Proceedings of the 7th International Conference on Genetic Algorithms (ICGA-97), pp. 238-245

A Real Coded Genetic Algorithm with an Explorer and an Exploiter Populations

Shigeyoshi Tsutsui[*1], Ashish Ghosh[*2], David Corne[*3] and Yoshiji Fujimoto[*4]

*1 Department of Management and Information Science, Hannan University, 5-4-33 Amamihigashi, Matsubara, Osaka 580, Japan, tsutsui@hannan-u.ac.jp

*2 Machine Intelligence Unit, Indian Statistical Institute, 203 B. T. Road, Calcutta 700035, INDIA, ash@isical.ernet.in

*3 Department of Computer Science, The University of Reading, Whiteknights, PO Box 225, Reading Berkshire, UK, D.W.Corne@reading.ac.uk

*4 Department of Applied Mathematics and Informatics, Faculty of Science and Technology, Ryukoku University, 1-5 Yokoya, Seta Ooe, Ohtsu, Shiga 520-21, JAPAN, fujimoto@math.ryukoku.ac.jp

Abstract

We introduce the concept of a bi-population scheme for real-coded GAs (bGAs) consisting of an explorer sub-GA and an exploiter sub-GA. The explorer sub-GA mainly performs global exploration of the search space, and incorporates a restart mechanism to help avoid being trapped at local optima. The exploiter sub-GA performs exploitation of fit local areas of the search space around the neighborhood of the best-so-far solution. Two types of bGA are presented, aimed at addressing different classes of problems. We also explore a method for adaptive load balancing between the two sub-GAs within a bGA, which uses knowledge of the number of restarts occurring in the explorer sub-GA. The proposed technique exhibits performance significantly superior to standard GAs on two complex highly multimodal problems.

1 Introduction

A key aspect of a Genetic Algorithm (GA) is that it maintains a population of candidate solutions that evolve over time (Goldberg, 1989; Grefenstette, 1992). The population allows the GA to continue to explore a number of regions of the search space that appear to be associated with high performance solutions. As early exploration of the search space gradually gives way to exploitation of a small number of fit regions later on in the search process, conventional GAs tend to face difficulty in solving certain kinds of problem, such as highly multimodal functions, functions with high levels of epistasis, and deceptive functions (Goldberg, 1989; Whitley, 1991). On such problems, typical premature *takeover* of exploitation occurs in conventional GAs, before the early exploration stage has had a chance to set the population in the right direction. Many kinds of modified GA have been proposed to address this and related issues, such as CHC (Eshelman, 1991), mGA (Goldberg et al., 1990), delta coding (Mathias and Whitley, 1995) and fGA (Tsutsui et al., 1993; 1997b). These efforts have mainly been applicable to binary or Gray coded chromosomes.

In recent years, several researchers have concentrated on using real-valued genes. It is reported that, for some problems, real-valued encodings and associated techniques outperform conventional bit string approaches (Davis, 1991; Eshelman and Schaffer, 1993; Wright, 1991; Janikow and Michalewicz, 1991; Radcliffe, 1991). Although real-coded GAs show promise for some problems, they also find it hard to solve highly epistatic multimodal problems. As with any GA, one approach towards dealing with this issue is to concentrate on better ways to balance the degrees of exploration and exploitation at different stages of the search process.

One novel approach to this problem is to use separate populations for exploration and exploitation. Preliminary ideas along these lines were reported in Tsutsui et al (1997a), which presented a bi-population GA scheme (bGA). The bGA has two populations; one of these populations, the explorer sub-GA, tries mainly to explore the whole search space and maintain a useful degree of global diversity, while the other population, the exploiter sub-GA, exploits the neighborhood of the best solution obtained so far. This paper reports on continuing experimentation with this idea in a more systematic and elaborate manner. We report on the development of two slightly different versions of the bGA scheme, aimed at dealing with different kinds of application. The scheme is shown to be successful in finding global optima for two difficult multimodal complex function optimization problems.

The paper is organized as follows. We describe the basic evolutionary algorithm model, as used in the two sub-GAs of the bGA, in Section 2. In Section 3, two types of bGA model are introduced. Then, in Section 4 empirical results and their analysis are given. Finally, concluding remarks are made in Section 5.

2 Basic Evolutionary Model

The bGA is of course not restricted to a particular underlying evolutionary model, but we will briefly describe here the model used in this study. Evolution in both the explorer and the exploiter sub-GAs is similar to that of CHC (Eshelman, 1991) and (μ+λ)-ES (Schwefel, 1981). Let the population size be N, and let the population at time t be represented by $P(t)$. The population $P(t+1)$ is produced as follows: A set of $N/2$ pairs of chromosomes is copied from $P(t)$ to $I(t)$ (intermediate population at time t). Genetic operators are applied to the chromosomes in $I(t)$, generating N offspring which are placed in $I'(t)$. Rank-based selection is then used to select N individuals from the $2N$ in $P(t)$ and $I'(t)$ to form $P(t+1)$. Elitism is employed to make sure that the best solution so far is always included in $P(t+1)$.

