A Genetic Algorithm without Parameters Tuning and its Application on the Floorplan Design Problem

Hiroshi Someya
Department of Computational Intelligence and Systems Science,
Graduate School of Interdisciplinary Science and Engineering,
Tokyo Institute of Technology,
4259, Nagatsuta, Midori-ku,
Yokohama, 226-8502, Japan.
E-mail:hiroshi@es.dis.titech.ac.jp
Phone:+81-45-924-5211

Masayuki Yamamura
Department of Computational Intelligence and Systems Science,
Graduate School of Interdisciplinary Science and Engineering,
Tokyo Institute of Technology,
4259, Nagatsuta, Midori-ku,
Yokohama, 226-8502, Japan.
E-mail:my@dis.titech.ac.jp
Phone:+81-45-924-5212

Abstract

Genetic Algorithm (GA) has been applied to many difficult optimization problems. It is known that GA can find the globally optimum solution rapidly if the population holds both varieties and concentration sufficiently. However, it is difficult to satisfy both requirements at the same time, because they are often in the relation of tradeoff each other. Although existing methods have several parameters to control this tradeoff balance, tuning them before search is also difficult. In this paper, we propose GSA, Genetic algorithm with Search area Adaptation, which controls the tradeoff balance dynamically. We have applied GSA to the floorplan design problem. The experimental results have shown the effectiveness of this approach.

1 INTRODUCTION

Stochastic optimization methods, such as Simulated Annealing (SA) and Genetic Algorithm (GA), have been applied to many difficult combinatorial optimization problems. These methods have to be equipped with both global and local search abilities for good performance. However, it is difficult to satisfy these requirements at the same time, since they are often in the relation of tradeoff each other. Although this difficulty is overcome with tuning several parameters in existing approaches, tuning them properly is not easy.

In SA, the tradeoff balance is controlled by cooling schedule with parameters, which control acceptance probability of state transition, called temperature. In earlier stage of search, the temperature is set high for global search. Then, it is decreased gradually, as the search continues. Thus, the performance of SA heavily depends on the cooling schedule. On the other hand, it is also reported that a universally good cooling schedule for any structure of solution space has not been found (Konishi 97). Therefore, we cannot avoid adjusting parameters for each problem instance with trial and error. In addition to that, it is known that the performance of SA depends on an initial solution. Since it starts searching from only one initial point in the search space and does not have operators for global search, it can search only a limited area (Koakutsu 92). Regarding the ability to search globally, GA has some advantages over SA, since it starts searching with a population, multiple points in the search space, and has an operator called crossover, which enables it to search over wide region. However, for the proper tradeoff balance, adjusting parameters, such as crossover rate and mutation rate, is necessary for GA.

The purpose of this study is to propose a method that can show good performance without tuning parameters for each problem instance. If a method can search adaptively, its parameters should depend not on the structure of the solution space of each problem instance but on the dimensions of the solution space. This paper proposes GSA, Genetic algorithm with Search area Adaptation, which adapts to the structure of solution space and controls the tradeoff balance between global and local search abilities dynamically. We have applied GSA to several floorplan design problem instances, whose dimensions of solution space are the same and structures of solution space are different, with not changing parameters. The experimental results have shown better performance than several existing methods and we have confirmed the effectiveness of our approach.

2 GENETIC ALGORITHM WITH SEARCH AREA ADAPTATION

GSA is developed from GA, considering the roles of genetic operators. GSA searches worthy regions in a

