ELSEVIER

European Journal of Operational Research 84 (1995) 645–661

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

Guillotineable bin packing: A genetic approach

Berthold Kröger *

Department of Mathematics and Computer Science, University of Osnabrück, P.O. Box 4469, 49 049 Osnabrück, Germany

Abstract

For a constrained, two-dimensional Bin Packing Problem this paper introduces a sequential and a parallel genetic algorithm. A guillotine constraint is directly reflected by the encoding mechanism and thus is ensured at any stage of the algorithm. As an enlargement, the concept of meta-rectangles is proposed and incorporated into the algorithm. Each meta-rectangle temporarily fixes (rectangular) hyperplanes of existing solutions. By this the hierarchical structure of guillotineable packing schemes is exploited to reduce the problem's complexity without affecting the quality of the generated solutions. The presented algorithm is able to generate almost optimal packing schemes and even in its sequential version the algorithm empirically is proven to be superior to different approaches like random search or simulated annealing.

Keywords: Two-dimensional bin packing; Parallel genetic algorithm; Simulated annealing

1. Introduction

Genetic algorithms [4,8] and simulated annealing [1] define heuristic approaches which apply analogies from nature to successfully tackle complex optimization problems [1,11,12]. Basically both strategies describe iterative improvement techniques. Existing solutions (the *ancestors*) are randomly modified to *offsprings* which might replace their parents. For two-dimensional Bin Packing Problems – similar to the one considered in this paper – both strategies were tried, too [5,14,15].

However, most of these approaches suffer from the unfavorable structure to represent and manipulate packing schemes. This makes the creation of almost optimal solutions quite unlikely: Smith [15] uses permutations of rectangles to encode the instances of a Bin Packing Problem. Thus, the original problem becomes a sequencing problem and heuristics have to transform those permutations into packing schemes. In contrast, Dowsland [5] and Oliveira [14] suggest a more 'neural-like' behavior of their simulated annealing algorithms. Instead of guaranteeing the existence of admissible solutions at any time, the items to be packed may temporarily overlap. Varying the ancestors then aims at both, the elimination of overlap and the minimization of the criterion function.

* Now with the Research Center of Deutsche Telekom AG, P.O. Box 100003, D-64276 Darmstadt, Germany.

SSDI 0377-2217(95)00029-1

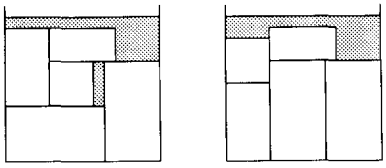

Fig. 1. Non-guillotineable (left) and guillotineable packing scheme (right).

A different approach to encode the instances of a certain Bin Packing Problem is proposed in this paper. It is suitable for genetic algorithms as well as for simulated annealing approaches and it supports the generation of almost optimal solutions as it reflects their characteristic features [9]. In addition, merely feasible solutions are considered at any stage of the algorithms. Thus, the optimization process can concentrate on its major task of minimizing the criterion function.

The two-dimensional Bin Packing Problem which is considered throughout this paper demands to generate a guillotineable [2] packing scheme of minimal height for a set of rectangles into a two-dimensional open-ended bin B of width W. All rectangles are non-oriented and may be rotated. The packing scheme has to be orthogonal [2] and w.l.o.g. it can be required to be bottom-left-justified. Fig. 1 presents two orthogonal packing schemes: a non-guillotineable (left) and a guillotineable (right). A packing scheme is said to be guillotineable, if and only if it can be created by recursively bipartitioning the Bin B with straightlined, guillotineable cuts. Each of these cuts intersects a rectangular area within B into two likewise rectangular pieces. According to Dyckhoff's typology [6] this problem belongs to a 2/V/O category.

The following chapter presents a sequential genetic algorithm for the Guillotine Bin Packing Problem (GBPP) mentioned above. By means of some experimental results the quality of the algorithm is compared to other heuristics including random search and simulated annealing. In Section 3 parallelization schemes for genetic algorithms are applied and evaluated. The innovative concept of meta-rectangles is introduced in Section 4. These meta-rectangles temporarily fix (rectangular) hyperplanes of existing solutions and thus reduce the problem's complexity. Some final experimental results indicate that these restrictions speed up the algorithm and even may increase the quality of the generated solutions.

