Inteligencia Artificial 40(2008), 73-85

INTELIGENCIA ARTIFICIAL

http://erevista.aepia.org/

Greedy Seeding Procedure for GAs Solving a Strip Packing Problem

Carolina Salto[1], Enrique Alba[2], Juan M. Molina [2] and Guillermo Leguizamón[3]
[1] Universidad Nacional de La Pampa
General Pico, Argentina
saltoc@ing.unlpam.edu.ar
[2] E.T.S.I. Informática, Universidad de Málaga
Málaga, España
{eat,jmmb}@lcc.uma.es
[3] Universidad Nacional de San Luis
San Luis, Argentina
legui@unsl.edu.ar

Abstract In this paper, the two-dimensional strip packing problem with 3-stage level patterns is tackled using genetic algorithms (GAs). We evaluate the usefulness of a knowledge-based greedy seeding procedure used for creating the initial population. This is motivated by the expectation that the seeding will speed up the GA by starting the search in promising regions of the search space. An analysis of the impact of the seeded initial population is offered, together with a complete study of the influence of these modifications on the genetic search. The results show that the use of an appropriate seeding of the initial population outperforms existing GA approaches on all the used problem instances, for all the metrics used, and in fact it represents the new state of the art for this problem.

Keywords: Genetic Algorithms, Strip Packing, Seeding.

1 Introduction

The two-dimensional strip packing problem (2SPP) arise in many real-world applications such as in the paper or textile industries. Typically, the 2SPP consists of a set of M rectangular pieces, each one defined by a *width* and a *height*, which have to be packed in a larger rectangle with a fixed width W and unlimited length, designated as the *strip*. The search is for a layout of all the pieces in the strip that minimizes the required strip length with the following restrictions: all pieces have to be packed with their sides parallel to the sides of the strip, without overlapping, and rotations are not allowed. This problem is similar to the one of cutting the pieces out of the strip by means of orthogonal cuts, minimizing the consumed strip.

Additionally, another constraint is included in our problem: we consider only 3-stage level (guillotine) packing patterns. In these patterns, pieces are packed by horizontal levels (parallel to the bottom of the strip). Inside each level pieces are packed bottom left justified and, when there is enough room in the level, pieces with the same width are stacked one above the other. Many real application of 2D cutting and packing in the glass, wood, and paper industries consider 3-stage level patterns, for that the importance of incorporating this restriction in the problem formulation.

The 2SPP is NP-hard [9]. A few exact approaches for this problem are known [6, 14], however, metaheuristics are the usual approach to solve it. Regarding the existing surveys of metaheuristics in

ISSN: 1137-3601(print); 1988-3064(on-line)
c AEPIA and the authors

the literature, Hopper and Turton [8, 9] review the approaches developed to solve 2D packing problems using GAs; Simulated Annealing, Tabu Search, and artificial Neural Networks are also considered. They conclude that Evolutionary Algorithms (EAs) [4, 15] are the most widely investigated metaheuristics in the area of cutting and packing. Lodi et al. [13] consider several methods for the 2SPP in their survey and discussed also mathematical models; specially the case where the items have to be packed into rows forming levels are discussed in detail. Other representative works about GAs applied to solve level packing problems are [5], [10], [11], and [16] and for 3-stage guillotine cuts are [17], [18], [21], [22], and [23]. The majority of the approaches imposing this last restriction deal with bin packing problems, for that it is difficult to find previous works in strip packing to compare with.

In this article we use a GA as the general driving force to locate the region in which a solution of minimum length is located. GAs deal with a population of tentative solutions, on which genetic operators are applied in an iterative manner to progressively compute new solutions of higher quality. We study a hybrid approach where a GA is combined with a heuristic placement routine. The GA is used to determine the sequence in which the pieces are to be packed, and the placement routine determines the layout of the pieces onto the strip in order to generate a 3-stage guillotine pattern.

The efficiency of a GA could be improved by increasing the quality of the initial population. This quality depends on two important issues: the average fitness of individuals and the diversity in the population. By having an initial population with better fitness values, better final individuals can be found faster [1, 2, 19]. We here investigate the advantages of seeding the initial population using a set of greedy rules including information of the problem (such as the pieces width, the pieces area, etc.) resulting in a more specialized initial population.

The main goal of this paper is to build an improved GA to solve larger problems than the ones found in the literature at present. As well, to quantify the effects of including a seeding procedure into the algorithms, looking for the best trade-off between exploration and exploitation in the search process. Issue which is considered the key point to perform accurately and efficiently on complex applications.

The organization of the paper is as follows. The components of the used algorithm are described in Section 2. Section 3 describes the greedy generation of the initial population. In Section 4, we explain the parameter settings of the algorithms used in the experimentation. Section 5 reports on the algorithm performances. Finally, in Section 6, we give some conclusions and analyze future search directions.

2 A hybrid GA for the 2SPP

In Algorithm 1 we can see the structure of the basic steady-state GA ($(\mu + 1)$-GA) we use for solving the 2SPP. This algorithm creates an initial population $P(0)$ of μ solutions in a random (uniform) way, and then evaluates these solutions. The evaluation uses a placement algorithm to arrange the pieces in the strip to construct a feasible packing pattern. After that, the population goes into a cycle where it undertakes evolution. This cycle involves the selection of two parents by binary tournament and the application of recombination and mutation operators to create a new solution ($P'(t)$). The new generated individual replaces the worst individual in the population only if it is fitter. In this study, the stopping criterion for the cycle is to reach a maximum number of evaluations ($max\_evaluations$). The best solution is identified as the best individual ever found which minimizes the required strip length.