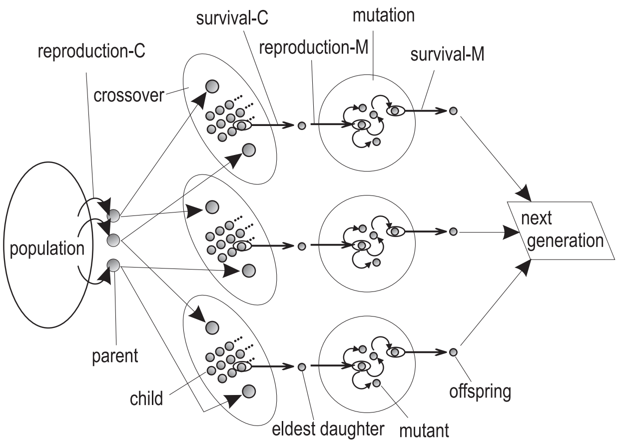

Figure 1: GSA algorithm(1).

```
make initial population and let counter = 0.;
while (counter < MAX){
        // MAX is the stop condition.;
    choose 3 parents (p0, p1, p2);
    repeat 3 times {
        if      (1st repeat) { 2parents are p0 & p1; }
        else if (2nd repeat) { 2parents are p1 & p2; }
        else if (3rd repeat) { 2parents are p2 & p0; }
        do {
            generate some children using crossover
                and select the best child.;
        }while(the best is worse than
                the worst in the population);
        counter += the number of generated children;
    }
    repeat 3 times {
        repeat R_m times {
            // R_m is the parameter of stop condition.;
            generate a mutant using mutation.;
        }
        select an offspring from mutants.;
        counter += the number of generated mutants;
    }
    replace 3 parents with 3 offsprings.;
}
```

Figure 2: GSA algorithm(2).

solution space adaptively for good solutions. The procedure of GSA is shown in Figure 1 and 2. One cycle of generation is divided into the following two phases:

Crossover search phase : This phase consists of three parts, selection for reproduction (called reproduction-C), crossover, and selection for survival (called survival-C). The role of this phase is to find a region that has high probability of containing good solutions between two parents. The procedure of searching a region depends on the topology of the search space. In Euclidian space, the region might be on the line segment connecting two parents.

Mutation search phase : This phase also consists of three parts, selection for reproduction (called reproduction-M), mutation, and selection for survival (called survival-M). The role of this phase is to search the region, which is selected in crossover search phase, for good solutions.

In the rest of this paper, we use the term "individual" as a solution candidate for a given problem. We use also following terms. An individual chosen through reproduction-C is called **Parent**. An individual generated by crossover is called **Child**. An individual selected through survival-C is called **Eldest daughter**. An individual selected through reproduction-M or generated by mutation is called **Mutant**. An individual selected through survival-M is called **Offspring**.

2.1 Crossover Search Phase

This phase is given a role of "finding a small region that is worth of intensive search between two parents." For global search, a middle region between two parents is selected if possible. The procedure of this phase is described as follows:

step1 Choose three individuals from the population randomly (reproduction-C), and make three pairs.

step2 For each pair, do *step3*∼*step6*.

step3 Let $f_a = 50, f_b = 50$.

step4 Generate a certain number of children that inherit f_a% or f_b% of the characteristic from each parent respectively using crossover.

step5 If the best child produced through the crossover operation is worse than the worst individual in the population, decrease f_a and increase f_b by a certain amount, and go back to *step4*.

step6 Select the best child as the eldest daughter (survival-C).

The crossover operator depends on problem domains. We assume it is already designed as to have excellent abilities of characteristic preservation and inheritance ratio control. If infeasible solutions are appeared, they are evaluated as the worst in all selections of GSA.

This procedure is based on the following images. If a child that is better than the worst in the population is generated in the middle of their parents, it is expected that good individuals exist in the child's neighborhood. Thus, the neighborhood region is searched intensively in mutation search phase. This image is illustrated in Figure 3. Otherwise, a few more children that are a little close to either of their parents are generated. In Figure 4, since child-1 is worse than the worst in the population, its neighborhood is not searched. Then, children-2 are generated. Since they are also worse, children-3 are generated. Since at least one of them is better, its neighborhood is searched in mutation search phase. If a child that is better than the worst in the population have not appeared until

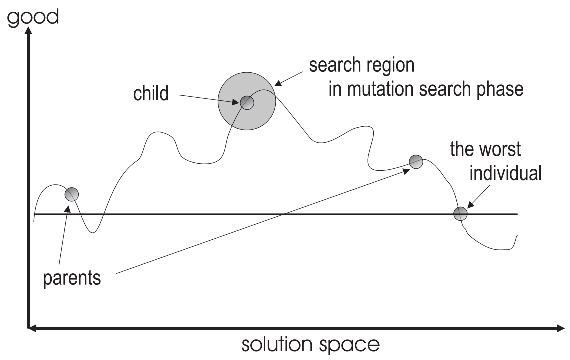

Figure 3: An image of GSA(1).

