SOLVING THE RECTANGULAR PACKING PROBLEM BY AN ADAPTIVE GA BASED ON SEQUENCE-PAIR

Koichi HATTA *Shin'ichi WAKABAYASHI* *Tetsushi KOIDE*

Faculty of Engineering, Hiroshima University
4-1, Kagamiyama 1 chome, Higashi-Hiroshima 739-8527, Japan
E-mail: {hatta, wakaba, koide}@ecs.hiroshima-u.ac.jp

ABSTRACT

In this paper, we propose a genetic algorithm (GA) to solve the rectangular packing problem(RP), in which the sequence-pair representation is adopted as the coding scheme of each chromosome. New genetic operators for RP is presented to explore the search space efficiently. The proposed GA has an adaptive strategy which dynamically selects an appropriate genetic operator during the GA execution depending on the state of an individual. Experimental results show the effectiveness of our adaptive genetic algorithm compared to simulated annealing (SA).

1. INTRODUCTION

The rectangular packing problem (RP) is a well-known discrete combinatorial optimization problem in many applications such as VLSI layout placement [11], etc. RP is known to be a difficult and time-consuming problem, since the number of possible placements of rectangles increases exponentially as the number of rectangles increases. Many approaches have been proposed to solve RP in a practical computation time. One important key to struggle to solve the problem is the representation of an instance of RP. Recently, *sequence-pair* has been proposed as the representation of a solution of this problem, which is particularly suitable for stochastic algorithms such as genetic algorithm (GA) and simulated annealing (SA) [6, 5, 7, 12].

GA is known to be a robust search algorithm, which deals with the individuals (chromosomes) of candidate solutions (population) encoded in the problem independent representation [3]. During the genetic process, new candidate solutions are composed by using the genetic operators such as crossover and mutation. When solving the specific application by a GA, it is often necessary to use complex representation of individual and genetic operators tailored to the problem to search solutions efficiently.

Since GA has an adaptive feature in nature, types of genetic operators as well as their parameters may be changed during the genetic process. An adaptive GA is the GA which dynamically adapts itself to an instance of a given problem during the GA execution.

In this paper, an adaptive GA for the rectangular packing problem is presented. The adaptive GA proposed in this paper, EAGA for short, uses a new measure of superiority of an individual, called *elite degree* [4]. EAGA supports two types of crossover operators, and selects the proper crossover operator and determines the mutation rate for each individual during the algorithm execution, since these operators and parameter values affect GA performance substantially.

There have been some previous works to use GAs to solve the rectangular packing problem or its related problems [1, 9, 10, 8]. In those previous works, a binary tree representation of an individual was adopted, and genetic operators used in the GAs were fixed. Compared to those previous works, the main feature of the proposed GA is an adaptive strategy, in which two types of crossover operators are supported, and one type of operator is selected to each pair of individuals during the algorithm execution. Another feature of our algorithm is to use a sequence-pair representation of an individual. Experimental results show that the proposed GA produces good placements.

This paper is organized as follows. In Section 2, the rectangular packing problem is defined, and sequence-pair is briefly explained. In Section 3, after giving the outline of newly proposed genetic operators, the adaptation strategy is explained. Section 4 shows results of the simulation experiments for test data.

2. PRELIMINARIES

2.1. Rectangular Packing Problem

The rectangular packing problem RP is an important discrete combinatorial optimization problem in many applications such as VLSI layout design [6].

A *module* $m_i \in M, (0 \leq i < n)$ is a rectangle with a given height and width in real number. A *packing* of a set of modules is a non-overlapping placement of given modules. The minimum bounding rectangle of a packing is called the *chip*. The problem is to find a packing of M onto a chip with the minimum area. This problem is known to be NP-hard, and hence good heuristics are generally solicited.

2.2. Sequence-Pair

An elegant coding scheme called *sequence-pair* has been proposed for RP[6, 5]. A sequence-pair $(\Gamma+, \Gamma-)$ for a set of n module is a pair of sequences of the n module names. A sequence-pair imposes a horizontal/vertical (H/V) constraint for every pair of modules. The oblique-grid notation visually shows H/V constraints specified by a given sequence-pair figure 1.

