ELSEVIER

European Journal of Operational Research 145 (2003) 530–542

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

www.elsevier.com/locate/dsw

Discrete Optimization

Application of a mixed simulated annealing-genetic algorithm heuristic for the two-dimensional orthogonal packing problem

T.W. Leung [a], Chi Kin Chan [b], Marvin D. Troutt [c,*]

[a] *Diocesan Girls' School, Kowloon, Hong Kong*
[b] *Department of Applied Mathematics, The Hong Kong Polytechnic University, Kowloon, Hong Kong*
[c] *Department of Management and Information Systems, Kent State University, Kent, OH 44240, USA*

Received 23 August 2000; accepted 10 January 2002

Abstract

In this paper a pure meta-heuristic (genetic algorithm) and a mixed meta-heuristic (simulated annealing-genetic algorithm) were applied to two-dimensional orthogonal packing problems and the results were compared. The major motivation for applying a modified genetic algorithm is as an attempt to alleviate the problem of pre-mature convergence. We found that in the long run, the mixed heuristic produces better results; while the pure heuristic produces only "good" results, but produces them faster.

Keywords: Meta-heuristics; Mixed heuristics; Simulated annealing; Genetic algorithm; Two-dimensional orthogonal packing problem; Difference process strategy

1. Introduction

The two-dimensional orthogonal packing problem consists of packing rectangular pieces of predetermined sizes into a large but finite rectangular plate (the *stock plate*), or equivalently, cutting small rectangular pieces from the large rectangular plate. We wish to find "packing patterns" that minimize the unused area (*trim loss*). The problem has obvious relevancy to the textile, paper, and other industries, and in the three-dimensional case, is related to the problem of packing boxes into a container.

The problem has been formulated as an integer program. For that approach and variations, see Beasley (1985), Tsai (1993) or Christofides (1995). Recently, different meta-heuristics have been applied to problems of this kind, for instance, see Parada et al. (1998), Lai and Chan (1997), Jakobs (1996), Glover et al. (1995), Dowsland (1993, 1996). For an introduction to meta-heuristics, see Osman and Laporte (1996). Based on the papers of Jakobs (1996), also Lai and Chan (1997), we have carried out extensive comparisons of these heuristics (Leung et al., 2000). An important building

* Corresponding author. Tel.: +1-330-672-1145; fax: +1-330-672-2953.
E-mail address: mtroutt@bsa3.kent.edu (M.D. Troutt).

0377-2217/03/$ - see front matter
PII: S0377-2217(02)00218-7

block in any area of research is a valid comparison among the different theories or models that were proposed in the field. Such a comparison is missing in the area of meta-heuristics for the two-dimensional orthogonal packing problem.

In our investigation, we found that the pure genetic algorithm meta-heuristic usually produces good results. Nevertheless, we also found that this heuristic converges very fast; thus, good results were produced at early stages. That is, while good, these results are improvable if what we call "premature convergence" can be avoided. For this purpose, we employ a mixed meta-heuristic and test its performance.

2. Methodology

A packing pattern may be represented by a permutation, which corresponds to the sequence in which the small rectangles are packed. Other representations may be found in Beasley (1985) or Dowsland (1993, 1996). In this paper we concentrate on the permutation representation. Now if we have a permutation, we know the order of packing the small pieces, but we may still consider different strategies for packing the pieces. In Jakobs's (1996) paper, the problem of packing rectangular pieces into a rectangle of finite width but infinite height was considered (the two-dimensional bin-packing problem). He used a genetic algorithm to find permutations of small trim loss (measured by the height occupied by the pieces). After a permutation was obtained, he used the bottom left (BL) strategy (placement policy) to pack the pieces. Lai and Chan (1997) used simulated annealing to find permutations of small trim loss (measured by percentage of unused area). After a permutation was obtained, these authors used the difference process (DP) strategy (placement policy), detailed below, to pack the rectangular pieces. As the DP strategy has the ability to fill holes by small pieces, the resultant pattern is usually more compact. Hence in this paper we will only employ the DP placement strategy. We summarize our work as follows:

1. We develop a mixed heuristic, combining the simulated annealing and genetic algorithm approaches; the mixed heuristic is denoted by MSAGA. A simple genetic algorithm (SGA) approach will also be used for comparison purpose. These heuristics are used to find permutations of small trim loss. This process is also called encoding, and a permutation thus found is a genotype.
2. We use the DP placement strategy to pack the pieces corresponding to a particular permutation. This process is also called decoding, and the pattern thus formed is also called a phenotype. We then calculate the trim loss corresponding to that phenotype, and then we go back to step 1.