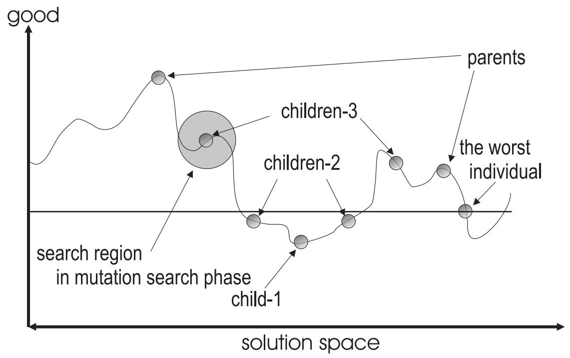

Figure 4: An image of GSA(2).

a certain stop condition is satisfied, the best child in this phase is selected as the eldest daughter.

This procedure controls searching region dynamically. A child that is worse than parents is allowed to survive. Thus, the parents can get out of local minima.

2.2 Mutation Search Phase

This phase is given the role of "generating good individuals by searching around a single individual intensively," under the constraint of not searching toward worse. This phase works in the following order:

step1 For each eldest daughter, do *step2*∼*step6*.

step2 Let $i = 0, j = 0$. Select the eldest daughter as mutant[0] (reproduction-M).

step3 Generate mutant[$i+1$] from mutant[i] using mutation operator. Then, add 1 to j.

step4 If mutant[$i+1$] is better than mutant[0], add 1 to i.

step5 Go back to *step3* if $j <= R_m$ (parameter).

step6 If the best mutant is better than the worst in the population, it is selected as an offspring (survival-M). Otherwise, at first, make a set including all individuals in the population and the best of the overall individuals appeared. Then, the chromosome of the individual chosen from the set randomly is copied to an offspring (survival-M).

step7 Replace all parents with all offsprings.

The mutation operator depends on problem domains. We assume it is already designed to search locally.

The above steps are based on the following images. In case mutant[i] is better than mutant[0], a globally good region or "a hill" of good individuals might be discovered in the direction of mutant[i]. So the hill should be searched as widely as possible. In this case, since mutant[i] is allowed to be worse than mutant[$i-1$], it is expected that the mutants extend for a maximum of R_m steps from mutant[0] beyond local minima. The other case, the hill might not be discovered. Therefore, regions in the other directions should be searched. If a mutant that is better than the worst in the population is not appeared, the eldest daughter might be trapped in local minima, and the process of searching the region stops. In this way, it is expected to search efficiently around an eldest daughter.

2.3 Diversity Maintaining and Characteristic Preservation in GSA

GSA achieves several good features around diversity maintaining and characteristic preservation as follows:

Since Simple GA, which is one of the well-known generation alternation models, is not equipped with selection for survival, there is high probability that children inherited useful characteristics from parents cannot survive. Thus, useful characteristics in the population decrease. GSA is equipped with the selection.

When a specific superior individual inhabits in a population for a long time, similar characteristics included in it increase in the population rapidly. Thus, the population converges on local minima. Therefore, it is desirable that any individual cannot inhabit for a long time. In many models equipped with selection for survival (Satoh 96), this case is inevitable since the selections work in both parents and children. On the other hand, in GSA, selections for survival work in between children or in between mutants respectively. Then, parents are replaced by offsprings forcibly. Besides, generation alternations perform locally.

When several children inherit characteristics from their parents so much, they have similar characteristics. Thus, their surviving simultaneously is not necessary or even undesirable from the viewpoint of diversity maintaining (Nagata 97). In GSA, only one of the children, including similar characteristics, can survive.

3 EXPERIMENTS

We apply GSA to the floorplan design problem. We discuss its performance through comparisons with the performances of several existing methods based on SA or GA.

3.1 Floorplan Design Problem

In VLSI circuit layout, it is important to determine a rough placement of modules, which are functional blocks composed of a lot of circuits. The floorplan design problem is to place modules on a plane chip and decide the aspect ratio of the region including each module so as to minimize both the chip area and total wire length. The floorplan design problem is known as NP-complete. In this paper, the floorplan design problem is formulated as follows (Wong 86, Scherwani 95):

- All modules and a chip are rectangular.
- The aspect ratio of all modules and a chip are $0.5 \sim 2.0$.
- A chip structure is restricted to slicing structure.
- Each wire connects only two modules. (called pin-pair)

