PERGAMON

Computers & Industrial Engineering 40 (2001) 201–214

computers & industrial engineering

www.elsevier.com/locate/dsw

Applications of genetic search and simulated annealing to the two-dimensional non-guillotine cutting stock problem

T.W. Leung[a,*], C.H. Yung[b], Marvin D. Troutt[c]

[a]*Department of Computing and Mathematics, Hong Kong Institute of Vocational Education (Chai Wan), 30, Shing Tai Road, Chai Wan, Hong Kong, People's Republic of China*
[b]*Lingnan University, People's Republic of China*
[c]*Graduate School of Management, Kent State University, USA*

Abstract

We applied a genetic algorithm and a simulated annealing approach to the two-dimensional non-guillotine cutting stock problem and carried out experimentation on several test cases. The performance and efficiency of these two heuristic algorithms on this problem were compared.

Keywords: Genetic algorithm; Simulated annealing; Bottom left algorithm; Difference process

1. Introduction

The two-dimensional non-guillotine cutting stock problem (CSP) 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. It is non-guillotine since the 'cuts' may not go from one end to another. We are interested to find 'cutting patterns' that minimize the unused area (trim loss). The problem has obvious relevancy to the paper, textile or 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 its history and other variations, see Beasley (1985), Christofides (1995) or Tsai (1993). Recently evolutionary algorithms have been applied to problems of this kind, for instance, see Glover, Kelly and Laguna (1995), Jakobs (1996), Lai and Chan (1997) and Parada, Sepulveda, Solar and Gomes (1998). Our work is based on the papers of Jakobs (1996) and Lai and Chan (1997).

In short, a cutting pattern may be represented by a permutation, which corresponds to the sequence in

* Corresponding author. Fax: +852-2595-8165.
E-mail address: twleung@vtc.edu.hk (T.W. Leung).

0360-8352/01/$ - see front matter
PII: S0360-8352(01)00021-3

which the small rectangles are packed. Now if we have a permutation, we know the order of packing the small pieces, but we may still consider different algorithms for packing the pieces. In Jakobs (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). And after a permutation was obtained, he used the bottom left (BL) algorithm to pack the pieces. In the paper of Lai and Chan (1997), they used simulated annealing to find permutations of small trim loss. After a permutation was obtained, they used the difference process algorithm (DP) to pack the rectangular pieces. We considered that perhaps it may be worthwhile to combine their work, thus we may compare the performances of different algorithms. Also by varying the parameters, we may find a good combination of procedures. We summarize our work in two steps below:

1. Use a genetic algorithm or simulated annealing to find permutations of small trim loss. The process is also called encoding, and a permutation thus found is a genotype.
2. Use different packing algorithms to pack the pieces corresponding to a particular permutation. The process is also called decoding, and the pattern thus formed is a phenotype. We then calculate the trim loss corresponding to that genotype, and go back to step one.

For the sake of simplicity, we have made 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. In other words, there is no rotation when placing the pieces into the plate.
3. The length and width of each piece does not exceed the corresponding dimensions of the stock plate.
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 edge of the plate. This limitation is not serious since in practice the actual dimensions can be scaled up to become integers by multiplying by a large enough 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, i.e. there is no restriction that two pieces cannot go together.

We shall first describe the decoding procedures and the evolutionary algorithms, then give a summary of our results.

2. The BL algorithm

We first put a rectangular piece at the right upper corner of the stock plate. We now try to move it down as far as possible, (i.e. as long as it is not blocked by another piece), then we move it as far as possible to the left, then we try to move it down, then to the left again, and so on, until we can go no further. The final position is where the rectangular piece will stay. In case a rectangular piece is not placed entirely into the stock plate, it will be discarded (unpacked).

For example, suppose we try to pack the following five rectangular pieces into a stock plate of width

170 and height 120, we put the BL corner of the stock plate at the origin. The dimensions of the pieces are as follows:

Piece	Width	Height
1	100	50
2	60	100
3	80	80
4	100	60
5	70	20

We try to pack the pieces using the permutation (genotype) (1, 2, 3, 4, 5) by the BL algorithm. First, piece one goes all the way down, then all the way to the left, thus its BL corner coincides with the origin (0, 0), and its top right corner is at (100, 50). Clearly, these two points specify the position of piece one. Now piece two may also go all the way down, but when it tries to turn left, it is eventually blocked by piece one, thus piece two is specified by the points with coordinates (100, 0) and (160, 100). When we try to pack piece three by the BL algorithm, we find that the top corners of the piece will go out of bounds; thus, we unpack it. Eventually we pack piece four, specified by the points with coordinates (0, 50) and (100, 110) and piece five, specified by the points with coordinates (100, 100) and (170, 120). See Fig. 1 for a pictorial representation.

