On Solving Rectangle Bin Packing Problems Using Genetic Algorithms

Shian-Miin Hwang, Cheng-Yan Kao*, Jorng-Tzong Horng
Dept. of Computer Science and Information Engineering
National Taiwan University,Taipei, Taiwan
*All correspondence should be sent to the second author

Abstract This paper presents an application of genetic algorithms in solving rectangle bin packing problems which belong to the class of NP-Hard optimization problems. There are three versions of rectangle bin packing problems to be discussed in this paper: the first version is to minimize the packing area, the second version is to minimize the height of a strip packing, and the final version is to minimize the number of bins used to pack the given items. Different versions of genetic algorithms are developed to solve the three versions of problems. Among these versions of genetic algorithms, we have demonstrated two ways of applying the genetic algorithms, either to solve the problem directly or to tune an existing heuristic algorithm so that the performance is improved. Experimental results are compared to well-known packing heuristics FFDH and HFF. From these results, we know that both methods can be useful in practice.

I. INTRODUCTION

. The obvious industrial applications of stock cutting have been an important stimulus to the research of two dimensional packing. Further motivation has been driven by the advances in VLSI technology in which layouts on chips pose a number of important combinatorial packing problems.

. It is well known that an efficient algorithm for finding optimal solutions for bin packing problems has proved to be quite difficult to find. In fact, the decision version of the bin-packing problem "Given C, L, and an integer bound K, can L be packed into K or fewer bins of capacity C?" is NP-complete, this means that it is unlikely that efficient optimization algorithms can be found for these problems. Thus researchers have turned to the study of approximation algorithms, that is, algorithms which, although not guaranteed to find an optimal solution for every instance, usually find near-optimal solutions for most cases.

. Genetic Algorithms(GAs) [3], [4], [5], [6] developed by John Holland in 1975 are techniques for optimization and machine learning. A GA is composed of a reproductive plan which provides an organizational framework for representing the pool of genotypes of a generation. After the successful genotypes are selected from the last generation, a set of genetic operators are used in creating the offsprings of the next generation. Whenever some individuals exhibit better then average performance, the genetic information of these individuals will be reproduced more often. GAs work with a rich database of population and simultaneously climb many peaks in parallel during the search so that the probability of trapping into a local minimum is reduced significantly.

```
Procedure Genetic_Algorithm
begin
  t = 0
  initialize P(t)
  evaluate P(t)
  while (not termination-condition) do
  begin
    t = t + 1
    select P(t) from P(t-1)
    recombine P(t)
    evaluate P(t)
  end
end
```

Fig. 1 A simple genetic algorithm

The structure of a simple GA is shown in Fig. 1. The GA simulates an evolutionary process with n individuals which represent n points in a large search space. From the engineering point of view , GAs are an iterative process where each iteration has two steps, evaluation and generation. In the evaluation step, domain knowledge is used to determine the fitness of a candidate, a measure of its quality. Then an evaluation function maps a candidate solution into the nonnegative real numbers. The generation step includes a selection operator and several modification operators. The selection operator chooses individuals with a probability that corresponds to the relative fitness. Two chosen individuals, called the parents, produce children using the genetic operator crossover. The crossover operator exchanges substring of the codes of the parents at the same randomly determined point or points; however, it does not create any new genetic material in the knowledge base. The mutation operator, on the other hand, randomly changes a component in the structure introducing a new material into the knowledge base. From another point of view, the mutation operator acts as a local search close to the current point in the search space while the crossover operator causes larger jumps in the search space. Finally, the descendants replace some individuals in the population after the generation step is done.

A GA for a particular problem must have the following components:

(1) a genetic representation for potential solutions to the problem,

(2) a way to create an initial population of potential solutions,

(3) an evaluation function that plays the role of the environment, rating solutions in terms of their fitness,

(4) genetic operators that alter the composition of children during reproduction,

(5) values for various parameters that the genetic algorithm uses (population size, probabilities of applying genetic operators, etc.)

. In this paper we will try to solve the three versions of rectangle bin-packing problems, with slightly differences on their goals, by using GAs. We will denote the version to minimize the packing area as RBP1, denote the strip packing problems as RBP2, and denote the version to minimize the number of bins used as RBP3.