An example of a floorplan is illustrated in Figure 5. The slicing structure of a chip is obtained by recursively partitioning a rectangle into two parts either by a vertical line or a horizontal line. This slicing structure is represented by Polish expression. The horizontal or vertical division of a rectangle is expressed using the notation "+" or "$*$" respectively. For example, the notation "AB+" is equivalent to a rectangle that is divided into A and B horizontally. Manhattan distance is used to compute the length of each wire. The aim of the floorplan design problem is to design the slicing structure in order to minimize the expression (1).

$$cost = \sum_{i=1}^{m} x_i y_i + \lambda \sum_{i,j=1}^{m} c_{ij} d_{ij} \tag{1}$$

m	:	The number of modules
x_i, y_i	:	The width and height of the region including module i
λ	:	A constant to control the relative importance of area and wire length
c_{ij}	:	The number of wires between module i and j
d_{ij}	:	Manhattan distance between module i and j

3.2 Previous Works

Several works have been proposed to solve the floorplan design problem. GAPE (Cohoon 88, 91) and ISA (Koakutsu 92, 94) have shown better performance than SA (Wong 86).

GAPE uses multiple processors, each of which is assigned to a subpopulation. Each processor executes GA on its subpopulation until it has converged. Then, some solutions emigrate to the other subpopulations. The crossover operators, $CO_1 \sim CO_4$, used in GAPE are as follows. CO_1 and CO_2 copy one parent's either operands or operators to an offspring. CO_3 generates an offspring by coping a subtree. CO_4 generates two offsprings to exchange subtrees. Its selections are based on roulette wheel.

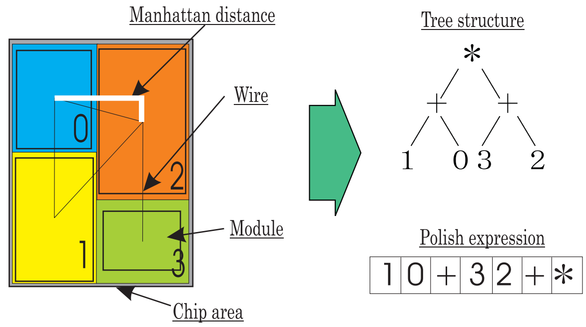

Figure 5: A slicing structure and Polish expression.

ISA, which is based on parallel SA, has features of both GA and SA. From time to time, two solutions, each of which have higher or lower cost value than the average of the population, are selected from the population as parents. Then, three solutions, called children, are generated from parents by crossovers, $CO_1 \sim CO_3$, used in GAPE. Each of parents is replaced by the best solution selected from a set including the parents and the children.

In order to focus on the effectiveness of GSA, we used operators defined in GAPE or in ISA.

3.3 Designing Crossover and Mutation for the Floorplan Design Problem

To design crossover for the floorplan design problem, *step3*$\sim$*step5* at section 2.1 are modified as follows:

step3 Let $f_a = \lfloor m/\ 2 \rfloor$, $f_b = f_a + m \bmod 2$.

step4 Extract all subtrees included in each parent, under the constraint that satisfies the expression (2). f is the number of operands of the subtree.

$$f = f_a \quad or \quad f = f_b \tag{2}$$

If each parent, P_0 or P_1, includes the same size subtrees, generate children from all pairs of all the subtrees by crossover as the following steps.

① Copy the operators from P_0 into the corresponding positions in the child.

② Copy the subtree from P_1 into the corresponding positions of the subtree of P_0 in the child.

③ Copy unused operands from P_0, by making a left-to-right scan, to complete the child.

step5 If the best child is worse than the worst in the population or crossover has failed, take 1 from f_a and add 1 to f_b. If $f_a >= R_f$ (parameter) and $f_b <= m - R_f$, go back to *step4*.

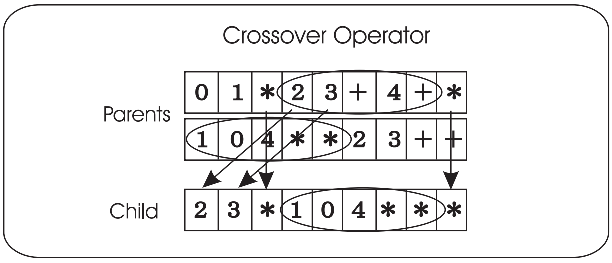

Figure 6: An example of the crossover.

Table 1: Models of the floorplan.

\	m	w	λ	a	c_{ij}
n1	20	0	1.0	fixed	0
n2	20	14	1.0	fixed	0∼5
n3	20	0	1.0	flexible	0
n4	20	72	1.0	flexible	0∼5
model3	20	31	1.0	flexible	0∼1

w : The number of pin-pairs, ($c_{ij} \geq 1$).
a : aspect ratio.

