Development of the parallel optimization method based on genetic simulated annealing algorithm

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

Department of Mechanical Engineering, National University of Singapore,
10 Kent Ridge Crescent, Singapore, 119260
Email: engp1604@nus.edu.sg

Abstract. This paper presents a parallel genetic simulated annealing (PGSA) algorithm that has been developed and applied to optimize continuous problems. In PGSA, the entire population is divided into subpopulations, and in each subpopulation the algorithm uses the local search ability of simulated annealing after crossover and mutation. The best individuals of each subpopulation are migrated to neighboring ones after certain number of epochs. An implementation of the algorithm is discussed and the performance evaluation is made against a standard set of test functions. PGSA shows some remarkable improvement in comparison with the conventional simulated annealing, parallel genetic algorithm.

1 Introduction

Genetic algorithm (GA) was developed by Holland [1] and is an adaptive global search method that mimics the metaphor of natural biological evolution. GA operates on a population of potential solutions by applying the principle of survival of the fittest to aim for approximations towards an optimal solution. Owing to their ability to achieve the global or near global optimal, this algorithm has been applied to a large number of combinatorial optimization problems.

The successful application of GA depends on the population size or the diversity of individual solutions in the search space. If GA cannot hold its diversity well before the global optimum solution is reached, it may be difficult for GA to find it, and sometimes it even results in the premature convergence to the local optimum solution. Although maintaining diversity is the predominant concern of GA, it also reduces its performance. Many researchers tried to find a trade-off between the diversity and performance (exploration and exploitation) of GA.

An alternative approach is to combine GA with other optimization techniques, such as simulated annealing (SA). SA is a general-purpose stochastic optimization method that has proven to be quite effective in finding the global optima for many different NP-hard combinatorial problems. In this paper, a new hybrid of GA and SA, referred to as genetic simulated annealing (GSA) is developed to avoid the premature convergence of GA by exploiting the local selection strategy of SA. Then, GSA is parallelized to improve its computation performance further. The principal purpose of

this paper is to demonstrate parallel GSA is a powerful optimization strategy that overcomes the inherent weaknesses of conventional GA and SA.

2 Genetic Simulated Annealing

GA and SA are both independently valid approaches toward problem solving with certain strengths and weaknesses. Although GA can begin with a population of solutions in parallel, 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, but the higher temperature and the lower cooling rate reduce the performance of SA. In addition, parallelism 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 [2] had shown that the hybrid of GA and SA can perform better for ten difficult optimization problems than either GA or SA independently. Sirag and Weisser [3] proposed a unified thermodynamic genetic operator to solve ordering problems. The unified operator is applied to the conventional GA operation of crossover and mutation to yield offspring. This operator can ensure greater population diversity at high temperature and less population diversity at low temperature. 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. Varanelli and Cohoon [5] used a similar hybrid method of GA and SA. In addition, Chen, Flann and Watson [6] also proposed a hybrid method, which maintains one solution per Processing Element (PE). Each PE accepts a visiting solution from other PEs for crossover and mutation. For the selection process, the SA cooling schedule and system temperature are used to decide whether the new generated individual is to be accepted or not. In this method, they used the local selection of SA to replace the conventional selection process of GA. Furthermore, it needs large amount of PEs for more complicated optimization problem, but computers with large amount of PEs may not be available for most researchers.

Each of the above approaches 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, which shows another tighter coupling of these two paradigms in the sense that SA controls a number of distinct GAs running in parallel. Although GSA is related to the hybrid method developed by Chen, Flann and Watson [6], there are some important differences between GSA in this paper and Chen's GSA (C-GSA). In C-GSA, each PE maintains one solution, and each PE accepts a visiting solution from other PEs for crossover and mutation. In GSA, each process maintains its own subpopulation of solutions; and different processes exchange their best solutions after certain number of epochs. So the communication overhead between processors is smaller. In C-GSA, a normal SA-type probabilistic selection procedure is used to retain the proof of convergence of

SA. In GSA, a Markov chain is used to realize the local selection of SA, which can improve the selection performance of SA.