II. Literature Review

. So far, we have found two papers that solve the rectangle bin packing problems by using GAs. The first work is done by D. Smith [8]. The goal of his GA is to put as many blocks into a single rectangular region as possible. Experimental results have shown that this GA can produce the same packing density 300 times faster than their previously developed deterministic bin packing algorithm which used some heuristics and dynamic programming techniques. However, the genetic encoding does not allow for the recognization of characteristic features of packing schemes in their encoding, as most of these characteristics are hidden in an algorithm to place a sequence of rectangles. Thus this approach cannot support the inheritance of certain features by the offsprings.

. The second paper we have found is by B. Kröger, P. Schwendering and O. Vornberger [1]. They try to solve the strip bin packing problems (RBP2 in this paper) by using GAs. Their representation is much more complex than that of Smith. They also devise a special crossover operator for their representation. This representation of packing is better than the list representation because the features of the parents are specified more explicitly in the chromosome, and thus the offsprings can inherit features from their parents so that the building block hypothesis can be satisfied. They implement their GA on a parallel machine transputer. Experimental results show that this GA is able to solve large bin packing problems in reasonable time and that smaller instances are likely to be solved optimally. However, the complexity of the representation makes the design of the genetic operators more difficult and the time complexity of evaluating a chromosome is still too high. Also, the cost of applying the associated genetic operators to this representation is much more than the associated genetic operators for list representation.

III. On Solving RBP1: Minimizing the Packing Area

The problem is stated below: Given a set of rectangles, we wish to pack them into a rectangular area, so that no two items overlap and so that the packing area is minimized. For all items, rotation by 90 degree is allowed. Square packing is preferred.

We will denote our GA for solving RBP1 as GA1.

A. The Representation

. For GA1, we conceptually use a slicing tree (See Fig. 2) to represent a solution. A slicing tree is an oriented rooted binary tree. Each internal node of the tree is labeled either * or +, corresponding to either a vertical or a horizontal cut, respectively. Each leaf corresponds to a basic rectangle or item and is labeled by a identification number between 1 and n, where n is the problem size. A slicing tree can be viewed either from top down or from bottom up fashion. From a top down point of view, a slicing tree specifies how a given rectangle is cut into smaller rectangles by horizontal and vertical cuts. From a bottom up point of view, a slicing tree describes how smaller rectangles are combined. The operator * and + are no more than left-right and top-down relations for two adjacent rectangles, respectively. Corresponding to each slicing tree, there exists a polish expression to describe it. The polish expression can be easily obtained through post-order traversal of the slicing tree. Thus, in the implementation of GA1, we actually use the polish expression as our internal representation of a packing. Althugh there are certain packing that cannot be described by slicing trees (see Fig. 3), we still believe that the slicing tree can represent most good and near optimal packings.

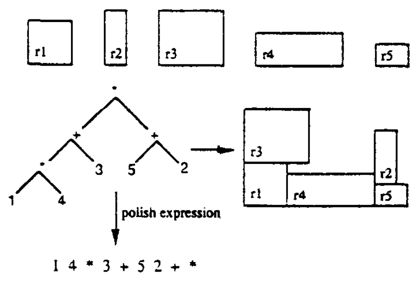

Fig. 2. A slicing tree and its corresponding packing and polish expression

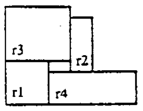

Fig. 3. A packing that can't be represented by a slicing tree

. A polish expression can be easily converted to an actual packing by using a stack of rectangles. A rectangle is specified by its width and height. We evaluate a polish expression by scanning it from left to right, if an operand (a number to index the item) is encountered, push the corresponding rectangle to the top of the stack; if an operator is encountered, pop two rectangles from the stack as it's operands and apply the operator, then push the newly generated rectangle (the bounding rectangle of its two operands) back to the stack. When the scanning process is over, the rectangle left on the stack is the bounding rectangle. The application of operator * and + is shown in Fig. 4. They are no more than a sum or max operation. Eventually, all of the operation can be finished in O(n) time.