A sequence pair is easily utilized as the representation of a candidate solution for stochastic algorithms such as genetic algorithm (GA) and simulated annealing (SA) [5].

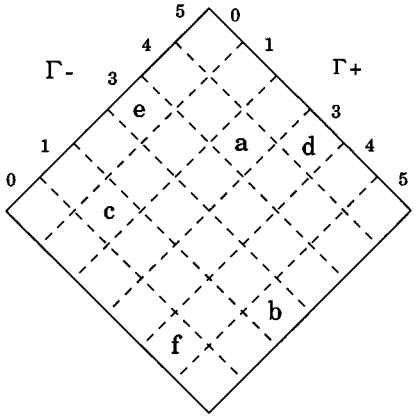
3. GENETIC APPROACH FOR RP

GA is a stochastic search method based on natural genetics, which deals with the individuals composed of candidate solutions (*population*), each of which is generally encoded in a problem independent representation. During the genetic process, new candidate solutions are evaluated by their fitness and reproduced using the genetic operators such as crossover and mutation [3].

3.1. Representation

As mentioned, we adopt sequence-pair to represent each individual of the proposed GA. Let $x^i (0 \leq i < P)$ be i-th individual in the population. x^i represents a candidate solution of the packing problem RP, and is denoted by $x^i = (\Gamma+^i, \Gamma-^i, \Theta^i)$. $(\Gamma+^i, \Gamma-^i)$ is the

sequence-pair whose respective elements are composed of module names, that is, $(\gamma+^{i}_{0}, \gamma+^{i}_{1}, \cdots, \gamma+^{i}_{j}, \cdots, \gamma+^{i}_{n-1})$ and $(\gamma-^{i}_{n}, \gamma-^{i}_{n+1}, \cdots, \gamma-^{i}_{j}, \cdots, \gamma-^{i}_{2n-1})$. Θ^{i} shows orientation of modules, that is, $\theta^{i}_{j} = \{0, ..., 7\}, (2n \leq j < 3n)$ [1]. Each position of an individual is called the *locus* of chromosome. Figure 2 shows an example of the chromosome in GA.



(Γ+, Γ-)=(ecadfb, fcbead)

Figure 1. An example of oblique-grid notation.

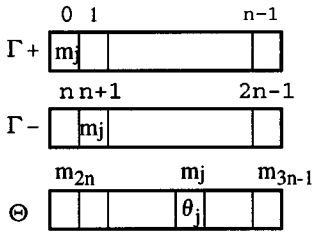


Figure 2. A chromosome.

3.2. Fitness

The *fitness value* of an individual is the packing area obtained from the sequence-pair of the individual. The packing area is calculated from the sequence-pair of an individual by constructing the H/V constraint graphs (figure 3), and finding the longest paths on H/V constraint graphs, the orientation of modules, which are specified by Θ^{i}, are also considered.

3.3. Crossovers

We present two crossover operators, called OPX and UPX, for the proposed adaptive GA. OPX is designed for preserving the characteristics of parents to offsprings. On the other hand, UPX is designed for explore the search space widely.

3.3.1. 1 point + PMX based Crossover (OPX)

OPX is based on two classical crossover operators, 1 point crossover and PMX. During the crossover phase, OPX is performed on Γ+, Γ− and Θ, independently. Given a crossover point, OPX separates Γ+ into subsequences of Γ+′ and Γ+″, and

[1] In the current definition of the rectangular packing problem, the number of types of meaningful module orientation is 2. However, to make it easy to extend the problem to a VLSI floorplanning problem, in which the wire length among modules is considered, 8 types of module orientation are considered.

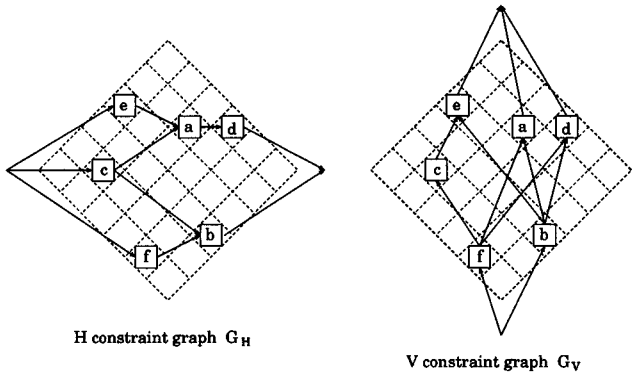


Figure 3. H/V constraint graphs.

exchanges between parents. For Γ−, crossover point is determined according to the modules in Γ+′. For Θ, the orientation of modules in Γ+′ are exchanged between parents. Let x^{p1} and x^{p2} be the parents selected for recombination. We also call x^{p1} and x^{p2} parent 1 and parent2, respectively. Let x^{c1} and x^{c2} be the offsprings reproduced from crossover. The procedure OPX is as follows.