The trim loss for packing the pieces by this procedure equals

$$\frac{\text{area of the stock plate} - \text{sum of areas of pieces } 1, 2, 4 \text{ and } 5}{\text{area of stock plate}} = 0.098.$$

Suppose we want to pack n pieces into a stock plate, and that $m-1$ pieces have already been packed. To pack the mth piece, we first want to move it down as far as possible; this is done by comparing the two coordinates of its BL corner with the $2(m-1)$ corners of the pieces already in the stock plate. After that, we want to go left. Since its two coordinates have been changed, we need to compare with the $2(m-1)$ pieces again. It is easy to see that the mth piece can move at most $2m$ times, (a move is a displacement of maximum length). Thus the number of comparisons made in packing the n pieces by the BL algorithm is of order $O(tn^2)$, where t is the average number of moves for each piece. Of course when a rectangular piece has moved, it is not necessary to compare the piece with those above it, nor with those on its right side. Thus, our estimate is approximate.

The BL algorithm has a limitation. For instance, no matter what permutations we employ, we cannot pack the following patterns (Fig. 2). Recently Liu and Teng (1999) advocated a new version of BL algorithm, trying to alleviate this situation.

3. The DP

In packing a new piece using the BL algorithm, the positions of the pieces already in the stock plate are considered. In the DP, while trying to pack a new piece, the empty 'spaces' are considered. In short, a space is a largest possible rectangular area not yet occupied by any rectangular pieces. At the very beginning there is only one space, its BL corner is at the origin. After the first piece is packed, there are (usually) two new spaces generated. We then calculate the distances (called the anchor distances) between the BL corners of the spaces and that of the origin, and we pack the second piece, if possible,

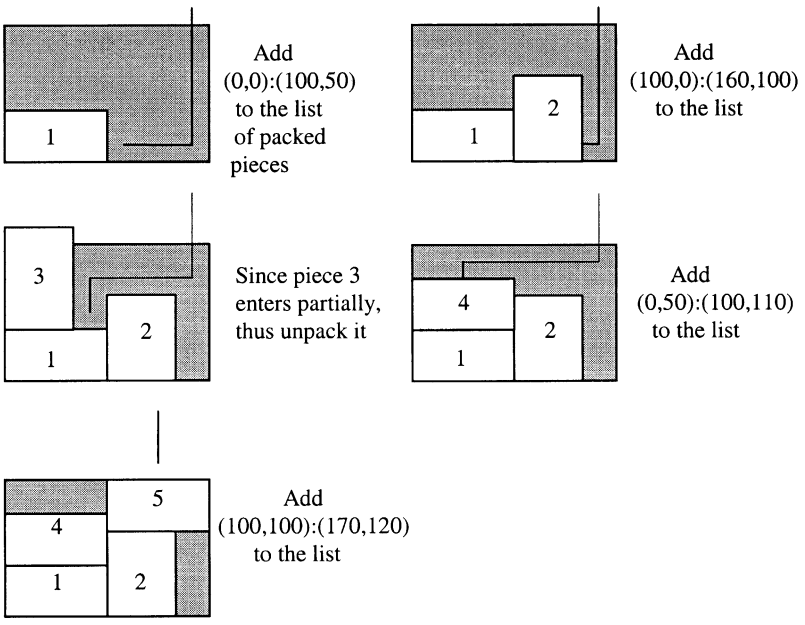


Fig. 1.

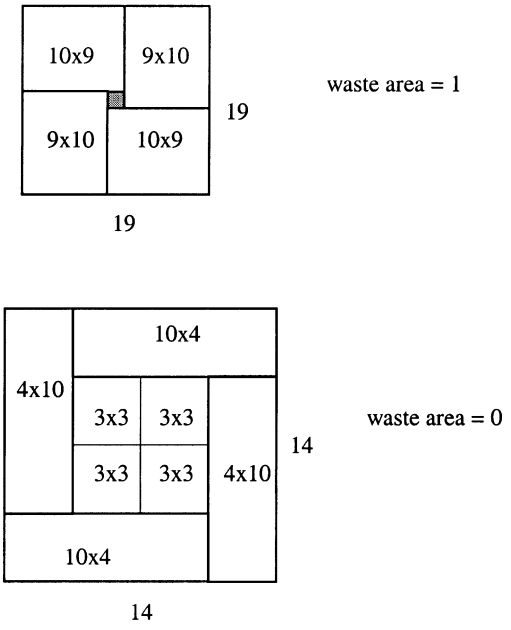


Fig. 2. Patterns that cannot be generated by BL algorithm.

in the space nearer to the origin, (to pack as compactly as possible). Every time a new piece is packed, new spaces are generated. We then keep track of a space list, calculate the anchor distances again, and try to pack another piece accordingly. The process continues until all pieces have been tried. We give an example.