Table 2: Values of parameters.

\	GAPE		SA, ISA		GSA	
	N	8	L	50	p	65
	n	200(xN)	N	250	R_f	2
	S	30	T_s	500	R_m	25
	G	50	T_e	0.1	-	-
	C	0.5	α	0.9	-	-
	M	0.75	M	5000	-	-

GAPE		
N	:	the number of subpopulations
n	:	population size
S	:	the size of solutions sent to other subpopulation
G	:	the number of generations par epoch.
C	:	crossover rate
M	:	mutation rate

SA,ISA		
L	:	population size
N	:	selection frequency
T_s	:	initial temperature
T_e	:	final temperature
α	:	reduction ratio for the temperature
M	:	inner loop number

GSA		
p	:	population size
R_f	:	limitation of copied subtree length
R_m	:	the number of generated individuals through mutation search phase

An example of the crossover is illustrated in Figure 6. We consider that subtrees are building blocks of slicing tree. Thus, this crossover operator was designed based on CO_4 used in GAPE. f_a and f_b can control a ratio of inheritance from each parent to a child. If f_a and f_b are near $m/$ 2, children inherit characteristics from parents equally. If they are near 0 or m, children inherit characteristics partially.

To design mutation for the floorplan design problem, *step3* at section 2.2 is modified as follows:

step3 Generate mutant[$i+1$] from mutant[i], using a mutation operator selected out of the following three types, *M1*∼*M3* (Koakutsu 92, 94), randomly. Then, add 1 to j.

- *M1* Swap arbitrary two operands.
- *M2* Switch an operator from $*(+)$ to $+(*)$.
- *M3* Swap an operator and a neighboring operand.

3.4 Experimental Conditions

The dimensions of the solution space of the floorplan design problem depends on the number of modules. The structure of the solution space depends on the followings:

- The number of wires.
- Flexibility of modules.
- Variety of module size.

We selected five problem instances, model3 used in (Koakutsu 92), and *n1*∼*n4* created randomly but considered above three factors. Each of them has 20 modules. Table 1 summarizes the setting of each problem instance. The width and height of each module in *n1* and in *n2* are integers selected out of 1∼4 randomly. The area of each module in *n3* is an integer chosen out of 2∼14 at random. For variety, the area of each module in *n4* was set 2, 4, 6, ... , 40 respectively. The cost of *n4* has been about ten times larger than others, so it is measured by ten. The numbers of pin-pairs in *n2* and in *n4* were decided randomly. The area of each module in model3 is 2∼9.

GSA is compared with GAPE, SA and ISA over the best in the overall trials, the worst in the overall trials and the average of the best in each trial. They run 300 trials. We have implemented GAPE, following the description in (Cohoon 91). Also, we have used the original source code of ISA. To be fair in our comparisons, each of them is allowed to generate about 400,000 new solutions in each trial as written in (Koakutsu 92). The initial populations are generated randomly. The parameters shown in Table 2 were tuned in model3, and they are used also in *n1*∼*n4*.

4 RESULTS & DISCUSSION

4.1 Effectiveness of GSA

The experimental results are shown in Table 3. The lower table shows the ranking of performance.

Table 3: Experimental results.

\		GAPE	SA	ISA	GSA
n1	Best	135	135	135	135
	Worst	140	144	140	140
	Ave	136.95	138.04	138.883	137.933
n2	Best	191.5	191.5	191.5	191.5
	Worst	206.5	238	214	212
	Ave	195.543	204.152	198.295	192.385
n3	Best	137.28	137.055	137.02	136.964
	Worst	139.488	138.123	138.008	137.838
	Ave	137.926	137.61	137.593	137.364
n4	Best	230.493	229.979	228.789	227.407
	Worst	256.044	255.426	243.296	240.84
	Ave	242.437	238.792	235.625	233.475
model3	Best	184.961	182.967	185.087	182.947
	Worst	213.299	218.857	209.117	208.197
	Ave	201.487	197.847	195.837	194.620