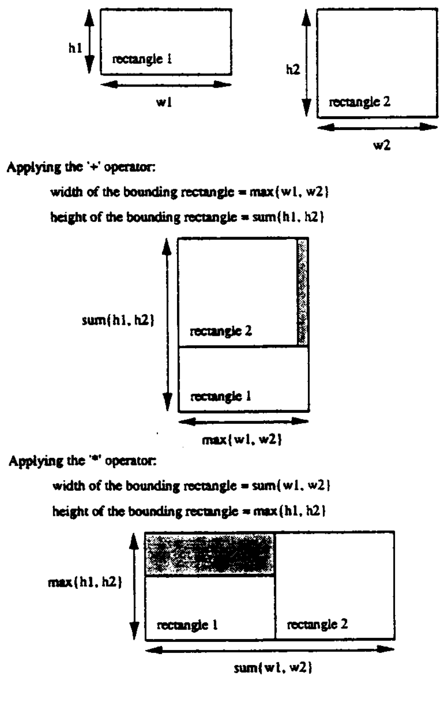
h1
rectangle 1
w1
h2
rectangle 2
w2
Applying the '+' operator:
width of the bounding rectangle = max(w1, w2)
height of the bounding rectangle = sum(h1, h2)
sum(h1, h2)
rectangle 2
rectangle 1
max(w1, w2)
Applying the '*' operator:
width of the bounding rectangle = sum(w1, w2)
height of the bounding rectangle = max(h1, h2)
max(h1, h2)
rectangle 1
rectangle 2
sum(w1, w2)

Fig. 4 applying the + and * operators to generate the bounding rectangle.

B. The Genetic Operators

. Based upon our polish expression representation, we design a set of genetic operators, including a crossover and several mutations, to manipulate it. The crossover operator in GA1 is called hybrid crossover. The hybrid crossover works as follows: First, it decompose the polish expression into two parts, the index part and the operator part. The index part is an ordered list, just like a list of cities to be visited in the traveling salesman problem. Thus we apply the partially matched crossover (pmx) [5] on the two selected parents to generate their offsprings and use the implementation of pmx by Lin [7]. The operator part specifies the type of operators (+ or *) and their position in the polish expression. If we have n items to be packed, then we have n-1 operators because + and * are both binary operators. The operator part are manipulated by uniform crossover [3] . If we regard the relative position of items as schemata, it is easy to see that both the children inherit some portion of the chromosome from each parent. If the parents are good packings, then the children may be good ones, too. See Fig. 5 and Fig. 6 for an example of hybrid crossover.

```
          Parent 1   1  4  3  2  +  5  *  +  *
          Parent 2   5  1  *  4  +  3  •  2  +
=>
                    Index Part        Operator Part
Parent 1            1  4 | 3  2  5 |   +4  *5  +5  *5
Parent 2            5  1 | 4  3  2 |   *2  +3  *4  +5
                           template    1   1   0   0
                        Ø (pmx)            Ø(ux)
Child 1             1  5 | 4  3  2 |   *2  +3  +5  *5
Child 2             4  1 | 3  2  5 |   +4  *5  *4  +5
=>
          Child 1   1  5  *  4  +  3  2  +  *
          Child 2   4  1  3  2  +  *  5  *  +
```

Fig. 5 An example of hybrid crossover

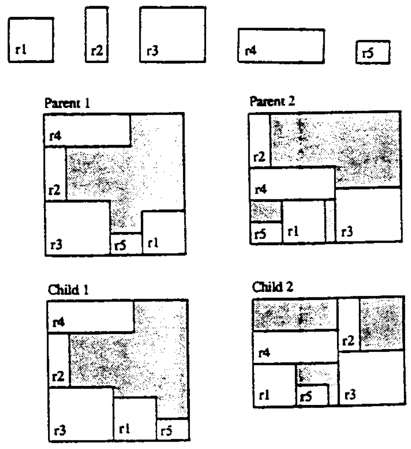

Fig. 6. The corresponding packings of Fig. 5.