3 Models of Bi-population GAs

In this section we describe the models for two slightly different bi-population GAs, the relations and differences between them, and the problem types they are aimed to solve. Also we discuss the relationship of the present work to existing literature.

3.1 Components of bGAs

As illustrated in Fig. 1, a bGA consists of three components: (i) an explorer sub-GA, (ii) an exploiter sub-GA and (iii) a monitor. The explorer sub-GA explores the whole search space, and the exploiter sub-GA searches the neighborhood

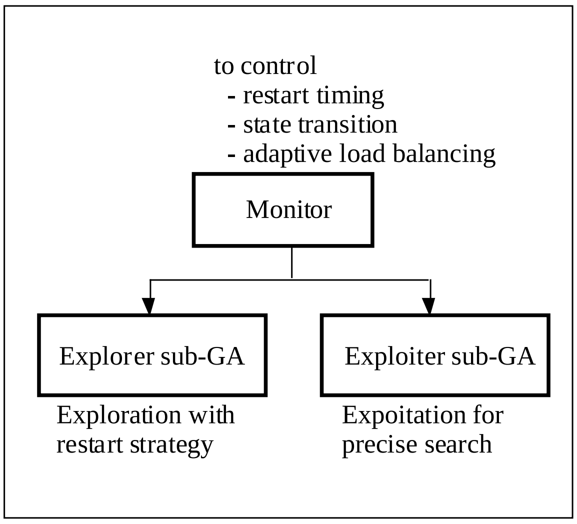

to control
- restart timing
- state transition
- adaptive load balancing
Monitor
Explorer sub-GA
Exploiter sub-GA
Exploration with restart strategy
Expoitation for precise search

Fig. 1 Block diagram of a bi-population GA

of the best solution obtained so far. If certain pre-specified convergence conditions are satisfied before completion of search, the explorer sub-GA is *restarted*. In general, the bGA can be in different well defined states, depending on the current activity of the two sub-GAs. The monitor observes the current search process and makes decisions about the following: (a) when to restart the explorer sub-GA, (b) transitions between different bGA states, and (c) adaptive load balancing between the sub-GAs.

3.2 Types of bGAs and Their Dynamics

In this subsection we consider two types of bGA models aimed at different types of application.

3.2.1 Type I

Fig. 2 shows the state transition diagram of a Type I bGA. When search starts, only the explorer sub-GA is used. We call this state *E0-STATE*. When the pre-specified restart condition is satisfied, all the individuals of the explorer sub-GA are copied to the exploiter sub-GA and the explorer sub-GA is restarted after randomly re-initializing all the individuals and incrementing the *restart counter* (which is initially set to zero). Thus the best so far solution is maintained in the exploiter sub-GA. The exploiter sub-GA then continues search with the old population members. We call this state *EE-STATE*.

In EE-STATE two sub-GAs are run concurrently. During this concurrent running process, whenever the restart condition is satisfied, the explorer sub-GA restarts and the