\		GAPE	SA	ISA	GSA
n1	Best	-	-	-	-
	Worst	-	×	-	-
	Ave	◎	△	×	○
n2	Best	-	-	-	-
	Worst	◎	×	△	○
	Ave	○	×	△	◎
n3	Best	×	△	○	◎
	Worst	×	△	○	◎
	Ave	×	△	○	◎
n4	Best	×	△	○	◎
	Worst	×	△	○	◎
	Ave	×	△	○	◎
model3	Best	△	○	×	◎
	Worst	△	×	○	◎
	Ave	×	△	○	◎

◎	...	Winner	○	...	2nd
△	...	3rd	×	...	Worst
-	...	Draw			

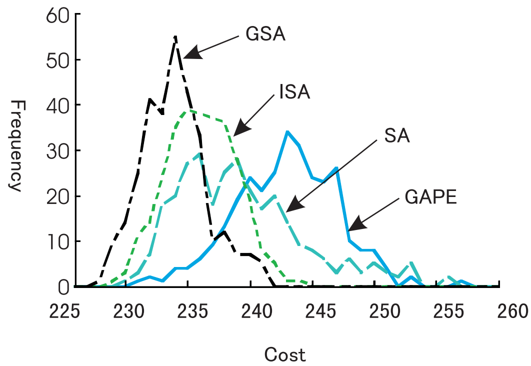

Figure 7: Frequency distribution of results (*n4*).

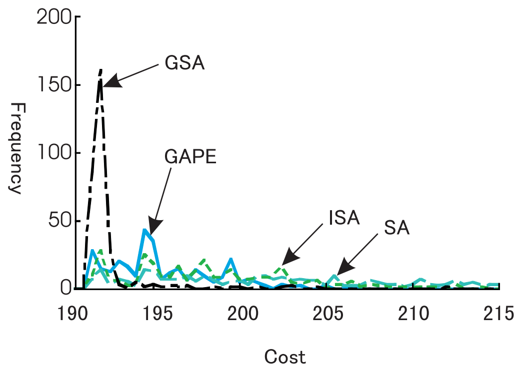

Figure 8: Frequency distribution of results (*n2*).

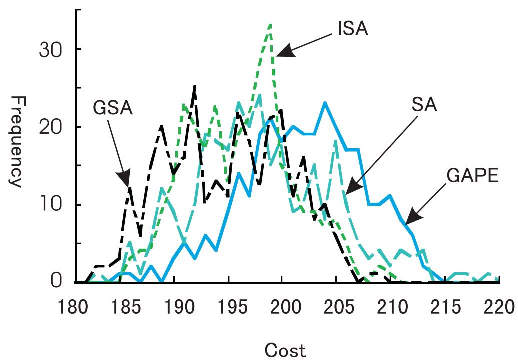

Figure 9: Frequency distribution of results (model3).

We can observe the effectiveness of GSA with tuning parameters, because GSA shows good performance in model3. In *n1*, the performances of all methods are about the same. But, in *n3* and in *n4*, GAPE shows poor performance, and so does SA and ISA in *n2*. The frequency distributions of results in *n4*, *n2* and model3 on Figure 7, 8 and 9, show the features of each method distinctly. Although the worst of GSA is worse than the worst of GAPE in *n2*, Figure 8 shows clearly that GSA is superior to GAPE. Therefore, it is confirmed that GSA shows good performance not only with tuning parameters but also without it. The best solution found by GSA through overall trials in model3 is displayed in Figure 10. Then, we examined the behavior of GSA using the following procedure:

step1 Make a set of individual pairs, the number of which is the combinations of population size taken two at a time.

step2 For each pair, do *step3*∼*step4*.

step3 Extract all subtrees that are included in both individuals of the pair. Then, make a set of them.

step4 The subtree set is classified into five groups according to the sum of the length of the subtrees included.

Figure 11 shows the ratio of the number of the subtree sets in each group while searching model3. In the early stage of search, the large subtree sets included in more

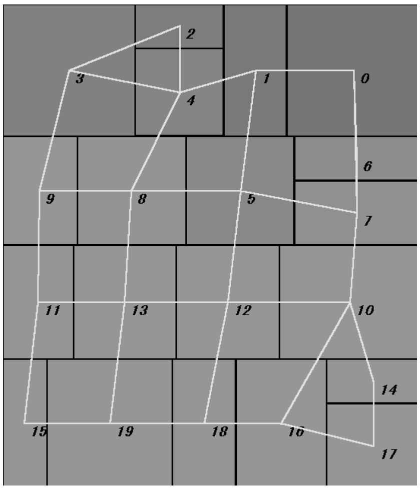

Figure 10: An example of model3.