In GA, two chromosomes are replaced by a pair of their offspring after the crossover and mutation operators; while in GSA, after the crossover and mutation, there are four chromosomes: two parents and two offspring, from which two chromosomes are chosen to form the new population. The selection criterion is based on the fitness values of individuals. Strings with higher fitness value have a greater probability of surviving to the next generation. However, individuals, that have the fitness values less than the best obtained, are not necessarily discarded, and they are instead selected 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 the two offspring. f_{best} and f_{worst} are the best fitness value and the worst one of two parents respectively; f_1 and f_2 are the fitness values of two offspring. During the course of the Markov chain at temperature T_t, the fitness value f_i (i =1, 2) of the trial chromosome is only compared with f_{worst}. If $\min\{1, \exp(-(f_i-f_{worst})/T_t)\}$ is greater than r (a randomly generated number between 0 and 1), f_i is accepted to replace the worst individual. Then the worst chromosome and the best one are updated before the course of the Markov chain finishes, as illustrated in Fig. 1, where $P(t)$ is the population of individuals of generation t and α is the cool rate of SA. At initial stage, when manipulating the cooling schedule of SA properly, parents will be replaced by their offspring whether they are much fitter or not; thus this algorithm can maintain the diversity and alleviate the premature convergence problem. On the other hand, at later stage, the chances for the fitter parents to be replaced decrease greatly, because the temperature is lower. In this way, the current best individual always remains in the next generation. Therefore the adverse effects (destroy useful information of the last generation) of the mutation operation can be reduced.

3 Parallel Genetic Simulated Annealing and its implementation

GSA can eliminate the premature convergence of GA to some extent. However, the initial temperature should be set higher in order to get the global optimum for those complicated problems with millions of local optima. This reduces the performance of GSA. In addition, for problems with a large search space and costly fitness function evaluation, there is a great need for fast convergence to the optimal solution. Thus, a parallel version of GSA (PGSA) is presented to improve its efficiency.

There are several ways to parallelize GA [7], and the parallel GA (PGA) has been developed and successfully applied to optimize practical problems by many researchers [8]. PGA can be classified into three categories: global single-population master-slave GA, coarse-grained and fine-grained PGA according to the ways in which parallelism is exploited in GA and the nature of the population structure and recombination mechanisms used [9]. In the master-slave GA, there is only a single population, but the evaluation of fitness function is distributed among several processors. The fine-grained parallel GA treats each individual as a separate breeding unit. Each individual may mate with those selected from a small local neighborhood. Since the neighborhoods overlap, fit individuals will migrate throughout the

population. The coarse-grained parallel GA is very popular and widely used. Studies of coarse-grained PGA are the mainstream in current studies about parallel distributed GA model. Thus in this paper this model has been used to parallelize GSA.

```
begin
  t = 0
  initialize P(t) and temperature T_t
  evaluate P(t);
  while (not termination-condition) do
    begin
      t = t + 1
      select P(t) from P(t-1)
      select individuals for reproduction from P(t)
      do reproduction
        select two unused individuals P_1, P_2
        crossover & mutation; generate two children C_1,C_2
        evaluate C_1, C_2
        for i:= 1 to 2 do
          if min{1, exp(-(f_i-f_worst)/T_t)}> random[0,1)
          accept C_i & replace the corresponding parent
          update the new best and worst points
      until all selected parents finish reproduction
      T_t+1 = T_t × α; 0 <α < 1
    end
end
```

Fig. 1. Algorithm of Genetic Simulated Annealing

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

```
begin
    t = 0
    initialize P(t)
    evaluate P(t)
    while (not termination-condition) do
        reproduction process of GSA
        if( migration-condition satifies ) then
            dest = neighbor(myrank)
            slns_migrate = migrant_individual(P(t))
            send_string(dest, x_migrate)
            slns_recv = recv_string()
            slns_delete = delete_individual(P(t))
            replace_string (x_delete, x_recv, P(t))
        endif
    end
end
```

Fig. 2. Pseudo code of the parallel genetic algorithm

3.1 Representation

PGSA uses a real-value coding scheme to represent the individual, that is to say, each chromosome vector is coded as a vector of real-value point numbers of the same length as the solution vector. Each point is initially selected within the desired domain, and reproduction operators of GA are carefully designed to preserve this constraint.

