World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:1, No:10, 2007

Development of Heterogeneous Parallel Genetic Simulated Annealing Using Multi-Niche Crowding

Z. G. Wang, M. Rahman, Y. S. Wong and K. S. Neo

***Abstract*—** In this paper, a new hybrid of genetic algorithm (GA) and simulated annealing (SA), referred to as GSA, is presented. In this algorithm, SA is incorporated into GA to escape from local optima. The concept of hierarchical parallel GA is employed to parallelize GSA for the optimization of multimodal functions. In addition, multi-niche crowding is used to maintain the diversity in the population of the parallel GSA (PGSA). The performance of the proposed algorithms is evaluated against a standard set of multimodal benchmark functions. The multi-niche crowding PGSA and normal PGSA show some remarkable improvement in comparison with the conventional parallel genetic algorithm and the breeder genetic algorithm (BGA).

***Keywords*—** Crowding, genetic algorithm, parallel genetic algorithm, simulated annealing.

I. Introduction

THE simple genetic algorithm (SGA) was firstly developed by Holland [1]. Owing to its ability to achieve global or near global optima, this algorithm has been applied to a large number of combinatorial optimization problems. However, searching for extrema in a multimodal space is different from locating the extremum of a unimodal function. When attempting to optimize a multimodal function, SGA tends to converge to a single solution, in some cases SGA only converges to an optimum in the local neighborhood.

Simulated annealing (SA) is an alternative to escape from a local optimum. However, there are many applications where multiple optima of optimization problems need to be found. Given multiple optimal solutions, users are able to choose the corresponding high-quality solutions based on their preference. These optimal solutions may also suggest innovative alternative solutions to practical problems. Only a limited amount of research has been conducted on the use of GA to locate multiple optima of a multimodal function, and the parallel subpopulation method is an effective way to solve this type of problems [2].

When applying GA to multimodal optimization problems, success in the application depends on the preservation of good individuals into the next generation and the maintenance of diversity of individual solutions in the search space. If GA cannot hold its diversity well before the global optimum is reached, it may prematurely converge to a local optimum. SA is an effective way to preserve good individuals into the next generation, and crowding strategy is an alternative approach to maintain the diversity in the population and postpone premature convergence.

This paper presents a hybrid algorithm of GA and SA, referred to as genetic simulated annealing (GSA). In the proposed method, SA is used to select the individuals for next generation and control the mutation rate. GSA is parallelized to find multiple optimal solutions for the multimodal functions. Notably, multi-niche crowding strategy is also used to maintain population diversity of parallel GSA (PGSA). The paper aims to demonstrate that multi-niche crowding PGSA is a powerful optimization strategy in comparison to other advanced search algorithms.

II. Related Work

A. Hybrid of GA and SA

GA and SA are both independently valid approaches toward problem solving with certain strengths and weaknesses. GA can begin with a population of solutions in parallel, but it suffers from poor convergence properties. By contrast, SA has better convergence properties if the starting temperature is sufficiently high and the temperature cooling rate is low. However, the higher temperature and the lower cooling rate reduce the performance of SA. In addition, parallelization cannot be easily exploited in SA.

Recently many researchers tried to combine GA and SA to provide a more powerful optimization method that has both good convergence control and efficient parallelization. Chen and Flan [3] had shown that the hybrid of GA and SA can perform better for ten difficult optimization problems than either GA or SA independently. Mahfoud and Goldberg [4] also introduced a GA and SA hybrid. Their hybrid runs SA procedures in parallel, which uses mutation as the SA neighborhood operator and incorporates crossover to reconcile solutions across the processors. A similar hybrid method of GA and SA was also used by Varanelli and Cohoon [5]. In addition, Chen *et al*. [6] also proposed a hybrid method, which maintained one solution per Processing Element (PE). Each PE accepted a visiting solution from other PEs for crossover

Manuscript received July 6, 2005.

Z. G. Wang is with the Department of Mechanical and Aeronautical Engineering, University of California, Davis, CA, 95616, USA (Tel: +1-530-752-8253; fax: +1-530-752-8253; e-mail: zgwang@ucdavis.edu).

M. Rahman, Y. S. Wong and K. S. Neo are with the Department of Mechanical Engineering, National University of Singapore, Singapore, 119260 (e-mail: mpemusta@nus.edu.sg, mpewys@nus.edu.sg, and mpeneoks@nus.edu.sg, respectively).

and mutation. For the selection process, the SA cooling schedule and system temperature were used to decide whether the new generated individual was accepted or not. In this method, they used the local selection of SA to replace the conventional selection process of GA. Recently; Hiroyasu *et al*. [7] proposed an algorithm involving several processors. In each processor, SA was employed. The genetic crossover was used to exchange information between individuals at fixed intervals. Based on parallel simulated annealing in [4], [7], Baydar [8] developed a parallel simulated annealing algorithm using the survival of the fittest method, and acceptable results were found with this algorithm.

B. Niching Strategies in GA

De Jong [9] used a scheme called crowding for the pre-selection technique. In crowding, selection and reproduction are the same as in the SGA, but replacement is different. In crowding, a group of C individuals is firstly selected randomly from the population, where C is called the crowding factor. Then, an individual is compared to every member of the selected group, and the most similar member of that group is replaced. Crowding is essentially a successive replacement strategy, which helps to maintain the population diversity and eliminate premature convergence. A number of means of implementing niching in GA has also been devised in [10].