In most cases, applying a GA means devising a customized representation for encoding a candidate solution. We encode a packing pattern into a chromosome as a sequence of pieces that defines the input for the layout algorithm. Therefore, a chromosome will be a permutation $\pi = (\pi_1, \pi_2, ..., \pi_M)$ of M natural numbers (piece identifiers). In order to generate a 3-stage level pattern, a modified *next-fit* heuristic (NF) is used here —in the following referred as *modified next-fit*, or *MNF*— which was proved to be very efficient in [18, 20]. This heuristic gets a sequence of pieces as its input and constructs the packing pattern by placing pieces into stacks, and then stacks into levels in a greedy way, i.e., once a new stack or a new level is started, previous ones are never reconsidered. See Figure 1 for an illustrative example. Deeper explanation of the MNF procedure can be found in [23].

GAs are guided by the values computed by an objective function for each tentative solution until the optimum or an acceptable solution is found. In our problem, the objective is to minimize the strip length needed to build the layout corresponding to a given solution π. An important consideration is that two packing patterns could have the same length —so their fitness will be equal— nevertheless, from the

Algorithm 1 Genetic algorithm

```
GA
$t = 0$; {current generation}
initialize($P(t)$);
evaluate($P(t)$);
while ($t < max\_evaluations$) do
   $P'(t)$ = evolve($P(t)$); {recombination and mutation}
   evaluate ($P'(t)$);
   $P(t+1)$ = select($P'(t), P(t)$);
   $t = t + 1$;
end while
```

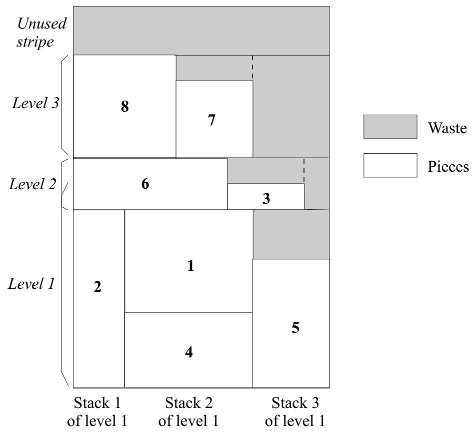

Figure 1: Packing pattern for the permutation 2 4 1 5 6 3 8 7.

point of view of reusing the trim loss, one of them can be actually better because the trim loss in the last level (which still connects with the remainder of the strip) is greater than the one present in the last level of the other layout. Therefore we are using the following fitness function:

$$F(\pi) = strip.length - \frac{l.waste}{l.lenght * W} \tag{1}$$

where *strip.length* is the length of the packing pattern corresponding to the permutation π, *l.waste* is the area of reusable trim loss in the last level l of the packing pattern and *l.lenght* is the length of l. Hence, $F(\pi)$ is both simple and accurate.

Regarding the genetic operators, the recombination operator used here, *Best Inherited Level Recombination (BILX)* (introduced in [23] as BIL), incorporates some problem-specific knowledge into their mechanism in order to improve the trim loss. The BILX operator transmits the best levels of the parent to the child, i.e., those with the highest filling rate (*fr*) or, equivalently, with the least trim loss. This rate is calculated as follows, for a given level l:

$$fr(l) = \sum_{i=1}^{n} \frac{width(\pi_i) \times length(\pi_i)}{W \times l.length} \tag{2}$$

where $\pi_1, ..., \pi_n$ are the pieces in l, $width(\pi_i)$ and $length(\pi_i)$ are the respective dimensions.

Actually, BILX works as follows. Let nl be the number of levels in one parent $parent_1$. In the first step the filling rates of all nl levels from $parent_1$ are calculated. After that, a probability of selection, proportional to its filling rate, is assigned to each level and a number $(nl/2)$ of levels are selected from $parent_1$. The pieces π_i belonging to the inherited levels are placed in the first positions of the child. Meanwhile, the remaining positions are filled with the pieces which do not belong to that levels, in the order they appear in the other parent $parent_2$.

Table 1: Rules to generate the initial population

Rule	Description
1	sorts pieces by decreasing width.
2	sorts pieces by increasing width.
3	sorts pieces by decreasing height.
4	sorts pieces by increasing height.
5	sorts pieces by decreasing area.
6	sorts pieces by increasing area.
7	sorts pieces by alternating between decreasing width and height.
8	sorts pieces by alternating between decreasing width and increasing height.
9	sorts pieces by alternating between increasing width and height.
10	sorts pieces by alternating between increasing width and decreasing height.
11	the pieces are reorganized following a modified BFDH heuristic.
12	the pieces are reorganized following a modified FFDH heuristic.

On the other side, the idea behind the mutation operator is to change the location of some pieces so that the final cost is reduced; it was successfully tested in [23]. Named as *Best and Worst Stripe Exchange (BW_SE)*, this mutation changes the location of the best and the worst level. The pieces of the best level (the one with highest filling rate) are allocated in the first positions of the new packing pattern while the pieces of the worst level are assigned to the last positions. The middle positions are filled with the remaining pieces in the order they appeared in the original packing pattern. In BW_SE, the movements can help to the involved levels or their neighbors to accommodate pieces from neighboring levels, thus improving their trim loss.

3 Initial Seeding

The performance of a GA is often related to the quality of its initial population. This quality depends on two important issues: the average fitness of individuals and the diversity in the population. By having an initial population with better fitness values, better final individuals can be found faster [1, 2, 19]. Besides, high diversity in the population inhibits early convergence to a locally optimal solution.

There are many ways to arrange this initial diversity. The idea in this work is to start with a seeded population created by following some building rules, hopefully allowing us to reach good solutions in the early stages of the search. The rules will have include some characteristics from the problem such as piece sizes, and also incorporate ideas from the BFDH and FFDH heuristics [12].

Individuals are generated in two steps. In the first step, the packing patterns are randomly sampled from the search space with a uniform distribution. After that, each of them is modified by one rule, randomly selected, with the aim of improving the piece location inside the random packing pattern. Each application of a rule yields a (possibly) different solution because of the randomization used in the first step.

