLE I: GAGLM Parameter Setting

Parameter	Definition	Value
μ	Population size	50
p_c	Crossover Probability	0.6
p_m	Mutation Probability	0.1
m	No. of *GM* columns	$50n$
l	*LM* size	$50n$
N_1	No. of individuals used by GM-*Mutagenesis* and LM-*Mutagenesis*	2
N_2	No. of individuals used by BCI-*Mutagenesis*	2
α	Advanced Gene Matrix percentage for $GAGLM_M^A$	$3/m$
NM_{maxIt}	No. of maximum iterations used by Local Search in GAGLM$_L$	$5n$

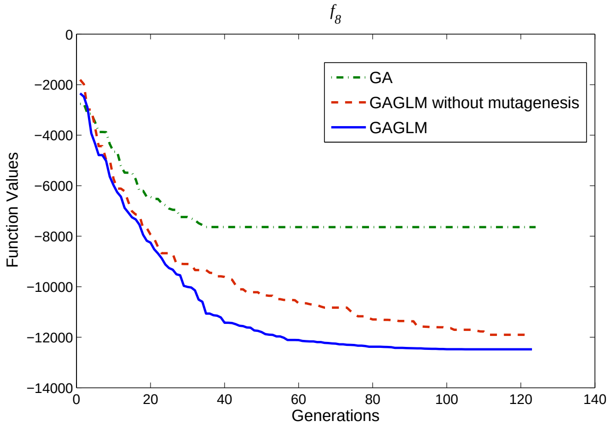


Fig. 5: An Example of the Mutagenesis Performance

B. Performance Analysis of the Mutagenesis Operation

Figure 5 reveals that the proposed mutagenesis operation can improve the performance of GA. It shows that the mutagenesis operation helps in reducing the solution costs.

C. Performance Analysis of Automatic Termination

More preliminary experimental results are depicted in Figures 6-7 and Figures 8-9 to show the role of the exploration and automatic termination by using *GM* and *LM*, respectively. In these figures, the solid vertical line "Full GM" refers to the generation number at which a full *GM* is obtained, the solid vertical line "Empty LM" refers to the generation number at which an empty *LM* is obtained, and while the dotted vertical line "Termination" refers to the generation number at which the exploration process terminates. The code still kept running after reaching the dotted line without stopping in order to check the efficiency of the automatic termination. As we can see, no more significant improvement in terms of function values can be expected after the dotted line for all test functions. Thus, we can conclude that our automatic termination played a significant role. Moreover, these figures show the robustness of the proposed method, in the sense that it obtains nearly optimal solutions.

IV. CONCLUSIONS

This paper has shown that the use of gene and landmark matrices effectively assists an algorithm to achieve wide exploration and deep exploitation before stopping the search. This indicates that our main objective to equip evolutionary algorithms with automatic accelerated termination criteria has largely been fulfilled. Moreover, the proposed intensification schemes based on mutagenesis of the gene and landmark matrices and the best child inspiration have proved to be efficient in our experiments. Therefore, the newly added elements in the proposed method contribute in achieving good performance.

ACKNOWLEDGMENT

This work was funded by the National Plan for Science, Technology and Innovation (MAARIFAH) – King Abdulaziz City for Science and Technology – the Kingdom of Saudi Arabia, award number (13-INF544-10).

APPENDIX

A. Sphere Function (f_1)

Definition: $f_1(\mathbf{x}) = \sum_{i=1}^{n} x_i^2$.
Search space: $-100 \le x_i \le 100,\ i = 1, \ldots, n$.
Global minimum: $\mathbf{x}^* = (0, \ldots, 0),\ f_1(\mathbf{x}^*) = 0$.

B. Schwefel Function (f_2)

Definition: $f_2(\mathbf{x}) = \sum_{i=1}^{n} |x_i| + \Pi_{i=1}^{n} |x_i|$.
Search space: $-10 \le x_i \le 10,\ i = 1, \ldots, n$.
Global minimum: $\mathbf{x}^* = (0, \ldots, 0),\ f_2(\mathbf{x}^*) = 0$.