Niching was introduced into GA primarily to maintain population diversity. Cedeno [11] indicated that multi-niche crowding (MNC) has the ability to converge simultaneously to multiple solutions by encouraging competition between individuals within the same locally optimal group. This objective is achieved by encouraging mating and replacement within members of the same niche while allowing for some competition for population slots among the niches. Further details of MNC are given in Section IV.

III. Parallel Genetic Simulated Annealing

A. Genetic Simulated Annealing

Each of the above approaches to hybridize GA and SA described in Section II.A has its own strengths, because some good characteristics of GA and SA are maintained when combining GA and SA together. In this paper, a new GA and SA hybrid, GSA, is presented.

After crossover and mutation for a couple of individuals, there are four chromosomes: two parents and two offspring. In conventional GA, two parents are replaced by their offspring. But in GSA, two chromosomes are chosen to form the next generation from these four individuals. The selection criterion is based on the fitness values of these four individuals. Individuals with higher fitness values have a greater probability of surviving into the next generation. Those with less fitness values are not necessarily discarded. Instead, a local selection strategy of SA is applied to select them with a probability related to the current temperature (as in simulated annealing). In this selection process, a Markov chain is executed, which is composed of two offspring. Four parameters (f_{best}, f_{worst}, T_t, f_i) are involved to describe this selection process:

f_{best} — the best fitness value of two parents;
f_{worst} — the worst fitness value of two parents;
f_i — the fitness value of one offspring (i= 1, 2);
T_t — control temperature;

During the course of the Markov chain at temperature T_t, the fitness value f_i (i =1, 2) of the trial chromosome is compared with f_{worst}. Chromosome i is accepted to replace the worst individual, if the following requirement is met:

$$\min\left\{1,\ e^{\frac{f_i - f_{worst}}{T_t}}\right\} \geq r$$

where r is a randomly generated number between 0 and 1.

If chromosome i is accepted, the worst chromosome and the best one are updated and then the course of the Markov chain continues until completion. After the implementation of the Markov chain, the best and the worst individuals are survived into the next generation.

In SGA, mutation simply changes the value for a particular gene with a certain probability. It helps to maintain the vast diversity of the population and also prevents the population from stagnating. However, at later stages, it increases the probability that good solutions will be destroyed. Normally, the mutation rate is set to a low value (e.g., 0.01) so that accumulated good candidates will not be destroyed. This negative effect of mutation has been eliminated for GSA, because the local selection of SA is applied after mutation, such that at the later stage, only better solutions are retained after mutation. Therefore, the initial value of mutation probability can be larger than the recommended values in [12]. In this study, the mutation probability p_m of GSA is initially set to a higher value, and a simple annealing process is then used to adjust p_m. After every certain generations, the mutation probability p_m is updated with $p_m \times \alpha$ until it reaches to a certain value, where α is the cooling rate of SA. Thus, at the initial stage, when manipulating the cooling schedule of SA properly, the initial higher temperature can ensure that parents will be replaced by their offspring after crossover and mutation whether they are much fitter or not. More importantly, the initial higher mutation probability is capable of improving population diversity greatly, which can eliminate the premature convergence problem of conventional GA. On the contrary, at the later stage the mutation probability and the temperature become lower, and the chances for the fitter parents to be replaced decrease greatly. In this way, the current best individuals may continue to remain in the next generation. Thus, the possibility of removing potentially useful individuals in the last generation because of the mutation operation can be reduced. The pseudo-code of GSA is illustrated in Fig. 1, where $P(t)$ is the population of individuals at generation t, and n is the string length of chromosome.

In addition, good parallelizable property of GA is applied to parallelize GSA, which will be explained in Section III.B.

Thus, GSA shows tighter coupling of GA and SA as SA controls a number of distinct GAs running in parallel.

1: $t = 0$
2: initialize $P(t)$ and temperature T_t
3: evaluate $P(t)$
4: **while** not termination-condition **do**
5: $t = t + 1$
6: select $P(t)$ from $P(t\text{-}1)$
7: select individuals for reproduction from $P(t)$
8: **repeat**
9: select two unused individuals P_1, P_2
10: crossover & mutation; generate two children C_1,C_2
11: evaluate C_1,C_2
12: **for all** $i = 1$ to 2 **do**
13: **if** $\min\{1, exp((f_i\text{-}f_{worst})/T_t)\} > random[0,1)$ **then**
14: accept C_i and replace the corresponding parent
15: update the new best and worst points
16: **end if**
17: **end for**
18: **until** all selected parents finish reproduction
19: $T_{t+1} = T_t \times \alpha;\ 0 < \alpha < 1$
20: **if** *the modulus of t divided by 10 == 0&&* $p_m > 1/n$ **then**
21: $p_m = p_m \times \alpha$
22: **end if**
23: **end while**

Fig. 1 Pseudo code of genetic simulated annealing

B. Parallel Genetic Simulated Annealing

There are several ways to parallelize GA [13], and the parallel GA (PGA) has been developed and successfully applied to optimize many practical problems [14]. According to the nature of the population structure and recombination mechanisms used, PGA can be classified into four categories: single-population master-slave PGA, coarse-grained PGA, fine-grained PGA and hierarchical hybrids [13]. In single-population master-slave PGA, there is a single population, and the evaluation of fitness values is distributed among several processors. The fine-grained PGA treats each individual as a separate breeding unit; and the individuals may mate with those selected from a small local neighborhood. Since the neighborhoods overlap, fit individuals will migrate through the population. The coarse-grained PGA is very popular and widely used. In a coarse-grained PGA, the entire population is divided into several subpopulations. Each subpopulation runs a conventional GA independently and concurrently on its own subpopulation. After several epochs, best individuals migrate from one subpopulation to another according to a migration topology. The hierarchical PGA combines coarse-grained PGA with master-slave or fine-grained PGA, so that it has the benefits of its components [13]. In this paper, the idea to parallelize GA is borrowed to implement the parallelization of GSA. A master-slave/coarse-grained PGA, which combines master-slave PGA and coarse-grained PGA together, is used to parallelize GSA.