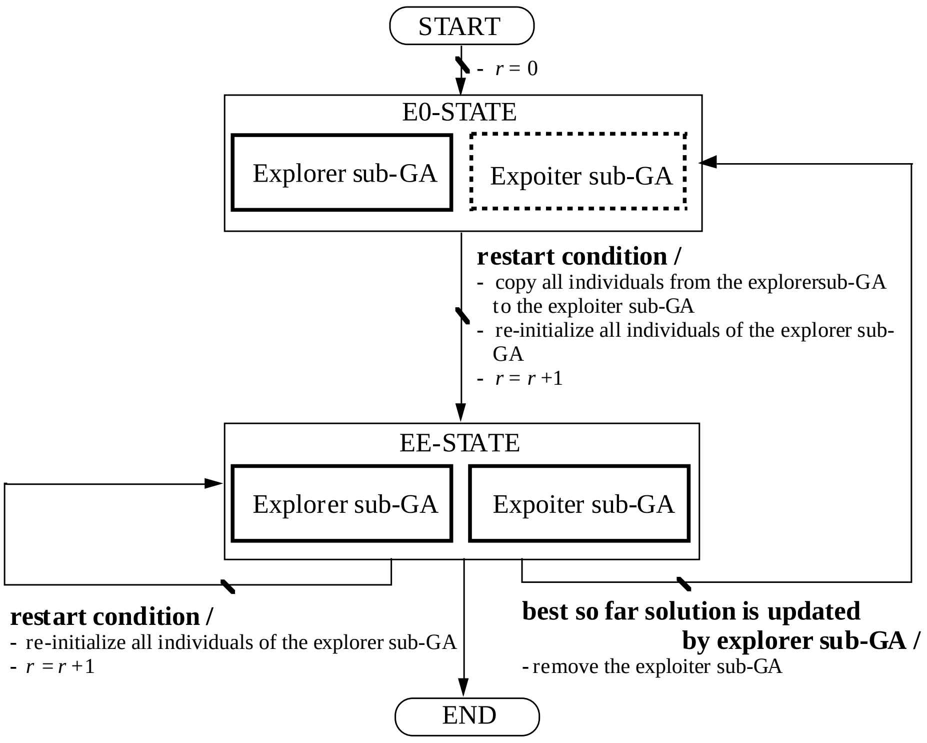

START
- r = 0
E0-STATE
Explorer sub-GA
Expoiter sub-GA
restart condition /
- copy all individuals from the explorersub-GA to the exploiter sub-GA
- re-initialize all individuals of the explorer sub-GA
- r = r +1
EE-STATE
Explorer sub-GA
Expoiter sub-GA
restart condition /
- re-initialize all individuals of the explorer sub-GA
- r = r +1
best so far solution is updated by explorer sub-GA /
- remove the exploiter sub-GA
END

Fig. 2 State transition diagram of Type I bGA

restart counter r is incremented by one. On the other hand, for situations where the best solution of the explorer sub-GA is better than the best solution of the exploiter sub-GA, the current exploiter sub-GA becomes useless and the state transition from EE-STATE to E0-STATE occurs.

These state transitions continue until the termination condition is satisfied. In the present work, restart of the explorer sub-GA is done only if the current best fitness value is improved by more than the pre-specified value Δf (say) during the last K_h (say) generations.

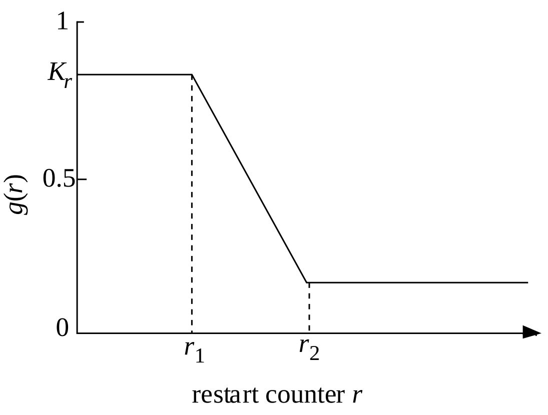


Fig. 3 Fraction of time given to explorer sub-GA

As two sub-GAs run concurrently in the EE-STATE, questions naturally arise about computational time-sharing, whether or not we assume a serial implementation. Intuitively, at the initial stage of the search more time should be given to the explorer sub-GA, and towards the final stages of search, more time should be allowed to the exploiter sub-GA.

We can use the restart counter r as a parameter to help implement these intuitions as a model of adaptive load balancing between the two sub-GAs. When r is small (there have not been many restarts) we give more time to the explorer sub-GA; when r is larger (the explorer has been restarted several times, and so it is perhaps a good idea to start concentrating on exploitation) we give more time to the exploiter sub-GA. Let $g(r)$ be the fraction of time given to the explorer sub-GA, and so 1-$g(r)$ is given to the exploiter sub-GA. We use the following function for calculating $g(r)$.

$$g(r) = \begin{cases} K_r & \text{if } r < r_1 \\ K_r - \dfrac{2K_r - 1}{r_2 - r_1} \times (r - r_1) & \text{if } r_1 \le r \le r_2 \\ 1 - K_r & \text{otherwise,} \end{cases} \qquad (1)$$

where $0.5<K_r<1.0$ and $1<r_1<r_2$. As shown in Fig. 3, when r is less than or equal to r_1, the bGA searches mainly in the exploration mode. When r exceeds r_2, the algorithm searches

mainly in exploitation mode. When r is between r_1 and r_2, time is shared by both the sub-GAs depending on the value of r.

The present model is suitable for problems where we have no knowledge about the optimal or target value.