For the sake of simplicity, we make the following assumptions:

1. All pieces have fixed orientation, i.e., a piece of length l and width w is different from a piece of length w and width l if l does not equal w.
2. The pieces must be placed into the stock plate orthogonally, thus the sides of the small rectangles are parallel to the stock plate.
3. The length and width of each piece does not exceed the corresponding dimensions of the stock plate, i.e., the plate has enough space for holding each piece when the plate is empty.
4. The dimensions of the pieces and the stock plate are integers. Therefore the placements or cuts on the stock plate are to be done in integer steps along the horizontal and the vertical edges of the plate. This limitation is not serious since in practice the actual dimensions can be scaled up to become integers by using a sufficiently large factor.
5. All cuts on the stock plate are infinitesimally thin, i.e. the edges of the pieces do not occupy any area.
6. Each piece may be positioned at any place in the stock plate and in proximity to any other piece in the plate. That is, there is no restriction that two pieces cannot be contiguous.

We first briefly describe the DP placement strategy and the meta-heuristics and then give a summary of our results.

3. The difference process strategy

In the difference process strategy, while trying to pack a new piece, the "empty spaces" are considered. In short, a space is a rectangular area of largest possible size, with sides parallel to the stock plate, which is not yet occupied by any rectangular pieces. One may observe that in the packing process, there are only finitely many spaces. At the very beginning, there is only one space and its bottom left corner is precisely at the origin. After the first piece is packed, there are (usually) two new spaces generated, (unless the height or the width of the first piece is the same as that of the stock plate). We then calculate the distances (called the anchor distances) between the bottom left corners of the spaces and that of the origin, and we pack the second piece, if possible, in the space nearer to the origin (packing as compactly as possible).

Every time a new piece is packed, we calculate how this piece intersects with the existing spaces. From each existing space, which intersects with the piece non-trivially, it is not hard to see that at most three new rectangular unused areas are generated. From the new areas generated, we check if one is entirely contained in the other, and remove it in that case. This is the so-called elimination process in Lai and Chan (1997). In this manner, we update our space list whenever a piece is packed.

4. Genetic search algorithm

A genetic search algorithm is a heuristic search process that resembles natural selection. There are many variations and refinements, but any genetic algorithm has the features of reproduction, crossover and mutation. Initially a population is selected, and by means of crossovers among members of the population or mutations of members, the better of the population will remain, because of the "survival of the fittest". This idea was first proposed by Holland (1975), among others. For its many applications and details, see Goldberg (1989), Dasgupta and Michalewicz (1997), and Mazumder and Rudnick (1999).

The idea of applying genetic search to the bin-packing problem was suggested by Jakobs (1996), who employed the BL strategy as the decoding procedure. Using the terminology of genetics, a population is a set of feasible solutions of the problem. A member of the population is a genotype, a chromosome, a string or in fact a permutation which corresponds to the order of packing rectangular pieces. When a genotype is decoded, a packing pattern is formed, and it is called a phenotype. We may calculate its *fitness* value (percentage of total area of the stock plate packed by pieces) and its trim loss. We describe the procedures as follows:

1. First we set the mutation rate, p_m. Then we create an initial population by a random process.
2. If the maximum number of possible iterations has been exceeded, then we stop the process and choose the best solution in the population as the final solution of the problem.
3. We calculate the fitness values of the strings in the population and choose two from them by using a (biased) roulette wheel.
4. We then take a random number r between 0 and 1. If $r \leqslant p_m$, produce an offspring from the first of the two chosen strings by a mutation process and update the population by replacing the worst string in the population by this offspring. Go to step 2. If $r > p_m$, apply a suitable crossover process to the two chosen strings and produce an offspring. Update the population by replacing the worst string in the population by this offspring. Go to step 2.

A few term descriptions and specifics are as follows:

Population. There is no clear indication as to how large a population should be. The considerations are: if the population is too large, there may be difficulty in storing the data, but if the population is too small, there may not be enough strings for good crossovers. In our experiments, the populations range from 10 to 100.

Biased roulette wheel (*or proportional selection*). First, we calculate the fitness values of the strings in the population. By taking the fitness value of a

member divided by the sum of the fitness values of the population, we get the relative fitness value of a member. Then we partition $[0,1]$ into subintervals, with each interval having length equal to the relative fitness of a string in the population. Two random numbers from $[0,1]$ are then chosen. The two values will fall into two subintervals, and we then choose the two strings corresponding to those two subintervals.

Mutation. If the entire population has only one type of string (or strings are very similar), then the crossover of two strings is unlikely to produce new strings that are different and hopefully superior. To escape from this scenario, the mutation operator may be used. The mutation operator randomly selects two genes (entries) in a string, and swaps the positions of these two genes to produce an offspring. Then this offspring will replace the worst string in the population. This process provides a mechanism to escape from local optima. In our case, if the randomly chosen number is smaller than the pre-determined mutation rate, then we apply the mutation operator to one of the two chosen strings. Thus, if the mutation rate is large (close to 1.0), then it is more likely that the process will be applied, and vice versa. If the mutation rate is 1.0, then there are no crossovers, reminiscent of the swapping or shifting operations in simulated annealing. Here we have a larger population to start with; thus, intuitively, it should be possible to get a good solution sooner.