C. Schwefel Function (f_3)

Definition: $f_3(\mathbf{x}) = \sum_{i=1}^{n} (\sum_{j=1}^{i} x_j)^2$.
Search space: $-100 \le x_i \le 100,\ i = 1, \ldots, n$.
Global minimum: $\mathbf{x}^* = (0, \ldots, 0),\ f_3(\mathbf{x}^*) = 0$.

D. Schwefel Function (f_4)

Definition: $f_4(\mathbf{x}) = \max_{i=1,\ldots n}\{|x_i|\}$.
Search space: $-100 \le x_i \le 100,\ i = 1, \ldots, n$.
Global minimum: $\mathbf{x}^* = (0, \ldots, 0),\ f_4(\mathbf{x}^*) = 0$.

E. Rosenbrock Function (f_5)

Definition: $f_5(\mathbf{x}) = \sum_{i=1}^{n-1} \left[100\left(x_i^2 - x_{i+1}\right)^2 + (x_i - 1)^2 \right]$.
Search space: $-30 \le x_i \le 30,\ i = 1, 2, \ldots, n$.
Global minimum: $\mathbf{x}^* = (1, \ldots, 1),\ f_5(\mathbf{x}^*) = 0$.

F. Step Function (f_6)

Definition: $f_6(\mathbf{x}) = \sum_{i=1}^{n} (\lfloor x_i + 0.5 \rfloor)^2$.
Search space: $-100 \le x_i \le 100,\ i = 1, 2, \ldots, n$.
Global minimum: $\mathbf{x}^* = (0, \ldots, 0), f_6(\mathbf{x}^*) = 0$.