The rules for the initial seeding are listed in Table 1. These rules are proposed with the aim of producing individuals with improved fitness values and also for introducing diversity into the initial population. Hence, sorting the pieces by their width (Rule 1 and 2) will hopefully increase the probability of stacking pieces (see Figure 2 (left)), and then produce denser levels. On the other hand, sorting by height (Rule 3 and 4) will generate levels with less wasted space above the pieces, especially when the heights of the pieces are very similar (see Figure 2 (middle)). Rules 7 to 10 have been introduced to increase the initial diversity. Finally, Rules 11 and 12 relocate the pieces with the goal of reducing the trim loss inside a level (see Figure 2 (right) where Rule 12 was applied). As we will see, these rules are not only useful for initial seeding: several of them can be used as a simple greedy algorithm for a local search to help during the optimization process. The MBF_Adj operator is an example which is based on Rule 11.

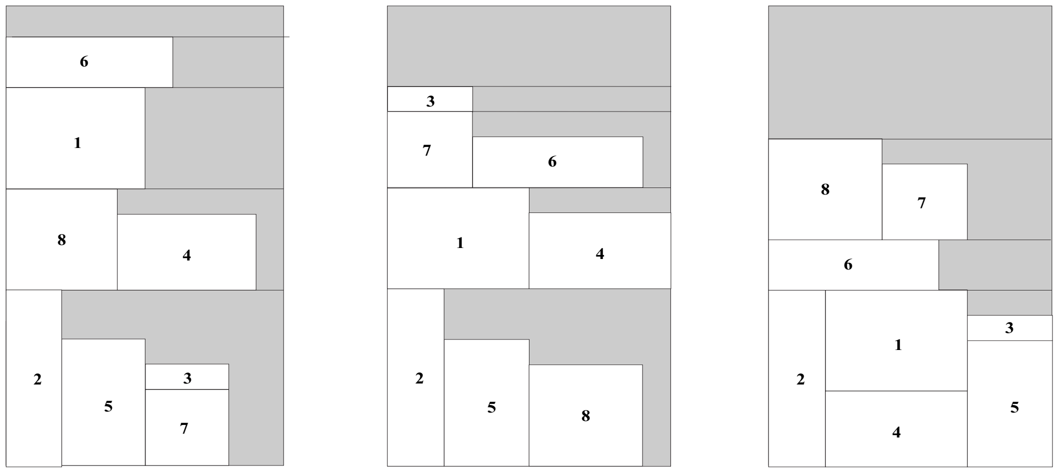

Figure 2: Packing patterns after the application of Rule 2 (left), Rule 4 (middle) and Rule 12 (right) to the original permutation 2 4 1 5 6 3 8 7.

4 Implementation

The specific $(\mu + 1)$-GA we have implemented is analyzed here using three different methods of seeding the initial population: (i) by means of a random generation (GA), (ii) by applying one determined rule from the Table 1 (GA_i where i stands for a rule number) and (iii) by randomly applying a rule taken from Table 1 for each individual of the population (GA_{Rseed}). We also consider the random generation of individuals as other applicable rule in this case, i.e., once a individual is generated, no rule is applied.

The population size was set to 512 individuals. The maximum number of evaluations was fixed to 2^{16}. The recombination probability was set to 0.8, while the mutation probability was set to 0.1. Parameters (population size, stop criterion, probabilities, etc.) were not chosen at random, but rather by an examination of values previously used with success (see [21]).

The algorithms were implemented inside MALLBA [3], a C++ software library fostering rapid prototyping of hybrid and parallel algorithms. The platform was an Intel Pentium 4 at 2.4 GHz and 1 GB RAM, linked by Fast Ethernet, under SuSE Linux with 2.4.19-4GB kernel version.

We have considered five randomly generated problem instances with M equals to 100, 150, 200, 250, and 300 pieces and a known global optimum equals to 200 (the minimum length of the strip). These instances belong to the subtype of level packing patterns but the optimum value does not correspond to a 3-stage guillotine pattern. They were obtained by an own implementation of a data set generator, following the ideas proposed in [25]. The length-to-width ratio of all M rectangles is set in the range $1/3 \leq l/w \leq 3$. These instances are publicly available at `http://mdk.ing.unlpam.edu.ar/~lisi/2spp.htm`.

5 Computational Analysis

In this section we summarize the results of applying the proposed algorithm with its seeding variants to all the problem instances. Our aim is to offer meaningful results and check them from a statistical point of view.

For each algorithm we have performed 30 independent runs per instance using the parameter values described in the previous section. Also, the evaluation considers two important issues for any search process: the capacity for generating new promising solutions and the pace rate of the progress in the surroundings of the best found solution (fine tuning or intensification). For a meaningful analysis we consider the average fitness values and the entropy measure (as proposed in [7]), which is computed as follows:

$$entropy = \frac{\sum_{i=1}^{M} \sum_{j=1}^{M} (\frac{n_{ij}}{\mu}) \ln(\frac{n_{ij}}{\mu})}{M \ln M} \qquad (3)$$

Table 2: Experimental results for the GA using different seeding methods.