2. Sequential genetic algorithm

In genetic algorithms [4,8] a population Π of artificial individuals (each describing a complete packing scheme) is submitted to the simulation of an evolutionary process for several generations. During each generation selected individuals are manipulated either by the application of a *mutation* operator or they mate and produce offsprings Ω by means of a *crossover* operator. The life-span of individuals is ruled by a *reduction* process which – at the end of each generation – determines members of the current population Π and from the set Π_Ω of newly generated offsprings to form the population of the subsequent generation.

In general, both selective decisions within a genetic algorithm, the choice of ancestors to manipulate and the reduction to a new population are quality-driven. A fitness function which is derived from the criterion function to be optimized serves as a measure to evaluate individuals. Then, a successive improvement of the population's mean quality can be achieved. However, since genetic algorithms still are heuristic strategies, it cannot be guaranteed that an optimal solution will be found.

The listing in Fig. 2 presents the scheme of a genetic algorithm as it will be considered for the remainder of this paper. The algorithm includes procedures `evaluate`, `select`, `mutate`, crossover and `reduce` which represent the evaluation, selection, mutation, crossover and reduction operators mentioned above. The probability of applying the mutation operator can be adjusted with the `mutation.prob` parameter. Two other parameters, `max.gens` and `max.offs` are needed to determine upper bounds for the total number of generations resp. the amount of offsprings per generation.

Besides the operators already mentioned, a hillclimbing process is applied, too. Its task is to

```
generation:= 0
Π:= initialize.population()
FOR i:=1 to |Π| DO
   fitness(I_i):= evaluate(I_i)
WHILE generation < max.gens DO
   generation:= generation + 1
   Π_Ω:= ∅ ;  no.offspring:= 0
   WHILE no.offspring ≤ max.offs DO
      dad:= select(Π)
      IF random(0,100) < mutation.prob
         THEN Ω:= mutate(dad)
         ELSE mom:= select(Π)
              Ω:= crossover(mom, dad)
      Ω:= hillclimber(Ω)
      fitness(Ω):= evaluate(Ω)
      Π_Ω:= Π_Ω ∪ {Ω}
      no.offspring:= no.offspring + 1
   Π:= reduce(Π, Π_Ω)
```

Fig. 2. Scheme of a genetic algorithm.

cause an immediate improvement of the quality of each newly generated offspring.

Simulated annealing approaches can be understood as specialized genetic algorithms: Here, only single-individual populations ($|\Pi| = 1$) are considered and that's the reason why the creation of just one offspring per generation (by means of a mutation) is suggestive. Different to genetic algorithms, the selection of ancestors is omitted and the reduction operator is based on a boltzmann probability [1].

The following subsections give a comprehensive explanation of all major operators which are used in a realization of the algorithm in Fig. 2 for the GBPP. As mentioned in the introduction, the discovery of high quality solutions strongly depends on the structure which is used to describe (encode) and manipulate existing solutions. For this reason, a description of the encoding mechanism has to precede the explanation of the genetic operators mutation and crossover. For more details concerning both, the encoding and the operators see [9].

2.1. Encoding

Different to most other heuristics genetic algorithms do not manipulate the outward appearance of a solution (the *phenotype*) itself but an appropriate encoding (the *genotype*). The major motivation for this is given by the fact that phenotypic representations usually do not reveal the characteristic features of the solutions and consequently do not enable the determined improvement of these characteristics.

Unlike the approaches mentioned in the introduction an appropriate encoding for the GBPP should immediately reflect the structure of a solution, but still be flexible enough to enable its manipulation by certain genetic operators. In addition, if already the representation can ensure that the resulting packing scheme is guillotineable, time consuming feasibility tests or repair operators may be omitted. As a consequence, not the absolute positions of rectangles, but their relative arrangement within the bin should pass into the genetic description.