We have proposed four different mutation operators:

Mut1: rotate an item
Mut2: randomly exchange two items
Mut3: move an operator
Mut4: complement an operator

Mutation 1 is to randomly choose an item and rotate it by 90 degrees. Mutation 2 is to randomly choose two items and exchange them. Mutation 3 is to move a randomly chosen operator to a new position. Mutation 4 is to replace a randomly chosen operator by the complement operator (the complement of + is * and vice versa). All of these mutation operators are applied with a given probability. See Fig. 7 for examples of mutation operators.

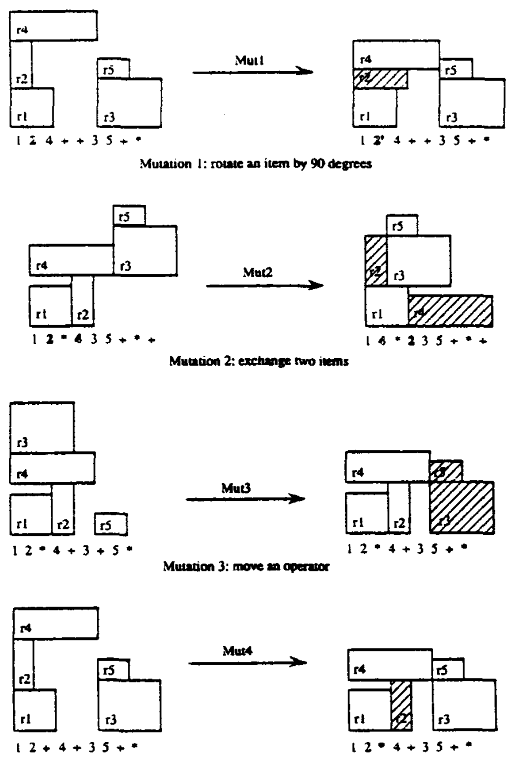

Fig. 7 Illustrations of four mutations

. However, Applying the hybrid crossover and Mut3 operator may cause illegal offspring. To solve this problem, we propose an algorithm to adjust the positions of operators to make the illegal polish expression legal. The idea of this algorithm is simply to count the operands(indices) and operators(+ or *). Since both the operators + and * are binary operators, the application of + or * consumes two operands and will produce a new operand. Thus, when scanning the polish expression from left to right, the accumulated number of operators cannot exceed the accumulated number of operands minus one at any point of the polish expression. See Fig. 8 for an example of applying the adjustment algorithm.

before adjustment	2 3 + * 4 5 1 + *
count	1 2 1 x invalid
after adjustment	2 3 + 4 * 5 1 + *
count	1 2 1 2 1 2 3 2 1 valid

Fig. 8. An example of applying the adjustment algorithm

C. The Evaluation Function

. We first apply the packing algorithm to a polish expression to obtain its bounding rectangle and then use this information to evaluate the corresponding packing. However, this information may not be enough to reveal the preference that the square packing is preferred. To meet this preference we add a penalty function. If the bounding rectangle is square or near square, the penalty is zero. When the difference between the width and height of the bounding rectangle becomes larger, the penalty grows larger, too. For example, given an previously specified allowable aspect ratio (e.g. AAR = 1.2), we may define the penalty as follows.

```
if((width/height)>AARor(width/height)<1/AAR)
  penalty = (width - height) * (width - height);
else
  penalty = 0;
```

Thus the fitness function is:

fitness=1/(area of the bounding rectangle, + penalty).

D. Initialization

. The initialization should generate the initial population to represent the entire solution space statistically. To initialize the population, we randomly generate an polish expression with n operands and (n-1) operators. The n operands are just a random permutation of 1, 2, 3, ..., n, and for each of the n-1 operators, we randomly generate its type (* or +) and its position in the polish expression. Of course, the randomly generated expression may not be a legal polish expression, therefore we need to apply the adjustment algorithm to make it legal.

E. The Parameters

. The population size is empirically set to 2 times the size of the problem (i.e. number of items to be packed). The steady-state reproduction strategy is used with 20 percent of the population updated in each generation. The newly generated offsprings are put back into the generation by deleting the least-fit chromosomes. No duplication of chromosome is allowed to maximize the diversity.

The rate for crossover is set to be 0.3 and the rate for all four mutations is set to be 0.7. The four mutations are selected with equal probability.