The dimensions of the rectangular pieces we want to pack are given as follows:

Piece	Width	Height
1	200	100
2	100	50
3	100	100
4	110	60
5	100	120

The width of the stock plate is 300 and the height 200, with its BL corner situated at the origin. We apply the DP to the permutation (genotype) (1, 2, 3, 4, 5).

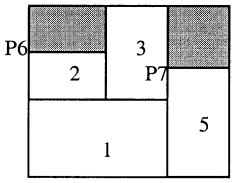
Initially there is only one space, denoted by (0, 0): (300, 200), corresponding to the coordinates of the BL corner and the top right corner of the stock plate. The anchor vector is (0, 0), of distance zero from the origin. Thus we pack the first piece by placing its BL corner at the origin. After packing the first piece, we get two new spaces, denoted by (200, 0): (300, 200) and (0, 100): (300, 200). The anchor vectors are (200, 0) and (0, 100), respectively. Since the second vector is closer to the origin, we pack piece two in the second space, with its BL corner at the point (0, 100). The outcomes of applying the algorithm to the genotype is summarized in the following table:

Space list	Anchor vector	$(\text{Distance})^2$	Decision
(0, 0): (300, 200)	(0, 0)	0[a]	Pack piece one at (0, 0)
(200, 0): (300, 200)	(200, 0)	40,000	
(0, 100): (300,200)	(0, 100)	10,000[a]	Pack piece two at (0, 100)
(200, 0): (300, 200)	(200, 0)	40,000	
(100, 100): (300, 200)	(100, 100)	20,000[a]	
(0, 150): (300, 200)	(0, 150)	22,500	Pack piece 3 at (100, 100)
(200, 0): (300, 200)	(200, 0)	40,000	
(0, 150): (100, 200)	(0, 150)	22,500	No space is large enough to hold piece four, unpack piece four
(200, 0): (300, 200)	(200, 0)	40,000	
(0, 150): (100, 200)	(0, 150)	22,500[a]	Pack piece five at (0, 150)
(0, 150): (100, 200)	(0, 150)	22,500	
(200, 120): (300, 200)	(200, 120)	54,400	

[a] Indicates the space nearest to the origin.

Hence we pack pieces 1, 2, 4, and 5 into the stock plate, with trim loss equal to the area of unused space over the total area, which is 0.2167. The final packing pattern is given in Fig. 3.

In the execution of the DP, we need to keep an updated space list, and calculate the anchor distances of the spaces. Every time we try to pack a new piece, we need to consider if the spaces can hold the piece, and try to pack it as compactly as possible. After packing a new piece, we need to count how many spaces the piece has overlapped, and generate a new space list. In general, it is more complicated to implement the DP than the BL algorithm. Moreover, because there exist many possible ways of how a



Space List	Anchor Vector	(Anchor Distance)^2
(0,150):(100,200)	P6	$0^2+150^2=22500$
(200,120):(300,200)	P7	$200^2+120^2=54400$

Trim Loss = 0.2167

Fig. 3.

piece may overlap with spaces, it is difficult to count the number of spaces. Nevertheless, because this process helps us to fill in 'holes', it usually produces better packing patterns.

As in the BL algorithm, there are packing patterns that the DP cannot generate (Fig. 4).

4. Simulated annealing

The simulated annealing approach for solving optimization problems has been around for a quite a while. It has been used to tackle many NP-hard problems such as the traveling salesman problem, VLSI layout, etc.; see Ansai and Hou (1997) Rayward-Smith, Osman, Reeves and Smith (1996) and Schwefel (1994). The main idea is that in the process of searching for good solutions, there is a mechanism that prevents us from getting stuck at poor local optima. Specifically, if we move from one solution to an inferior solution, with change in value Δc ($\Delta c > 0$) of a certain cost function, then the move to the inferior solution is still accepted if

$$\exp(-\Delta c/T) > R,$$

where T is a control parameter (usually called the temperature) and R is a uniform random number between zero and one. Initially, T is of a relatively high value; thus, there is larger chance to accept

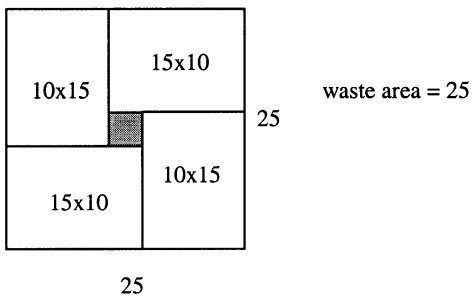


Fig. 4. Pattern that cannot be generated by DP.