An encoding mechanism which satisfies the above demands takes advantage of the slicing tree structure of guillotineable packing schemes. A similar structure is used for the creation of floorplans during the VLSI design cycle [17].

The recursive bipartition of the bin with guillotine cuts results in a slicing tree structure, whose leaf (not split) nodes correspond to the rectangles to be packed. All interior nodes define the hierarchy of guillotine cuts needed to create the packing scheme. Due to the scheme's orthogonality, each cut line either runs horizontally (parallel to the lower border of B) or vertically (orthogonal to B's lower border). This type of a cutline is added as a label to each non-leaf node of a slicing tree. A label 'v' ('h') is used to denote a vertical (horizontal) cut. Fig. 3 shows an example of the slicing tree structure. Note that the partitioning process may leave some 'unused' areas within the bin which are not covered by any rectangle.

The actual encoding which finally is applied in the genetic algorithm to solve the GBPP represents slicing trees in a different way. Preorder traversal of the tree and output of all entries at

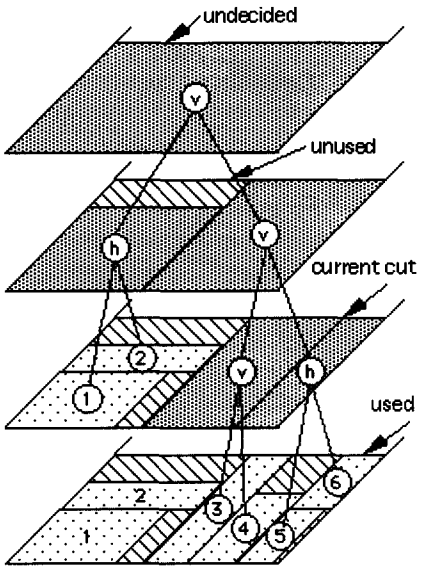

Fig. 3. Slicing tree structure.

the nodes being visited, leads to a string over the alphabet $\{1, 2, \ldots, n, 'h', 'v'\}$. The string has length $2n-1$, if n rectangles are to be packed. In addition to each leaf node a '$\rightarrow$' resp. '$\uparrow$' sign specifies whether the orientation of the corresponding rectangle is upright ('$\uparrow$') or not ('$\rightarrow$'). In Fig. 4 the resulting preorder string for the slicing tree of Fig. 3 is depicted.

Besides its ability that just the existence of a preorder string ensures the 'guillotineability' of a solution, the selected encoding mechanism offers several additional advantages:

First, subsequent to random variations, simply parsing the string reveals whether the string represent a valid slicing tree. Second, complete subtrees are included within preorder strings as non-disrupted substrings. As a consequence, these components which a solution space analysis has found to be the major building blocks of good solutions, can be grasped compactly and manipulated efficiently. Finally, the slicing tree structure allows to ignore complete subtrees, when a preorder string has to be evaluated. Local modifications within a subtree (e.g. the rotation of rectangle 1 in Fig. 3) merely affect each extensive subtree. There is no need to repeatedly analyze all remaining subtrees (e.g. the subtree covering rectangles 3–6 in Fig. 3) after such local modifications. This permits very effective evaluation procedures.

v	h	1	2	v	v	3	4	h	5	6
		→	→			↑	↑		↑	↑

Fig. 4. Preorder string of the slicing tree in Fig. 3.

2.2. Mutation operators

Similar to nature, the task of mutation in a genetic algorithm is to effect random variations of the genetic material. Mutation enables the (re-)creation of genes which are lost in the current population and which cannot be gained if only the existent material is combined.

In general, mutation aims at preserving the genetic manifold within a population of individuals. However, the schematic algorithm of Fig. 2 (and the simulated annealing approaches, too) apply mutation operators to perform a local search, alternatively to the crossover operator. The effect of mutation is locally restricted since each offspring inherits most of its genetic material from its ancestor, and thus the distance (measured with a proper metric in the solution space) between the two individuals keeps small.