In the master-slave/coarse-grained PGSA, the host program runs on the master processor, which decides on the global termination criterion. The whole population is equally divided into several subpopulations among the slave processors. Each slave processor runs a sequential GSA independently within its own subpopulation on one processor. The pseudo code of PGSA is shown in Fig. 2, where $P(t)$ is the population of individuals of generation t, *myrank* is the rank of the processor, $slns_{migrate}$, $slns_{recv}$ and $slns_{delete}$ are the migrants, received individuals and the individuals to be replaced, respectively. If the migration conditions are satisfied, each processor, such as the source processor, implements the function *neighbor* to find the destination processors according to a migration topology. The migrant individuals ($slns_{migrate}$) are selected and sent to the destination processors. After the migrant individuals ($slns_{recv}$) are received on the destination processor, the individuals to be deleted ($slns_{delete}$) are determined and replaced by received individuals ($slns_{recv}$). The same program is executed on each processor, but on different data (their own population) until the global optimum is achieved.

1: $t = 0$
2: initialize $P(t)$
3: evaluate $P(t)$
4: **while** not termination-condition **do**
5: reproduction process of GSA
6: **if** migration-condition satisfies **then**
7: $dest = neighbor(myrank)$
8: $slns_{migrate} = migrant\_individual(P(t))$
9: $send\_string(dest, x_{migrate})$
10: $slns_{recv} = recv\_string(\ )$
11: $slns_{delete} = delete\_individual(P(t))$
12: $replace\_string\ (x_{delete}, x_{recv}, P(t))$
13: **end if**
14: **end while**

Fig. 2 Pseudo code of parallel genetic simulated annealing

Although PGSA is related to the parallel hybrid method developed by Chen *et al*. [6], there are some important differences between PGSA in this paper and Chen's PGSA (C-PGSA). In C-PGSA, each PE maintained one solution, and each PE accepted a visiting solution from other PEs for crossover and mutation. In PGSA, each processor maintains its own subpopulation of solutions and different processors exchange their best solutions after certain number of epochs. Thus, the communication overhead between processors is much smaller. In C-PGSA, a normal SA-type probabilistic selection procedure is used to retain the proof of convergence of SA. In PGSA, a Markov chain is used to realize the local selection of SA, which can improve the selection performance of SA.

C. Implementation Details of PGSA

In this study, optimization problems of real-valued functions are considered. And the real-value coding scheme is employed to represent the chromosome. Each chromosome vector is coded as a vector of real-value point numbers of the

same length as the solution vector. Let $x = (x_1, x_2, \ldots, x_n)$ be the encoding of a solution, where x_i ($\in \boldsymbol{R}$) represents the value of the ith gene in the chromosome x. Initially, x_i is selected within the desired domain, and reproduction operators of GA are carefully designed to preserve this constraint. In the reproduction process of PGSA, a tournament selection approach is used to select individuals for the next generation. In the tournament approach, a sub-group is initially selected randomly from the population. Then, a 'tournament' competition takes place in this sub-group, and the winner is inserted into the next population. Other implementation details of PGSA are described in the following sections.

1) Crossover and mutation

The conventional crossover operator combines substrings belonging to their parents. For real-value encoding, this type of crossover does not change the value of each variable; so it cannot perform the search with respect to each variable. Therefore, it is not suitable in this study and consequently, a modified crossover operator, called convex recombination [15], is used. It operates as follows.

Consider that the crossover takes place at the positions i and j ($i < j$), let $x = (x_1, \ldots, x_n)$ and $y = (y_1, \ldots, y_n)$ be the parent strings. Then, the offspring u and v are of the form:

$$u_k = \begin{cases} ax_k + (1-a)y_k, & i < k < j \\ x_k, & \text{otherwise} \end{cases} \quad (1)$$

$$v_k = \begin{cases} ay_k + (1-a)x_k, & i < k < j \\ y_k, & \text{otherwise} \end{cases} \quad (2)$$

where a is a random number in the interval [0, 1]. In this paper, the two-point crossover is applied to each couple of individuals.

For the real-value coding scheme, different mutation operators can be used, such as uniformly distributed mutation, Gaussian mutation, range mutation and non-uniform mutation. The first two mutation methods were used in this study. In the uniformly distributed mutation, the mutation operator randomly chooses a number z in the interval defined by [$-A$, A], where A is called the mutation range. The new point is given by: $x_m = x + z$ [16]. In Gaussian mutation, a random value z is chosen from a normal Gaussian distribution $N(0, \sigma)$, where σ is the standard deviation [17]. Uniformly distributed mutation is more commonly used for searches in a large region. The Gaussian mutation performs better searches in a small local area [15]. In this study, at the initial stage, uniformly distributed mutation is used. When the decreasing rate of the average fitness values is less than 0.01, Gaussian mutation is used.