Crossover. The crossover operation corresponds to the concept of mating. It is hoped that the crossover (mating) of good parents may produce a good offspring. Thus, the crossover operation is a simple yet powerful way of exchanging information and creating new solutions. We employ two different crossover operators, modified from Jakobs (1996) and Goldberg (1989). Because in this problem chromosomes are permutations corresponding to sequences of packing the pieces, it makes sense intuitively to preserve subsequences of effective orderings.

Crossover operator A: For each pair of strings (parents), two cut points are generated randomly along the positions of the strings. The elements (genes) lying between these two cut points of the first string are taken out. The vacant positions are filled with respect to the same order by elements (of the second string), which also appear between the cut points on the first string. This gives the first child. Now certain elements of the second string are removed. They are replaced by elements of the first string between the cut points, with respect also to the same order of appearance. For example, suppose $(6,4,5,3,8,9,1,2,7)$ is the first parent string and $(3,9,4,6,8,5,2,7,1)$ is the second parent string. The cut points enclose 3, 8, 9 of the first string. These elements are removed. In the second parent string, the elements are in the order 3, 9, and 8. Thus the first child is $(6,4,5,3,9,8,1,2,7)$. Now 3, 9 and 8 of the second string have been removed and are then replaced by 3, 8, and 9, respectively, (elements within cut points of the first parent). Thus the second child is $(3,8,4,6,9,5,2,7,1)$. We see that the first child is close to the parent, hence in case the parent is a good solution, the child remains close. While the second is in a sense further apart. This may help us to diversify.

Crossover operator B: First, generate randomly a position on one of the parent strings, (call it the first parent). From the mapping of the first parent string to the second, we form a cycle starting from the gene at the position generated. If the elements of the cycle form a proper subset of all the elements of the same string, we can swap all the corresponding elements positioned by the cycle to form two new strings. For example, suppose again that the first parent is $(6,4,5,3,8,9,1,2,7)$, and the second is $(3,9,4,6,8,5,2,7,1)$ Starting from the second position of the first string, we get a cycle $(4,9,5)$. Upon swapping, we get two children $(6,9,4,3,8,5,1,2,7)$ and $(3,4,5,6,8,9,2,7,1)$.

These two operators are then selected with equal probabilities. We denote this heuristic by SGA.

5. Simulated annealing and genetic algorithms

After many reported experiments in the literature, genetic algorithms have been found to be efficient and robust. In addition, it is usually straightforward to encode a problem suitable for application of genetic algorithms. Nevertheless,

genetic algorithms also have their shortcomings. In fact, if the worst members are discarded after each generation, the population will tend to quickly become homogeneous. If the mutation probability is small then chromosomes of large variation in fitness may not be produced. This phenomenon may be called "pre-mature convergence". Also, because the population becomes homogeneous very soon, crossovers may not produce offspring of sufficiently large variation, and the search process may become very slow. Several authors have suggested modified genetic algorithms to address this difficulty, for example, introduction of an elite group from the population (Thierens and Goldberg, 1994; Chan et al., 1999), or applications of disruptive selections (Kuo and Hwang, 1996).

There is yet another alternative. Some authors (Lin et al., 1993) have suggested inserting another operator, namely, a Boltzmann-type operator, after the crossover and mutation operations. This operator will induce competition between parents and children. The idea is, if the children are better, we accept the children, but if the children are not as good, we may still accept the children with some positive probability. This helps to prevent the population from becoming homogeneous too soon. There is a close analogy between this approach and the thermodynamic process of annealing (cooling of a solid). Thus, it is also called the simulated annealing (SA) approach. It was Metropolis et al. (1953) who first proposed this idea, and 30 years later, Kirpatrick (1983) observed that this approach could be used to search for feasible solutions of quite general optimization problems. The main idea is that the SA strategy may help prevent being trapped at poor solutions associated with local optima of the fitness function.

By adding this Boltzmann-type operator, or in other words, a SA operator, we obtain a mixed simulated annealing-genetic algorithm (MSAGA). The steps of MSAGA are detailed as follows:

1. Randomly initialize a population of potential solutions or chromosomes.
2. Evaluate the fitness of each chromosome in the population.
3. Select chromosomes for reproduction with probabilities proportional to fitness.
4. Apply crossover to the selected chromosomes to produce new chromosomes (children).
5. Apply mutation to the new chromosomes.
6. Evaluate the new chromosomes.
7. Apply the SA operator to decide which two of the parents and children (four chromosomes altogether) remain.
8. Decrease the temperature as explained further below.
9. If the stopping criterion is reached, return the best solution. Otherwise, go to step 3.

The procedures are same as SGA, except when applying the SA operator. We describe further on this point.

Temperature and iterations. First, we set the number, say N, of iterations (generations) to carry out the algorithm. Then we set what are known as the initial temperature, T_0, and the final temperature, T_1 $(T_0 > T_1)$. We decrease the temperature T after every generation, usually by a proportion α (cooling rate), so that after N generations, the temperature becomes $T_1 = \alpha^N T_0$.