1. Generate one crossover point $xp, (0 \leq xp < n)$ randomly in the first sequence Γ+. Let $\Gamma+'^{p2}$ be the subsequence of parent 2 between the locus 0 and xp in $\Gamma+^{p2}$. Let the $\gamma+^{p2}_{j}, (0 \leq j < xp)$ be the module name in $\Gamma+'^{p2}$. Find the location k, $(0 \leq k < n)$ which $\gamma+^{p2}_{j}$ is appeared in $\Gamma+^{p1}$. Exchange the $\gamma+^{p1}_{j}$ and $\gamma+^{p1}_{k}$ in $\Gamma+^{p1}$ about all j. Let $\Gamma+''^{p1}$ be the subsequence between xp and n of $\Gamma+^{p1}$. Copy the $\Gamma+'^{p2}$ to x^{c1} on locus between 0 and xp, and copy $\Gamma+''^{p1}$ to x^{c1} on locus between xp and n.
2. Let the M_x be the set of module in the $\Gamma+'^{p2}$. Find the location $j(n \leq j < 2n)$ in parent2 which $m \in M_x$ is appeared in $\Gamma-^{p2}$. Find the location $k, (n \leq k < 2n)$ which m appeared in $\Gamma-^{p1}$. Exchange the $\gamma-^{p1}_{j}$ and $\gamma-^{p1}_{k}$ in $\Gamma-^{p1}$ about all $m \in M_x$. Copy the module name $m \in M_x$ appeared in $\Gamma-^{p2}$ to x^{c1} on same locus. And copy the $M - M_x$ in $\Gamma-^{p1}$ to x^{c1} on same locus.
3. For modules $m_j \in M_x$ included in the $\Gamma+^{p2}$, θ_j of x^{c1} is θ_j of x^{p2}. For modules $m_i \in \{M - M_x\}$, θ_i of x^{c1} is θ_i of x^{p1}.
4. Reproduce x^{c2} by the same manner as x^{c1}.

3.3.2. Uniform + PMX based Crossover (UPX)

UPX is based on both Uniform crossover and PMX. UPX is similar to OPX about the removal of conflict of module name, but different in terms of the generation of crossover points and exchange of the orientation of module. The UPX is defined by replacing step 1 and step 3 of OPX with the following steps.

1. Generate the $mask_j = \{0, 1\}, (0 \leq j < n)$ randomly on the first sequence Γ+. $\gamma+^{p2}_{j}$ is the module name on locus j in $\Gamma+^{p2}$. Let M_x be the set of module name $\gamma+^{p2}_{j}$ when $mask_j = 1$. Find the location $k, (0 \leq k < n)$ which $m \in M_x$ is appeared in $\Gamma+^{p1}$. Exchange the $\gamma+^{p1}_{k}$ and $\gamma+^{p1}_{j}$ in $\Gamma+^{p1}$ about all $m \in M_x$. Copy $m \in M_x$ appeared in $\Gamma+^{p2}$ to x^{c1} on same locus. And copy the $m \in \{M - M_x\}$ in $\Gamma+^{p1}$ to x^{c1} on same locus.
3. Generate the $mask_j = \{0, 1\}, (2n \leq j < 3n)$ randomly for Θ. if $mask_j = 1$, copy θ^{p1}_{j} to θ^{c1}_{j}. Otherwise, copy θ^{p2}_{j} to θ^{c1}_{j}.

An example of UPX is illustrated in Figure 4. Note that OPX and UPX preserve the absolute locations of modules to be crossed over in parent $p2$ on the oblique grid representation in the crossover phase.