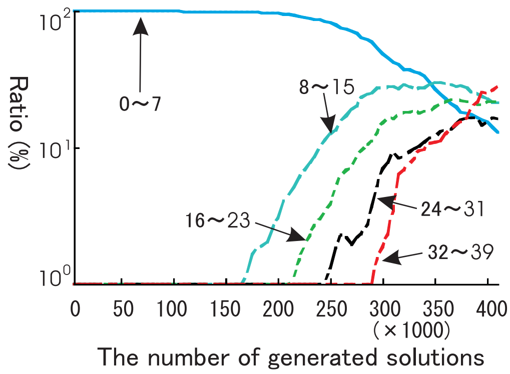

Figure 11: The transitions of the ratio of the subtree sets included in more than two individuals.

than two individuals did not appear. Thus, at this stage of search, subtrees with useful characteristics are still to be seen with maintaining the diversity in the population. Toward the final stage, the number of large subtree sets have increased gradually, meaning useful characteristics have grown.

4.2 Application on Benchmarks

We apply GSA to *ami33* and *ami49*, MCNC benchmark data (Kozminski 91), to confirm the performance of GSA in a large solution space. Table 4 summarizes the setting of each benchmark. Multi-terminal nets are decomposed into pin-pairs. The cost values are measured by one million. Experimental conditions are the same as the previous section except stop conditions. We set the stop conditions about 650,000 in *ami33* and about 1,000,000 in *ami49*. They are determined in the proportion to the number of modules respectively. The parameters used in this experiments are shown in Table 5. The numbers of trials are 300 in *ami33* and

Table 4: Models of the floorplan (*ami33*, *ami49*).

\	m	w	*(nets)*	λ	a	c_{ij}
ami33	33	528	(123)	0.1	fixed	0∼22
ami49	49	435	(408)	0.1	fixed	0∼16

Table 5: Values of parameters (*ami33*, *ami49*).

\	**GAPE**		**SA, ISA**		**GSA**	
ami33	N	8	L	50	p	50
	n	150(xN)	N	1000	R_f	3
	S	15	T_s	5	R_m	25
	G	50	T_e	0.05	-	-
	C	0.3	α	0.9	-	-
	M	0.7	M	14900	-	-
ami49	N	8	L	50	p	60
	n	100(xN)	N	1500	R_f	3
	S	20	T_s	10	R_m	18
	G	50	T_e	0.01	-	-
	C	0.5	α	0.9	-	-
	M	0.9	M	15200	-	-

Table 6: Experimental results (*ami33*, *ami49*).

\		GAPE	SA	ISA	GSA
ami33	Best	361.75	349.333	349.784	349.055
	Worst	386.876	365.721	361.036	362.661
	Ave	374.737	355.608	355.243	354.472
ami49	Best	255.386	219.779	221.891	219.419
	Worst	293.969	252.139	237.867	238.582
	Ave	275.695	231.719	230.662	229.187

100 in *ami49*. The experimental results are shown in Table 6. GSA shows good performance also in large solution space.

4.3 Analysis of the Structure of Solution Space using Random Search & Hill Climing Method

We apply "Random search & Hill climbing Method (R&H)" to the problem instances to analyze the structures of the solution space of each problem instance. R&H works as follows:

step1 Repeat *step2*∼*step4* p times.

step2 Generate a solution randomly. Let $h = 1$.

step3 Select a transition operator out of *M1*∼*M3* randomly, and apply it to the solution. If the evaluation of the solution is improved or equal to the value before transition, the transition is accepted. Otherwise, it is rejected. Then, add 1 to h.

step4 If $h < r$, go back to *step3*.

We use four parameter sets called R&H1∼R&H4. Table 7 shows them. When R&H1 shows the best performance, we can consider that the problem instance has a complex structure of solution space and a method

Table 7: Values of parameters (R&H).

\		R&H1	R&H2	R&H3	R&H4
p		400	100	50	10
r	model3, *n1* ∼*n4*	1,000	4,000	8,000	40,000
	ami33	1,625	6,500	13,000	65,000
	ami49	2,500	13,000	65,000	100,000

p : The number of retry
r : The number of state transition

Table 8: Experimental results (R&H).