Alg	$M = 100$			$M = 150$			$M = 200$			$M = 250$			$M = 300$		
	avg_i	***best***	$\boldsymbol{avg}_{\pm\sigma}$	avg_i	***best***	$\boldsymbol{avg}_{\pm\sigma}$	avg_i	***best***	$\boldsymbol{avg}_{\pm\sigma}$	avg_i	***best***	$\boldsymbol{avg}_{\pm\sigma}$	avg_i	***best***	$\boldsymbol{avg}_{\pm\sigma}$
GA	417.26	234.66	240.21 ± 4.81	504.83	238.43	247.94 ± 5.30	485.36	243.52	250.94 ± 4.82	488.54	241.68	249.32 ± 4.19	531.41	241.04	258.10 ± 6.10
GA_1	353.30	229.71	241.84 ± 9.37	408.04	231.73	243.70 ± 5.97	381.72	223.71	240.08 ± 6.15	354.59	230.79	237.54 ± 3.45	409.64	234.70	248.01 ± 5.78
GA_2	298.17	229.70	235.63 ± 5.51	368.69	236.73	245.88 ± 4.10	352.22	229.57	240.29 ± 4.59	331.22	227.74	237.90 ± 4.55	345.29	241.24	246.00 ± 3.35
GA_3	325.88	229.63	233.85 ± 4.98	274.10	226.39	229.17 ± 1.88	294.71	222.54	228.01 ± 2.48	284.16	222.41	228.08 ± 2.43	273.27	227.79	236.89 ± 3.45
GA_4	282.92	226.70	228.80 ± 0.83	240.37	226.18	227.82 ± 0.90	244.53	221.36	225.32 ± 2.04	239.70	221.59	224.80 ± 1.51	239.11	224.79	230.62 ± 2.65
GA_5	381.35	240.60	251.90 ± 5.37	427.78	257.62	271.73 ± 6.99	415.78	249.63	261.49 ± 5.28	377.27	242.56	255.23 ± 5.37	401.09	255.64	268.16 ± 5.92
GA_6	371.87	249.54	260.05 ± 5.40	377.75	257.61	267.92 ± 4.14	359.10	241.37	253.18 ± 5.91	357.14	248.56	258.88 ± 4.65	375.08	272.59	286.64 ± 5.30
GA_7	396.65	236.68	244.61 ± 6.47	486.68	239.61	258.42 ± 7.75	452.28	236.61	249.62 ± 6.61	412.23	243.27	251.62 ± 4.74	454.39	260.31	273.58 ± 5.63
GA_8	299.20	229.72	238.15 ± 3.58	274.69	241.49	246.04 ± 1.70	275.21	233.68	238.32 ± 1.62	262.78	231.21	234.61 ± 1.62	259.63	241.63	244.70 ± 1.48
GA_9	384.48	233.74	246.34 ± 6.22	499.78	245.62	258.00 ± 6.09	460.07	237.29	250.37 ± 5.15	450.02	246.60	255.76 ± 5.91	486.95	259.57	272.17 ± 5.01
GA_{10}	353.97	237.33	246.60 ± 6.74	353.53	255.68	272.59 ± 7.13	369.00	241.35	251.28 ± 5.45	321.23	240.66	253.39 ± 5.85	336.15	269.65	275.64 ± 3.74
GA_{11}	277.93	229.32	235.17 ± 2.97	290.17	232.77	241.57 ± 4.02	278.52	227.66	233.69 ± 3.01	273.96	226.68	232.63 ± 2.79	283.35	229.37	234.40 ± 3.11
GA_{12}	281.64	226.26	238.15 ± 4.29	294.10	237.69	246.23 ± 4.32	282.66	229.65	236.37 ± 3.56	277.04	229.65	235.21 ± 2.99	288.46	232.60	241.03 ± 3.76
GA_{Rseed}	340.35	222.75	230.18 ± 2.28	368.81	226.10	228.56 ± 1.66	358.60	220.74	224.85 ± 2.67	339.49	218.68	224.23 ± 1.81	360.13	222.63	230.71 ± 2.94

where n_{ij} represents the number of times the piece i is set into the position j in the population of size μ. This function takes values in [0..1] and a value of 0 indicates that all the individuals in the population are identical.

Table 2 shows the results obtained for the different methods for generating the initial population (GA, GA_i, and GA_{Rseed}) for all instances. The most relevant metrics used in this comparison are: the average objective value of the initial population (column avg_{ini}), the best found feasible solution (column *best*) and the average value of the best found feasible solution, in the 30 independent runs, along with its standard deviation (column $avg_{\pm\sigma}$).

From that results we can point out that any seeded GA starts the search process from better fitness values than in the case of randomly generated initial populations (GA). Best initial populations, in average, are obtained using GA_4 but having poor genetic diversity (see Figure 3 which shows the mean entropy values in the initial population for each algorithm and different M values). Rule 4 arranges the pieces by their height, so the pieces in each level have similar heights and consequently the free space inside each level tends to be small. On the other hand, a poor performance is obtained with both GA_7 and GA_9, which have the worst initial population means as well as poor genetic diversity. The random selection of rules to generate the initial solution (GA_{Rseed}) works quite well (being in the medium positions of the rule ranking) with an acceptable initial population diversity (entropy value near to 0.8).

Figure 3 shows that most of the seeded GAs present poor initial genetic diversity (less than 0.5), except GA_{12}, GA_{11}, and GA_{Rseed} with an initial diversity (near to 1) close to one of the non seeded GA. GA_{12} and GA_{11} apply NF and BF heuristics (respectively) to a randomly generated solution, i.e., these heuristics do not take piece dimensions into account hence the original piece positions inside the chromosome suffer few modifications. On the other hand, GA_{Rseed} combines all the previous considerations with random generation of the piece positions, so high genetic diversity was expected. This algorithm presents entropy values near 0.8, and the average objective value of its initial population is in the middle of the rule ranking.

As supposed, the GA_{Rseed} significantly reduces the number of evaluations required to find good solutions (see Figure 4): they are near almost half as many on average as with GA. To confirm these observations, we used the t-test, which indicates that the difference among the algorithms are significant under this metric (p-values near to 0). GA_7, GA_9, and GA present similar effort in order to locate their

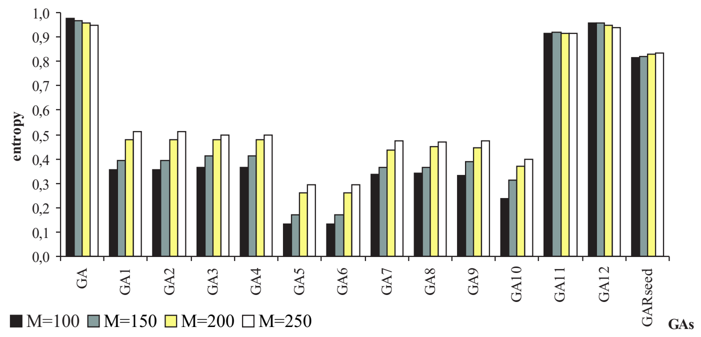

Figure 3: Average entropy of initial population.