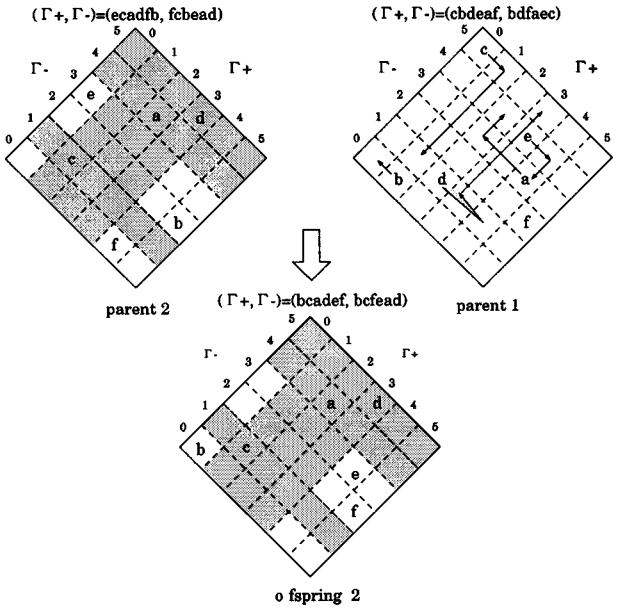


Figure 4. An example of UPX.

3.4. Mutation

We present the mutation operator MM for RP. Mutation is occurred with mutation rate p_m per each gene of $\Gamma+$ and Θ. For each sequence-pair, MM exchanges module names, that corresponds to the movement of a module to some random location on the oblique grid. For the orientation of each module, MM flips the height and width of placed module randomly.

Let x^p and x^c be the parent and offspring chosen for the mutation phase, respectively.

1. Generate $mp1, (0 \leq mp1 < n)$ randomly for mutation. let the m be the module name $\gamma+^p_{mp1}, (0 \leq mp1 < n)$ in $\Gamma+^p$. Generate $mp2, (0 \leq mp2 < n)$ at random. Exchange the $\gamma+^c_{mp1}$ and $\gamma+^c_{mp2}$ after x^c is copied from x^p.
2. Find the location $j, (n \leq j < 2n)$ which m is appeared in Γ^p-. Generate $mp3, (n \leq mp3 < 2n)$ at random. Exchange the $\gamma-^c_j$ and $\gamma+^c_{mp3}$ in x^c.
3. If $\theta^p_{mp4}, (2n \leq mp4 < 3n)$ is chosen for mutated with p_m orientation in Θ. Replace θ^p_{mp4} to a random number between 0 to 7.

An example of MM is illustrated in figure 5.

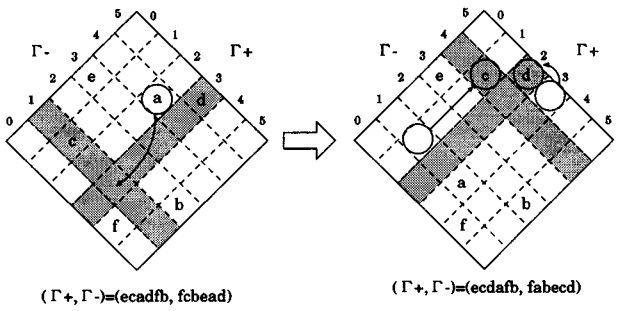


Figure 5. An example of mutation MM.

3.5. Adaptation

In general, some accurate measure for an instance of the problem is required for any adaptive strategy. The fitness value could be used as one of such influential measures. But fitness value is considered not to be suitable for combinatorial optimization problem such as RP, because the search space of the combinatorial optimization problem might have many local optima.

As the measure suitable for the combinatorial optimization problem such as RP, we have proposed *elite degree*, which indicates how many superior (i.e., "elite") individuals are included in its ancestors [4]. Elite degree was devised to estimate the potential superiority of an individual reproduced during the GA execution.

Based on elite degree, the proposed adaptive GA, EAGA for short, dynamically selects an appropriate operator from two types of crossover operators explained in the previous subsection.