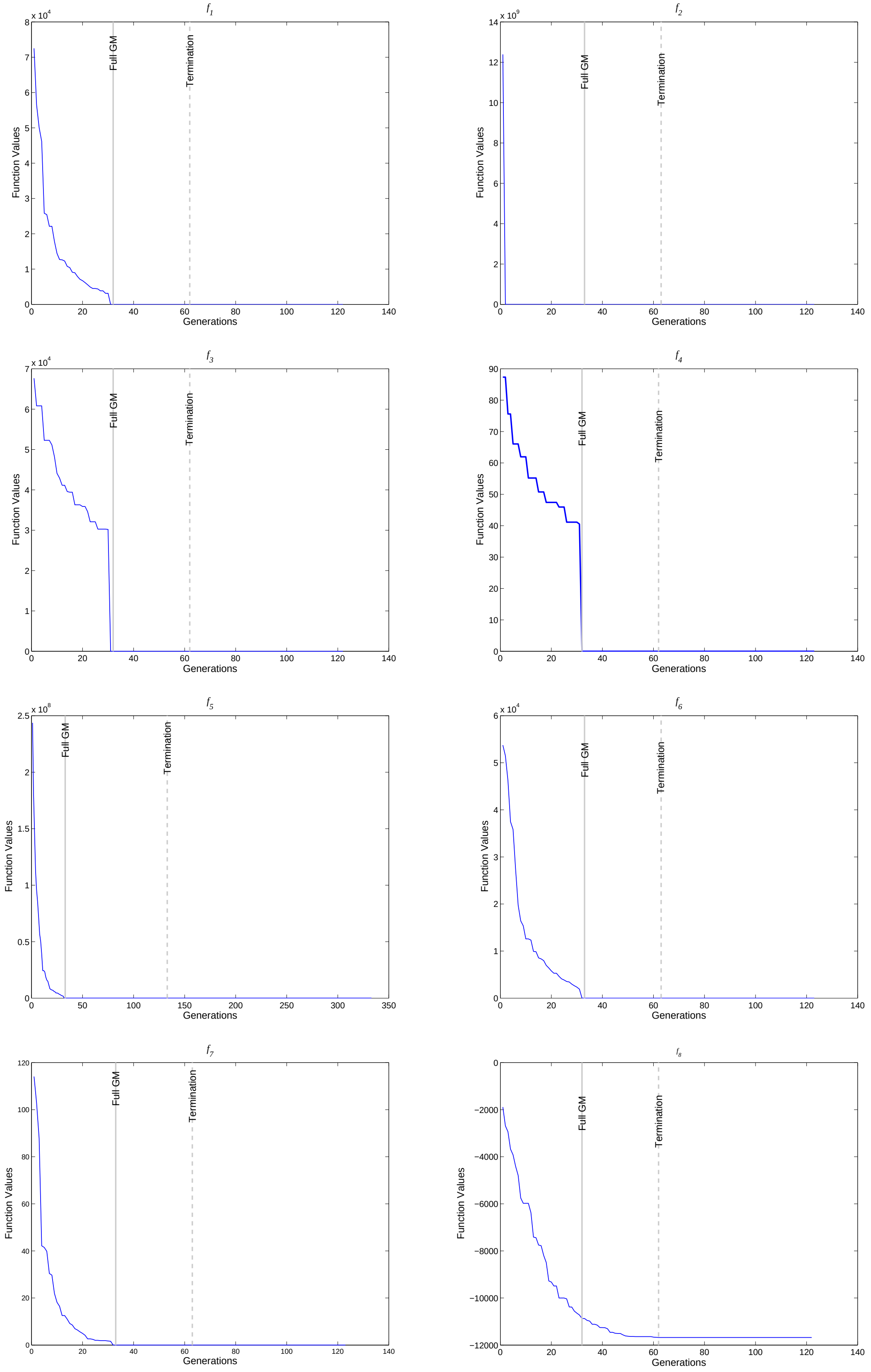


Fig. 6: Automatic Termination Performance Using GM (f_1 – f_8)

