Resolution of Strip-Packing Problems with Genetic Algorithms

Article *in* Journal of the Operational Research Society · November 2000

DOI: 10.1057/palgrave.jors.2601019 · Source: OAI

Alberto Gomez
University of Oviedo
94 PUBLICATIONS **815** CITATIONS

Resolution of Strip-packing problems with genetic algorithms.

A. Gómez and D. de la Fuente

University of Oviedo, Spain

This paper studies Strip-Packing problems. Its aim is to optimise the position of a number of rectangular shapes on a base surface in order to minimise wastage of material. As the problem is a complex NP-Complete one, a heuristic based on genetic algorithms (GA) is used to solve it. The main problem is the wide variety of genetic algorithms available in the literature, which makes it hard to know which variation is best suited to this type of problem. We conclude that using a cyclic crossover GA with fitness by area and variable mutation works best for this problem.

Key-words: strip-packing; genetic algorithms; heuristics; combinatorial; optimisation

Correspondence: Alberto Gómez. E.T.S. Ingenieros Industriales. Campus de Viesques. 33204 Gijón, Asturias, Spain.

Phone: +34.985.18.21.06 Fax: +34.985.18.20.10

e-mail: agomez@etsiig.uniovi.es

Introduction.

This paper confronts a real industrial problem — that of positioning a number of rectangular shapes on a base surface so as to optimize the use of the material the base surface is made of. Confronting the problem arose from the difficulties that a company based in Asturias (Spain) faces in its attempt to optimize the use of its raw materials, and the solution involves supplying a computer program with the position of the different shapes. The program is then responsible for transferring the information to the cutting machine. It should be noted that all the pieces are rectangular.

Cutting problems like the one described above are common to many industries. The textile, leather, shipbuilding and metal industries are examples of industries where obtaining pieces from a base surface is a common activity. Since the 60s, many researchers have tackled the problem, aiming to develop algorithms to resolve it.

The raw materials that industry uses are commonly available in certain standard sizes. These sheets usually have to be cut to the required size before they can be used in the production process they are a necessary part of. The obvious aim of this phase of the process is to make optimum use of the raw material. Gilmore and Gomory[1,2] and their work to resolve these problems in a single dimension stand out amongst early research in the field; they widened the scope of their focus in 1965 to two-dimensional problems[3]. Analysis of this kind of problem has since then spread rapidly. However, no global solution to all problems of this type exists owing to their extreme complexity.

One way of finding a solution is to divide the problem into several sub-problems[4], and then attempt to solve each one separately. This is precisely what we propose to do in this paper, focussing on just one of these problems – Strip-Packing, which adapts itself perfectly to the characteristics of the industrial processes we have to solve problems for. Our aim may be defined as follows:

One has a strip of W width and infinite height and a list of rectangles $L_n=(R_1,R_2,...,R_n)$ with $n\geq1$. Each of them has a length not exceeding W. The aim is to place the rectangles on the strip keeping the height the rectangles reach to a minimum. Figure 1 shows an example:

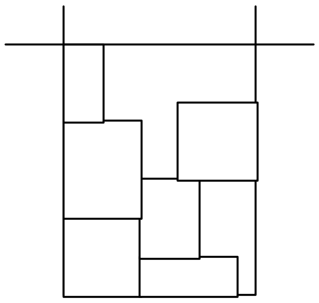


Figure 1: S*trip-packing example*.

When analysing these problems, certain limitations must apply:

- Rectangles cannot overlap with each other or with the end of the strip.
- The rectangles are packed with their sides parallel to the sides of the strip (90º rotation is not allowed)
- The height the packing reaches should be minimised.

Several authors have dealt with solutions to this problem, though none have come up with a method that provides the optimum solution in every case. The heuristic methods designed by Coffman[5] and Jakobs[6] might be mentioned from amongst approaches that have been put forward.

Genetic algorithms.

Genetic Algorithms (abbreviated to GAs) were developed towards the end of the 60s through the work of Holland[7], and were intended to solve the numerous problems that certain sectors of industry were facing that were difficult to solve with the methods available at that time.