F. Experimental Results and Discussions

. The test problems are randomly generated as follows: a rectangle is specified by (area, ar), where area is the area of the rectangle and is generated randomly from 1 to 100, and ar is the aspect ratio of the rectangle with value randomly generated from 1 to 4. We test GA1 for six problems with size ranging from 10 to 60 on a 80286-based PC-AT. Each problem is run for 10 times . The experimental results are listed in Fig. 9. The average packing density is around 88%. The solution quality can be improved as the running time increased. For example, Fig. 10 shows an sample packing of 30 items with packing density of 95.6% full. This packing was obtained in 40 minutes running time of GA1.

problem	size	packing density	time elapsed
p1	10	92.74	3 mins
p2	20	89.16	5 mins
p3	30	90.62	10 mins
p4	40	87.69	18 mins
p5	50	85.20	30 mins
p6	60	85.99	57 mins

Fig. 9 Experimental results of GA1

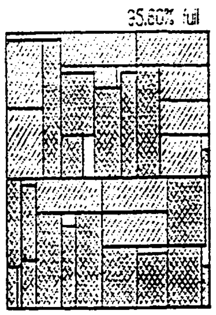

Fig. 10 A sample packing of 30 rectangular items

. For most problems, the mutation rate of traditional GAs is supposed to be very low, e.g. 0.5%, and the GA leaves most of the works of searching to the crossover operator. However, in the domain of bin packing this is not true because it is very difficult to combine the good features of two good packings. Thus, we take another point of view and use the high mutation rate instead. Conceptually, the crossover operator causes a larger jumps in the search space and the mutation operator acts as a local search close to the current approximate solution in the search space.

IV. On Solving RBP3: Minimizing the Number of Bins Used

. In this section, we demonstrate another way to use GAs. The two GAs developed in this chapter are called GA3 and GA4, respectively. They do not try to solve the problem directly, but to solve it from the point of view of an existing heuristic packing algorithm. In this case, GA acts like a heuristic improver more than a problem solver. The idea of our heuristic packing algorithm is from algorithm Hybrid First Fit (HFF) [2]. Empirical results of GA3 and GA4 are compared with that of HFF. Experiences tell us that this is a simple and efficient way to improve the performance of a heuristic algorithm.

. The problem is stated as below: Let L = {r1, r2, ..., rn} be a set of rectangular items, each item r have height h(r) and width w(r). A packing P of L into a collection {B1, B2, ..., Bm} of H * W rectangular bins is an assignment of each items to a bin and a position within that bin, such that (i) each rectangle is contained entirely within its bin, with its sides parallel to the sides of the bin, (ii) no two items in a bin overlap, and (iii) the number of bins used is minimized. For all items, rotation by 90 degrees is allowed.

A. HFF Heuristic

. The algorithm HFF is proposed by F.R. Chung, M.R. Garey and D.S. Johnson [2]. A more comprehensive name for the algorithm is FFDH * FFD since it is a combination of these two algorithms. It works as follows: First create a strip packing for L using FFDH and strip width W, thereby obtaining a collection {b1, b2, ..., bk} of blocks of nonincreasing heights h1 ≥ h2 ≥ ... ≥ hk, each containing a subset of the rectangular items. If we view these blocks as a new collection of rectangles L' = {b1, b2, ..., bk} with h(bi) = hi and w(bi) = W, 1 ≤ i ≤ k, we have an instance of the one-dimensional problem and can apply FFD to pack the blocks (and hence the rectangles they contain) into H * W bins. See Fig. 11 for an example of HFF.

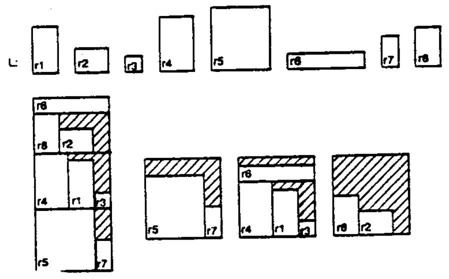

Fig. 11 An example of HFF

. From the descriptions of HFF above, we know that HFF is an off-line algorithm since it reorders the items in non-increasing height before it packs them. The arrangement of non-increasing height ordering of items is also a heuristics embedded in HFF. Thus, we decompose HFF into two parts: the first part is to

V. On Solving RBP2: Minimizing the Height of A Strip Packing

. In this section, we continue the idea from the previous sections, and develop 3 GAs to solve the strip packing problems RBP2. We denote them as GA2, GA5 and GA6. The GA2 is modified from GA1 by using a different evaluation function. GA5 and GA6 are modified from GA3 and GA4 with different embedded packing heuristics and evaluation function. Experimental results are compared with the classical deterministic strip packing algorithm FFDH.