3.2.2 Type II

There are many real-world applications where we either know the optimal fitness value, or we wish to reach a known target value. For example, in curve-fitting applications we are aiming to minimize error, and so targeting a fitness value of 0. Or, in design optimization, we are aiming to determine design parameters which satisfy known specifications (Tsutsui et al., 1993, 1997b). The bGA would presumably benefit greatly from knowledge of a target value, using this knowledge to better control and direct state transitions and adaptive load balancing. Fig. 4 shows the schematic representation of a Type II bGA which makes use of this knowledge. The target fitness value is used to avoid *over-exploration* when the search is already near enough to the target to be able to reach it (hopefully) by exploitation. Let $f_{\text{threshold}}$ be a specified threshold value used to test the present status of the search process. Now if the best so far performance value is less than $f_{\text{threshold}}$, we assume that the search is proceeding in the right direction and there is no further need for exploration. In this case, the state transition to *0E-STATE* occurs. In 0E-STATE, only the exploiter sub-GA is used and search proceeds until other termination conditions are satisfied. The load balancing strategy between the two sub-GAs is identical with that of the Type I bGA, but in future work we will examine the use of $f_{\text{threshold}}$ within

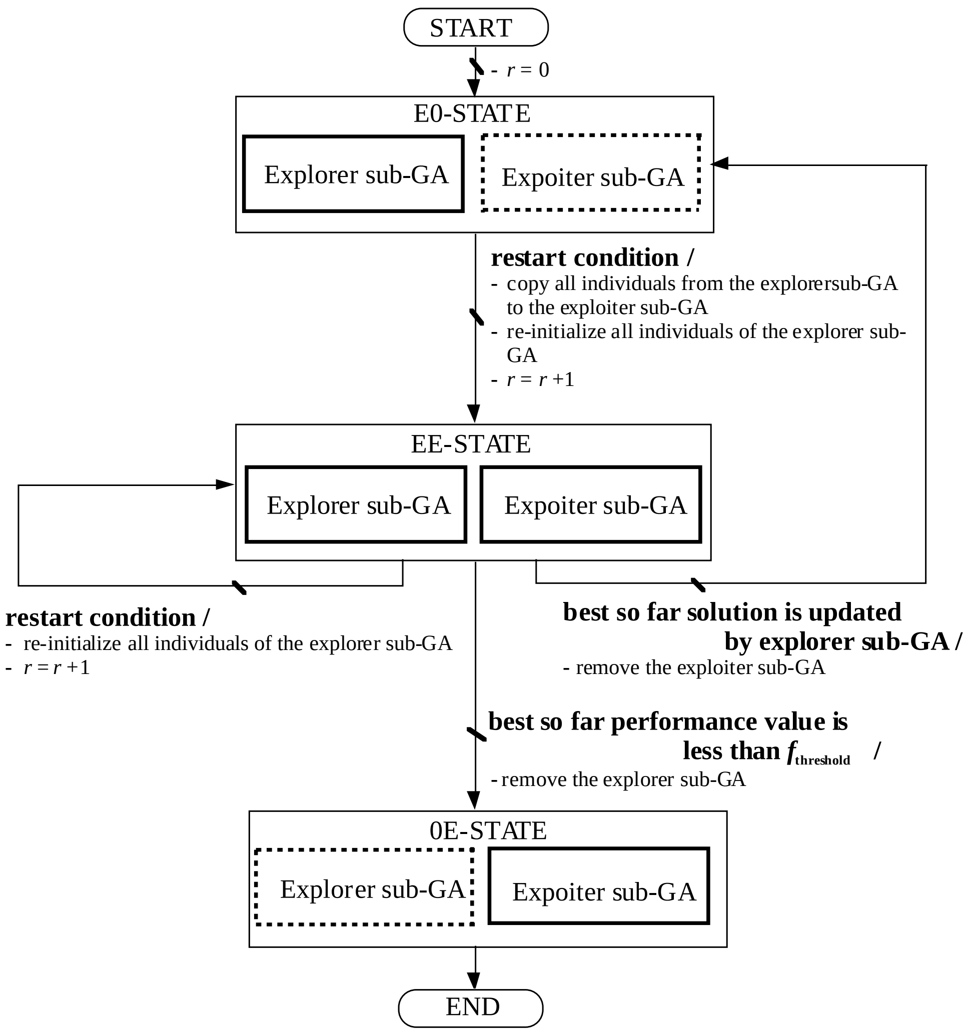

START
- r = 0
E0-STATE
Explorer sub-GA
Expoiter sub-GA
restart condition /
- copy all individuals from the explorersub-GA to the exploiter sub-GA
- re-initialize all individuals of the explorer sub-GA
- r = r +1
EE-STATE
Explorer sub-GA
Expoiter sub-GA
restart condition /
- re-initialize all individuals of the explorer sub-GA
- r = r +1
best so far solution is updated by explorer sub-GA /
- remove the exploiter sub-GA
best so far performance value is less than f threshold /
- remove the explorer sub-GA
0E-STATE
Explorer sub-GA
Expoiter sub-GA
END

Fig. 4 State transition diagram of Type II bGA

the load balancing strategy.