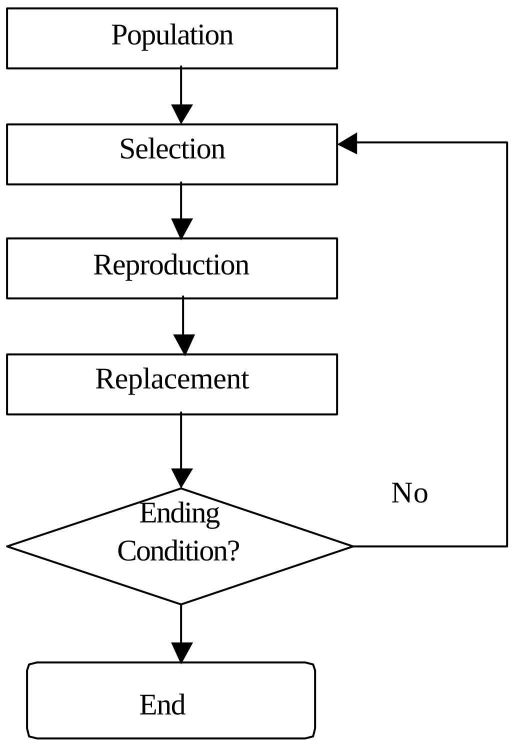


Figure 2: a GA loop

GAs are intended to resolve search and optimisation problems by reaching a solution which, though not the best one, is at least an approximation to the best one. This is why GAs are said to provide quasi-optimal solutions. GAs work by following a loop basically similar to the one in figure 2. They start with an initial population of solutions (usually randomly generated), and the loop is successively applied, so that each loop constitutes a GA generation. The loop consists of three basic steps – Selection, Reproduction and Replacement[8,9].

The *selection* step consists of sampling an initial population so that another population, called the auxiliary, is generated, containing the same number of individuals as the initial population. Improving the population is the aim of this step, that is, the best individuals – those that have a higher aptitude function value – are favoured when generating the auxiliary generation.

So-called Genetic operators, the most common of which are crossover and mutation, are applied during the *reproduction* step. Generally speaking, the crossover operator works by taking two breeders and interchanging part of their strings; this means that by crossing their strings two new individuals – the offspring – are created. The mutation generator, for its part, is applied to a breeder and alters its string, for example by changing some of its genes. A new offspring is generated in this way.

When reproduction has finished there are two independent populations – a breeder population, and an offspring population. The *replacement* process consists of forming a new population as a result of mixing the two initial ones. The most usual ending condition for GAs is to set a maximum number of generations; the GA stops when this number is reached.

Parallel Genetic Algorithms.

The first attempt to incorporate genetic algorithms into parallel computer architectures was probably made in 1981 by John Greffenstette[10]. There have been many subsequent applications of parallel genetic algorithms (PGAs), some of which are the following:

- Global Parallelization. This model has a single population and uses various computers for calculating fitness. Part of the total population is worked on at each individual computer using this methodology. Genetic operators are also used in parallel on occasions.

- Fine-Grain GA: an individual is assigned to each computer and is processed in parallel with the others. Each individual is part of a system of multiple sub-populations, the relationships of which are determined by the topology of the network the PGA is applied to.
- Coarse Grain GA: this is similar to the above one, the difference being that each computer works with a sub-population instead of with a single individual.

The main differentiating feature of PGAs is the method of migration they employ, that is, the conditions that have to be fulfilled for interchanges between sub-populations to occur. There are several alternatives:

The Island system. There is no migration between sub-populations. Each of them evolves independently. This system has been shown to have no major advantages over non-parallel systems.

Synchronous systems. The PGA is made up of a series of sub-populations which intercharge individuals. Two parameters guide this interchange: the migration rate and the migration interval.

Asynchronous systems. As most computers where PGAs are implemented have different work loads, it is impossible to synchronise communications between sub-populations, so this occurs at different instances in time.

Solving the Problem.

The solution to the problem that we addressed will now be dealt with. The GA implementation applied by Jakobs[6] in 1996 will be used as a starting point and an attempt to improve Jakobs contribution will be made.