2) Migration policy, migration rate, migration topology and migration frequency

In the implementation of PGSA, some parameters of concern are: migration policy which determines how to select individuals to migrate, migration rate or the number of individuals to migrate, the frequency of migration, and the migration topology.

Cantú-Paz [13] indicated that the elitist strategy performed better than tournament selection. Thus, the elitist strategy is used to choose the migration individuals. The top 1% of the best individuals are migrated to replace the worst individuals of other subpopulations. If 1% of population size is not an integer, it will be rounded off to the next integer that is greater than its fractional value. A ladder neighborhood relation is used to implement PGSA, and the 8-processor structure of this relation is shown in Fig. 3, which shows that every slave processor is connected with other five slave processors. The advantage of this topology is that it can spread the migrant individuals quickly among the slave processors. The frequency of migrating individuals is given in Section V.A.

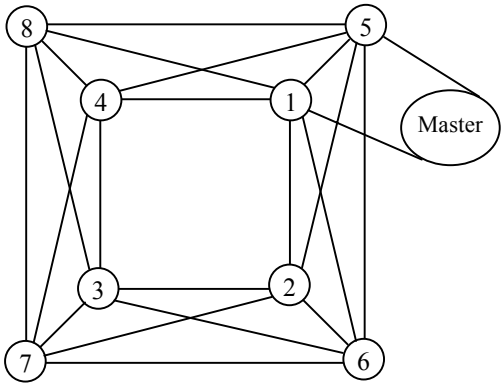

Fig. 3 Schematic diagram of the implementation of PGSA

IV. Multi-niche Crowding in PGSA

A. Similarity Metric

Phenotypic distance metric is used here as the similarity metric. For the real-value coding scheme, the Euclidean distance between two individuals is employed to measure their similarity. The smaller the distance between two individuals, the more similar they are.

For the given two individuals: $x = (x_1, \ldots, x_n)$ and $y = (y_1, \ldots, y_n)$, the distance between them is defined by:

$$d(x, y) = \sqrt{\sum_{i=1}^{n} \left(\frac{x_i - y_i}{x_i^u - x_i^l}\right)^2} \quad (3)$$

where n is the length of the chromosome, x_i^u and x_i^l are the upper and lower bounds of variable x_i, which is used here to normalize the distance $d(x, y)$.

B. Multi-niche Crowding in PGSA

The concept of MNC [11] is briefly reviewed here. In MNC, not only the selection but also replacement has been modified with some type of crowding. There are two steps for this crowding selection approach. Firstly, an individual A is selected for mating. Secondly, its mate M is chosen with the crowding selection instead of the fitness proportionate reproduction (FPR) of SGA. Based on the similarity to A, M is selected from a group of C_s individuals, which is composed of randomly chosen individuals from the population. The mate M must be the most similar one to A in the selected group.

After picking the mater of A, the genetic operators of crossover and mutation are then applied; one pair of offspring is generated. For each of these two offspring, MNC is again used to select an individual from the population for replacement by this offspring. During the replacement step, a replacement policy called "worst among the most similar" is used in MNC [11]. Firstly, from the population, C_f groups are created by randomly choosing s individuals per group. Then, in each group, the individual which is most similar to the offspring is identified. Finally, C_f individuals are identified as the most 'similar' to the offspring. These C_f individuals are candidates for replacement by virtue of their similarity to the offspring. From this group of most similar individuals, the one with the lowest fitness has been replaced with the offspring. The offspring could possibly have a lower fitness than the individual being replaced. In the implementation of MNC, based on the values used in [11], the following parameter settings are used: $C_s = 2$, $C_f = 6$, and $s = 3$.

Because of selection pressure caused by FPR, sometimes PGSA still cannot maintain good diversity in the population, especially for some difficult to solve multimodal functions. So MNC has been considered to maintain population diversity for PGSA. In implementation of PGSA, MNC was only applied among half the number of slave processors to select mating individuals and the individuals to be replaced by offspring. Among the other half number of slave processors, normal PGSA as described in section III.B was used in order to maintain the good local selection ability of SA. With the incorporation of MNC into PGSA, a new heterogeneous parallel algorithm, called MNC-PGSA, is obtained. This MNC-PGSA algorithm maintains good diversity in the population and inherits good convergence from PGSA.

V. Results and Discussion

Trafalis and Kasap [18] used 19 well-known global optimization problems to evaluate the performance of their global search functions. Recently, Wang et al [19] attempted a well-established set of nine functions to test their search algorithms. With focus on the search ability and scalability of the algorithm proposed in this paper, three more complex multimodal benchmark functions from available literatures [19], [20] have been used to compare the performance of MNC-PGSA and PGSA with other algorithms, shown in Table I. All tests were run on SUN Workstation network consisting of 42 SUN Blade 2000 workstations with a fast Ethernet interconnect (100 M-Bytes/sec). The main hardware and software of the Sun Blade 2000 consists of an Ultra-SPARC III Cu 900 MHz processor, 2 G-Bytes memory, 4 G-Bytes swap memory, and Solaris 8 operating system.