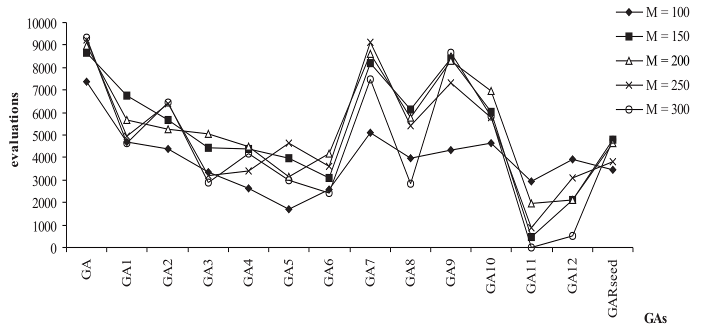

Figure 4: Mean number of evaluations to reach the best value for each instance.

best solutions: all of them have quite similar initial population averages (see Table 2) but GA presents best final solution qualities.

With respect to mean execution times there are a slight difference compared to any seeded GA due to the initialization phase, but this difference is negligible.

After the 30 runs of GA_{Rseed} for all M values, we analyzed the contribution of the solutions generated by each rule to the average fitness in the initial population. Regarding that measurement, the rule rank, from best to worst, was as follows: random, 9, 7, 5, 1, 6, 10, 2, 8, 3, 12, 11 and 4 (see Figure 5), although the differences in percentages are quite small. The no application of rules, i.e., preserving the random piece position, are first allocated in the ranking, followed by the ones sorting pieces by alternating between increasing/decreasing width and height. By surprise, rules incorporating the BF and FF heuristics appear in last positions, when the expectation was that they could create good packing patterns due to the improvement in the layout they achieved. Also a remarkable point here is the rank position of Rule 4 which appeared in the last position, although its use for the initialization of the whole population brings to a better performance than the rest.

Furthermore, a neat conclusion of this study is that the GA_{Rseed} (random seeded initial populations) significantly outperforms the GA with a traditional initialization in all the metrics (the p-value is close to 0) and also the rest of the seeded GAs. This suggests that the efficiency of a GA could be improved simply by increasing the quality of the initial population.

5.1 Additional Discussion

If we seek among all the proposed algorithms, we can state that the GA_{Rseed}, which initializes the population considering the whole set of rules including information of the problem, converges to a better final solution, allowing a highly reduced number of evaluations comparing to that of GA using a random

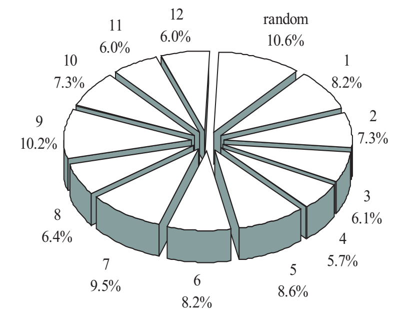

Figure 5: Rule Rank.

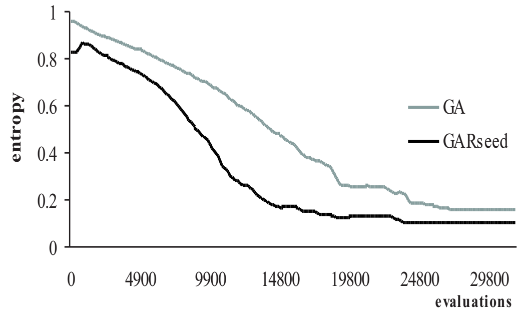

Figure 6: Population entropy for M=200.

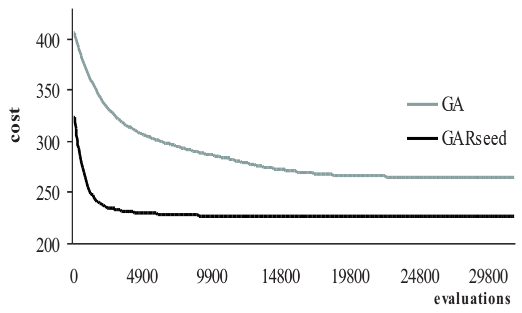

Figure 7: Average population fitness for M=200.

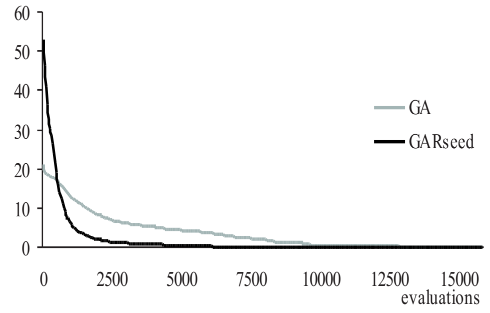

Figure 8: Standard deviation of fitness for M=200.

sampled initial population. Therefore, for the next study we compare the performance of GA_{Rseed} against the plain GA.

Owing to the stochastic nature of the evolutionary process, all the results shown here were obtained by averaging 30 independent runs of the same experiments. Curves are drawn for the GA and GA_{Rseed} algorithms. Figure 6 plots the entropy values versus the number of evaluations meanwhile Figure 7 plots the fitness versus computational effort. In order to assess the statistical significance of the results, standard deviations of the fitness are also presented (Figure 8). These curves correspond to the instance with M=200, because similar results are obtained with the rest of the instances. From Figure 7 and 8, we observe that the distances between the curves for GA_{Rseed} and those for the GA are always larger than the standard deviations. One can thus conclude that, the GA_{Rseed} has a fast convergence to the best