The example shown in figure 3 was proposed by Jakobs[6] in his work. He begins with a surface of 40*15 units, which he divides randomly into 25 rectangles. Using the GA he proposes, he then attempts to place the rectangles on a surface measuring 40 * height.

The meaning of this height should be clear if the work is to be understood. As previously stated, the aim of the operation is to place all the pieces on as small surface as possible, and in this particular case, height is the only value that can be modified, as width is fixed (40 units in the example). The lowest height that Jakobs achieves is 17 units, and our aim in this study is to improve on this result.

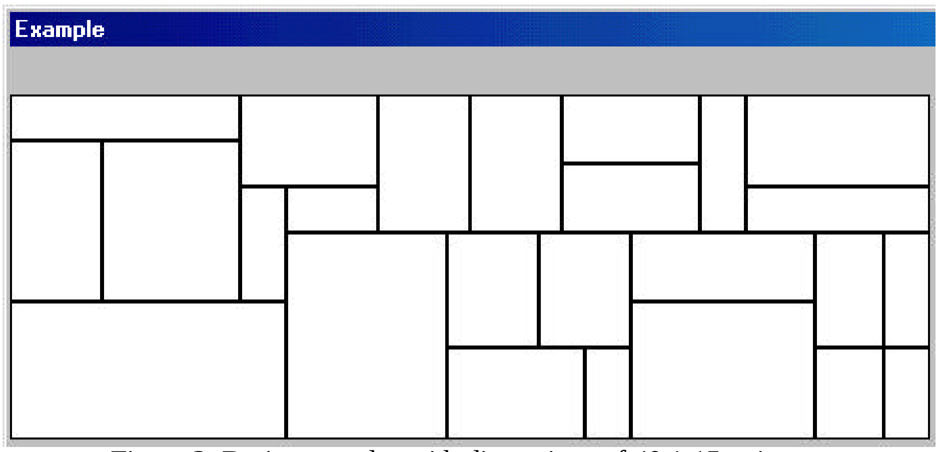


Figure 3: Basic example, with dimensions of 40 * 15 units.

The different variants applied to improve the solution will next be dealt with. Note that a simple GA[8] has been used instead of the steady-state one implemented by Jakobs. The latter is a little faster, but the average quality of the solutions obtained is a little inferior to classic GA-generated solutions.

Codification.

The structure of the data selected for search space codification of an optimisation problem is an important element of Genetic Algorithms. GAs usually work with binary strings, which in some way represent one of the potential solutions to the problem. These strings are generally not the most appropriate solutions to the packing problem.

An adequate way to represent the search space of the problem is using the "Π" permutation,

$$\Pi = (i_1 , \dots , i_n)$$

Where the i_j are the numbers assigned to the rectangles.

This permutation represents the sequence in which the rectangles are packed into the container. Moreover, it is easy with this data structure to create new permutations by changing the sequence, and the new permutation is, moreover, a feasible individual.

To give a clearer idea of the codification, let us consider that four rectangles of given dimensions are to be packed into a container. These rectangles are numbered from 1 to 4. An individual (genotype) that may be the solution to the problem will be given by the following permutation: $\Pi = (1,3,4,2)$, which means that rectangle 1 is the first to go into the container, followed by 3, by 4, and finally by 2.

Once the sequence of the rectangles going into the container is known, the steps to be followed to put a generic rectangle into the container, bearing in mind this algorithm, will be as follows:

1) The rectangle occupying the first place in the permutation is placed in the lower left-hand corner of the container.

2) The remaining rectangles are taken in the order given by the permutation. Starting in the top right hand corner of the container, they are lowered as much as possible until they meet another rectangle and cannot be lowered any further. They are then moved as far to the left as possible, until some limit stops their progress. The rectangles are placed in this way until none remain to be placed (figure 4).

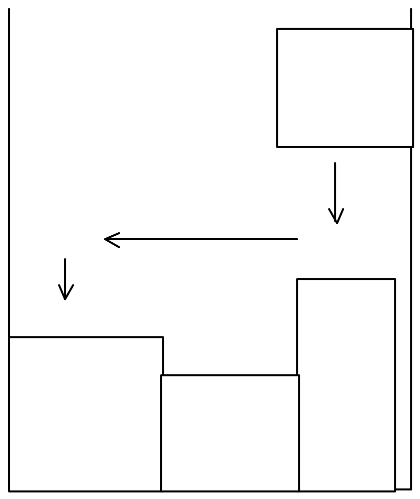


Figure 4: Packing sequence

The maximum number of packing models that can be obtained with n rectangles is n!. There are n! rectangle sequences. In practice, this maximum number of possible models is never reached because some permutations coincide with the packing model itself.