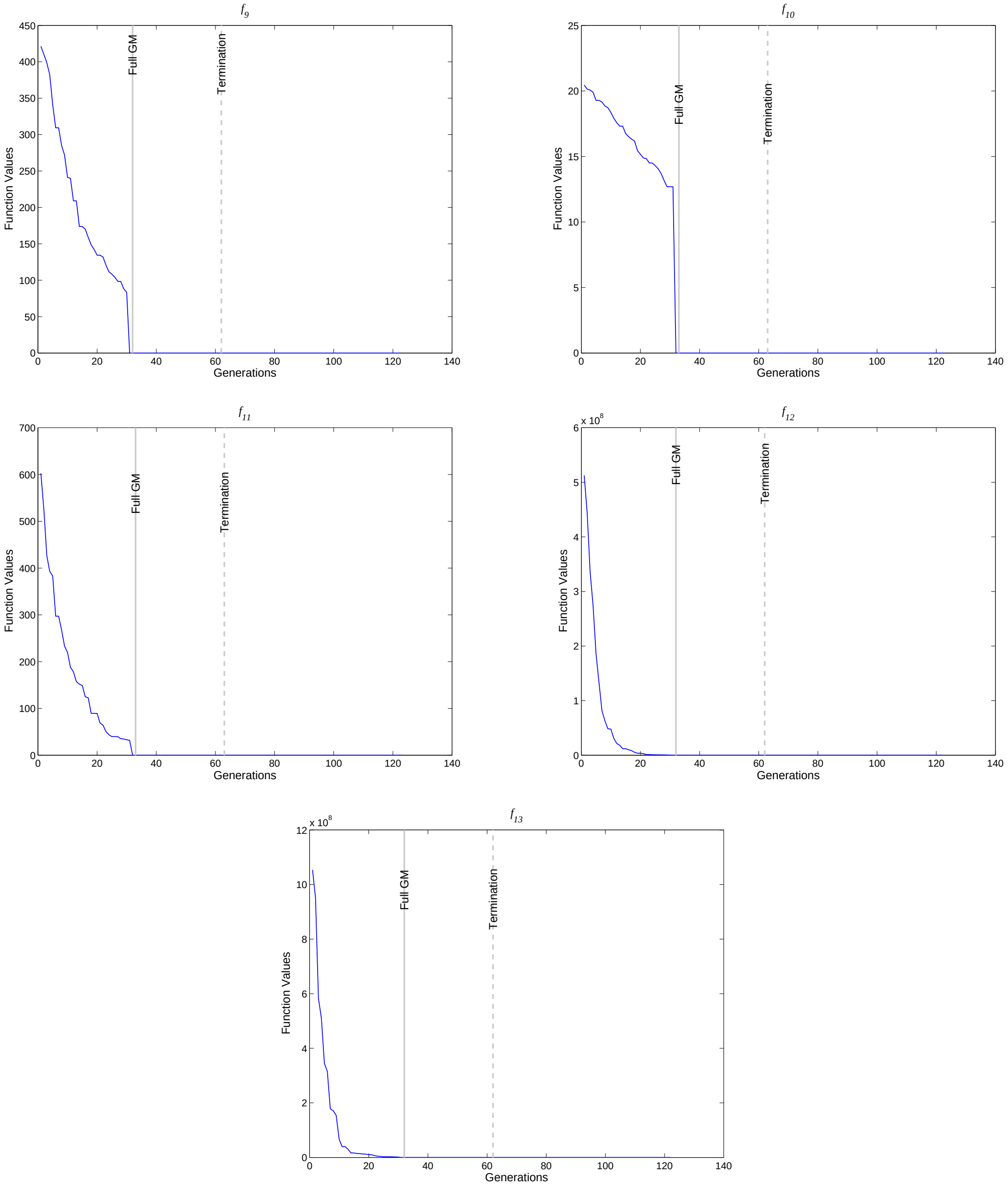


Fig. 7: Automatic Termination Performance Using GM (f_9 – f_{13})

G. Quartic Function with Noise (f_7)

Definition: $f_7(\mathbf{x}) = \sum_{i=1}^{n} i x_i^4 + random[0,1)$.
Search space: $-1.28 \leq x_i \leq 1.28,\ i = 1, \ldots, n$.
Global minimum: $\mathbf{x}^* = (0, \ldots, 0),\ f_7(\mathbf{x}^*) = 0$.

H. Schwefel Functions (f_8)

Definition: $f_8(\mathbf{x}) = -\sum_{i=1}^{n} \left(x_i \sin \sqrt{|x_i|} \right)$.
Search space: $-500 \leq x_i \leq 500,\ i = 1, 2, \ldots, n$.
Global minimum: $\mathbf{x}^* = (420.9687, \ldots, 420.9687),\ f_8(\mathbf{x}^*) = -418.9829n$.

I. Rastrigin Function (f_9)

Definition: $f_9(\mathbf{x}) = 10n + \sum_{i=1}^{n} \left(x_i^2 - 10\cos\left(2\pi x_i\right) \right)$.
Search space: $-5.12 \leq x_i \leq 5.12,\ i = 1, \ldots, n$.
Global minimum: $\mathbf{x}^* = (0, \ldots, 0), f_9(\mathbf{x}^*) = 0$.

J. Ackley Function (f_{10})

Definition: $f_{10}(\mathbf{x}) = 20 + e - 20e^{-\frac{1}{5}\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}} - e^{\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)}$.
Search space: $-32 \leq x_i \leq 32,\ i = 1, 2, \ldots, n$.
Global minimum: $\mathbf{x}^* = (0, \ldots, 0); f_{10}(\mathbf{x}^*) = 0$.