With the exception of operator MUT4 all mutation operators proposed subsequently preserve the overall structure of the input string. The operators behave almost intelligent as they guarantee the creation of valid preorder strings. Thus, testing the feasibility of all non-MUT4-based offsprings can be omitted.

To mutate preorder strings the following operators are on offer. They increase or adapt the functionality of some operators which are proposed by Wong et al. [17] for a simulated annealing algorithm.

MUT1: swap two adjacent subtrees.

MUT2: invert each cutline's orientation label within a non-empty substring ('h' becomes 'v' and vice versa).

MUT3: invert each rectangle's orientation label within a non-empty substring ('$\rightarrow$' becomes '$\uparrow$' and vice versa).

MUT4: swap an adjacent pair of a rectangle- and a cutline-item.

Fig. 5 presents examples for an application of

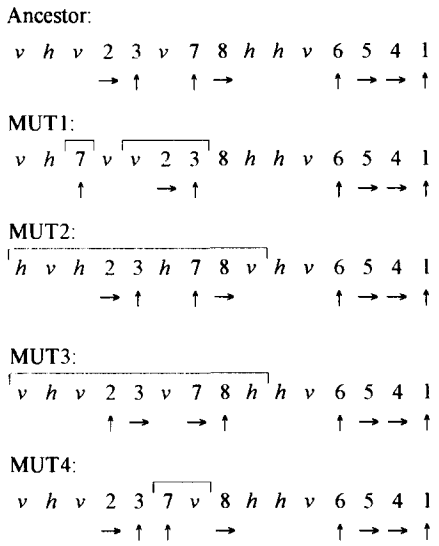

Fig. 5. Mutation operators for preorder strings.

the operators MUT1-MUT4 on a given ancestor string. Each modified substring is displayed by a bracket.

An additional operator MUT5 combines the functionality of MUT2 and MUT3: In a non-empty substring the orientation entries of both, each line of a cut and each rectangle are inverted. This corresponds to the rotation of a partial packing scheme as a whole for 90°.

Stochastic effects influence the mutation in two ways: First, a random choice is made to determine the operator which shall be applied and, second, the substrings or items to be mutated are randomly selected, too.

2.3. Crossover operator

To simulate the sexual propagation of individuals a recombination operator is included into genetic algorithms. Its task is to combine the genetic material of usually two parents in order to produce an offspring which inherits certain good characteristics of its ancestors. As recombination commonly implies the crosswise junction of segments from both parents' gene strings, the term *crossover* is often used, too.

The major motivation for doing crossover is the realization of some kind of heredity. Features which were once adapted by ancestors shall be transmitted from parents to their offsprings. Thus, knowledge of preceding generations might directly be passed to offsprings and there is no need for the descendants to repeat the learning processes of their parents. The most important question arising in this context is: What makes up the 'good characteristics' of packing schemes which are worth to be inherited by an offspring?

For this purpose, a solution space analysis for the GBPP was intended to reveal the similarities of high quality packing schemes [9]. A comparison of sets of locally optimal packing schemes indicates that

- compact, small-sized partial packing schemes form the knowledge of individuals. As a consequence, merely complete partial structures should be preserved during crossover. Referring to the encoding mechanism mentioned above this strategy corresponds to the passing of substrings (subtrees) as a whole. Thus, no separation of the slicing tree structure and the labels of terminal nodes is made.
- the size of each subtree inherited by an offspring should remain small; at most 4 rectangles are included. Instead of transmitting more complex subtrees several small-sized substrings should be transferred simultaneously.

According to these results, an innovative crossover operator will be introduced in the following. The operator is devided into three consecutive steps:

First, all those subtrees which shall be transmitted to the offspring have to be determined from both parent encodings. This choice is guided by the basic principle to discover as many distinct partial solutions within both parents as possible in order to pass these sub-solutions without any modifications to the offspring. Referring to the preorder string encoding these distinct partial solutions correspond to subtrees which do not have any rectangle in common. As suggested by the solution space analysis, only subtrees which contain at most 4 rectangles and achieve a packing density of at least 90% at their root node are candidates for this selection. The packing density at a node k is defined as the ratio of the total used area to the rectangular area required for k (including waste). See Fig. 3 for an example.