Initial Population.

One of the first tests that was carried out was to verify the importance of initiating the population randomly or not. Initiation was done using two criteria. In the first the population is initiated randomly, and in the second the initial population is generated by ordering the rectangles from the widest to the narrowest.

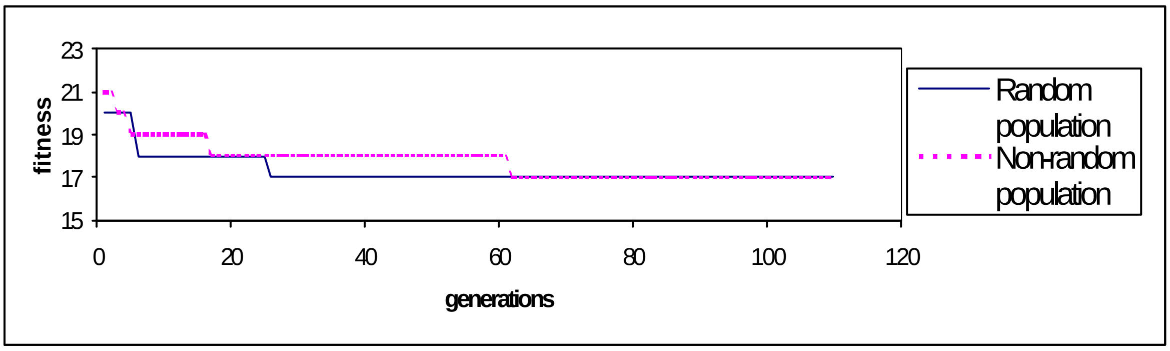


Figure 5: Evolution of fitness with random and non-random population.

Figure 5 shows the evolution of best fitness of the population when generated both randomly and non-randomly. The results of the different experiments carried out showed that the importance of initial position ordering is minimal for problems involving few rectangles. In this respect, the only point to be noted is the obvious one that the system finds an acceptable solution earlier if the position of some of the rectangles is prepared. However, after 500 generations the quality of the solutions using both methods is the same.

Crossover variants.

Coinciding with work published on resolving the travelling salesman problem using GAs with decimal codification, we decided to implement different types of crossover; their results in the combinatory field are widely recognised. Four types of crossover were, in fact, implemented (see Appendix for results).

- One-point crossover: one crossover point is selected and there is information exchange from the breeders.

- PMX crossover (partial matching crossover, proposed by Goldberg and Lingle [11]). Given two parent chromosomes, the operator copies a sub-string of one of the parents directly in the same positions for the offspring. The remaining positions are filled in with the values that have still not been used in the same order as they are found in one of the parents.

  For example, if we have two chains, p1 (1,2,4,6,3,7,5,8) and p2 (5,4,1,7,2,6,8,3), and if the randomly selected p1 sub-chain to be inserted in p2 is (4,6,3), this establishes a relation with the sub-chain (1,7,2) that has the same position in p2. So the sequence of operations would transform p" into (5,4,4,6,3,6,8,3); repetitions are eliminated, leaving (5,*,4,6,3,*,8,*); replacement leaves (5,1,4,6,3,*,8,*); as 4 occupied the position of 1 in the sub-chain. By continuing we obtain (5,1,4,6,3,7,8,2).
- OX crossover (order crossover, proposed by Davis[12]). Crossover by basic order consists of choosing a section of one of the parents for each descendent whilst simultaneously maintaining the relative order of all the rectangles of the other. In the above example, it is assumed that the same sub-chain as before is chosen, so the sequence becomes (*,*,4,6,3,*,*,*) and (*,*,1,7,2,*,*,*). Next, for each parent one begins from one of the cross points and the rectangles of each parent are copied, maintaining the relative order and omitting those already present. Once at the end of the chain, you begin at the beginning of the chain till the starting point is reached: Our example ends up as (7,2,4,6,3,8,5,1) and (6,3,1,7,2,5,8,4).
- CX crossover (cycle crossover, proposed by Oliver[13]). Each rectangle successively inherits the position of one of its parents. How this works will be explained with an example. Imagine the following parents: p1 (1,2,3,4,5,6,7,8,9) and p2 (4,5,2,1,8,7,6,9,3). It is extremely helpful for the explanation to construct a table of positions, and in this particular case this would be as follows:

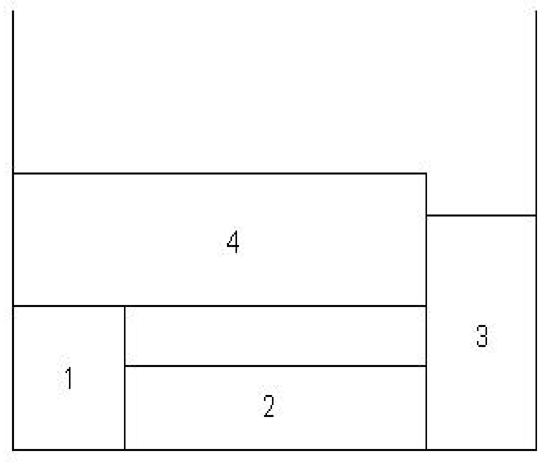
	1 2 3 4 5 6 7 8 9
v	1º 2º 3º 4º 5º 6º 7º 8º 9º
w	4º 3º 9º 1º 2º 7º 6º 5º 8º

It operates by completing the so-called “succession cycles”. The operator consists of four stages

1. For the first descendent you start from the first rectangle of the first parent (1,x,x,x,x,x,x,x,x); this means giving the fourth rectangle the fourth position of this descendent (1,x,x,4,x,x,x,x,x) and the first succession cycle thus finishes.
2. The second rectangle is randomly chosen from one of the parents, for example, the first parent again. This means giving rectangle 5 the fifth position, which in turn means giving position 8 to the eighth, 9 to the ninth, and third to the third, which completes the second succession cycle and yields p1(1,2,3,4,5,x,x,8,9)
3. The final cycle is as follows: you now start from the sixth rectangle of the second parent to obtain the first descendent (1,2,3,4,5,7,6,8,9)
4. The second descendent is obtained by complimentariness with the first (4,5,2,1,8,6,7,9,3)

Type of fitness.

It was decided that the height that the rectangles reach within the container would be the basis for calculating fitness. However, as different combinations can have the same fitness but the quality of these combinations may not necessarily be the same, different variants had to be implemented to attempt to improve the quality of the information provided by the fitness. Figure 6 and 7, show two ways of placing 4 rectangles on a surface. Although the height of the solution is identical in both cases, this is not true as regards the quality of the solution, since the solution shown in figure 7 makes better use of the material. In order to try to reflect this in the fitness, the following improvements are proposed:



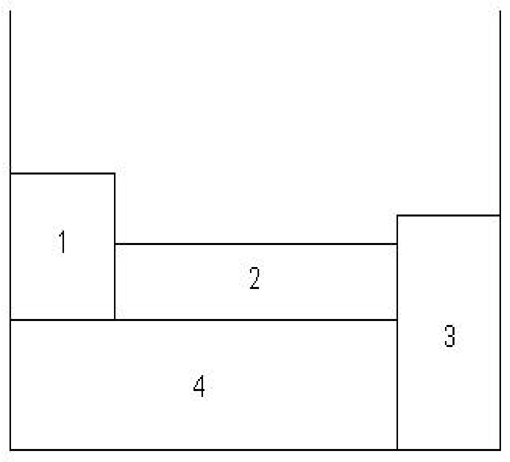


Figure 6: Placing of 4 pieces (note the space between the 4 rectangles).

Figure 7: Placing of 4 pieces

- A consideration of the total area occupied by the rectangles. This consisted of adding a value to the height that the rectangles reached. This value was obtained by multiplying the area covered by the rectangles by a weight (in the case in question 0.01).