An extensive performance evaluation for functions F1-F3 has been done for PGA by Mühlenbein *et al*. [21]. In their investigation, the efficiency of the search method was demonstrated by varying the problem size n. Mühlenbein and Schlierkamp-Voosen [16] increased the problem size n further to investigate the advantage of scaling their search method. The number of function evaluations was defined as efficiency criterion in [16] and [21]. For comparison, in this section, the efficiency of search methods is also defined by the number of function evaluations needed to obtain the optimal solutions. When the minimal value of each function on the master processor is reached, it will send the termination signal to all slave processors. After these processors have received the termination signal, they stop running and send the number of function evaluations done so far to the master processor to sum them up.

TABLE I
MULTIMODAL BENCHMARK FUNCTIONS

Function	Function equation	Parameter intervals
F1	$f_1(x) = nA + \sum_{i=1}^{n}[x_i^2 - A\cos(2\pi x_i)]$	$-5.12 \le x_i \le 5.12$
F2	$f_2(x) = \sum_{i=1}^{n}[-x_i \sin(\sqrt{\lvert x_i \rvert})]$	$-500 \le x_i \le 500$
F3	$f_3(x) = \sum_{i=1}^{n}[-x_i^2/4000] - \prod_{i=1}^{n}\cos(x_i/\sqrt{i}) + 1$	$-600 \le x_i \le 600$

In the implementation of PGSA, the host program, which runs on the master processor, decides on the global termination criterion. After certain iterations, each slave processor sends its top 1% of the best individuals to the master processor. After receiving best individuals from all slave processors, the stop flag of PGSA is set on the master processor, if the following criterion is fulfilled:

$$\left| f_{best}^{k} - f_{best}^{k-\Delta k} \right| \le \varepsilon \left| f_{best}^{k} \right| \quad (4)$$

or $$\left| f_{best}^{k} - f_{best} \right| \le \varepsilon \quad (5)$$

where f_{best}^{k} is the best fitness value of an individual on the master processor at generation k, and the stop criterion is checked at every Δk generations; f_{best} is the global minimum and ε is a constant value 10^{-3}. For F1 and F2, Inequality (4) was used as the termination criterion. For F3, Inequality (2) was chosen as the termination criterion, where $f_{best} = 0$. After setting the termination flag, the master processor sends it to every slave one. According to the termination flag, each slave processor decides whether GSA continues to run or not.

A. Experiment Setup

Both GA and SA have many internal control parameters. Thus, PGSA, which is composed of GA and SA, also has many control parameters, which can be described as follows:

$$PGSA = (P_0, \lambda, \mu, \sigma, \delta, \tau, GSA, T_0, \alpha, t) \quad (4)$$

where P_0 is the initial population, λ is the number of subpopulations, μ is the population size of each subpopulation, σ is the migration interval in number of generations, δ is the number of neighbors, T_0 is the initial temperature, α is the cooling rate and t is the termination criterion.

In order to show the robustness of PGSA, there is no need to tune all these parameters to a specific function. Based on the recommended values from previous work [13], [21], the parameter settings for F1 and F2 are shown in Tables II and III, respectively, where p_m is the mutation probability. For all

experiments, the crossover probability is 0.65, and $\delta = 5$.

In case of Function F3, it is comparatively much more difficult to get the global minimal value. The parameter settings for F3 are as follows: λ =16, σ = 20, p_m = 0.3, T_0 = 600, α = 0.85 and crossover rate is 0.65. The population size on each processor μ is equal to 50 when the problem size n < 100, and μ =100 when $n \geq 100$.

TABLE II
PARAMETER SETTINGS FOR RASTRIGIN'S FUNCTION F1

Parameters	Problem size (n)						
	20	50	100	200	400	500	1000
λ	8	8	16	16	16	16	32
μ	20	20	20	40	40	100	100
σ	10	20	20	40	40	50	50
p_m	0.1	0.05	0.05	0.05	0.65	0.80	0.80
T_0	200	400	500	1000	2000	2000	2000
α	0.85	0.85	0.85	0.85	0.85	0.85	0.85

TABLE III
PARAMETER SETTINGS FOR SCHWEFEL'S FUNCTION F2

Parameters	Problem size (n)					
	10	50	100	150	200	400
λ	8	16	16	16	16	32
μ	20	50	100	150	200	200
σ	10	30	50	50	50	50
p_m	0.1	0.05	0.05	0.05	0.05	0.05
T_0	200	400	600	800	1000	1200
α	0.85	0.85	0.85	0.85	0.85	0.85

B. Results and Discussion for Lower Dimension Problems

Functions F1 and F2 with lower dimensions are easily solved by MNC-PGSA, PGSA and PGA as shown in Figs. 4 and 5. But in [21], the global optimum of F2 was not found in 4 of the 50 runs. In this study, MNC-PGSA and PGSA have found the optimum of F2 in all 50 runs. Because PGSA can maintain a good diversity with a higher mutation probability at the initial stage, it can eliminate premature convergence to suboptimal minima. At the later stage, the local selection strategy of SA can ensure that best solutions are not discarded after crossover and mutation operators. Therefore, PGSA can approach or converge on the global minimum with less number of function evaluations than PGA. In addition to the advantages of PGSA, MNC-PGSA can also maintain good population diversity with the crowding strategy.