. The strip packing problems is stated below: Given a set of rectangles pi, with height hi and width wi, the goal is to pack them into a vertical strip of width C, so as to minimize the total height of the strip needed. For all items, rotation by 90 degrees is allowed.

A. Modification of GA1

. The RBP2 can be viewed as a more constrained version of RBP1 with the constraint of keeping the bin width fixed. There are two obvious ways of handling the constraints in a GA: either (1) requiring that they are satisfied for every solution generated or (2) allowing constraint violation for the intermediate solutions at the expense of some penalty. Kröger, Schwendering and Vornberger [1] have proposed a GA to solve the strip packing problems with the BOTTOM-LEFT scheme embedded in the genetic operators to meet the bin width constraint and to generate good packings. Here, we try to modify the GA1 developed in previous chapter to accommodate the bin width constraint by adding an penalty function term.

. Since RBP2 is nothing more than a special case of RBP1 with the packing width not exceeding a constant C. If we meet this constraint when minimizing the total packing area, we minimize the height of the strip. Thus the idea from GA1 can be directly used here. The only modification is to embed the packing width constraint in the evaluation function, and the goal can be still to minimize the total packing area as it was before.
To minimize the area, we have
 fitness = 1 / (area of packing)

To meet the constraint, we add a penalty function term as follows:

```
if (width of packing ≤ C)
  penalty = 0;
else
. penalty = (width of packing - C) * height of packing;
```

The resulting evaluation function is:
 fitness = 1 / (area of packing + penalty).

The other parts of GA1 are left unchanged.

B. Modifications of GA3 and GA4

. We apply the concepts used in GA3 and GA4 to solve RBP2. GA5 is modified from GA3 and is a combination of GA and First-Fit packing algorithm. GA6 is modified from GA4 and is a combination of GA and Best-Fit packing algorithm. Because what we care in RBP2 is the height of the strip packing, the fitness function can be expressed as below :
. fitness = 1 / (height of the strip packing)

C. Experimental results & Discussions

. The test problems are the same as described in previous chapter. Ten problems with different sizes are tested by the three GAs. The running time of GA2 is roughly equal to the running time of GA1 for the same problem size, and the running time of GA5 and GA6 are comparable to GA3 and GA4. The results are compared with those obtained by FFDH and shown below:

Problem	size	FFDH	GA2	GA5	GA6
p1	10	25	17	17	15
p2	20	37	31	31	31
p3	30	51	46	46	46
p4	40	62	57	56	56
p5	50	71	68	66	64
p6	60	79	80	73	73
p7	70	97	?	90	93
p8	80	110	?	106	104
p9	90	127	?	120	122
p10	100	139	?	133	133

* the notation ? denotes that the results are not good and thus need further improvement

Fig. 13 Experimental results of GA2, GA5 and GA6

From the results above, we recognized that the solution quality of GA2 decrease the problem size sets larger. The results are not good. We guess that this may be because of that the bin width constraint was dealt with as a penalty function, which prunes the solution space too much and thus limits the GA2 to do the search efficiently. That is to say, to deal with the bin width constraint as a penalty may not work for this problem. To make GA solve the strip packing more efficient, we may need to incorporate the constraint in the representation and genetic operators so that every offspring meets the constraints. The paper by Kröger, Schwendering and Vornberger [1] demonstrate this idea. The other possible reason is the weight for the penalty function term is too small. This may be tested by increasing the weight of the penalty function term.

. Although the results of GA2 does not beat FFDH for every instance, the GA2 still have an advantage over FFDH in that the performance of FFDH is more stable than that of FFDH. For some problem instances, FFDH may obtain a worst case quality solution, but GA2 is expected to adapt itself to have a steady performance.