3.3 Relation to Existing Work

Restart strategies have been discussed elsewhere (Eshelman, 1991; Mathias and Whitley, 1995). The delta coding method used by Mathias and Whitley (1995) is an iterative genetic search strategy that sustains search by periodically re-initializing the population. It also remaps the search hyperspace with each iteration and it is reported that it shows good performance especially when used with Gray coding. The CHC method by Eshelman (1991) is a safe search strategy where restart of the search process is done if it gets stuck at local optima by re-initializing the population with individuals generated by mutating the best solution obtained so far (also keeping the best one). In another study, Tsutsui et al. (1993, 1997b) reported a search space division and restart mechanism so as to avoid getting stuck in local traps.

Although the use of a restart strategy is a feature of the bGA, its main purpose is to maintain a suitable balance between exploration and exploitation during the search process by means of two populations. Multiple population models have been described before (Tanese 1989; Whitley et al. 1990; Gorges-Schleuter, 1991), and are often the choice of underlying GA implementation in application oriented papers, however the populations are not assigned different roles. In contrast, the most closely related previous work appears to be that on *memetic* algorithms (Moscato and Norman, 1992; Radcliffe and Surry, 1994) and other hybridizations of a GA with local search. In these approaches, GA exploration is balanced by exploitation of every chromosome via local search, essentially incorporating local search into the evaluation function. In a memetic algorithm, exploration and exploitation remain tightly interlocked and concurrent, with overall load balancing between these two stages of search and still subject to the vagaries of standard GA dynamics. The bGA can be seen as a (rather radical) variation on such an idea, in which only very few chromosomes are subject to local search, and in which the local search is actually performed by a GA. In addition, a monitoring mechanism makes careful decisions about concentration of effort between the two stages.

4 Empirical Study

In this section, we show some experimental results to confirm the utility of the proposed scheme using two highly multimodal test functions. Initially, we will first discuss the exact models of the explorer and exploiter sub-GAs.

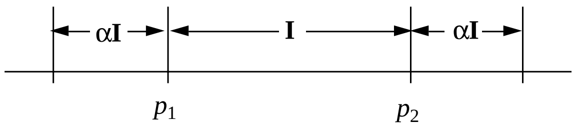
4.1 Explorer and Exploiter sub-GA Models

The basic evolutionary model used in this study is described in Section 2. Here, we describe the genetic operators and the control parameters for the explorer and the exploiter sub-GAs.

Several crossover operators have been proposed in the literature (Wright, 1991; Eshelman and Schaffer, 1993; Ono, et al., 1996) for real coded GAs. BLX-α (Eshelman and Schaffer, 1993) as shown in Fig. 5 is one of them and is reported to work well on a wide range of problems. In this study BLX-α is commonly used for both the sub-GAs. BLX-α creates children of parents p_1 and p_2 which lie on the line joining the parents, but not between them, as shown in Fig. 5.

With regard to mutation, several mutation methods such as creep mutation (Davis, 1991) have been proposed. For the explorer sub-GA, a *coarse-grained mutation* method, and for the exploiter sub-GA a *fine-grained mutation* method, seem to be suitable choices. However, to use different mutation methods for different sub-GAs will increase the number of control parameters to be tuned. Instead, we use the following *relative mutation* method. In this method, the distance | I | (see Fig. 5) between parents is multiplied by M (M>1.0) with mutation probability. This allows coarse mutation in the explorer sub-GA because the distance between parents will be relatively large, and fine mutation in the exploiter GA where distances between parents will be relatively small.

For efficient exploitation, intuition suggests that the population size of the exploiter sub-GA should be rather less than N, which is the population size of the explorer sub-GA. We therefore examined the effects of different population sizes for the exploiter sub-GA, namely: N, N/2, N/4 and N/



BLX-α uniformly picks new individuals with values that lie in [I-αI, I+αI], where p_1 and p_2 are two parents

Fig. 5 BLX-α

10.

The following parameters and experimental conditions were used. The crossover operator was BLX-α with α = 0.5. Mutation probability was 0.1 and M = 3.0. The population size of the explorer sub-GA (N) is fixed to 100. As regards the restart conditions of the explorer sub-GA, K_h was 50, and Δf was 0.1. The number of simulations used for each experiment was 10. The values of K_r, r_1 and r_2 of Equation (1) were 0.7, 5 and 6, respectively.