SA operator. The SA operator is applied as follows. First we compare parent 1 and child 1, if child 1 is better, we replace parent 1 by child 1. If parent 1 is better, we may also replace parent 1 by child 1 with a certain chosen probability. The same procedure will then be applied to parent 2 and child 2.

Suppose the fitness values (percentages of area of the stock plate filled by the pieces) corresponding to the child is F_c and to the parent F_p, with $F_p > F_c$, (parent is better). Let the current temperature be T. A random number, r, between 0 and 1, is selected and then we accept the child if

$$\exp\left[\frac{F_c - F_p}{TF_p} \cdot 100\%\right] > r.$$

Thus, a child may be selected with some probability even if it is inferior compared to the parent. It is easy to see that, initially when the temperature is high, there is a higher probability of accepting an inferior child. The above exponential function is occasionally called a Boltzmann function, thus the operator is also called a Boltzmann-type operator. Of course, if a child produced is better, then it is accepted. We put one further restriction on this operator. Since the discarded parent or

child may be lost forever, in case the parent is the best in the population, then we will not carry out the operation, i.e., the parent will be kept.

We next compare the performances of the genetic algorithm (SGA) and the mixed simulated-annealing genetic algorithm (MSAGA) on the orthogonal packing problem.

6. Experimental results

We have altogether 19 test problems. The first eight problems came from Lai and Chan (1997) and Jakobs (1996). The 9th and the 10th problems were our own constructions. They all have a known optimal solution of zero trim loss, so that the absolute performances of the various algorithms can be determined. The number of rectangular pieces in each stock plate ranges from 10 to 50. The data sets for these 10 problems are given in Tables 1–10. The 11–19th problems are from Tables 13–15 of Hopper and Turton (2000). The number of rectangular pieces in these problems ranges from 70 to 200. From our experimental results, it is also quite likely they all have zero trim loss.

The algorithms were written in C and run on a 586 personal computer with 400 MHz. For each test problem, all the meta-heuristics were run 15

Table 1
Test Problem 1 – Size of the stock plate: Width = 400, Height = 200

Piece	Width	Height	Piece	Width	Height
1	200	100	6	100	60
2	100	50	7	60	60
3	100	50	8	40	100
4	100	100	9	160	40
5	100	120	10	200	40

Table 2
Test Problem 2 – Size of the stock plate: Width = 400, Height = 200

Piece	Width	Height	Piece	Width	Height
1	100	50	9	200	40
2	100	50	10	100	90
3	100	30	11	100	50
4	100	70	12	50	60
5	100	100	13	50	60
6	100	50	14	50	60
7	100	50	15	50	60
8	200	40			

Table 3
Test Problem 3 – Size of the stock plate: Width = 400, Height = 400

Piece	Width	Height	Piece	Width	Height
1	170	50	11	50	150
2	170	50	12	150	110
3	30	90	13	150	110
4	30	90	14	50	150
5	120	40	15	120	40
6	120	40	16	120	40
7	50	150	17	30	90
8	150	110	18	170	50
9	50	150	19	170	50
10	150	110	20	30	90

Table 4
Test Problem 4 – Size of the stock plate: Width = 70, Height = 80

Piece	Width	Height	Piece	Width	Height
1	17	9	11	21	9
2	17	9	12	24	15
3	18	9	13	25	15
4	18	9	14	10	31
5	24	15	15	11	31
6	25	15	16	14	20
7	24	16	17	16	20
8	25	16	18	10	20
9	10	31	19	9	25
10	11	31	20	40	5

Table 5
Test Problem 5 – Size of the stock plate: Width = 70, Height = 80

Piece	Width	Height	Piece	Width	Height
1	29	8	14	16	20
2	29	8	15	16	20
3	29	8	16	9	36
4	12	16	17	25	4
5	12	16	18	29	14
6	5	12	19	29	5
7	22	18	20	16	19
8	22	18	21	23	21
9	19	16	22	22	4
10	19	16	23	22	4
11	19	4	24	11	13
12	15	4	25	11	13
13	15	4			

Table 6
Test Problem 6 – Size of the stock plate: Width = 120, Height = 45

Piece	Width	Height	Piece	Width	Height
1	36	18	14	6	12
2	12	21	15	24	18
3	18	21	16	24	9
4	6	15	17	18	9
5	30	6	18	18	9
6	18	12	19	6	18
7	12	6	20	24	12
8	21	27	21	24	6
9	12	18	22	9	15
10	12	18	23	9	12
11	12	15	24	6	15
12	12	15	25	6	12
13	18	12			

times and we took averages of the results. As we were interested in the long-term behavior of these heuristics, each run was set to consist of 30,000 iterations.