inferior solutions. T is slowly reduced to values such that most inferior solutions will be rejected. For a fixed T, we first search for a new solution, accept it or reject it according to the criterion just described, until we reach equilibrium. This phase is called an inner loop. The control parameter T is then moved down to another value, and another phase of inner loop is carried out. The entire process in moving the initial temperature to the final temperature consists of a set of outer loops. There is a close analogy between this approach and the thermodynamic process of annealing (cooling of a solid). It was Metropolis, Rosenbluth, Rosenbluth, Teller and Teller (1953) who first proposed this idea, and after thirty years, Kirpatrick, Gelatt and Vecchi (1983) observed that this approach could be used to search for feasible solutions of an optimization problem, with the objective of converging to an optimal solution.

In applying the simulated annealing approach to the cutting stock problem, we take the following steps.

1. We first create an initial string, decode it using a suitable decoding algorithm, and calculate its trim loss. We also set the initial temperature.
2. If the maximum number of iterations has been exceeded, i.e. we have arrived at the final temperature, then we take the best solution and the process is stopped, otherwise we go to step three.
3. We move from the string at hand to a neighboring string using a suitable neighborhood move.
4. If the new string is a string with smaller trim loss, we accept it as a new solution, otherwise we accept it with a probability.
5. We check if the equilibrium has been reached (an inner loop has been completed). If so, we reduce the temperature using a suitable cooling schedule and move to another outer loop and go to step two. Otherwise we go to step three.

Of course steps three to five form an inner loop. An inner loop is contained in steps two to five, which correspond to an outer loop. We need to clarify a few points.

Initial and Final Temperatures. Because we measure the trim losses in percentages, the temperatures should be comparable to these figures. Thus, the initial temperatures are chosen from 10 to 80, and we set the final temperature to be one. In case the initial temperature is 80, and the new solution is 20% worse than the old one, then $\exp(-20/80) = 0.78$, and hence there is still a relatively large chance that the inferior solution be accepted. While at the final temperature ($= 1$), even if the new solution is only 1% worse off than the old one, because $\exp(-1/1) = 0.36$, there is not much chance that the inferior solution will be accepted.

Cooling Schedule and Maximum Iterations in the Outer Loop. For the sake of comparison, we also set the number of iterations M in the outer loop to be 2000. The performance of this algorithm also depends on the cooling schedule, which is essentially the temperature updating function. Two schedules were employed: (a) Proportional decrement scheme and (b) Lundy and Mees (1986) scheme. In the proportional decrement scheme, temperatures at the k and $k + 1$ steps of the outer loop, T_k and T_{k+1}, are related by

$$T_{k+1} = \alpha T_k \qquad \text{with } 0 < \alpha < 1.$$

While in the Lundy and Mees scheme, T_k and T_{k+1}, are related by

$$T_{k+1} = T_k/(1 + \beta T_k),$$

with $\beta > 0$ a suitably chosen parameter.

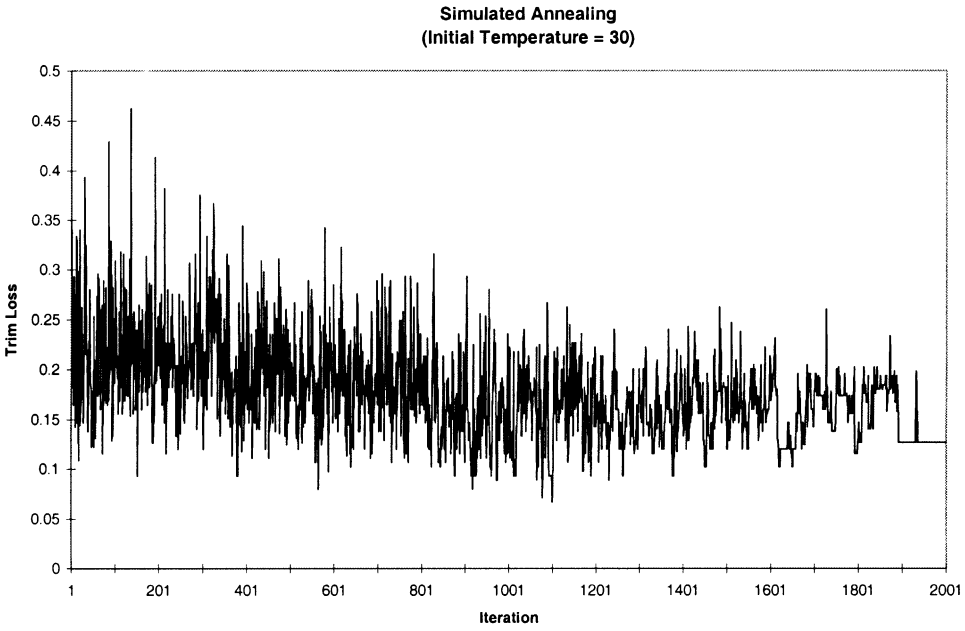


Fig. 5. Plot of trim losses versus iterations in simulated annealing.