3.2 Selection

Based on evaluating the fitness of candidates, individuals are appropriately selected for recombination. This first step is fitness assignment by calculating the objective function. Sometimes the fitness value needs to be scaled for further use. Scaling is important to avoid early convergence caused by dominant effect of a few strong candidates in the beginning, and to differentiate relative fitness of candidates when they have very close fitness values near the end of run [10]. In GA, there are mainly three selection procedures: proportional selection, tournament selection and rank-based selection [11]. Proportional approach is usually called “roulette wheel” selection. Fitness values of individuals represent the width of slots of the wheel, and selection is based on the slot widths of individuals [12]. Individuals with larger slot widths will have a higher probability to be selected. In tournament approach, at first a

sub-group is randomly selected from the population with or without replacement. Then, a "tournament" competition is taken place in this sub-group, and the winner is inserted into the next population. This process is repeated until the new population is formed [11]. For rank-based approach, the individuals are sorted based on their fitness values, and the rank N is assigned to the best individual and the rank 1 to the worst one. Then based on their ranks, the selection probability is assigned to the individuals. Rank-based selection behaves in a more robust manner than the proportional one. Therefore in this paper, rank-based fitness selection is chosen as the selection approach.

3.3 Crossover and Mutation

The basic operators for producing new individuals in GA are crossover and mutation. Crossover may produce better individuals that have some genetic material of both parents. In this paper, the traditional two-point crossover is applied to each couple of individuals. Mutation simply changes the value for a particular gene with a certain probability. It helps 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. The probability that mutation will occur is set to a low value (e.g., 0.01) so that good chromosomes are not destroyed. The best mutation rate is application-dependent but for most applications is between 0.001 and 0.1.

For the real-value coding scheme, there are different mutation operators that can be used, such as uniform mutation, Gaussian mutation, range mutation and non-uniform mutation. In Gaussian mutation, a random value from a Gaussian distribution to each element of an individual's vector is added or subtracted to the old value to create a new offspring with a probability of 0.5. Gaussian mutation is widely used, and so it is used in this study.

3.4 Migration

In the implementation of the coarse-grained PGSA in this study, there are some parameters to be concerned: the number of individuals to migrate, then how to select and how frequently to migrate them, and logical arrangement of the subpopulations. The most widely used migration scheme is the ring topology, where individuals are exchanged between directionally adjacent subpopulations. Two methods to choose the individual to migrate are often attempted: elitist strategy and probabilistic tournament selection. In the first method, the best individuals are sent to a neighboring subpopulation; and in the second one, the fitter individual is selected via a probabilistic binary tournament with a certain probability (i.e. the individual is randomly chosen with a chance p; when $0.5 < p < 1.0$, the individual is selected to migrate.). Gordan and Whitley [7] compared these two methods and indicated that elitist strategy performed better than tournament selection. Thus we used elitist strategy to choose migration individuals and the top 1~5% (according to population size) of best individuals are migrated to replace the worst individuals of other subpopulations.