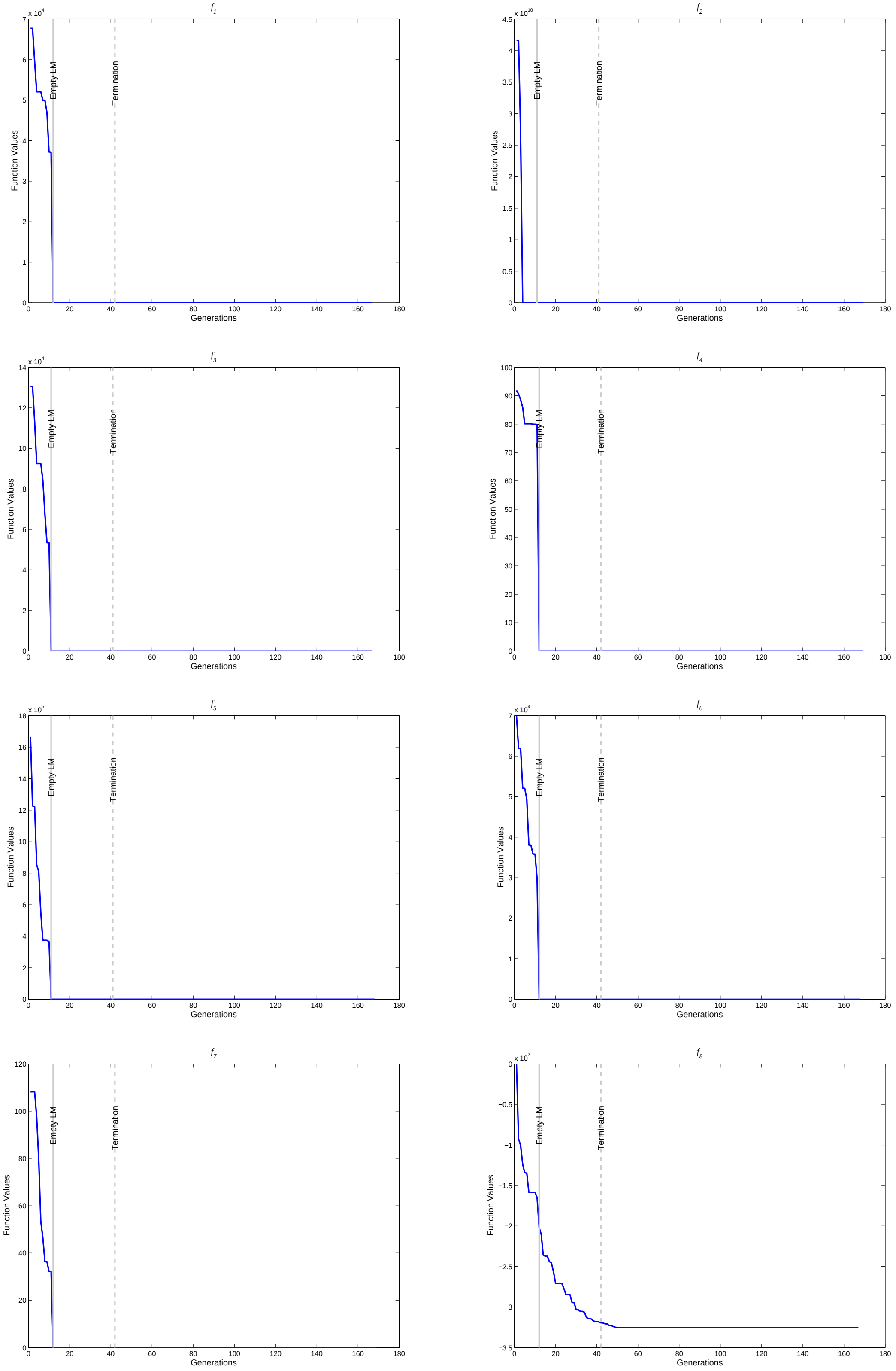


Fig. 8: Automatic Termination Performance Using LM ($f_1 - f_8$)