Neighborhood move. Basically two neighborhood moves were employed. The first one is to swap two 'genes' randomly in a string, for example, swapping the first and the fourth element of (1, 2, 3, 4), we get (4, 2, 3, 1). The second one is to find a gene randomly and put it in front of another randomly chosen gene. For example, if we put the last gene in front of the second in (1, 2, 3, 4), we get (1, 4, 2, 3).

Equilibrium. There are a variety of ways to define if equilibrium has been achieved. One may set an upper bound for the possible number of neighborhood moves, one may decide that an equilibrium has been attained if there is no improvement in the foregoing five moves, or one may say if enough genuine moves have already been happened, and so on. In our case, we set an inner loop consisting of at most five neighborhood moves.

A typical graph of trim losses versus iterations is depicted in Fig. 5. The peaks indicate escapes from local optima.

5. Genetic algorithm

A genetic algorithm is a heuristic search process that resembles natural selection. There are many variations and refinements but basically, any genetic algorithm contains reproduction, crossovers 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). For its many applications and details, see Dasgupta and Michalewicz (1997), Goldberg (1989) and Mazumder and Rudnick (1999).

Using the terminology of genetics, a population is a set of feasible solutions of a problem. A member of the population is a genotype, a chromosome, a string or a permutation, in our case corresponding to

the order of packing rectangular pieces. When a genotype is decoded, a packing pattern, called a phenotype, is formed. We may calculate its fitness value (percentage of total area of the stock plate packed by pieces) and its trim loss. In applying a genetic algorithm to the cutting stock problem, we have the following steps.

1. First we set the mutation rate p_m and create an initial population randomly.
2. If the maximum number of 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. Take a random number r between zero and one. If $r \leq p_m$, produce an offspring from the first of the two chosen strings by a mutation process, update the population by replacing the worst string in the population by this offspring and go to step two. 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 and go to step two.

We need to clarify a few concepts.

Population. There is no clear indication as to how large should a population be. 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, a population consists of 10 to 100 members.

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 total fitness value of the population, we get the relative fitness value of each member. Two strings are then chosen as parents, with probabilities corresponding to their relative fitness values.

Mutation. If the entire population has only one type of string (or strings are very similar), then the crossover of two strings does not produce any new strings. 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.

Crossover. The crossover operation corresponds to the concept of mating. It is hoped that the crossover (mating) of good parents may produce good offspring. Thus, the crossover operation is a simple yet powerful way of exchanging information and creating new solutions. We employ five different crossover operations. These are the Stefan Jakobs crossover (SJX), partially matched crossover (PMX), order crossover (OX), cycle crossover (CX), and order based crossover (OBX).

Given a father string and a mother string, and two integers p and q, the SJX takes q genes from the pth gene of the father string, and fills in the remaining genes of the offspring by the other genes of the mother string. For example suppose the father is (1, 2, 3, 4, 5, 6, 7, 8, 9), the mother is (6, 4, 2, 5, 3, 1, 8, 7, 9), $p = 2$ and $q = 3$. Then 2, 3, 4 are the first three genes of the offspring, and the other unused genes 6, 5, 1, 8, 7, 9 come from the mother, thus the offspring is (2, 3, 4, 6, 5, 1, 8, 7, 9).

For the PMX operator, at first two cut points are chosen at random for the parents. Then the genes of the father string bounded by the cut points will be copied to the same positions of the offspring, and the remaining genes of the offspring will be filled up by the mother string in the same order. But then the offspring may not be legal because of possibilities of repeated genes. In that case, the repeated genes will be replaced by genes corresponding to the mapping of the father and mother spring bounded by the cut

points. For example, suppose the father is (2, 4, 5, 3, 8, 9, 6, 1, 7), the mother is (3, 9, 8, 6, 5, 4, 2, 7, 1) and the cut points lie after the third gene and before the seventh gene. Thus, the father genes inside the cut points are 3, 8, 9, the mother genes outside the cut points are 3, 9, 8, 2, 7, 1. The offspring is (3, 9, 8, 3, 8, 9, 2, 7, 1). This would be illegal because 3, 8, 9, are the repeated genes. Note that 3, 8, 9 of the father genes map to 6, 5, 4, thus we replace 3 by 6, 8, by 5, and 9 by 4, to get the new offspring as (6, 4, 5, 3, 8, 9, 2, 7, 1).