4 Numerical Results and Discussion

A set of functions known from the literatures have been used as the test function to compare the performance of PGSA with other algorithms, which is shown in Table 1. De Jong's suite functions F1~F5 [14] are extensively used in the GA community; functions F6-F10 are also widely used for testing the performance of the global search algorithms. The number of function evaluation and the running time are chosen as performance criteria. The running time is the elapsed time from the time the first process has started to the time that last process has executed the last command of the algorithm. When the minimal value of each function reaches at any process, it will send the termination signal to all other processes. Then just before every process stops running, it sends the number of function evaluation done so far on it to process 0 (i.e. the rank of process is 0); process 0 sums them up after receiving these numbers. In order to compare the efficiency of PGSA with that of other methods implemented on different machines, the standard unit of time should be used to normalize the computing time. The unit time is 1000 evaluations of Shekel's function at the point (4, 4, 4, 4) as introduced by Dixon and SzegÖ [13]. In this paper, the SUN workstation network is used, and one unit of time is 0.0012 seconds on each processor.

Table 1. A set of standard test function

Function	Function equation	Parameter intervals
F1	$f_1(x)=\sum_{k=1}^{3}x_k^2$	$-5.12\le x_i\le 5.12,\ \ i=1,2,3$
F2	$f_2(x)=100(x_2-x_1^2)^2+(1-x_1)^2$	$-2.048\le x_i\le 2.048,\ \ i=1,2$
F3	$f_3(x)=\sum_{i=1}^{5}\text{interger}(x_i)$	$-5.12\le x_i\le 5.12,\ \ 1\le i\le 5$
F4	$f_4(x)=\sum_{i=1}^{30}ix_i^4+Gauss(0,1)$	$-1.28\le x_i\le 1.28,\ \ 1\le i\le 30$
F5	$f_5(x)=0.002+\sum_{j=1}^{25}\frac{1}{j+\sum_{i=1}^{2}(x_i-a_{ij})^6}$	$-65.536\le x_i\le 65.536, i=1,2$
F6	$f_6(x)=nA+\sum_{i=1}^{20}[x_i^2-A\cos(2\pi x_i)]$	$-5.12\le x_i\le 5.12,\ \ 1\le i\le 20$
F7	$f_7(x)=\sum_{i=1}^{10}[-x_i\sin(\sqrt{\lvert x_i\rvert})]$	$-500\le x_i\le 500,\ \ 1\le i\le 10$
F8	$f_8(x)=\sum_{i=1}^{10}[-x_i^2/4000]-\prod_{i=1}^{10}\cos(x_i/\sqrt{i})+1$	$-600\le x_i\le 600,\ \ 1\le i\le 10$
F9	$f_9(x)=100(x_2-x_1^2)^2+(1-x_1)^2$	$-2000\le x_i\le 2000,\ \ i=1,2$
F10	$f_{10}(x)=\sum_{i=1}^{3}[(x_{i+1}-x_i^2)^2+(1-x_i)^2]$	$-200\le x_i\le 200,\ \ i=1,2,3,4$

4.1 Experiment Setup

Both GA and SA have many internal control parameters, and their performance is very sensitive to such parameters [6]. Different parameters of GA or SA are used for different problems. The PGSA, parallel version of the hybrid of GA and SA, also has many control parameters. Thus in order to show the robustness of PGSA, there is no need to tune all these parameters to a specific function. The parameters, used in the test, are shown in Table 2. Function F8 is much more difficult to get the global minimal value, so the larger population size and mutation rate were used.

Table 2. Control parallel setting

Parameters	F1~F5	F6	F7	F8	F9
Number of subpopulations / processors	8	8	8	16	8
Number of individuals of a subpopulation	20	20	20	50	20
Migration interval (generations)	10	10	10	20	10
Mutation probability	0.05	0.05	0.05	0.3	0.05
Crossover probability	0.65	0.65	0.65	0.65	0.65
Initial temperature	200	200	200	200	200
Cooling rate	0.85	0.85	0.85	0.85	0.85
Genome representation	floating point vectors				

Furthermore in order to get the more representative results, the average values for the PGSA listed in the following tables were based on 30 runs across the test suite of functions.