Table 7
Test Problem 7 – Size of the stock plate: Width = 90, Height = 45

Piece	Width	Height	Piece	Width	Height
1	15	18	16	12	9
2	21	18	17	12	9
3	9	18	18	12	9
4	12	18	19	12	9
5	12	9	20	12	6
6	12	12	21	12	6
7	12	6	22	12	12
8	18	6	23	18	18
9	18	9	24	6	15
10	18	12	25	6	15
11	9	12	26	9	12
12	9	12	27	9	12
13	6	15	28	6	12
14	9	9	29	9	9
15	12	6	30	9	18

Table 8
Test Problem 8 – Size of the stock plate: Width = 65, Height = 45

Piece	Width	Height	Piece	Width	Height
1	9	12	16	17	12
2	9	12	17	14	6
3	9	12	18	14	9
4	9	12	19	6	15
5	6	12	20	6	15
6	6	12	21	9	9
7	6	12	22	9	18
8	6	12	23	15	9
9	6	12	24	15	6
10	12	9	25	15	12
11	12	9	26	9	15
12	6	9	27	9	6
13	6	9	28	9	6
14	6	9	29	6	6
15	17	6	30	11	12

Although there were only 19 test problems, the large number of parameter variations led to a large number of experiments. For instance, in carrying out SGA, we have the choices of the following parameters:

Mutation rate: 0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0.

Population size: 20, 30, 40, 50, 60, 70, 80, 100.

Overall results. Despite the recent controversies over the uses of evolutionary algorithms (Wolpert and Macready, 1997; Koehler, 1997), we found that the use of these meta-heuristics produce reasonable results for this problem class. Of all the test problems we considered, the trim losses go from 0% (easier problems with 10–15 rectangular pieces) to 6% (for more difficult problems with 30–50 rectangular pieces), which are within acceptable standards (Lai and Chan, 1997; Jakobs, 1996). In terms of trim losses and CPU times, we found that the results are quite robust, run after run, thus adding reliability to this kind of algorithm.

Particular results for genetic algorithms. There is no clear evidence that larger populations will produce better offspring. We tried populations of sizes from 20 to 100, and we found that the best results occurred when the sizes of the populations

Table 9
Test Problem 9 – Size of the stock plate: Width = 150, Height = 110

Piece	Width	Height	Piece	Width	Height	Piece	Width	Height
1	20	32	18	10	24	35	16	10
2	20	36	19	12	14	36	44	12
3	20	18	20	32	14	37	10	22
4	38	14	21	32	20	38	22	22
5	38	20	22	8	20	39	32	16
6	22	40	23	24	20	40	12	38
7	16	4	24	42	6			
8	20	18	25	36	24			
9	28	18	26	36	6			
10	12	24	27	30	20			
11	8	24	28	16	20			
12	48	16	29	10	38			
13	20	20	30	10	34			
14	18	20	31	20	34			
15	18	10	32	14	4			
16	18	8	33	24	22			
17	10	36	34	16	16			

Table 10
Test Problem 10 – Size of the stock plate: Width = 160, Height = 120

Piece	Width	Height	Piece	Width	Height	Piece	Width	Height
1	20	40	18	16	12	35	14	26
2	14	22	19	32	12	36	18	14
3	8	34	20	32	18	37	18	22
4	6	34	21	20	6	38	14	10
5	6	30	22	30	6	39	8	14
6	6	20	23	10	20	40	8	26
7	30	24	24	10	26	41	16	26
8	40	24	25	22	10	42	10	12
9	44	6	26	32	22	43	12	12
10	30	10	27	12	12	44	30	22
11	14	28	28	20	36	45	40	22
12	16	24	29	18	28	46	10	28
13	8	24	30	18	8	47	38	28
14	24	14	31	30	14	48	22	16
15	18	20	32	16	22	49	48	8
16	14	32	33	34	26	50	22	20
17	36	20	34	14	36			

are around 60 to 100. Apparently, if a population is well chosen then good offspring will be produced rather fast. In terms of choices of mutation rates, we found that higher mutation rates usually work better. Perhaps high mutation rates will prevent the population from becoming too homogenized. In our experiments, the best results occurred when the mutation rates were from 0.7 to 1.0.

SGA versus MSAGA. When applying SGA, we observed that initially the mean trim losses of the populations dropped very fast. Nevertheless, usually after 4000–5000 iterations, perhaps because the populations may have become too homogenized, the mean trim loss remained stable. While on the other hand, because in MSAGA inferior solutions were occasionally accepted, the drop was not as sharp. Nevertheless, we observed that the drop usually continued and eventually overtook the performance of the SGA cases. In Fig. 1, the mean trim losses of the population in a