Similar to the PMX operator, the OX operator also starts with two cut points. The genes of the father string bounded by the cut points are copied to the offspring in the same positions. The remaining positions are filled (with respect to the same order) by the mother genes that have not yet appeared in the offspring. For example, suppose the father is (6, 4, 5, 9, 8, 3, 1, 2, 7), the mother is (3, 9, 4, 6, 8, 5, 2, 7, 1), the cut points lie after the third gene and before the seventh gene of the father string. We first copy 9, 8, 3 as the fourth to sixth gene of the offspring. Now the unused mother genes are 4, 6, 5, 2, 7, 1, thus finally the offspring is (4, 6, 5, 9, 8, 3, 2, 7, 1).

The CX operator performs in such a way that each gene of the offspring comes from the father or the mother. First from the mapping of the father to the mother we form a cycle, starting from the first gene of the father. Then these genes are copied to the offspring, with the remaining genes filled by the mother string. For example, let the father and mother be (2, 4, 5, 3, 8, 9, 6, 1, 7) and (3, 9, 8, 6, 5, 4, 2, 7, 1), respectively. Starting from 2, we go to 3, then from 3 (of the father) we go to 6, and from 6 (of the father) we go to 2, and we are back. Thus the cycle is 2, 3, 6. Hence, we copy genes 2, 3, 6 of the father to the offspring, and the other genes 9, 8, 5, 4, 7, 1 come from the mother. The offspring is therefore (2, 9, 8, 3, 5, 4, 6, 7, 1).

The OBX operates as follows. First a cut point is determined. The left hand portion of the father string will then be copied to the offspring, the other positions will then be filled by the unused genes of the mother (with respect to the same order). For example, suppose the father is (2, 4, 5, 3, 8, 9, 6, 1, 7), the mother is (3, 9, 8, 6, 5, 4, 2, 7, 1) and the cut point lies after the fourth gene. Then 2, 4, 5, 3 become the first four genes of the offspring. The unused genes of the mother are 9, 8, 6, 7, 1. Thus the offspring is (2, 4, 5, 3, 9, 8, 6, 7, 1).

A typical graph of average trim losses versus iterations is depicted in Fig. 6. Note that the offspring produced by mutation or crossover will always stay in the population for at least one more iteration, even if it is worse than the worst member already in the population. Thus, the graph still has ups and downs, though it is not as sharp as in the simulated annealing case.

6. Experimental results

We have altogether eight test problems. They all have the optimal solution of zero trim loss. Thus, we can estimate easily the performances of various algorithms. The number of rectangular pieces in each stock plate ranges from 10 to 30. For instance, test problem eight is given below (Fig. 7).

The algorithms are written in C and run in a 586 personal computer with 166 Hz. For each test problem both the simulated annealing procedure and the genetic algorithm were run 30 times and we took the averages of these results. For each run, the number of iterations (outer iterations for the simulating annealing procedures) was set to 2000. The parameters of the simulated annealing procedures and the genetic algorithms are listed below.