Distinct subtrees are gained by a heuristic solving an Independent Set Problem [7] which is defined among all suitable subtrees from both parents. Two subtrees may belong to an independent set if they have no rectangle in common.

Subsequent to the decision on all subtrees which shall be transmitted (those are displayed by brackets in the dad- and the mom-string of Fig. 6), a slicing tree structure containing just all inherited subtrees is built. For this, the dad-string is reduced as far as it only includes those substrings which are contributed to the offspring. All remaining rectangles are deleted from the former dad-string together with their particular ancestor node in the slicing tree. After this, the resulting shortened dad-string forms the basis for the creation of an offspring. First, every subtree which was selected for inheritance from the mom-string is separately inserted into the shortened string. The sequence of insertions is determined randomly. Each of these insertions requires the inclusion of a new cut line to combine the inherited substring with the current string, too. All feasible positions for inserting a substring are analyzed tentatively and are given a priority value by an evaluation function. Finally, the one with best priority value is accepted. Here, just partial evaluations of incomplete substrings are performed with an analogous evaluation procedure which is proposed in the following section.

In Fig. 6 an example for the crossover of two parent preorder strings is presented.

As usually not each rectangle is covered by any inherited subtree, finally an insertion of single rectangles has to complete an offspring if necessary. For this, the same insertion strategy as mentioned above is applied to all missing rectangles.

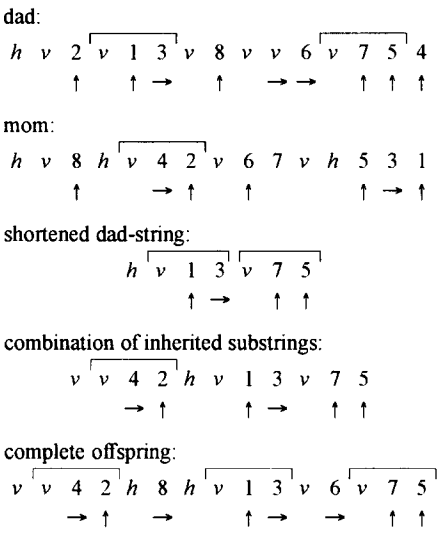

Fig. 6. Crossover operator for preorder strings.

2.4. Further operators

Besides mutation and crossover, the schematical algorithm in Fig. 2 applies some additional operators. Their operation very briefly is explained in the present section. A more detailed description can be found in [9].

The slicing tree structure offers a very efficient evaluation procedure for preorder strings (cf. Section 2.1, too). These trees specify the relative instead of an absolute position of rectangles. As a consequence, local changes within the tree, just require to (re-)evaluate those subtrees where these changes actually have taken place. All remaining, not modified subtrees keep their former shapes and extensions.

In general, the evaluation of a slicing tree requires a bottom-up pass throughout the tree. Given the horizontal extension $\mathrm{wd}(i)$ and the vertical extension $\mathrm{ht}(i)$ of each rectangle i within a packing scheme, analogous values can be calculated at each interior vertex of a slicing tree, too. Due to the guillotineability, each interior node again represents a partial packing scheme of rectangular shape. To evaluate the extensions at a vertex k (whose descendent nodes are denoted by k_1 and k_2; k_1, k_2 being already evaluated or remaining unchanged), the following formulas are applied:

$$\mathrm{wd}(k) = \begin{cases} \mathrm{wd}(k_1) + \mathrm{wd}(k_2) \\ \quad \text{if } k \text{ is of type '}v\text{'}, \\ \max\{\mathrm{wd}(k_1), \mathrm{wd}(k_2)\} \\ \quad \text{otherwise}, \end{cases}$$

$$\mathrm{ht}(k) = \begin{cases} \mathrm{ht}(k_1) + \mathrm{ht}(k_2) \\ \quad \text{if } k \text{ is of type '}h\text{'}, \\ \max\{\mathrm{ht}(k_1), \mathrm{ht}(k_2)\}, \\ \quad \text{otherwise}. \end{cases}$$

A solution is called *feasible* if $\mathrm{wd}(k) \leq W$ holds for each vertex k of a slicing tree. Then, the value of the criterion function equals $\mathrm{ht}(\tilde{k})$ at the tree's root $\tilde{k}$. As each decrease of $\mathrm{ht}(\tilde{k})$ must be reflected by a fitness increase, the actual fitness value for an individual has to be in reverse proportion to its $\mathrm{ht}(\tilde{k})$ value.