4.2 Test Functions

The test functions used are as follows:

a) FMS (Frequency Modulation Sounds) parameter identification problem: f_{fms} (Tsutsui et al., 1993, 1997b). Here the problem is to specify 6 parameters (a_1, w_1, a_2, w_2, a_3, w_3) of the FM sound model represented by

$$y(t) = a_1\sin(w_1 t\theta + a_2\sin(w_2 t\theta + a_3\sin(w_3 t\theta))), \quad (2)$$

with $\theta = 2\pi/100$. The function f_{fms} is defined as the summation of square errors between the evolved data and the model data as follows:

$$f_{\text{fms}} = \sum_{t=0}^{100}(y(t) - y_0(t))^2, \quad (3)$$

where the model data are given by the following equation:

$$y_0(t) = 1.0 \times \sin(5.0t\theta - 1.5 \times \sin(4.8t\theta + 2.0 \times \sin(4.9t\theta))). \quad (4)$$

Each parameter is in the range -6.4 to 6.35. The maximum number of trials was set to 160,000. This function is a highly complex multimodal (Fig. 6) function having strong epistasis, with minimum value f_{fms} = 0. $f_{\text{threshold}}$=0.0001 is assumed.

b) Griewank function: f_{Griewank} (Torn and Zilmskas, 1989). This function is defined as follows:

$$f_{\text{Griewank}} = \sum_{i=1}^{10} x_i^2/4000 - \prod_{i=1}^{10}\cos(x_i/\sqrt{i}) + 1. \quad (5)$$

parameter x_i is in the range -512 to 511. Maximum number of trials was set to 100,000. This function has its global minimum f_{Griewank} = 0 at x_i = 0, i = 1, ..., 10. This function is also highly multimodal and epistatic (Fig. 7). $f_{\text{threshold}}$=0.0001 is assumed here also.

4.3 Results and Analysis

Fig. 8 shows the mean best functional values achieved for the different approaches on f_{fms} achieved by the two types of bGA. A single pool GA without restart (non-bGA) is also tested for baseline comparison. Comparing the bGAs and the non-bGA we can see that the bGAs show much better performance. Comparing Type I and Type II bGAs, Type II bGAs showed better performance than Type I bGAs. This confirms the utility of Type II bGAs when we solve problems with a known target value. When the performance value of Type II bGAs becomes within the $f_{\text{threshold}}$ value, it rapidly converges and approaches the global optimum value with high precision, since it can concentrate on exploitation with the exploiter sub-GA only. As regard to the population size of the exploiter sub-GA, N/2 seemed to deliver the best