. For the results of GA5 and GA6, they beats that of FFDH for every cases. This is consistent with our discussions in previous chapter. Because GA5 and GA6 acts as tuners, their results can not worse than that of FFDH.

decide the packing order (non-increasing height order) and the second part is to packing the items according to the order presented to it. Obviously, the second part of HFF is an on-line algorithm and can be embedded in GA3 as part of the evaluation function. Then the work of GA3 is to find the optimal sequence of items for the packing algorithm. GA4 is modified from GA3 with slight differences in their embedded packing algorithm. What is in GA4 is a best-fit packing algorithm instead of first-fit algorithm in GA3.

B. The Representation

. Since most of the packing work is done by the packing algorithm, the only thing we can change is the order of items presented to it. That is, if we change the order of items presented to the packing algorithm, the solution quality may be improved. Thus, we represent a solution as an ordered list of items. With this representation, the solution of HFF is included in the solution space because if we let the order of items be non-increasing height, it would produce the same solution as that of HFF. Further, rotation of items by 90 degrees is allowed, so the representation is extended by added a rotation bit to each item in the solution vector to indicate their orientations. This extension of the representation not only provides an additional possibility to find solution that is better than that of HFF, but it also enlarge the solution space by the scale of 2n and the size of the final solution space becomes n! * 2n.

C. The Genetic Operators

. We apply the partially matched crossover (pmx) [5] on the extended list representation of chromosome and use the algorithm developed by Lin [7]. Two mutation operators are proposed, the first is random 2-swap mutation, which randomly selects two components in a chromosome and exchange them, and the second is the rotation mutation, which rotate a randomly selected gene of a chromosome by 90 degrees.

D. The Evaluation Function

. Before we evaluate a chromosome, we have to construct the actual packing using the embedded packing algorithm. The packing algorithm used in GA3 is a level-oriented first-fit (LFF) algorithm, inspired from algorithm HFF. Suppose the standard bin has width W and height H. It first uses a two-dimensional level-oriented first-fit algorithm to pack the set of items into a strip of width W. Next, decompose this packing into blocks corresponding to the levels created by the algorithm. Each block can be viewed as a rectangle of width W and height the height of the level. Thus, packing these blocks into rectangular bins of width W becomes a simple one-dimensional bin packing problem, where the size of an block is its height. Then we apply FFD to this one-dimensional problem. The packing algorithm used in GA4 is similar to that being used in GA3. The only difference is that it searches for the best fit area to pack the incoming item instead of using the firstly found area.

. The evaluate function is defined as below:

fitness=1/(bin_used*bin_height+average_height).

The first term bin_used * bin_height is to distinguish the packings by preferring a packing that uses a smaller number of bins. The second term average_height is to tell the difference between packings that uses the same number of bins and prefer the smaller average height of all bins.

E. Initialization

. This initial population is created by random permuting the n items, where n is the number of items to be packed. The orientation bit attached to each item is also generated randomly, meaning either no rotation or rotating by 90 degrees.

F. The Parameters

. For GA3 and GA4, the population size is set to be equal to the number of items to be packed empirically. The rate for crossover operator is 0.7 and the rate for mutation operators is 0.3. The two mutation operators, random 2-exchange mutation and rotation mutation, are selected with equal probability. The generational reproduction strategy is used for simplicity.

G. Experimental Results and Discussions

. The test problems are generated randomly with the width and the height of the items uniformed distributed from 1 to the width of the bin. We have run the algorithm for 10 problems, with problem sizes ranging from 10 to 100. Each problem is tested for 10 times. The results are summarized in Fig. 12.

	size	min*	HFF	GA3bins	saved	GA4 bins	saved
p1	10	2	2	2	0	2	0
p2	20	3	3	3	0	3	0
p3	30	6	7	6	1	6	1
p4	40	9	11	10	1	10	1
p5	50	12	15	14	1	14	1
p6	60	14	18	16	2	16	2
p7	70	16	22	19	3	19	3
p8	80	19	16	24	2	22	4
p9	90	22	29	27	2	25	4
p10	100	24	31	28	3	28	3

*min=È(total area of items)/(capacity of a bin)

Fig. 12 Experimental results of GA3 and GA4

. From the results above, we see that the solution quality is consistently better than that of HFF. The idea of using a GA to improve a heuristic packing algorithm is very simple, yet useful. It improve the average performance of the heuristic algorithm and have high probability to prevent the heuristic algorithm being trapped in the worst case.