- A reduction in the scale of the fitness interval, so that the optimum fitness is 1, thereby increasing the difference between the different solutions and obliging the algorithm to search for a better one.

- Setting fitness in the interval (1,100) so that poor solutions have the opportunity of remaining within the population, thereby increasing the population diversity.

Mutations.

Two types of mutation were implemented:

Order-based mutation: Based on the studies of Davis[14], this consists of permuting two rectangles of the same individual.

Variable mutation: a mutation exactly the same as the above-mentioned one is used, except that the mutation probability is modified as generations go by. Initially a very high mutation probability is used, so as to be able to verify the broadest possible range, and this probability is gradually reduced.

Results.

Some of the tests that were carried out along with the main results are next described. The research focused on the use of GA and PGAs for strip-packing. Several types of GA were used in order to find the best combination of crossover, mutation and fitness for this kind of problem. The height that the set of rectangles reached is used to gauge the quality of the GA, bearing in mind that the aim is to reduce, indeed minimise, the height.

A PC type computer with a Pentium 200MHz micro-processor was used to carry out the different experiments that are next described. Throughout this work tests were done with different types of randomly generated problems. The aim of these problems was to place on the base surface a number of pieces varying between 20 and 100 depending on the experiment. In the following summary of results that were obtained, Jakob's problem will be taken as the benchmark, and conclusions about the work will be commented upon in relation to this. It should be pointed out that the results observed in the other experiments are similar to those observed in this problem (see the Appendix). A final point to be made is that the computing time for 2000 generations was about 20 seconds.

Our first objective was to analyse whether it was ideal to use a variable mutation for this kind of problem. The experiments carried out confirmed that it was better to use a variable mutation, as this meant that in the initial generations the GA could widen its search area and in the last generations it could carry out a local search.

Once the type of mutation had been established, efforts were directed towards deciding the type of fitness to be used. It was deduced from the tests that the best option is to combine a fitness that takes height and the area used into account; the main problem with this method is that the weight of the area to be used is critical, and is strongly dependent on the number of rectangles to be placed on the surface. In the example used as a basis for this work, it was proved that the best results are achieved with a weight of 0.01.

As far as crossover is concerned, results are to some extent surprising. As all the types of crossover applied produced similar results it is difficult to decide which of them best adapts itself to the characteristics of the problem. Even one-point crossover gives good results.

The last of the experiments carried out focused on investigating whether PGAs could improve on the results. Values used by Belding[15] in his experiments were followed to set the parameters of the PGA. The population is made up of 24 sub-populations of 20 individuals each; the migration rate was 20% and the migration interval was 5. The results obtained are similar to those obtained with the simple GAs.

Finally, it is fitting to compare results obtained with those obtained by Jakobs. For the best of the GA structures, the algorithm succeeded in placing the pieces on a 40*16 surface 100% of the time, whereas in his work Jakobs only managed to reduce the base surface to a 40*17 rectangle.

As regards the question of the number of generations needed to achieve a satisfactory result, it was observed in the experiments that most times a height of 17 is achieved in the first 40 generations; in order to reach 16, more generations - generally over 500 - are required. The number of generations was limited in the experiments to 2000. Figure 8 shows one of the solutions obtained with a maximum height of 16 by way of example.