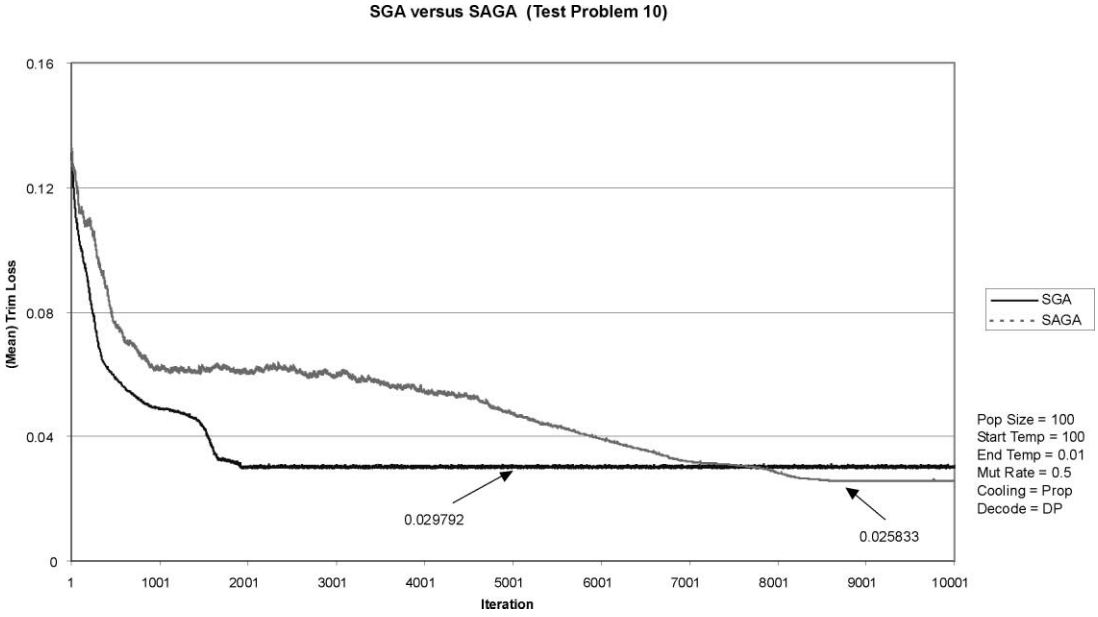

Fig. 1. Mean trim losses in a population versus iteration numbers.

particular run are plotted against iteration number. In Fig. 2, the lowest trim loss ever produced in a particular run is plotted against iteration number.

Table 11 summarizes results of comparisons between SGA and MSAGA. For each problem, we ran it 15 times, and each run consists of 30,000 iterations. In each run, we keep track of the best

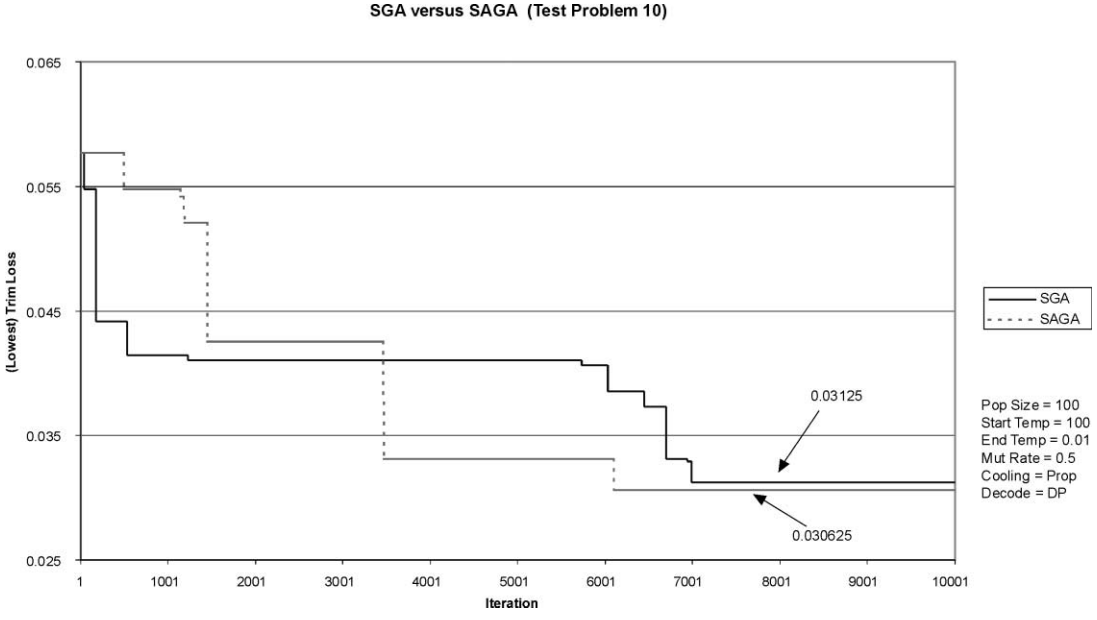

Fig. 2. Best trim losses in a population versus iteration numbers.

permutation (lowest trim loss) ever produced. After 15 runs, we then take the mean, and the minimum of these 15 numbers. The results are summarized in Table 11. Note that the "means" in the table differ from the "means" in Fig. 1. (In Fig. 1 the mean corresponds to the average of trim losses in a population.) These results show that the MSAGA method usually produces better results in