Griewank's function F3 is regarded as one of the most difficult test functions. It has its global minimum $f_{best} = 0$ at $x_k = 0$, and the local minima are located approximately at $x_k = m\pi\sqrt{k}$, where $k = 1, \ldots, n$, and m is any integer value. Four suboptimal minima (≈0.0074) exist at $\vec{x} = (\pm\pi, \pm\pi\sqrt{2}, 0, \ldots 0)$ in ten dimensions. The average number of function evaluations is 6600 by Griewank [20], but only one of the four sub-minima was found. An average of 59520 evaluations is needed to solve this problem by Mühlenbein et al. [21], but they did not comment on their results. MNC-PGSA and PGSA found the minimum values (< 0.001) of F3 with less than half of the number of function evaluations using PGA, as shown in Fig. 6. More importantly, in all 50 runs, these minimum values were found. Therefore, MNC-PGSA and PGSA are able to obtain much better solutions with a higher convergence speed than PGA. Fig. 6 shows that less number of function evaluations with MNC-PGSA was needed to converge to global optima for Function F3 than that for PGSA with the same problem size. Thus, MNC-PGSA performs better than PGSA.

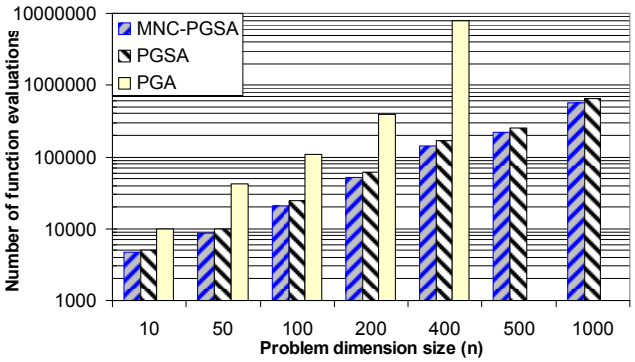

Fig. 4 Performance comparison between PGA [21], PGSA and MNC-PGSA for Rastrigin's Function

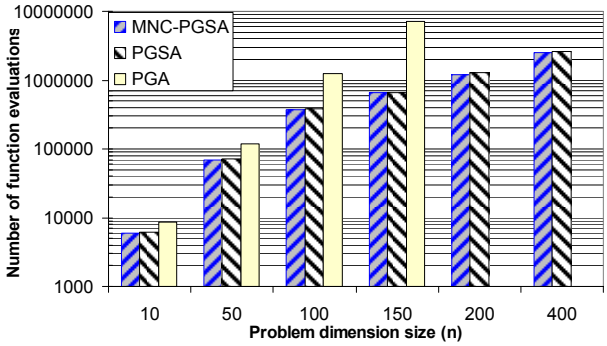

Fig. 5 Performance comparison between PGA [21], PGSA and MNC-PGSA for Schwefel's Function

C. Results and Discussion for Higher Dimension Problems

1) Computation results for F1 & F2 with high dimension

Mühlenbein *et al.* [21] found the global minimum of F1 with a dimension of 400 and F2 with a dimension of 150 on a 64-processor computer using PGA. For comparison in this paper, F1 and F2 with much higher dimension have also been attempted using PGSA. The parameter settings are listed in Tables II and III. The same termination criterion, Inequality (4), was used in order to directly compare the efficiency of MNC-PGSA, PGSA and PGA.

Function F1 with dimension 500 and 1000 and Function F2 with dimension 200 and 400 have also been investigated, which were not tried using PGA in [21]. It can be seen that the

number of function evaluations using MNC-PGSA and PGSA is much smaller than that using PGA. The performance of MNC-PGSA and PGSA gets better with the higher problem size n. In all cases, MNC-PGSA performs better than PGSA, but the difference in function evaluations between them is not so obvious; perhaps both of these two algorithms are effective for these two multimodal functions.

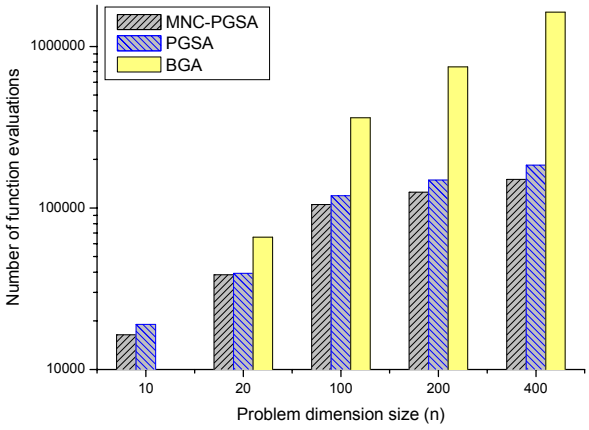

Fig. 6 Performance comparison between BGA [16], PGSA and MNC-PGSA for Griewank's Function

A typical run for Function F1 when n = 100 is shown in Fig. 7. Initially, the difference in performance between MNC-PGSA and PGSA is very small, but after 400 generations, it becomes larger. MNC-PGSA can converge to better optima with a faster speed. For PGSA, initially, the individuals of the subpopulation become more similar to one another than those in MNC-PGSA. In some cases, PGSA even converges around suboptimal minima. For PGSA, after 100 generations, the average and best values of the objective functions are much more similar. This means that less diversity exists in the population. However, for MNC-PGSA, the difference between the average and the best evaluation values is larger than that of PGSA, which indicates that more diverse individuals exist in subpopulations of MNC-PGSA. This can be attributed to the use of MNC, which can help to maintain better diversity in the population. As the optimization process proceeds for MNC-PGSA, owing to the better population diversity, it can converge to the different optimal solution within much smaller number of generations.