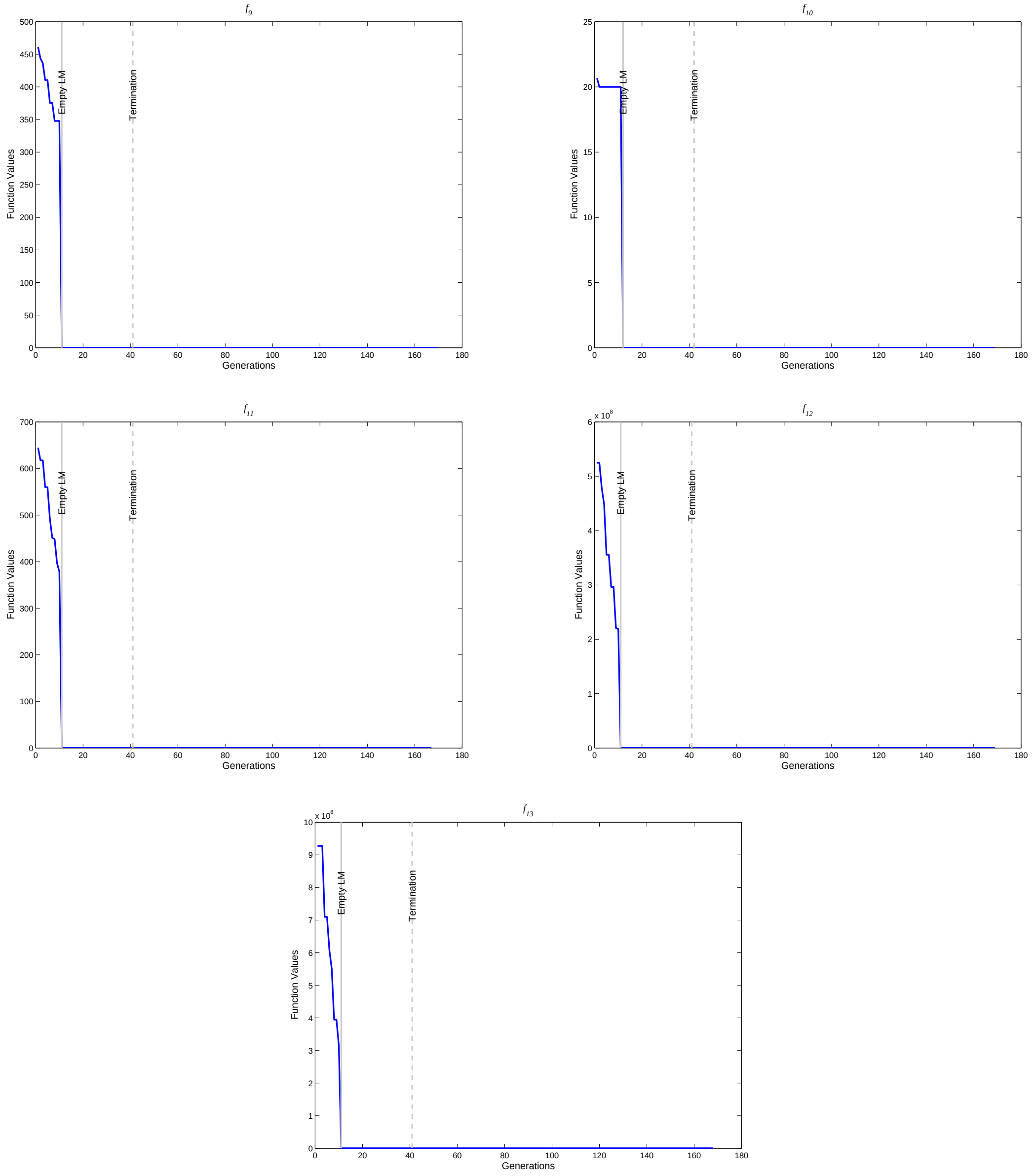


Fig. 9: Automatic Termination Performance Using LM (f_9 – f_{13})

K. *Griewank Function* (f_{11})

Definition: $f_{11}(\mathbf{x}) = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right) + 1.$
Search space: $-600 \leq x_i \leq 600,\ i = 1, \ldots, n.$
Global minimum: $\mathbf{x}^* = (0, \ldots, 0), f_{11}(\mathbf{x}^*) = 0.$

L. *Levy Functions* (f_{12}, f_{13})

Definition:

$$f_{12}(\mathbf{x}) = \frac{\pi}{n}\{10\sin^2(\pi y_1) + \sum_{i=1}^{n-1}\left[(y_i-1)^2(1+10\sin^2(\pi y_i+1))\right] + (y_n - 1)^2\} + \sum_{i=1}^{n} u(x_i, 10, 100, 4),\ y_i = 1 + \frac{x_i - 1}{4},\ i = 1, \ldots, n.$$

$$f_{13}(\mathbf{x}) = \frac{1}{10}\{\sin^2(3\pi x_1) + \sum_{i=1}^{n-1}\left[(x_i-1)^2(1+\sin^2(3\pi x_i+1))\right] + (x_n - 1)^2(1 + \sin^2(2\pi x_n)) + \sum_{i=1}^{n} u(x_i, 5, 100, 4),$$

$$u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m, & x_i > a; \\ 0, & -a \leq x_i \leq a; \\ k(-x_i - a)^m, & x_i < a. \end{cases}$$