VI. Conclusions and Future Works

. In this paper we demonstrated two different ways of using GAs to solve bin packing problems, either to solve the problem directly (as in GA1 and GA2) or to use the GA to tune an existed heuristic algorithm (as in GA3, GA4, GA5 and GA6) so that the performance of the heuristic algorithm is improved. The slicing tree representation for a packing in GA1 have the advantage of evaluation efficiency and totally freedom to search without the interference of a heuristic algorithm. And the GA2 shows that the method of adding the bin width constraint as a penalty function may not work alone. That is, we need different way to manipulate the bin width constraint, for example, to embed this constraint in the representation and the associated genetic operators. The value of GA3, GA4, GA5 and GA6 are their simplicity. We can use this concept to improve an algorithm without making too much effort.

. Using genetic approach to solve the rectangle bin packing problem has a monotone property. The more running time, the better solution you find. This is a practical advantage of this approach. Since GA maintains a population of candidate solutions, we can choose any good alternative solution from the population as our final solution if the best one (measured by combination of density and penalty violation in GA1) is discarded for some reasons. One disadvantage is that each time we run the algorithm, we will end up with a different packing, i.e. it is difficult to do reactive packing.

. The results of our GA approach to bin packing problems can be used directly in cutting-stock problems. And with appropriate modifications, GA1 can be used in the floorplan design of VLSI design [9]. The modification should be easy by embedding the flexible modules knowledge to the evaluation function.

. The first direction of research is to incorporate simulated annealing into GAs so that the performance is improved. The works by Lin[7] is a good demonstration of this kind of work.

. Another direction of research is to develop a more powerful representation so that GA1 can manipulate the packings that can't be represented by our slicing tree representation. If the representation is modified, the corresponding genetic operators should also be designed again. Besides, if we can devise a more efficient crossover operator, then both the quality of the solution and the running time will be improved.

. Yet another direction of research is to investigate different ways to manipulate the constraints in a GA.

. Finally, using the idea of GA3 to improve the 1-D bin packing algorithms and other heuristic algorithms is useful in practice.

References

[1] Berthold Kröger, Peter Schwendering and Olive Vornberger, "Parallel genetic packings of rectangles," proc. of the first workshop on Parallel Problem Solving from Nature (PPSN 1), Dortmund, FRG, Oct. 1-3, 1990, pp.160-164.

[2] Chung, F.R.K., Garey, M.R. and Johnson, D.J., "On packing two-dimensional bins," SIAM J. Alg. Disc. Meth., 3, 1982, pp.66-76.

[3] Davis, L. (Editor), Handbook of Genetic Algorithms, Van Nostrand Reinhold, Reading Mass, 1991.

[4] DeJong, K.A., "Genetic Algorithms: A 10 Year Perspective," Proceedings of the First International conference on Genetic Algorithms, 1985, pp. 169-177.

[5] Goldberg, D.E., Genetic algorithms: In search, Optimization and Machine Learning, Addison-Wesley Publishing Company, Reading Mass, 1989.

[6] Holland, J.H., Adaptation in Natural and Artificial Systems, the university of Michigan Press, Ann Arbor, 1975.

[7] Lin, F.-T. and Kao, C.Y., Incorporating the Genetic Approach to Simulated Annealing in Solving Combinatorial Optimization Problems, Ph.D. Dissertation, National Taiwan Univ., 1991.

[8] Smith, Derek, "Binpacking with Adaptive Search," Proc. of an Intern. Conf. on Genetic Algorithms and Their Application, Pittsburgh P.A., 1985, pp. 202-206.

[9] Wong, D.F., Leong H.W., and Liu, C.L., Simulated Annealing For VLSI Design, Kluwer Academic Publishers, 1988.