2) Computation results for F3 with higher dimension

Mühlenbein and Schlierkamp-Voosen [16] used breeder genetic algorithm (BGA) to optimize Functions F1 and F2 in higher dimension too. However, the number of function evaluations cannot be directly compared because a different termination criterion was used for BGA. In order to compare the efficiency of PGSA with that of BGA, the same termination criterion as that for BGA was used for Function F3. The computation results using BGA, PGSA and MNC-PGSA are listed in Fig. 6, which shows that less number of function evaluations is needed with PGSA and MNC-PGSA. Thus, the performance of PGSA and MNC-PGSA is better than that of BGA. The difference in performance between MNC-PGSA and PGSA for solving F3 is much more obvious than that for solving F1 and F2 with these two algorithms. Because it is much more difficult to obtain the global optima of F3 than F1 and F2, MNC-PGSA shows its higher efficiency in this case. This reiterates that MNC-PGSA is a powerful optimization method in comparison to PGSA and BGA. In [16], the number of function evaluations scales almost exactly with $n \cdot ln(n)$ for Function F3. With such scaling ability, the number of function evaluations increases quickly with the problem size. However, in this study, the increase rate of the number of function evaluations with the problem size for PGSA and MNC-PGSA is obviously smaller than that with BGA. With such scaling ability, the advantage of investigating the scaling of PGSA and MNC-PGSA has been demonstrated, i.e., PGSA and MNC-PGSA have better scalability.

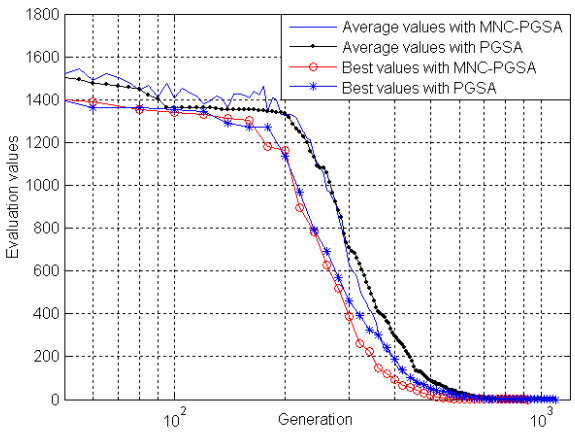

Fig. 7 Average and best function evaluation values for Rastrigin's function (n = 100)

D. Analysis of the speedup

Generally, when comparing the program performance between the parallel programs with the serial program, speedup is commonly used, which is defined as the ratio of the runtime to achieve a serial solution to a problem to the parallel runtime. Measurement of the speedup of PGSA on parallel processors is very difficult because of its probabilistic nature [21]. In this study, the speedup of PGSA was estimated based on average computation times of 50 runs, as shown in Table IV. When more processors are involved in computation, communication among processors causes the overhead, which will slowdown the speedup of parallel programs. Table IV shows that PGSA achieved almost linear speedup. Since the whole population is equally divided into several subpopulations among the slave processors, less computation time is needed to obtain the optimum. And the reduced computation time can even compensate for communication overhead among processors.

For F2 with a dimension 400, the global optimum could not be found for each case with the serial GSA. For Function F3, when the problem size was greater than 20, in most cases the serial GSA was not able to find the global optimum. Therefore the speedup analysis for F3 is not discussed in this section.

But PGSA could find the optimum for F2 and F3 with higher dimension within reasonable time; this demonstrates the significantly stronger global search ability of PGSA.

TABLE IV
AVERAGE COMPUTATION TIME WITH GSA AND PGSA

Function	n	Computation time (s)		λ	speedup
		GSA	PGSA		
F1	50	1.011351	0.110184	8	9.1787
	100	4.192071	0.218913	16	19.1495
	200	18.004712	1.005423	16	17.9076
	400	119.460274	7.451222	16	16.0323
	500	266.188604	14.167722	16	18.7884
	1000	1383.947593	36.713556	32	37.6958
F2	50	7.785258	0.446771	16	17.4256
	100	36.553699	2.155516	16	16.9582
	150	108.308197	5.264115	16	20.5748
	200	292.366368	14.096044	16	20.7410
	400	913.128460	30.328364	32	30.1081

F2 (n = 400): Global optimum was not found in 8 of 50 runs for the serial GSA.

VI. Conclusions

In this paper, a new GA and SA hybrid (GSA) is firstly presented, which inherits the strengths of GA and SA and overcomes their weaknesses. The extended ideas of simulated annealing were used in the adjustment of the mutation rate and the local selection of individuals which are retained in the next population after crossover and mutation. In GSA, at the initial stage, the higher mutation rate is helpful for maintaining population diversity. After crossover and mutation, the local selection of SA can ensure that good candidates still exist in the next generation at the later stage. Therefore by maintaining more diverse subpopulations at the initial stage, GSA mitigates the premature convergence of the standard GA. On the other hand, at the later stage, local selection strategy of SA ensures that increasing number of good candidates exists in the next generation. It can narrow the search space so that fast convergence can be achieved. PGSA is then described by implementing the parallelization of GSA. In addition, MNC has been incorporated into PGSA for the selection and replacement in reproduction process of PGSA. In MNC-PGSA, MNC can help to maintain population diversity throughout the search and converge to different local optima.