In general a schema preserving genetic operator is considered to be efficient for local search. In EAGA, for high elite degree individuals, the schema preserving genetic operator, e.g., OPX, is applied because they are expected to be near optima. On the other hand, for low elite degree individuals, the search space exploring genetic operator, i.e., UPX, is applied for the global search. We also change the application of probabilities of those two operators depending on the superiority of individuals to preserve good individuals from destruction of crossover. Let p_{ce} and p_{cn} be application probabilities of OPX and UPX. Then, we set p_{ce} to a much smaller value than p_{cn}. For mutation, we also change the application probabilities of mutation operator MM described in the previous section.

3.6. Overview of EAGA for RP

In EAGA for the rectangular packing problem, UPX or OPX is selected adaptively as the crossover operator. For mutation for RP, two mutation rates, p_{me} and p_{mn}, are supported and adaptively selected. The mutation operator itself is also slightly modified so that, for elite individuals, the ability of searching a better solution is strengthened. These operators and parameters applying to parents are selected as follows. In the crossover phase, if the sum of elite degrees of parents is equal to or greater than a given threshold, OPX is applied because OPX is expected to be less disruptive for characteristics of parents than UPX. Otherwise, UPX is applied. In the mutation phase, for high elite degree individuals, a lower mutation rate is applied. Furthermore, the mutation operation MM is applied in k times, and the best result will be adopted as a result of mutation, where k is a user-defined parameter. Otherwise, for non-elite individuals, a higher mutation rate is applied, and the mutation operation is applied once.

The overview of EAGA is given as follows.

Procedure EAGA for RP

1. Set the GA parameter values such as population size. Generate the initial population, in which each individual is represented as a sequence pair and the orientation of modules.
2. If the number of evaluations of individuals reaches to a given constant, output the best packing obtained so far and then stop.
3. Select the individuals from the population by roulette wheel selection.
4. Create the family tree of each individual and calculate the elite degree.
5. Choose two individuals randomly from the population with crossover rate pc. If the sum of elite degrees of individuals is equal to or greater than a given threshold, OPX is applied. Otherwise, UPX is applied.

6. For high elite degree individuals, apply the mutation MM with mutation late p_{me}. When mutation is performed, the mutation operator MM is applied in k times, and the best result among k trials is actually adopted as a result of mutation. For low elite degree individuals, the mutation rate p_{mn} is used where $p_{me} < p_{mn}$. For low elite degree individuals, the mutation operator is applied in one time.

7. Evaluate each individual by calculating the area of its corresponding placement.

8. Increment the generation, and go to 2.

3.7. Local Improvement

Hybridization is to combine the GA with some heuristic procedures, and is known to be effective for GA to improve the candidate solutions [2]. We introduce the local improvement procedure LI into EAGA in order to reduce the chip area. The procedure LI is performed in the evaluation phase (Step 7) of EAGA, which changes the orientations of modules on the critical path of H/V constraint graphs. At the evaluation phase, each candidate solution (individual) is replaced with the modified solution by LI as follows.

Procedure LI

7.1. For the candidate solution, calculate one longest path of the H/V constraint graph.

7.2. By back-tracking the longest path from the sink node, construct a set of modules M_c on the longest path.

7.3. For each module m in M_c' which is appeared in M_c and not included in both H constraint graph and V constraint graph, change its orientation if $w(m) < h(m), (h(m) < w(m))$ in V,(H) constraint graph, respectively, where $w(m)$, $h(m)$ are the width and height of the module m after placed.

7.4. For the modified solution, by calculating one longest path length of H/V constraint graph again, get the packing area, which is regarded as the new fitness value.

4. EXPERIMENT

To investigate the performance of the proposed GA, some experiments were conducted. We implemented EAGA for RP with C language based on GENESIS, a public domain GA software, and performs on UltraCOMPStation (model 170). The data used for experiment consists of 38 and 100 modules whose sizes are generated randomly. The optimum is known to be 10240 and 303700 unit area, respectively.