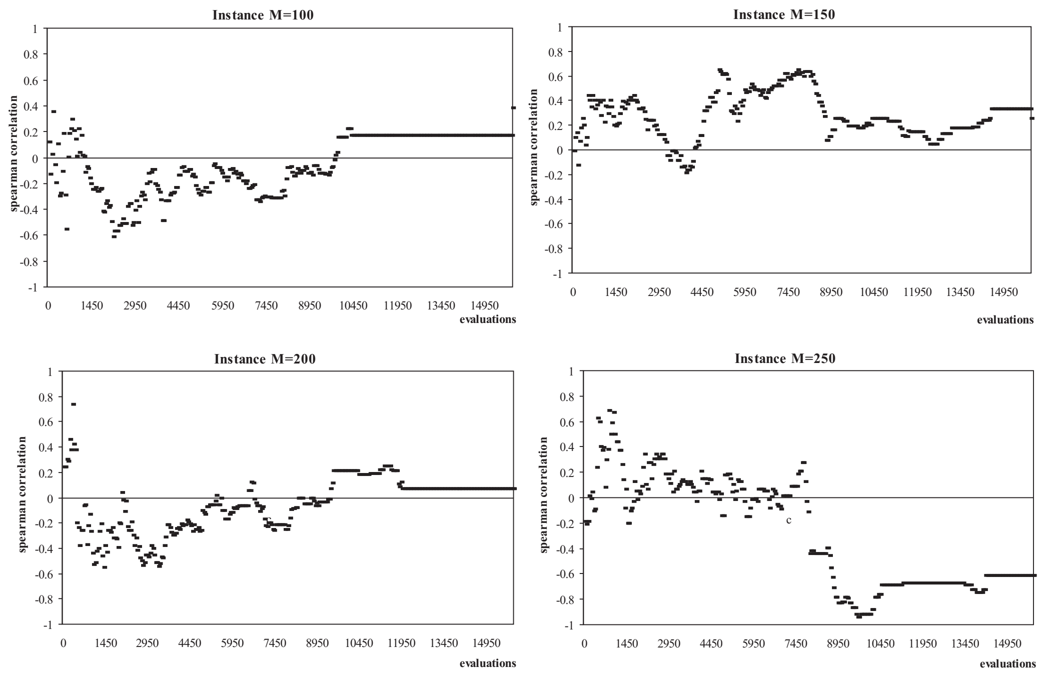

Figure 9: Evolving populations correlation between best fitness in each population and entropy measure. Each point represents the correlation between 15 populations from a 15 runs.

final solutions, but, for this algorithm, the genetic diversity decreases slower than standard deviations. Thus, the GA_{Rseed} maintains good values of genetic diversity for longer than the GA, however, GA_{Rseed} converge faster to the best final solution.

In order to provide a deeper explanation of what is happening during the search of the algorithms we analyze the relationship between the fitness and population entropy (genotypic diversity) which will have a strong effect on search difficulty. A way to examine this is by the use of the Spearman correlation measure (a nonparametric correlation test), which ranks two sets of variables and tests for a linear relationship between the variables rank. The Spearman correlation coefficient is computed (from [24]) as follows:

$$1 - \frac{6\sum_{i=1}^{N} d_i^2}{N \times (N^2 - 1)} \tag{4}$$

where N is the number of items, and d_i is the difference between each pair of fitness and entropy ranks. The correlation values range from +1 (perfect positive correlation), through 0 (no correlation), to -1.0 (negative correlation). For our study, if we see ideal low fitness values, which will be ranked in ascending order (1=best, ..., 15=worst) and high diversity, ranked in ascending order (1=lowest diversity and 15=highest diversity), then the correlation coefficient should be strongly negative. Alternatively a positive correlation indicates that either bad fitness occurs with high diversity or good fitness occurs with low diversity.

Figure 10 shows the correlation between entropy and best fitness for each generation. Note that each point represents the correlation between 15 populations, sampled from 15 runs where there is a dependency of later evaluations on preceding ones. Experiments with instance M=250 show a period of fluctuation until evaluation 8000, then the correlation coefficient became strongly negative, indicating that a good fitness is present with good diversity. Experiments with instances with M=100 and M=200 contain a period of fluctuation until evaluation 11000 approximately, after which entropy lost correlation with best fitness (coefficient correlations near to zero). For these two instances, the relationship between fitness and diversity becomes less important, probably due to other critical relationships. Finally, after a period of early fluctuations, the correlation for instance M=150 is positive, this can be owing to the fact

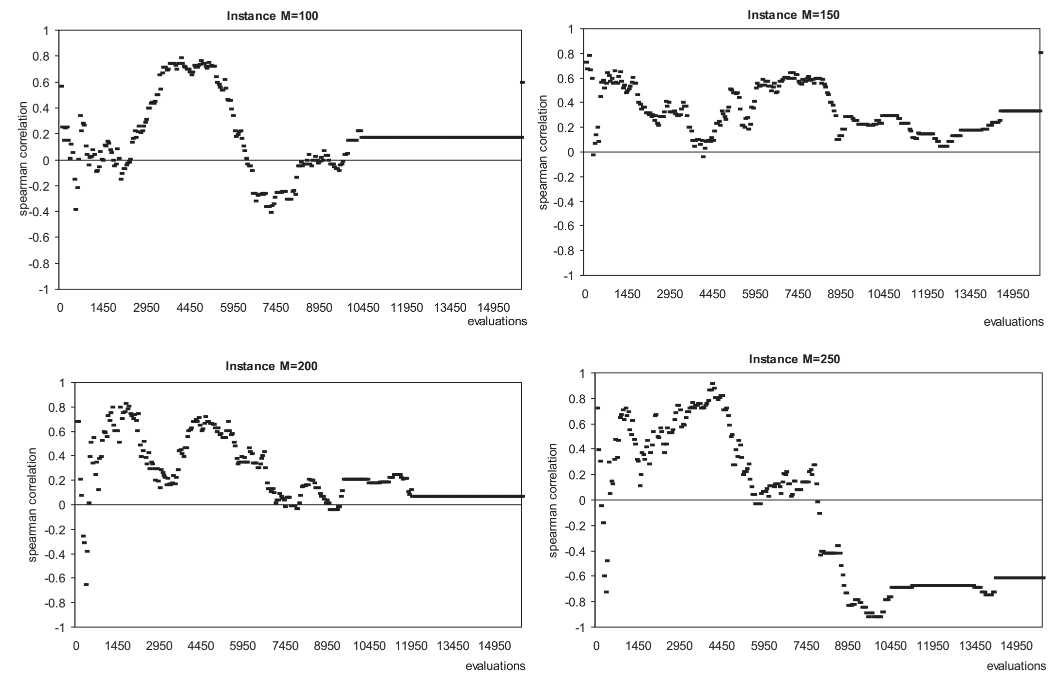

Figure 10: Evolving populations correlation between mean fitness in each population and entropy measure. Each point represents the correlation between 15 populations from 15 runs.