4.2 Results

Functions F1~F5 are easy to be solved by PGSA. For the more complex functions F6~F8, Griewank's function F8 is one of the most difficult test functions. The number of function evaluation and the final optimal values are given in Table 3. In all runs, less numbers of function evaluation are needed for PGSA to find the global minimum than that for PGA, and the computation time is also less with PGSA than that using PGA for all cases. In addition, the global minimum has been found to be accurate to at least four digits of accuracy in this study, whereas the global minimum in [15] only has been found to be accurate to at least three digits of accuracy. Therefore PGSA is found to obtain much better solutions with a higher convergence speed than PGA. The fast convergence of PGSA is due to the local selection decisions of SA after crossover and mutation, which can ensure that at the initial stage the subpopulations have great diversity and good candidates still exist in the next generation at the later stage. Thus PGSA can find the global optimum solution with less time.

Corana, Martini and Ridella [16] used the SA for the Rosenbrock functions F9 and F10 in two and four dimensions respectively. F9 and F10 were also tried in this paper using PGSA. The number of function evaluation is given in Table 4. In comparison with the calculation results of SA by Corana et al. [16], the number of function evaluation decreases remarkably using PGSA.

Table 3. Performance comparison between PGA and PGSA

Function	Numbers of function evaluation		Computation time	
	PGA	PGSA	PGA	PGSA
F1	1526	1356	1.57	1.35
F2	1671	1443	1.68	1.37
F3	3634	1476	3.43	1.50
F4	5243	4846	9.92	9.07
F5	2076	1701	2.79	2.18
F6	9900	9416	9.41	8.65
F7	8699	8528	6.61	6.24
F8	59520	57590	16.84	16.15

Table 4. Performance comparison of SA, AEA and PGSA

Function	Numbers of function evaluation		Final Function value	
	SA	PGSA	SA	PGSA
F9	484001	96320	4.2E-08	1.0E-06
F10	1264001	65750	5.9E-07	3.6E-06

The functions F6 and F7 of higher dimensions were also tried. Mühlenbein [15] found the global minimum of F6 of dimension 400 and F7 of dimension 150 on a 64 processor computer using PGA. For comparison in this paper, F6 and F7 with the same higher dimension have also been solved using PGSA. And F6 of dimension 500 and F7 of dimension 200 have not been dealt with by Mühlenbein [15]. The testing parameters are listed in Table 5, and the results are shown in Table 6. In all runs the global minimum has been found to at least four digits of accuracy.

From Table 6, it can be seen that it needs less numbers of function evaluation to reach the global minimum for PGSA than that for PGA. In addition, PGSA can still find the global minimum even with increasing complexity in problem dimension. It also means that PGSA has faster convergence than PGA.

Table 5. Control parallel setting

	F6					F7			
Dimension of the function	50	100	200	400	500	50	100	150	200
Number of subpopulations:	8	8	16	22	22	16	20	20	22
Number of individuals of a subpopulation	40	40	40	100	150	40	100	200	200
Migration interval (generations)	20	20	40	40	40	30	50	50	50
Mutation probability	0.01	0.01	0.01	0.3	0.3	0.01	0.01	0.01	0.01
Crossover probability	0.65	0.65	0.65	0.65	0.65	0.65	0.65	0.65	0.65
Initial temperature	200	500	1000	1500	2000	200	300	300	400
Cooling rate	0.85	0.85	0.85	0.85	0.85	0.85	0.85	0.85	0.85
Genome representation	floating point vectors								

Table 6. Performance comparison among PGA, AEA and PGSA

Function		Numbers of function evaluation		Computation time	
		PGA	**PGSA**	PGA	**PGSA**
F6	n=50	42753	*36826*	57.40	*49.45*
	n=100	109072	*82346*	129.69	*97.92*
	n=200	390768	*314635*	404.26	*304.45*
	n=400	7964400	*1514430*	4849.458	*1057.38*
	n=500	/	*2634236*	/	*1874.28*
F7	n=50	119316	*94695*	34.96	*28.76*
	n=100	1262228	*760604*	213.93	*143.48*
	n=150	7041440	*1543716*	1635.56	*279.56*
	n=200	/	*2607255*	/	*468.74*