The numerical results show that MNC-PGSA has faster convergence to global optimum solution than PGSA and PGA. The better performance of MNC-PGSA is attributed to the better population diversity than PGSA; thus, less number of function evaluations is needed to converge to the different optimal solution. In comparison to the other advanced search method such as BGA [16] using the same termination criterion, the performance of MNC-PGSA and PGSA is better. More importantly, MNC-PGSA performs better with the larger problem size. Thus, MNC-PGSA has a good scalability. In this study, three different functions have been tried with the proposed algorithms. The only difference between them lies in the objective function. The aforementioned PGSA and MNC-PGSA algorithms are applicable to new multimodal problems, needing basically incorporating appropriate objective functions and changing some parameters if necessary. So PGSA and MNC-PGSA have a good programmability.

References

[1] J.H. Holland, *Adaptation in Natural and Artificial Systems*. Ann Arbor: University of Michigan Press, 1975, ch.1.
[2] J.P. Li, M.E. Balazs, G.T. Parks, and P.J. Clarkson, "A species conserving genetic algorithm for multimodal function optimization," *Evolutionary Computation*, vol. 10, no. 3, pp. 207-234, Fall 2002.
[3] H. Chen, and N. Flann, "Parallel Simulated Annealing and Genetic Algorithms: A Space of Hybrid Methods," In *Proc. Int'l Conf. Evolutionary computation − PPSN III, Lecture Notes in Computer Science*, vol. 866, Berlin: Springer-Verlag, 1994, pp. 428-438.
[4] S.W. Mahfoud, and D.E. Goldberg, "Parallel recombinative simulated annealing: A genetic algorithm," *Parallel Computing*, vol. 21, no. 1, pp. 1-28, Jan. 1995.
[5] J.V. Varanelli, and J.C. Cohoon, "Population-Oriented Simulated Annealing: A Genetic/Thermodynamic Hybrid Approach to Optimization," in *Proc. 6th Int'l Conf. Genetic Algorithms*, M. Kaufman, San Francisco, Calif., 1995, pp. 174-181.
[6] H. Chen, N.S. Flann, and D.W. Watson, "Parallel genetic simulated annealing: a massively parallel SIMD algorithm," *IEEE Transactions on Parallel and Distributed Systems*, vol. 9, pp. 126-136, 1998.
[7] T. Hiroyasu, M. Miki, and M. Ogura, "Parallel Simulated Annealing using Genetic Crossover," in *Proc. IASTED Int'l Conf. on Parallel and Distributed Computing Systems*, Las Vegas, 2000, pp. 145-150.
[8] C. Baydar, "A hybrid parallel simulated annealing algorithm to optimize store performance," in *Workshop on GECCO 2002*, New York, 2002.
[9] K.A. De Jong, "An Analysis of the Behavior of a Class of Genetic Adaptive Systems," Ph.D. Thesis, University of Michigan, MI, 1975.
[10] D.E. Goldberg, *Genetic algorithms in search, optimization and machine learning*, Reading, Massachusetts: Addison –Wesley, 1989, pp. 1-145.
[11] W. Cedeno, and V.R. Vemuri, "Analysis of speciation and niching in the multi-niche crowding GA," *Theoretical Computer Science*, vol. 229, no. 1-2, pp. 177-197, 1999.
[12] J. Hesser, and R. Männer, "Towards an optimal mutation probability for genetic algorithms," in *Proc. Parallel problem solving from nature: 1st workshop*, *PPSN I, Lecture Notes in Computer Science*, vol. 496, Berlin: Springer-Verlag, 1991, pp. 23-32.
[13] E. Cantú-Paz, *Efficient and accurate parallel genetic algorithms*, Boston: Kluwer Academic Publishers, 2000, pp. 1-119.
[14] E. Alba, and J.M. Troya, "A survey of parallel distributed genetic algorithms," *Complexity*, vol. 4, no. 4, pp. 31-52, 1999.
[15] D. Dumitrescu, B. Lazzerini, L.C. Jain, and A. Dumitrescu, *Evolutionary computation*. Boca Raton: CRC Press, 2000, pp. 187-211.
[16] H. Mühlenbein, and D. Schlierkamp-Voosen, "Predictive models for the breeder genetic algorithm I. Continuous parameter optimization," *Evolutionary Computation*, vol. 1, no. 1, pp. 25-49, 1993.
[17] Th. Bäck, F. Hoffmeister, and H.-P. Schwefel, "A Survey of evolution strategies," in *Proc. Fourth Int'l Conf. Genetic Algorithms.* M. Kaufmann, San Mateo, Calif., 1991, pp. 2-9.
[18] T.B. Trafalis, and S. Kasap, "A novel metaheuristics approach for continuous global optimization," *Journal of Global Optimization*, vol. 23, no. 2, pp. 171-190, 2002.
[19] Z.G. Wang, Y.S. Wong and M. Rahman, "Development of a parallel optimization method based on genetic simulated annealing algorithm," *Parallel Computing*, vol. 31, no. 8-9, pp. 839-857, 2005.
[20] A.O. Griewank, "Generalized descent for global optimization," *Journal of Optimization Theory and Applications*, vol. 34, pp. 11-39, 1981.
[21] H. Mühlenbein, M. Schomisch, and J. Born, "The parallel genetic algorithm as function optimizer," *Parallel Computing*, vol. 17, pp. 619-632, 1991.