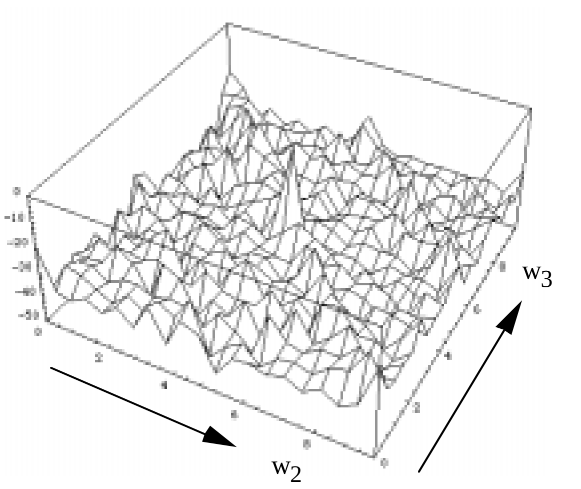


Fig. 6 Landscape of function f_{fms} on w_2-w_3 space

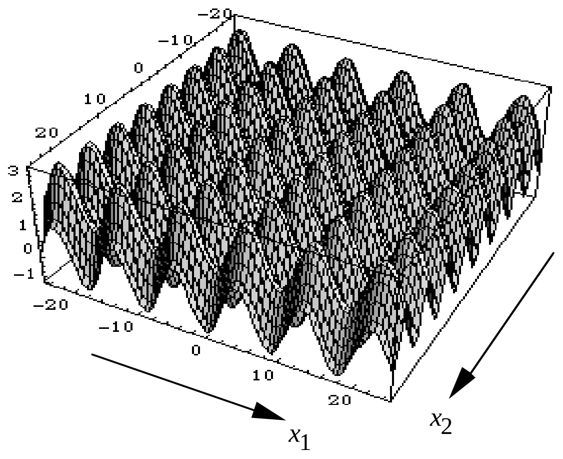


Fig. 7 Landscape of function f_{Griewank} on x_1-x_2 space

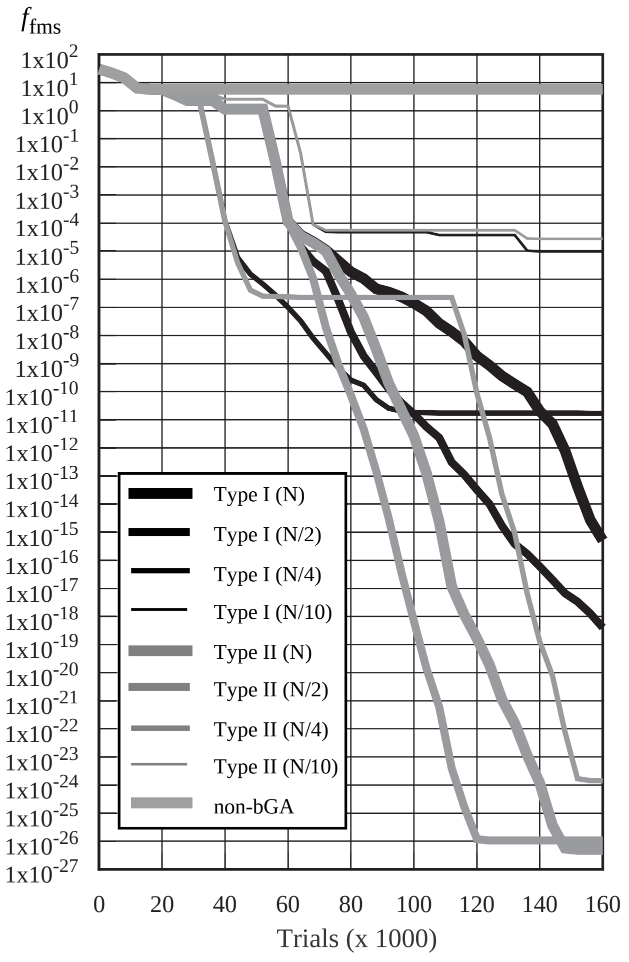


Fig. 8 Mean best functional value of f_{fms}

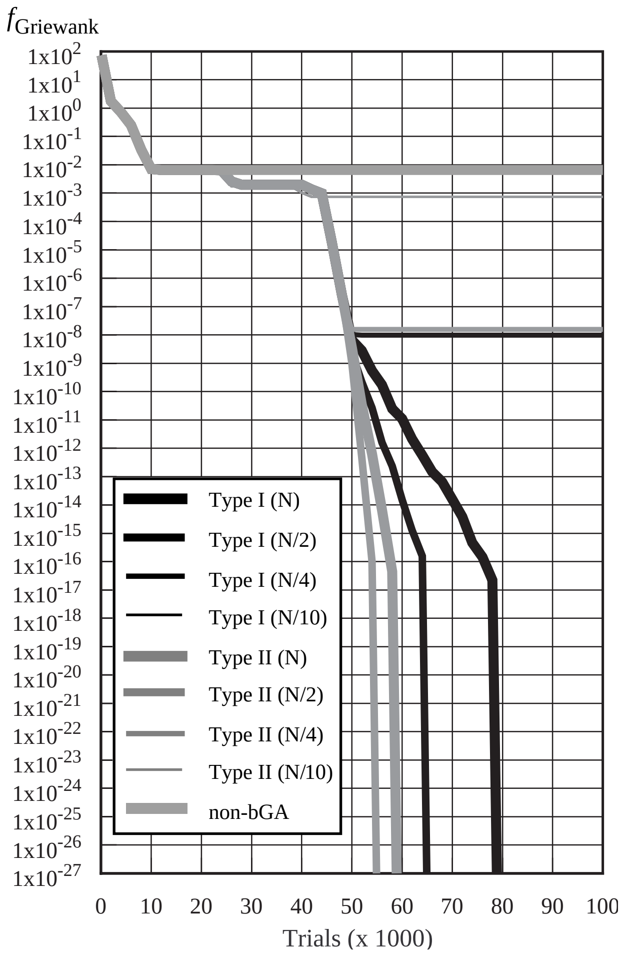


Fig. 9 Mean best functional value of f_{Griewank}

results. Smaller population sizes for the exploiter sub-GA seemed very prone to entrapment in local optima, particularly in the f_{fms} case. This is probably because f_{fms} has many local optima around the global optimal point (see Fig. 6), rendering local search with a rather small population size rather vulnerable to failure.

Fig. 9 shows the mean best functional values achieved for the different approaches on f_{Griewank}. For this function also, the non-bGA showed poorer performance. Since this function is little easier than f_{fms}, there is no big difference between the performance of Type I and Type II bGAs, although Type II bGAs were perhaps slightly better. As regards population size of the exploiter sub-GA, *N*/2 (= 50) again showed the best performance. Resizing the population to lesser values is seen to trap the search scheme to local optima again, corroborating the earlier findings.

5 Conclusions

The basic concept of a bi-population scheme for real-coded GAs which consists of an explorer sub-GA and an exploiter sub-GA is introduced. The main tasks for the sub-GAs are different; one mainly does exploration, and avoids being trapped in local optima by means of restart mechanism; and the other does exploitation by concentrating within the neighborhood of the best so far solution. An adaptive load balancing mechanism which allocates time between the explorer and exploiter sub-GAs is performed by a monitor which supervises the search process. The monitor also controls the state transitions during the search process. Two different bi-population GAs are explored and relations and differences between them and the problem types they are aimed to solve are briefly presented.