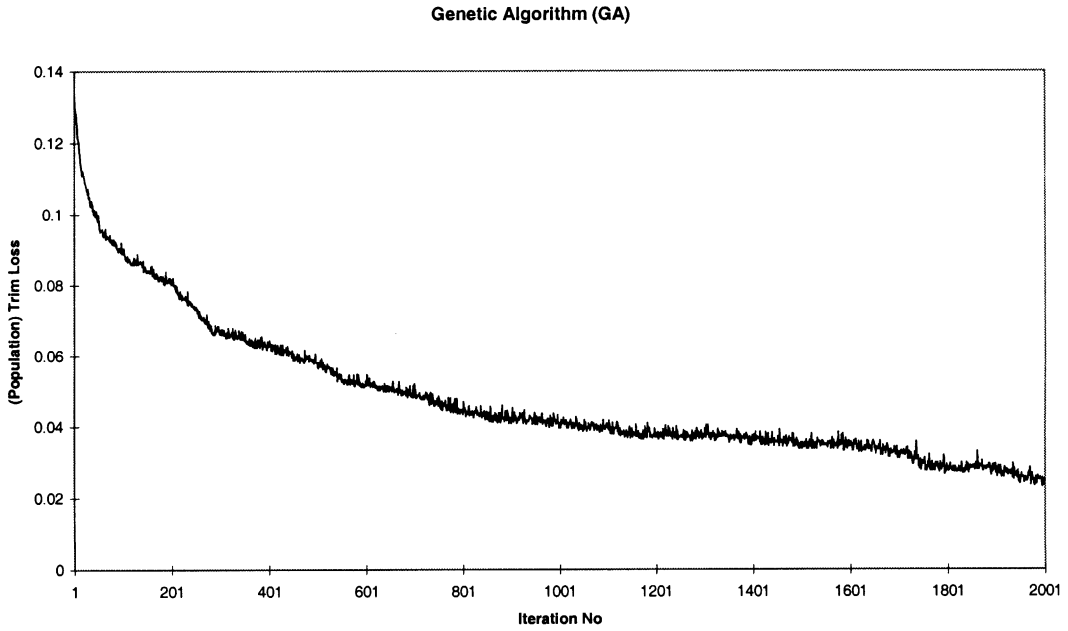


Fig. 6. Plot of trim losses versus Iterations in genetic algorithm.

6.1. Parameters for simulated annealing

Decoders: BL and DPs
Cooling Schemes: Proportional Decrement and Lundy and Mees
Neighborhood Moves: Swap and Shift
Initial Temperatures: 10, 20, 30, 40, 50, 69, 70, 80
Final Temperature: 1
Iterations in an Inner Loop: 5.

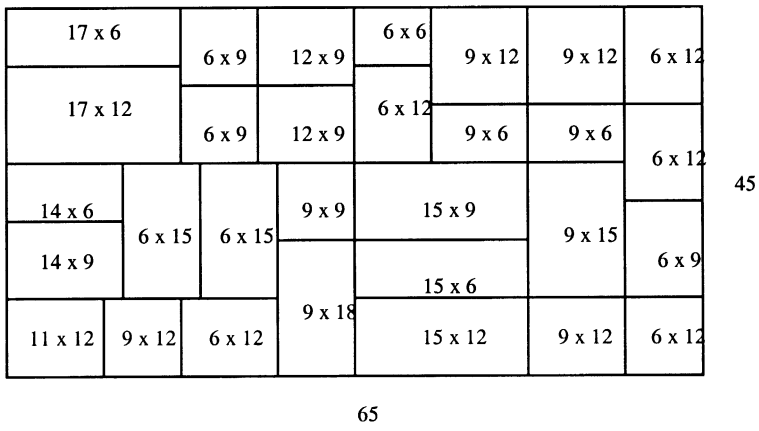


Fig. 7. Problem eight of the test problems.

6.2. Parameters for genetic algorithms

Decoders: BL and DPs
Crossover Operators: SJX, PMX, OX, CX, OBX
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.

Of course there are a lot of possible combinations for us to try. In order not to burden our readers with tables and tables of results, we shall only describe the most noteworthy observations.

Overall results. Despite the recent controversies over the uses of evolutionary algorithms (Koehler, 1997; Wolpert & Macready, 1997), we found that simulated annealing and genetic algorithms produce reasonable results for this problem. Of all the test problems we considered, the trim losses went from 0% (easier problems with 10 to 15 rectangular pieces) to 8% (more difficult problems with 25 to 30 rectangular pieces), which are within acceptable standards. The CPU times went from 6 s, (for easier problems using genetic algorithms) to 200 s, (for more difficult problems using simulated annealing). The CPU times using simulated annealing are comparable to those in the literature. But more importantly, in terms of trim losses and CPU times, we found that the results are quite robust, adding reliability to these kinds of algorithms.

Genetic algorithms versus simulated annealing. A large population size means the simultaneous handling of many solutions and increases the computation time per iteration. There is also concern of maintaining a large set of data. For instance, see Carter and Park (1994), Davis (1991), Pham and Karaboga (2000) and Reeves (1995). Nevertheless, in our problem, because the data structure is relatively simple, the size of populations does not pose a serious problem. Our results indicate that genetic algorithms outperform simulated annealing. We take test problem five as an example (we have similar findings in other test problems). Using simulated annealing and difference process decoding, with different neighborhood moves, cooling schemes and initial temperatures, the trim losses range from 7 to 10%. Using genetic algorithms, difference process decoding, mutation rate = 0.5, with various crossover operators and population sizes, the trim losses range from 4.5 to 5.5%. There is not much difference if different mutation rates are employed. Moreover, the CPU times of simulated annealing range from 190 to 205 s, while the CPU times of genetic algorithms range from 35 to 45 s. The difference in times is mainly due to the fact that in simulated annealing, there is an inner loop to run. It is more interesting to know why genetic algorithms outperform simulated annealing in terms of trim losses. We speculate that if the initial population is well chosen, crossovers and mutations will ensure that best strings will be found rather fast. While in the case of simulated annealing, the neighborhood move may in fact be regarded as a special case of mutation, and since we start from one string, it is less likely that the best string can be quickly found.