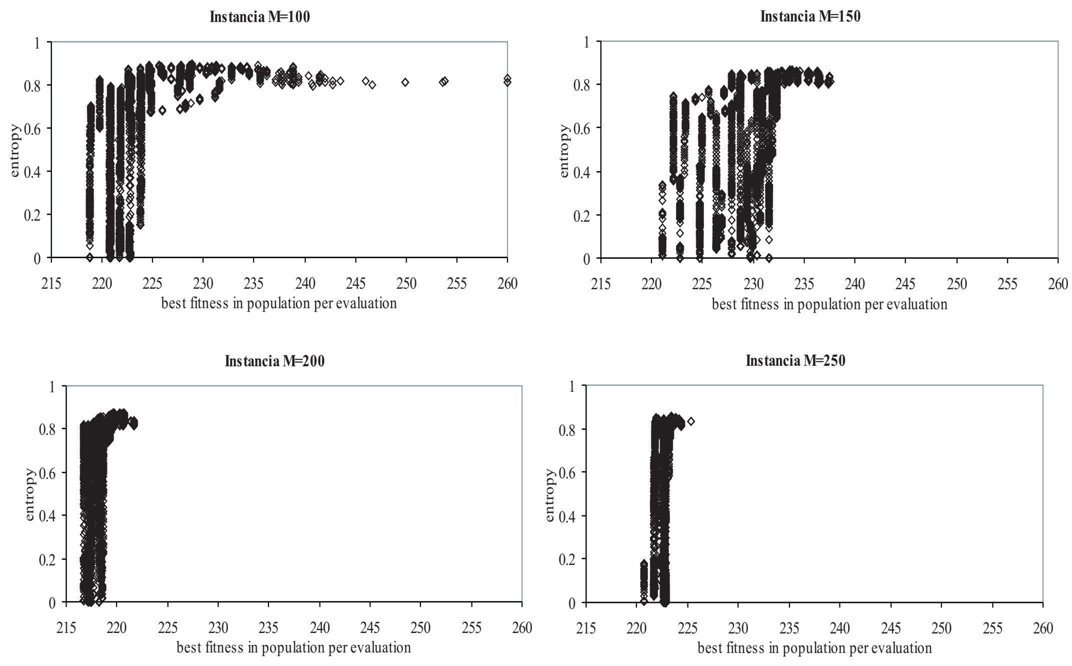

Figure 11: Best fitness plotted against the genetic diversity.

that, as the best fitness is achieved early in a run, the population contains an increasingly larger number of copies of the best fit individual.

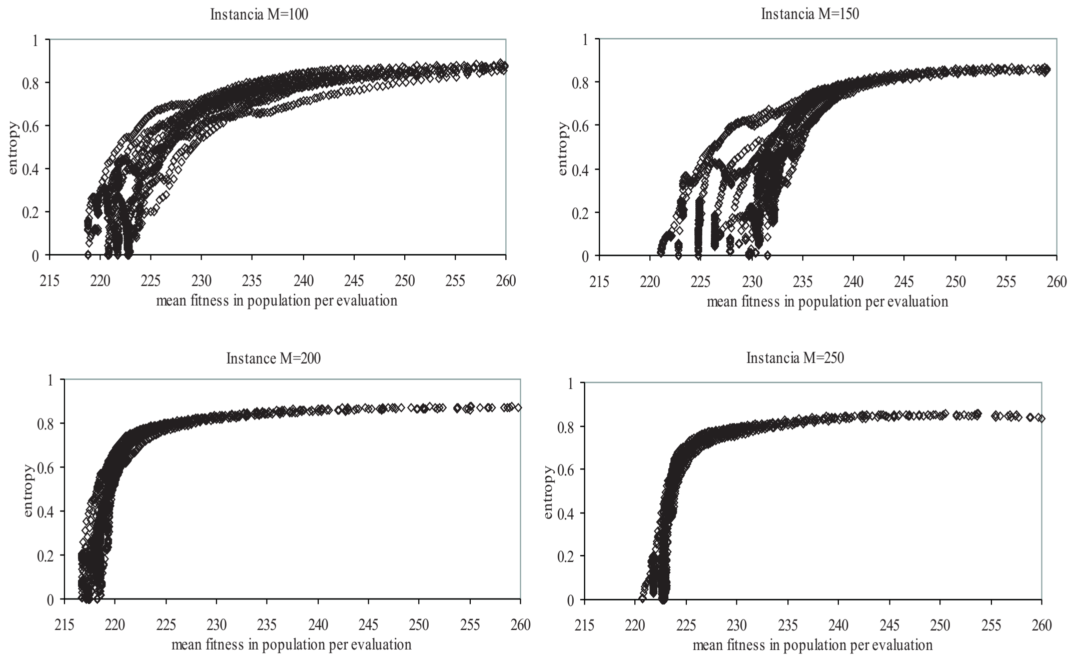

Figure 12: Mean population fitness plotted against that genetic diversity.

Figure 11 plots the best fitness found in the population along the x axis and genetic diversity on the y axis. From there we can see some populations with bad fitness (low values are better) occurring with higher entropy. Once the algorithm find a good fitness, the entropy values begin to decrease (the populations are less diverse), i.e., the population is made up of an increasingly larger number of copies of the best fit individual. Figure 12 plots the mean population fitness found in the population along the x axis and population's diversity on the y axis.

6 Conclusions

In this paper we have investigated different greedy methods of generating the initial population in a traditional GA to solve the 3-stage 2SPP. The study, validated from a statistical point of view, analyzes the capacity of the seeding policy to generate new potentially promising individuals and the ability to keep a diversified population. The problem-aware seeding includes pieces dimensions and known level heuristics.