Search space: $-50 \leq x_i \leq 50,\ i = 1, \ldots, n.$
Global minimum: $\mathbf{x}^* = (1, \ldots, 1),\ f_{12}(\mathbf{x}^*) = f_{13}(\mathbf{x}^*) = 0.$

REFERENCES

[1] Thomas Back, David B Fogel, and Zbigniew Michalewicz. *Handbook of evolutionary computation*. IOP Publishing Ltd., 1997.

[2] James Edward Baker. Adaptive selection methods for genetic algorithms. In *Proceedings of an International Conference on Genetic Algorithms and their applications*, pages 101–111. Hillsdale, New Jersey, 1985.

[3] Swagatam Das, Ajith Abraham, Uday K Chakraborty, and Amit Konar. Differential evolution using a neighborhood-based mutation operator. *Evolutionary Computation, IEEE Transactions on*, 13(3):526–553, 2009.

[4] Marco A Montes De Oca, Thomas Stützle, Mauro Birattari, and Marco Dorigo. Frankenstein's pso: a composite particle swarm optimization algorithm. *Evolutionary Computation, IEEE Transactions on*, 13(5):1120–1132, 2009.

[5] Andries P. Engelbrecht. *Computational Intelligence: An Introduction*. John Wiley - Sons, Chichester, England, 2003.

[6] Matthew S Gibbs, Holger R Maier, Graeme C Dandy, and John B Nixon. Minimum number of generations required for convergence of genetic algorithms. In *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on*, pages 565–572. IEEE, 2006.

[7] Fred Glover and Gary A Kochenberger. *Handbook of metaheuristics*. Springer Science & Business Media, 2003.

[8] Nikolaus Hansen. The cma evolution strategy: a comparing review. In *Towards a new evolutionary computation*, pages 75–102. Springer, 2006.

[9] Pierre Hansen, Nenad Mladenović, and José A Moreno Pérez. Variable neighbourhood search: methods and applications. *Annals of Operations Research*, 175(1):367–407, 2010.

[10] Abdel-Rahman Hedar and Masao Fukushima. Hybrid simulated annealing and direct search method for nonlinear unconstrained global optimization. *Optimization Methods and Software*, 17(5):891–912, 2002.

[11] Abdel-Rahman Hedar and Masao Fukushima. Minimizing multimodal functions by simplex coding genetic algorithm. *Optimization Methods and Software*, 18(3):265–282, 2003.

[12] Abdel-Rahman Hedar and Masao Fukushima. Heuristic pattern search and its hybridization with simulated annealing for nonlinear global optimization. *Optimization Methods and Software*, 19(3-4):291–308, 2004.

[13] Abdel-Rahman Hedar and Masao Fukushima. Tabu search directed by direct search methods for nonlinear global optimization. *European Journal of Operational Research*, 170(2):329–349, 2006.

[14] Abdel-Rahman Hedar, Bun Theang Ong, and Masao Fukushima. Genetic algorithms with automatic accelerated termination. *Department of Applied Mathematics and Physics, Kyoto University, Tech. Rep*, 2, 2007.

[15] Francisco Herrera, Manuel Lozano, and Daniel Molina. Continuous scatter search: an analysis of the integration of some combination methods and improvement strategies. *European Journal of Operational Research*, 169(2):450–476, 2006.

[16] Francisco Herrera, Manuel Lozano, and Jose L. Verdegay. Tackling real-coded genetic algorithms: Operators and tools for behavioural analysis. *Artificial intelligence review*, 12(4):265–319, 1998.