\		R&H1	R&H2	R&H3	R&H4
n1	Best	135	135	135	135
	Worst	140	140	140	140
	Ave	138.7	136.8	136.35	135.95
n2	Best	203	194	198	204.5
	Worst	233	230	238.5	256.5
	Ave	220.31	213.325	217.365	229.445
n3	Best	137.098	137.154	137.446	137.05
	Worst	138.148	138.06	138.086	139.168
	Ave	137.744	137.678	137.749	138.077
n4	Best	238.681	233.334	233.776	233.895
	Worst	250.669	246.124	250.531	256.926
	Ave	244.427	240.224	241.42	246.106
model3	Best	197.712	193.826	189.527	198.848
	Worst	219.24	215.631	219.68	232.824
	Ave	211.222	205.755	208.453	217.01
ami33	Best	366.447	357.192	358.518	359.489
	Worst	378.466	370.389	371.19	379.242
	Ave	373.954	365.302	365.094	369.453
ami49	Best	258.364	238.082	234.134	236.352
	Worst	275.398	256.618	256.67	265.516
	Ave	268.481	250.347	247.657	252.552

equipped with global search ability shows good performance. On the other hand, when R&H4 shows the best performance, the problem instance is considered to have a simple structure of solution space and a method equipped with local search ability is effective. Table 8 shows the experimental results for 100 trials.

R&H2 or R&H3 shows better performance than R&H1 and R&H4 in all problem instances except *n1*. Therefore, *n1* might have few local minima in the solution space. Thus, only local search ability is demanded in this case. On the other hand, the others require a proper balance between global and local search. The above analysis matches the experimental results of the section 4.1.

5 CONCLUSIONS

In this paper, we proposed GSA. We have confirmed that GSA shows good performance for the several floorplan design problem instances, which have different structures of solution space, with or without tuning parameters. We have selected problem instances whose structures of the solution space is clearly different. We have found good results for many other problem instances. To apply GSA to multiple combinatorial optimization problems for chip area and wire length is a future work. To find out the relation between parameters and the dimensions of solution space is also necessary.

Acknowledgements

The authors wish to thanks Associate Prof. S. Koakutsu (Chiba University) and Prof. Y. Kajitani (Tokyo Institute of Technology) for their helpful advices.

References

D.F.Wong, C.L.Liu (1986) : "A New Algorithm for Floorplan Design", Proc. IEEE 23rd Design Automation Conf., pp.101-107.

J.P.Cohoon, et al. (1988) : "Floorplan Design Using Distributed Genetic Algorithms", Proc. IEEE Int. Conf. on CAD, pp.452-455.

J.P.Cohoon, et al. (1991) : "Distributed Genetic Algorithms for the Floorplan Design Problem",IEEE Trans. on CAD, Vol.10, No.4, pp.483-492.

K.Kozminski (1991) : "Benchmarks for Layout Syntheis - Evolution and Current Status", Proc. 28th ACM-IEEE Design Automation Conf., pp.265-270.

S.Koakutsu, Y.Sugai and H.Hirata (1992) : "Floorplanning by Improved Simulated Annealing Based on Genetic Algorithm (in Japanese)", Trans. Institute of Electrical Engineers of Japan, Vol.112-C, No.7, pp.411-416.

S.Koakutsu, H.Hirata (1994) : "Genetic Simulated Annealing for Floorplan Design" Control and Information Sciences, Vol.197, pp.268-277.

N.A.Scherwani (1995) : "Algorithms for VLSI Physical Design Automation SECOND EDITION", Kluwer Academic Publishers.

H.Satoh, M.Yamamura, and S.Koabayashi (1996) : "A Genetic Algorithm with Characteristic Preservation for Function Optimization", Proc. IIZUKA'96, p.511.

K.Konishi, M.Yashiki and K.Taki (1997) : "An Application of Temperature Parallel Simulated Annealing to the Traveling Salesman Problem and Its Experimental Analsis (in Japanese)", IEICE Trans. Vol.J80-D-I No.2 pp.127-136.

Y.Nagata, S.Kobayashi (1997) : "Edge Assembly Crossover: A High-power Genetic Algorithm for the Travelling Salesman Problem", Proc. ICGA97, p450.

H.Someya, M.Yamamura (1999) : "A Genetic Algorithm for the Floorplan Design Problem with Search Area Adaptation along with Searching Stage (in Japanese)", Trans. Institute of Electrical Engineers of Japan, Vol.119-C, No.3, pp.393-403.