Our results show that an improvement in the GA performance was observed by using a problem-aware seeding, regarding both efficiency (effort) and quality of the solutions found. Moreover, the random selection of rules to build the initial population works properly, providing good genetic diversity of initial solutions. The performance of a GA is sensitive to the quality of its initial population. By having an initial population with better fitness values, we typically get better final individuals.

As future works we propose to investigate non-permutation representations and a direct mapping to the final disposition of the pieces, as well as to construct parallel versions of the algorithms studied in this work.

Acknowledgements

This work has been partially funded by the Spanish Ministry of Education and the European FEDER under contract TIN2005-08818-C04-01 (the OPLINK project, http://oplink.lcc.uma.es). We also acknowledge the Universidad Nacional de La Pampa and Universidad Nacional de San Luis, and the ANPCYT in Argentina from which we received continuous support. Moreover the first author thanks CONICET (National Research Council of Argentina) for a Doctoral fellowship.

References

[1] R.K. Ahuja and J.B. Orlin. Developing fitter genetic algorithms. *INFORMS Journal on Computing*, 9(3):251–253, 1997.

[2] R.K. Ahuja, J.B. Orlin, and A. Tiwari. A greedy genetic algorithm for the quadratic assignment problem. *Computers & Operations Research*, 27(3):917–934, 2000.

[3] E. Alba, J. Luna, L.M. Moreno, C. Pablos, J. Petit, A. Rojas, F. Xhafa, F. Almeida, M.J. Blesa, J. Cabeza, C. Cotta, M. Díaz, I. Dorta, J. Gabarró, and C. León. *MALLBA: A Library of Skeletons for Combinatorial Optimisation*, volume 2400 of *LNCS*, pages 927–932. Springer, 2002.

[4] T. Bäck, D. Fogel, and Z. Michalewicz. *Handbook of evolutionary computation.* Oxford University Press, New York, 1997.

[5] A. Bortfeldt. A genetic algorithm for the two-dimensional strip packing problem with rectangular pieces. *European Journal of Operational Research (article in press)*, 2005.

[6] S.P. Fekete and J. Schepers. On more-dimensional packing III: Exact algorithm. Technical Report ZPR97-290, Mathematisches Institut, Universität zu Köln, available from the first author at Department of Mathematics, 1997.

[7] J.J. Grefenstette. *Genetic Algorithms and Simulated Annealing*, chapter Incorporating problem specific knowledge into genetic algorithms, pages 42–60. Morgan Kaufmann Publishers, 1987.

[8] E. Hopper. *Two-dimensional packing utilising evolutionary algorithms and other meta-heuristic methods.* PhD thesis, University of Wales, Cardiff, U.K., 2000.

[9] E. Hopper and B. Turton. A review of the application of meta-heuristic algorithms to 2d strip packing problems. *Artificial Intelligence Review*, 16:257–300, 2001.

[10] S. Hwang, C. Kao, and J. Horng. On solving rectangle bin packing problems using genetic algorithms. *IEEE International Conference on Systems, Man, and Cybernetics - Humans, Information and Technology*, 2:1583–1590, 1994.

[11] B. Kroger. Guillotineable bin-packing: a genetic approach. *European Journal of Operational Research*, 84:645–661, 1995.

[12] A. Lodi, S. Martello, and M. Monaci. Recent advances on two-dimensional bin packing problems. *Discrete Applied Mathematicsournal of Operation Research*, 141:241–252, 2002.

[13] A. Lodi, S. Martello, and M. Monaci. Two-dimensional packing problems: a survey. *European Journal of Operational Research*, 141:241–252, 2002.

[14] S. Martello, S. Monaci, and D. Vigo. An exact approach to the strip-packing problem. *Informs Journal on Computing*, 15:310–319, 2003.

[15] M. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs.* Springer, third revised edition, 1996.

[16] C.L. Mumford-Valenzuela, J. Vick, and P.Y. Wang. *Metaheuristics: Computer Decision-Making*, chapter Heuristics for large strip packing problems with guillotine patterns: An empirical study, pages 501–522. Kluwer Academic Publishers BV, 2003.

[17] J. Puchinger and G.R. Raidl. Models and algorithms for three-stage two-dimensional bin packing. Technical report, Technische Universitat Wien, Institul fur Computergraphik und Algorithmen, 2004.

[18] J. Puchinger, G.R. Raidl, and G. Koller. *Solving a Real-World Glass Cutting Problem*, volume 3004 of *LNCS*, pages 162–173. Springer, 2004.

[19] C.R. Reeves. A genetic algorithm for flowshop sequencing. *Computers & Operations Research*, 22(1):5–13, 1995.

[20] C. Salto, J.M. Molina, and E. Alba. Sequential versus distributed evolutionary approaches for the two-dimensional guillotine cutting problem. *Proceedings of International Conference on Industrial Logistics (ICIL 2005)*, pages 291–300, 2005.

[21] C. Salto, J.M. Molina, and E. Alba. Analysis of distributed genetic algorithms for solving cutting problems. *International Transactions in Operational Research*, 13(5):403–423, 2006.

[22] C. Salto, J.M. Molina, and E. Alba. A comparison of different recombination operators for the 2-dimensional strip packing problem. *Proceedings of the XII Congreso Argentino de Ciencias de la Computacin (CACIC'06)*, 2006.

[23] C. Salto, J.M. Molina, and E. Alba. Evolutionary algorithms for the level strip packing problem. *Proceedings of the Workshop on Nature Inspired Cooperative Strategies for Optimization NICSO 2006*, pages 137–148, 2006.

[24] S. Siegel. *Nonparametric Statistics for the Behavioral Sciences*. New York: McGraw-Hill, 1956.

[25] P.Y. Wang and C.L. Valenzuela. Data set generation for rectangular placement problems. *EJOR*, 134:378–391, 2001.