BL algorithm versus DP. We found that the trim losses are smaller for the DP than for the BL decoding procedure. This is likely due to the fact that rectangular pieces are packed as compactly as possible and 'holes' or unused spaces may only be filled by applying the difference process approach. For example, in test problem 3, using simulated annealing with initial temperature = 10, with various neighborhood moves and cooling schemes, we found that by using the difference process decoding, the trim losses were close to 4%, while by using the BL decoding, the trim losses were close to 6%. However, there is also a catch. It is more difficult to implement the difference process in the computer

and it is more time consuming to check and update the space list; hence more computer time is needed. In fact, typically the computation time is more than double. Take again test problem three for example. Using simulated annealing with BL decoding, the computation times center around 53 s; while with difference process decoding, the computation times center around 130 s. In using genetic algorithms with BL decoding, the computation times were around 13 s, in contrast to use of the difference process, with times centering around 27 s.

Particulars for simulated annealing. We shall describe how the changes of parameters affect the performance of simulated annealing. First, in general, we found that the proportional decrement scheme performs better than that of Lundy and Mees, and the swapping neighborhood move works better than the shifting move. The best combination is the proportional decrement scheme with the swapping neighborhood move, and the worst is Lundy and Mees with the shifting move. For example, in test problem five with difference process decoding and various initial temperatures, the trim losses of the former cluster near 6.8%, while in the latter, 7.8%. Perhaps in the proportional decrement scheme, the rate of decrease in temperatures is not as steep as in the Lundy and Mees scheme, thus allows more time to jump out of local optima. Concerning the swapping move, a string has equal chance of going to any of its neighbors; while if we move a gene in front of another gene, the chances of a string going to its neighbors are not equal. This may explain why the swapping move usually works better.

We did not find any clear suggestion on the choice of initial temperature. Of course, the initial temperature should not be too high, otherwise at the early stages the process amounts to nothing but random search. But, it should not be too low also; otherwise it would be difficult to leave local optima. We believe the best initial temperatures should be near 50.

Particulars for genetic algorithms. There is no clear evidence that larger populations will produce better offspring. We tried populations of sizes from 20 to 80, and found that the best results occurred when the population sizes are around 40–60. Apparently, if a population is well chosen then good offspring will be produced rather fast. Concerning crossover operators, we found the SJX and PMX methods work slightly better, probably because these operators will preserve more characteristics of the parents.

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 cases the best results occurred when the mutation rates are from 0.7 to 1.0.

References

Ansai, N., & Hou, E. (1997). *Computational intelligence for optimization*, Dordrecht: Kluwer Academic Press.

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

Carter, B., & Park, K. (1994). On the effectiveness of genetic search in combinatorial optimization. *Computer Science Department, Boston University*, 94–100.

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*, Berlin: Springer.

Davis, L. (1991). *Handbook of genetic algorithms*, New York: Van Nostrand Reinhold.

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*, Reading, MA: Addison-Wesley.

Holland, J. (1975). *Adaptation in natural and artificial systems*, Michigan Press.

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, M. P. (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.
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.
Liu, D., & Teng, H. (1999). An improved BL-algorithm for genetic algorithm of the orthogonal packing of rectangles. *European Journal of Operational Research*, *112*, 413–420.
Lundy, M., & Mees, A. (1986). Convergence of an annealing algorithm. *Mathematical Programming*, *34*, 111–124.
Mazumder, P., & Rudnick, E. M. (1999). *Genetic algorithms for VLSI design, layout and test automation*, Englewood Cliffs, NJ: Prentice-Hall.
Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. W., Teller, A. H., & Teller, E. (1953). Equations of state calculation by fast computing machines. *Journal of Chemical Physics*, *21*, 1087–1092.
Parada, V., Sepulveda, M., Solar, M., & Gomes, A. (1998). Solution for the constrained guillotine cutting problem by simulated annealing. *Computers and Operations Research*, *25* (1), 37–47.
Pham, D. T., & Karaboga, D. (2000). *Intelligent optimization techniques*, New York: Springer.
Rayward-Smith, V. J., Osman, I. H., Reeves, C. R., & Smith, G. D. (1996). *Modern heuristic search methods*, New York: Wiley.
Reeves, C. R. (1995). *Modern heuristic techniques for combinatorial problems*, London: McGraw-Hill.
Schwefel, H. P. (1994). *Evolution and optimum seeking*, New York: Wiley.
Tsai, R. D. (1993). Three dimensional palletization of mixed box sizes. *IIE Transactions*, *25*, 64–75.
Wolpert, D. H., & Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE Transaction on Evolutionary Computation*, *1* (1), 67–82.