5 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. Then PGSA is described by implementing parallelized GSA. A set of standard test functions is attempted using this algorithm. The numerical results show that PGSA has faster convergence to global optimum solution than PGA and SA, so PGSA is superior to the conventional PGA and SA. In PGSA, the local selection of SA is involved after crossover and

mutation in each subpopulation, which can ensure that each subpopulation maintains greater diversity at the initial stage and good candidates still exist in the next generation at the later stage. Therefore by maintaining more diverse subpopulations at the initial stage, the PGSA mitigates the premature convergence of the standard GA. On the other hand, at the later stage, more and more good candidates exist in the next generation; it can narrow the search space so that the fast convergence can be achieved.

Reference:

1. Holland, J.H.: Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor, MI (1975)
2. Chen, H., Flann, N.: Parallel Simulated Annealing and Genetic Algorithms: A Space of Hybrid Methods. In: Davidor, Y., Schwefel, H.-P., Manner, R. (eds.): Proc. Int'l Conf. Evolutionary computation − PPSN III, Lecture Notes in Computer Science, Vol. 866. Springer-Verlag, Berlin (1994) 428-438
3. Sirag, D.J., Weisser, P.T.: Toward a unified thermodynamic genetic operator. In: Grefenstette, J.J. (ed.): Proc. Second Int'l Conf. Genetic Algorithms, Lawrence Erlbaum Associates, Hillsdale, N.J., (1987) 116-122
4. Mahfoud, S.W., Goldberg, D.E.: Parallel recombinative simulated annealing: A genetic algorithm, Parallel Computing, 21(1995) 1-28
5. Varanelli, J.V., Cohoon, J.C.: Population-Oriented Simulated Annealing: A Genetic/ Thermodynamic Hybrid Approach to Optimization. In: Eshelman, L.J. (ed.): Proc. Sixth Int'l Conf. Genetic Algorithms, M. Kaufman, San Francisco, Calif. (1995) 174-181
6. Chen, H., Flann, N.S., Watson, D.W.: Parallel genetic simulated annealing: a massively parallel SIMD algorithm, IEEE Transactions on Parallel and Distributed Systems, 9 (1998) 126-136
7. Gordon, V.S., Whitley, D.: Serial and parallel genetic algorithms as function optimizers. In: Forrest, S. (ed.): Proc. Fifth Int'l Conf. Genetic Algorithms, M. Kaufmann, San Mateo, Calif. (1993) 177-183
8. Alba, E., Troya, J.M.: A survey of parallel distributed genetic algorithms, Complexity, Vol.4(4) (1999) 31-52
9. Chipperfield, A., Fleming, P.: Parallel Genetic Algorithms. In: Zomaya, A.Y. (eds.): Parallel and distributed computing handbook, McGraw-Hill, New York (1996) 1118-1143
10. Goldberg, D.E.: Genetic algorithms in search, optimization and machine learning, Addison - Wesley (1989)
11. Blickle, T., Thiele, L.: A comparison of selection schemes used in evolutionary algorithms, Evolutionary Computation, 4(1996) 361-394
12. Pham, D.T., Karaboga D.: Intelligent optimisation techniques: genetic algorithms, tabu search, simulated annealing and neural networks**,** Springer, New York (2000)
13. Dixon, L.C.W., Szegö, G. P.: Towards global optimisation 2. In: Dixon L.C.W., Szegö, G. P. (eds.): Towards global optimisation 2, North-Holland, Amsterdam (1978) 1-15
14. De Jong, K.A.: An Analysis of the Behavior of a Class of Genetic Adaptive Systems, Ph.D. Thesis, University of Michigan, 1975
15. Mühlenbein, H., Schomisch, M., Born, J.: The parallel genetic algorithm as function optimizer, Parallel Computing, 17(1991) 619-632
16. Corana, A., Martini, M., Ridella, S.: Minimizing multimodal functions of continuous variables with the simulated annealing algorithm. ACM Trans. on Mathematical Software, 13(1987) 262-280