[17] Brijnesh J Jain, Hartmut Pohlheim, and Joachim Wegener. On termination criteria of evolutionary algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO-2001, San Francisco, CA, Morgan Kaufmann Publishers*, 2001.

[18] Amit Konar. *Computational intelligence: principles, techniques and applications*. Springer Science & Business Media, 2006.

[19] Manuel Laguna and Rafael Martí. Experimental testing of advanced scatter search designs for global optimization of multimodal functions. *Journal of Global Optimization*, 33(2):235–255, 2005.

[20] Chang-Yong Lee and Xin Yao. Evolutionary programming using mutations based on the lévy probability distribution. *Evolutionary Computation, IEEE Transactions on*, 8(1):1–13, 2004.

[21] Yiu-Wing Leung and Yuping Wang. An orthogonal genetic algorithm with quantization for global numerical optimization. *Evolutionary Computation, IEEE Transactions on*, 5(1):41–53, 2001.

[22] Jing J Liang, A Kai Qin, Ponnuthurai Nagaratnam Suganthan, and S Baskar. Comprehensive learning particle swarm optimizer for global optimization of multimodal functions. *Evolutionary Computation, IEEE Transactions on*, 10(3):281–295, 2006.

[23] Hui Liu, Zixing Cai, and Yong Wang. Hybridizing particle swarm optimization with differential evolution for constrained numerical and engineering optimization. *Applied Soft Computing*, 10(2):629–640, 2010.

[24] Jose A Lozano. *Towards a new evolutionary computation: Advances on estimation of distribution algorithms*, volume 192. Springer Science & Business Media, 2006.

[25] Manuel Lozano, Francisco Herrera, Natalio Krasnogor, and Daniel Molina. Real-coded memetic algorithms with crossover hill-climbing. *Evolutionary computation*, 12(3):273–302, 2004.

[26] M Hadi Mashinchi, Mehmet A Orgun, and Witold Pedrycz. Hybrid optimization with improved tabu search. *Applied Soft Computing*, 11(2):1993–2006, 2011.

[27] Nenad Mladenović, Milan Dražić, Vera Kovačevic-Vujčić, and Mirjana Čangalović. General variable neighborhood search for the continuous optimization. *European Journal of Operational Research*, 191(3):753–770, 2008.

[28] Pablo Moscato. Memetic algorithms: A short introduction. In *New ideas in optimization*, pages 219–234. McGraw-Hill Ltd., UK, 1999.

[29] Quang Huy Nguyen, Yew-Soon Ong, and Meng Hiot Lim. A probabilistic memetic framework. *Evolutionary Computation, IEEE Transactions on*, 13(3):604–623, 2009.

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

[31] Yew Soon Ong and Andy J Keane. Meta-lamarckian learning in memetic algorithms. *Evolutionary Computation, IEEE Transactions on*, 8(2):99–110, 2004.

[32] Yew-Soon Ong, Meng-Hiot Lim, Ning Zhu, and Kok-Wai Wong. Classification of adaptive memetic algorithms: a comparative study. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 36(1):141–152, 2006.

[33] Kenneth Price, Rainer M Storn, and Jouni A Lampinen. *Differential evolution: a practical approach to global optimization*. Springer Science & Business Media, 2006.

[34] A Kai Qin, Vicky Ling Huang, and Ponnuthurai N Suganthan. Differential evolution algorithm with strategy adaptation for global numerical optimization. *Evolutionary Computation, IEEE Transactions on*, 13(2):398–417, 2009.

[35] Krzysztof Socha and Marco Dorigo. Ant colony optimization for continuous domains. *European journal of operational research*, 185(3):1155–1173, 2008.

[36] Jasper Vrugt, Bruce Robinson, James M Hyman, et al. Self-adaptive multimethod search for global optimization in real-parameter spaces. *Evolutionary Computation, IEEE Transactions on*, 13(2):243–259, 2009.

[37] Xin Yao, Yong Liu, and Guangming Lin. Evolutionary programming made faster. *Evolutionary Computation, IEEE Transactions on*, 3(2):82–102, 1999.