The effectiveness of the proposed technique is shown by solving two complex multimodal function optimization problems, although further experimentation is needed to better test the performance of the bGA relative to other GAs, hillclimbing, and a standard GA with restart, for example. The explorer sub-GA with a larger population size and the exploiter sub-GA with a smaller population size showed the best performance. In the case, the population sizes were 100 and 50 respectively, but optimal sizes are of course likely to depend very much on the problem at hand and aspects of global and local fitness landscape structure.

Many opportunities for further research related to the present topic exist. More "smart" heuristics to reach 0E-STATE faster so as to save more time are under investigation. Strategies to avoid the explorer sub-GA getting trapped by strong attractors multiple times, employing traditional local search techniques for the exploiter sub-GA, are yet to be tried. Evaluating the effectiveness of bGAs on real problems, comparing them with other multi-population based schemes, and extending them for permutation problems also remain to be investigated. Co-evolution of the two sub-GAs will also yield ideas for future research.

Acknowledgments

The authors thank the anonymous reviewers for constructive and in-depth comments and criticism. This research is partially supported by the Ministry of Education, Science, Sports and Culture under Grant-in-Aid for Scientific Research on Priority Areas number 264-08233105.

References

Davis, L. (Ed) (1991). *Handbook of genetic algorithms*, Van Nostrand Reinhold, New York.

Eshelman, L. J. (1991). The CHC adaptive search algorithm: how to have safe search when engaging in nontraditional genetic recombination, *Foundations of Genetic Algorithms*, Morgan Kaufmann Publishers, pp.265-283.

Eshelman, L. J. and Schaffer, J. D. (1993). Real-coded genetic algorithms and interval-schema, *Foundations of Genetic Algorithms 2*, Morgan Kaufmann Publishers, pp.187-202.

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization and machine learning*, Addison-Wesley.

Goldberg, D. E., Deb, K. and Korb, B. (1990). Messy genetic algorithms revisited: studies in mixed size and scale, *Complex Systems*, Vol.4, pp. 415-444.

Grefenstette, J. J. (1992). Genetic algorithms for changing environments, *Proceedings of the Second International Conference on Parallel Problem Solving from Nature (PPSN II)*, Springer-Verlag, pp. 137-144.

Janikow, C. Z. and Michalewicz, Z. (1991). An experimental comparison of binary and floating point representations in genetic algorithms, *Proceedings of the Fourth International Conference on Genetic Algorithms (ICGA 91)*, Morgan Kaufmann Publishers, pp. 31-36.

Mathias, K. E. and Whitley, L. D. (1995). Changing representation during search: a comparative study of delta coding, *Evolutionary Computation*, Vol. 2, No. 3, pp. 249-278.

Moscato, P. and Norman, M. G. (1992). A memetic approach for the traveling salesman problem -- implementation of a computational ecology for combinatorial optimisation on message-passing systems, *Proceedings of the International Conference on Parallel Computing and Transputer Applications*, IOS Press, Amsterdam.

Ono, I, Yamamura, M. and Kobayashi, S. (1996). A genetic algorithm with characteristic preservation for function optimization, *Proceedings. of the IIZUKA 96*, pp. 511-514.

Radcliffe, N. J. (1991). Equivalence class analysis of genetic algorithms, *Complex Systems*, Vol. 5, pp. 183-200.

Radcliffe, N. J. and Surry, P. D. (1994). Formal memetic algorithms, *Evolutionary Computing: Selected Papers, Fogarty (ed)*, Springer Lecture Notes in Computer Science Volume 865, pp. 1-16.

Schwefel, H.-P. (1981). *Numerical optimization of computer models*, John Wiley & Sons, New York.

Torn, A. and Zilmskas, A. (1989). *Global optimization*, Springer-Verlag.

Tsutsui, S. and Fujimoto, Y. (1993). Forking genetic algorithm with blocking and shrinking modes, *Proceedings of the Fifth International Conference on Genetic Algorithms (ICGA 93)*, Morgan Caufmann Publishers, pp. 206-213.

Tsutsui, S., Ghosh, A., Fujimoto, Y. and Corne, D. (1997a). A bi-population scheme for real-coded GAs: the basic concept, *Proceedings of the First International Workshop on Frontiers in Evolutionary Algorithms, Vol.* 1, pp. 39-42.

Tsutsui, S., Fujimoto, Y. and Ghosh, A. (1997b). Forking GAs: GAs with search space divisions schemes, *Evolutionary Computation* (to appear).

Wright, A. (1991). Genetic algorithms for real parameter optimization, *Foundation of Genetic Algorithms*, Morgan Kaufmann Publishers, pp.205-218.

Whitley, D. (1991). Fundamental principles of deception in genetic search, *Foundations of Genetic Algorithms*, Morgan Kaufmann Publishers, pp. 221-241.