Table 11
SGA versus MSAGA

Test cases	SGA			Mixed SAGA		
	Decoder = DP, Pop. size = 100			Decoder = DP, Pop. size = 100 Start temp. = 100, End temp. = 0.01		
	Mutation rate			Mutation rate		
	0.3	0.7	1.0	0.3	0.7	1.0
Problem 1	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000
	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000
Problem 2	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000
	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000
Problem 3	0.02250000	0.01350000	0.01462500	0.01687500	0.01237500	0.02400000
	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000
Problem 4	0.03673810	0.02909520	0.02313100	0.00182143	0.01204760	0.02122620
	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000
Problem 5	0.03705950	0.03819050	0.03369050	0.02660710	0.03111900	0.03279760
	0.02142860	0.02589290	0.02428570	0.01571430	0.02142860	0.02142860
Problem 6	0.01288890	0.00911111	0.00611111	0.00177778	0.00266667	0.00400000
	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000	0.00000000
Problem 7	0.01896300	0.01777780	0.01718520	0.00948148	0.01185190	0.01777780
	0.01777780	0.00000000	0.00000000	0.00000000	0.00000000	0.01777780
Problem 8	0.02140170	0.02222220	0.01736750	0.01312820	0.00929915	0.01996580
	0.01230770	0.00000000	0.00000000	0.00000000	0.00000000	0.01230770
Problem 9	0.03335760	0.02946260	0.02799190	0.03107880	0.02614950	0.02555150
	0.02496970	0.02278790	0.02036360	0.01890910	0.01527270	0.01090910
Problem 10	0.02980560	0.02765280	0.02641670	0.02666670	0.02325000	0.02756940
	0.02208330	0.02250000	0.01895830	0.02270830	0.01958330	0.02083330
Problem 11	0.00993827	0.00971605	0.00839506	0.00979012	0.00833333	0.01024690
	0.00666667	0.00388889	0.00388889	0.00444444	0.00462963	0.00648148
Problem 12	0.00753086	0.00672840	0.00764198	0.00543210	0.00600000	0.00677778
	0.00111111	0.00259259	0.00407407	0.00148148	0.00259259	0.00370370
Problem 13	0.01308640	0.01116050	0.00998765	0.00918519	0.00924691	0.00962963
	0.00740741	0.00814815	0.00518519	0.00518519	0.00333333	0.00677777
Problem 14	0.01202080	0.01086810	0.01079170	0.00835417	0.00968750	0.01034720
	0.00729167	0.00645833	0.00687500	0.00562500	0.00583333	0.00656250
Problem 15	0.00706250	0.00675000	0.00695833	0.00438889	0.00530556	0.00733333
	0.00458333	0.00437500	0.00333333	0.00208333	0.00260417	0.00395833
Problem 16	0.01069440	0.00917361	0.00793056	0.00759722	0.00695139	0.00857639
	0.00854167	0.00583333	0.00447917	0.00468750	0.00447917	0.00427083
Problem 17	0.01643750	0.01545310	0.01411110	0.01506770	0.01312670	0.01487850
	0.01429690	0.01388020	0.01171880	0.01190100	0.01119790	0.01226560
Problem 18	0.00652083	0.00603472	0.00601562	0.00664757	0.00569271	0.00627778
	0.00375000	0.00484375	0.00375000	0.00283854	0.00442708	0.00453125
Problem 19	0.01076040	0.01034030	0.00962847	0.00982118	0.00920313	0.00959549
	0.00794271	0.00804688	0.00776042	0.00791667	0.00776042	0.00723958

Upper row: Mean of the best trim losses in 15 runs.
Lower row: Minimum of the best trim losses in 15 runs.

the long run. The differences are more pronounced when mutation rates are low, as crossover operators play a relatively more important role in that event.

7. Final remarks

Theoretically, we can tune the cooling process and the iteration numbers of a SA to achieve the global optimum (Mahfoud and Goldberg, 1995). Nevertheless, this implies that longer times may be required to achieve optimal or good solutions. Moreover, the serial nature of SA prevents us from employing parallelism. On the other hand, though GA exhibits implicit parallelism it is not easy to control the GA's convergence. Formally, there is also no proof of its global convergence. See, however, discussions in Rudolph (1994). It is therefore natural to ask if the cooling process may help to improve the convergence of the GA, and in practice, if this new modification may help to accelerate the process or to produce better results.

In practice, we observe that both the SA and the GA may converge prematurely to sub-optimal solutions. In the GA case, we use crossovers and mutations to maintain the diversity of a population. However, employment of these operators may also cause disruptions of good sub-optimal solutions. Moreover, if a population is becoming too homogenized, crossovers of members from a population may not be able to produce different and good solutions. Mutations may help to produce good solutions, but application of mutations may take a very long time to achieve a desirable solution. It is difficult to adjust the parameters, and appropriate adjustments may be problem-specific. Thus, we observe premature convergence in many of our experiments. From our experimental results (Leung et al., 2000), GA consistently performs better than SA. See, however, different views in Ingber (1992). We therefore try to add a cooling element in our GA process and to compare the results between GA and the mixed SAGA. Various authors have also employed hybrid strategies of this kind to handle different optimization problems (Wang and Zheng, 2001; Boseniuk et al., 1987; Lin et al., 1993).