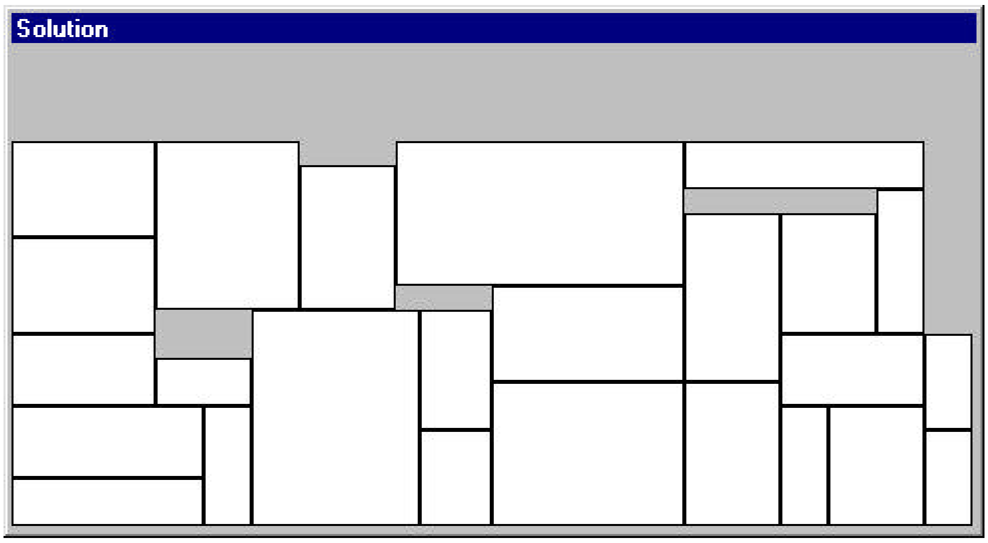


Figure 8: A solution at height 16

Conclusions

In this article we have focused our research on the search for the kind of genetic algorithm (GA) that best suits the strip-packing problem. From this research we conclude that using a cycle crossover type (CX), with fitness by area and variable mutation is the best GA to apply. Moreover, we have shown that parallel GAs do not make any significant contribution to solving the problems analysed.

Future lines of research could involve analysing new positioning of the pieces on the surface and generalising the problem by attempting to position not only rectangular pieces but also other *shapes* on the base surface.

Appendix: Results of the experiments.

Table 1 shows some of the experiments carried out, as well as results obtained with different types of combination of crossover, fitness and mutation. 20 experiments were done with each of these combinations, and the height that the best solution achieved is shown in the table.

The results of the 20 tests are given in the fourth and fifth column. The fourth shows the number of times the genetic algorithm succeeded in placing all the pieces on a 40*16 surface and the fifth column shows the number of times a 40*17 surface was used. The experiments have been carried out on the problem proposed by Jakobs to pack 25 pieces within a surface area of 40*15 (see figure 3).

The parameters used during the experiment were Pm = 0.3, Pc = 0.7, number of generations = 2000, population size = 100.

Crossover type	**Fitness type**	**Mutation**	**Results**	
			16	17
Normal crossover	Normal fitness	Order-based	18	2
		Variable	20	0
	Area fitness	Order-based	19	1
		Variable	20	0
PMX	Normal fitness	Order-based	17	3
		Variable	18	2
	Area fitness	Order-based	20	0
		Variable	20	0
OX	Normal fitness	Order-based	15	5

		Variable	17	3
	Area fitness	Order-based	19	1
		Variable	20	0
CX	Normal fitness	Order-based	19	1
		Variable	20	0
	Area fitness	Order-based	20	0
		Variable	20	0
Jakobs	Normal fitness	Order-based	17	3
		Variable	18	2
	Area fitness	Order-based	20	0
		Variable	20	0

Table 1: Heights reached in different tests using 25 pieces.

In table 2, the results obtained when attempting to place 50 rectangles, instead of 25 pieces, within the base surface area are shown. We used the problem with 50 pieces as proposed by Jakobs in his experiments. The structure of this table is identical to the previous one.

Crossover type	**Fitness type**	**Mutation**	**Results**	
			16	17
Normal crossover	Normal fitness	Order-based	1	19
		Variable	1	19
	Area fitness	Order-based	1	19
		Variable	1	19
PMX	Normal fitness	Order-based	0	20
		Variable	2	18
	Area fitness	Order-based	2	18
		Variable	3	17
OX	Normal fitness	Order-based	0	20
		Variable	0	20
	Area fitness	Order-based	0	20
		Variable	0	20
CX	Normal fitness	Order-based	1	19
		Variable	2	18
	Area fitness	Order-based	1	19
		Variable	4	16
Jakobs	Normal fitness	Order-based	0	20
		Variable	0	20
	Area fitness	Order-based	1	19
		Variable	2	18

Table 2: Height reached in different tests using 50 pieces.