We compared EAGA with SA and conventional GA. SA was implemented according to [5], and combined with the local improvement procedure LI. The parameter value were determined after running GA several times with some parameter sets. The parameter values for GA thus determined were as follows: population size 20, crossover rate 0.6, generation gap 0.6, scaling window 5, elitist strategy. The parameter values for EAGA were: non-elite crossover rate $p_{cn} = 0.6$, elite crossover rate $p_{ce} = 0.06$, non-elite mutation rate $p_{mn} = 0.2$, elite mutation rate $p_{me} = 0.1$, $D = 3$, where D is the threshold for adaptive crossover selection. In the mutation phase, the mutation operator is applied in 10 times for each elite individual ($k = 10$). We performed 10 runs, and average the best solutions. Table 1 shows the best, worst and average solution of best solutions and computation time. GA(OPX) and GA(UPX) are GAs in which only OPX or UPX is used as crossover operators, respectively. That is, in GA(OPX) and GA(UPX), no adaptation strategy were implemented.

From Table 1, the proposed method EAGA outputs good average solution compared with other methods including SA. For data d100, SA produced the best result, however, the computation time is two times longer than EAGA. Furthermore, EAGA is better than SA from the viewpoint of average solutions. For data d38, the dead space of the best solution obtained with EAGA is 3.6%, which is quite good. However, for data d100, the dead space of the best solution obtained with EAGA is 24.6%, that suggests there may be room for further improvement of EAGA.

Table 1. Best solution of EAGA

Data	Method	AREA (mm^2)			TIME
		Best	Worst	Ave	(sec)
d38	SA	11520	13600	12340	1158.0
	GA(OPX)	13440	14280	13832	475.3
	GA(UPX)	13440	14400	13936	474.2
	EAGA	10640	11560	11216	2650.4
d100	SA	387600	475700	423775	18176.7
	GA(OPX)	448500	484500	466110	1620.4
	GA(UPX)	446400	501600	472090	1658.2
	EAGA	402600	429000	420290	9420.3

5. CONCLUSION

We proposed the adaptive GA for the rectangular packing problem RP. We designed new crossover and mutation operators based on sequence-pair representation of individuals. We proposed an adaptive strategy to select appropriate genetic operators during the GA execution. Experimental results showed the effectiveness of our proposed GA.

Future work is to develop more powerful genetic operators to search efficiently. Applying the proposed GA to some real world problems such as the VLSI floorplanning problem is another research subject.

REFERENCES

[1] J. P. Cohoon and W. D. Paris: "Genetic placement," Proc. IEEE Int. Conf. on CAD, pp.422–425 (1986).

[2] L. Davis(ed.): "Handbook of Genetic Algorithms," Van Nostrand Reinhold (1991).

[3] D. E. Goldberg: "Genetic Algorithms in Search, Optimization, and Machine Learning," Addison-Wesley Publishing Company (1989).

[4] K. Hatta, K. Matsuda, S. Wakabayashi and T. Koide: "On-the-fly crossover adaptation of genetic algorithms," Proc. IEE/IEEE Genetic Algorithms in Engineering Systems: Innovations and Applications, pp.197–202 (1997).

[5] H. Murata, K. Fujiyoshi and M. Kaneko: "VLSI/PCB placement with obstacles based on sequence-pair," Proc. ISPD '97, pp.26–31 (1997).

[6] H. Murata, K. Fujiyoshi and Y. Kajitani: "VLSI module placement based on rectangle-packing by the sequence-pair," IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, Vol.15, No.12, pp.1518–1524 (1996).

[7] H. Murata and E. S. Kuh: "Sequence-pair based placement method for hard/soft/pre-placed modules," Proc. ISPD '98, pp.167–172 (1998).

[8] M. Rebaudengo and M. S. Reorda: "Gallo: A genetic algorithm for floorplan area optimization," IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, Vol.15, No.8, pp.943–951 (1996).

[9] V. Schnecke and O. Vornberger: "An adaptive parallel genetic algorithm for VLSI-layout optimization," Proc. Parallel Problem Solving from Nature, pp.859–868 (1996).

[10] K. Shahookar and P. Mazumder: "A genetic approach to standard cell placement using meta-genetic parameter optimization," IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, Vol.9, No.5, pp.500–511 (1990).

[11] K. Shahookar and P. Mazumder: "VLSI cell placement techniques," ACM Computing Surveys, Vol.23, No.2, pp.143–220 (1991).

[12] J. Xu, P. Ning Guo and C. Kuan Cheng: "Rectilinear block placement using sequence-pair," Proc.ISPD '98, pp.173–178 (1998).