In general we observe the MSAGA performs slightly better than simple GA. Initially, because we may keep inferior solutions, it is expected that the mean trim loss of the members in a population of MSAGA is not as good as that in the GA. However, in many cases, because the GA converges and quickly becomes homogenized very soon, the SA process helps to locate better solutions. Hence, we observe that the MSAGA eventually overtakes the GA (Figs. 1 and 2). This is an area to be explored further in later research.

Acknowledgement

Thanks to C.H. Yung who provided assistance with the programming work.

References

Beasley, J.E., 1985. An exact two-dimensional non-guillotine cutting tree search procedure. Operations Research 33, 49–64.

Boseniuk, T., Ebeling, W., Engel, A., 1987. Boltzmann and Darwin strategies in complex optimization. Physics Letters A 125, 307–310.

Chan, C.K., Cheung, B.K.S., Langevin, A., 1999. A modified genetic algorithm for the joint replenishment problem. In: Proceedings of the 3rd International Conference in Industrial Engineering, Montreal, Canada, pp. 1281–1289.

Christofides, N., 1995. An exact algorithm for general, orthogonal two-dimensional knapsack problems. European Journal of Operational Research 83, 39–56.

Dasgupta, D., Michalewicz, Z., 1997. Evolutionary Algorithms in Engineering Applications. Springer, Berlin.

Dowsland, K.A., 1993. Some experiments with simulated annealing techniques for packing problems. European Journal of Operational Research 68, 389–399.

Dowsland, K.A., 1996. Genetic algorithms – a tool for OR? Journal of the Operational Research Society 47, 550–561.

Glover, F., Kelly, J.P., Laguna, M., 1995. Genetic algorithms and tabu search: Hybrids for optimization. Computers and Operations Research 22 (1), 111–134.

Goldberg, D.E., 1989. Genetic Algorithms. Addison-Wesley, Reading, MA.

Holland, J., 1975. Adaptation in Natural and Artificial Systems. Michigan Press, Michigan.

Hopper, E., Turton, B.C.H., 2000. An empirical investigation of meta-heuristic and heuristic algorithms for a 2D packing problem. European Journal of Operational Research 128, 34–57.

Ingber, L., 1992. Genetic algorithms and very fast simulated reannealing: A comparison. Mathematical and Computer Modelling 16 (11), 87–100.

Jakobs, S., 1996. On genetic algorithms for the packing of polygons. European Journal of Operational Research 88, 165–181.

Kirpatrick, S., Gelatt, C.D., Vecchi, P.M., 1983. Optimization by simulated annealing. Science 220, 671–680.

Koehler, G.J., 1997. New directions in genetic algorithm theory. Annals of Operations Research 75, 49–68.

Kuo, T., Hwang, S.Y., 1996. A genetic algorithm with disruptive selection. IEEE Transactions on Neural Networks 26 (2), 299–307.

Lai, K.K., Chan, W.M., 1997. Developing a simulated annealing algorithm for the cutting stock problem. Computers and Industrial Engineering 32 (1), 115–127.

Leung, T.W., Chan, C.K., Troutt, M., 2000. Comparison of meta-heuristics for the two dimensional orthogonal packing problem. Working Paper, Department of Applied Mathematics, The Hong Kong Polytechnic University, Hong Kong.

Lin, F.T., Kao, C.Y., Hsu, C.C., 1993. Applying the genetic approach to simulated annealing in solving some NP-hard problems. IEEE Transactions on Systems, Man, and Cybernetics 23 (6), 1752–1767.

Mahfoud, S.W., Goldberg, D.E., 1995. Parallel recombinative simulated annealing: A genetic algorithm. Parallel Computing 21, 1–28.

Mazumder, P., Rudnick, E.M., 1999. Genetic Algorithms for VLSI Design, Layout and Test Automation. Prentice-Hall, Englewood Cliffs, NJ.

Metropolis, N., Rosenbluth, A.N., Rosenbluth, M.N., Teller, A.H., Teller, H., 1953. Equations of state calculation by fast computing machines. The Journal of Chemical Physics 21, 1087–1091.

Osman, I.H., Laporte, G., 1996. Metaheuristics: A bibliography. Annals of Operations Research 63, 513–623.

Parada, V., Sepulveda, M., Solar, M., Gomes, A., 1998. Solution for constrained guillotine cutting problem by simulated annealing. Computers and Operations Research 25 (1), 37–47.

Rudolph, G., 1994. Convergence properties of canonical genetic algorithms. IEEE Transactions on Neural Networks 5 (1), 96–101.

Tsai, R.D., 1993. Three dimensional palletization of mixed box sizes. IIE Transactions 25, 64–75.

Thierens, D., Goldberg, D.E., 1994. Elitist recombination: An integrated selection recombination GA. In: Proceedings of the First IEEE Conference on Evolutionary Computation ICEC'94.

Wang, L., Zheng, D.Z., 2001. An effective hybrid optimization strategy for job–shop scheduling problems. Computers and Operations Research 28, 585–596.

Wolpert, D.H., Macready, W.G., 1997. No free lunch theorems for optimization. IEEE Transactions on Evolutionary Computation 1 (1), 67–82.