To further emphasise the results obtained in this work, we show, in graphic form, (figure 9), the results achieved with 20 other problems. The number of pieces used in these tests vary between 20 and 100. It should be pointed out that "type 1" indicates the test carried out using normal fitness, and an order-based mutation. "Type 2" refers to the test carried out using normal fitness and variable mutation. In "type 3", we have used fitness by area, and Order-based mutation. Lastly, "Type 4" results are with fitness by area and variable mutation.

Figure 9 shows the number of experiments in which the best solution has been achieved with a pre-established codification. For example, the first rectangle indicates that with normal crossover, normal fitness, and an Order-based mutation, of the 20 problem, only in 5 have we reached the best solutions at least twice. Finally, it can be seen in this figure that the best solutions were obtained using a CX crossover with fitness by area and variable mutation.

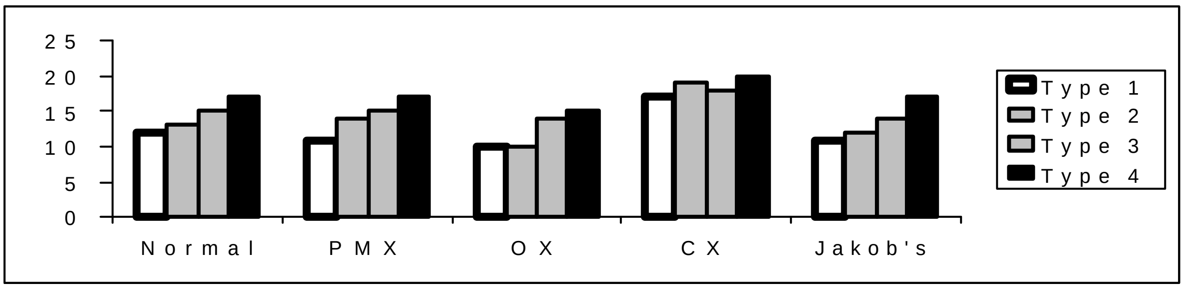


Figura 9: Results of tests with various types of problems.

References.

1. Gilmore PC. and Gomory RE (1961). A linear programming approach to the cutting-stock problem. *Operations Research* **9**: 724-746.
2. Gilmore PC. and Gomory RE (1963). A linear programming approach to the cutting stock problem. *Operations Research* **11**: 863-888.
3. Gilmore PC. and Gomory RE (1965). Multi stage cutting stock problems of two and more dimensions. *Operations Research* **13:** 94-112.
4. Dyckhoff H (1990). A typology of cutting and packing problems. *European Journal of Operational Research* **44**: 45-159.

5. Coffman EG and Shor PW (1990). Average-case analysis of cutting and packing in two dimensions. *European Journal of Operational Research* **44**: 134-144.
6. Jakobs S (1996). On genetic algorithms for the packing of polygons. *Eur. J. O. R.*. **88**: 165-181.
7. Holland J (1975). *Adaptation in Natural and Artificial Systems*. Ann Arbor: University of Michigan Press.
8. Goldberg D (1989). *Genetic Algorithms in Search, Optimization & Machine Learning*. Addison-Wesley Co., Inc, Reading, MA.
9. Michalewicz Z (1994). *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag.
10. Grefenstette J (1981). *Parallel adaptive algorithms for function optimization.* Technical Report CS: 81-19, Vanderbilt University, Nashville, TN.
11. Goldberg DE. and Lingle R (1985). Alleles, Loci, and the TSP. Presented at the First International Conference on Genetic Algorithms, Lawrence Erlbaum Associates, Hillsdale, NJ. pp. 154-159.
12. Davis L (1985). Applying Adaptive Algorithms to Epistatic Domains. Presented at the International Joint Conference on Artificial Intelligence, pp. 162-164.
13. Oliver IM, Smith DJ and Holland JRC (1987). A Study of Permutation Crossover Operators on the Travelling Salesman Problem. Presented at the Second International Conference on Genetic Algorithms, Lawrence Erlbaum Associates, Hillsdale, NJ. pp. 224-230.
14. Davis L (1991). Handbook of Genetic Algorithms. *Van Nostrand Reinhold*, New York,
15. Belding T (1995). The distributed genetic algorithm. Presented at the Sixth International Conference on Genetic Algorithms, San Mateo, CA: Morgan Kaufmann Publishers.