Considering preorder strings, a bottom-up evaluation of individuals can be simulated if the string is passed backwards. For each leaf node l its $\mathrm{wd}(l)$ and $\mathrm{ht}(l)$ values are pushed as a single item onto a stack. To calculate the shape at an interior node k both topmost stack entries have to be used and replaced with an item containing the values for the current vertex k. For this calculation the above formula is applied.

In order to guide the selective decisions within a genetic algorithm more directly into the desired direction, an artificial distinction between equal valued individuals is made. For this, the average packing density at all non-leaf nodes affects the fitness of individuals, too. If two individuals yield identical evaluations $\mathrm{ht}(k)$, that one which has the higher average packing density gets the better fitness value.

The tasks of the hillclimbing operator are twofold: First, the hillclimber aims at improving the fitness of each recently mutated or recombined offspring and, second, it shall transform infeasible into feasible solutions. To reach these purposes, the hillclimber acts as an iterative improvement strategy. It combines three component operators for the preorder string manipulation.

Two of the hillclimber operators, iterat_MUT4 and iterat_MUT5, denote variations of the corresponding mutation operators MUT4 resp. MUT5. Instead of selecting a single application within the string, these iterative variants systematically analyze all feasible positions and tentatively perform the corresponding changes at each suitable location. All manipulations which improve the fitness are accepted, all uneffective tries are rejected and the former string is reconstructed.

In addition, a replacement operator is incorporated into the hillclimber to rearrange single rectangles within the bin. At each application a move is tried for each rectangle. For this, the rectangle is removed from the string first, and all suitable positions are examined for its reinsertion. Changes which improve the fitness or which displace overlapping rectangles are accepted. If no improvement is found, a rectangle keeps its former position in the string.

To realize a hillclimbing strategy the operators iterat_MUT5, 'replace' and iterat_MUT4 are applied in the given order to a string until none of these operators obtains a fitness improvement.

By means of the selection and the reduction operators the genetic algorithm's search is guided into the most promising parts of the solution space. Thus, their decisions should be quality driven. The selection operator determines a single individual from the current population Π according to a linear ranking strategy [4,8]. And the reduction operator permits the survival of the $|\Pi|$ fitness-best individuals from $\Pi \cup \Pi_\Omega$ as long as their genetic material shows a sufficient difference. At least 3 'distinct' [9] subtrees of a hereditable size are required unless a preorder string is separated due to similarity reasons.

The initial population is gained by generating valid preorder strings at random.

2.5. Results of sequential algorithms

The comparison of different heuristics for the Bin Packing Problem in the course of this section is going to reveal that the genetic algorithm, even in its sequential version, produces solutions of superior quality. Although longer runtimes may be needed to calculate these solutions, reasonable fields of applications can be suggested for the use of sequential genetic algorithms.

To compare the algorithms they all were applied to 12 randomly generated instances including between $n = 25$ and $n = 60$ items to be packed. All borderlengths are uniformly distributed within the interval of 1–40 units. On the average, the variances in the ratio of all rectangles smaller to their larger side reach "medium" values of 3–7 per instance. The bin's width is 100 units. Each instance was repeatedly solved for at least 10 times and thus, all forthcoming results are averaged over their repetitive runs.

In order to ensure a fair comparison between the different approaches, all iterative strategies

were offered an identical amount of tentatively generated packing schemes. Table 1 presents these maximum numbers of trials which vary for different values of the problem size n. Having reached these limits the algorithms terminate and just the best packing scheme which was calculated so far is considered for most analyses. All values in Table 1 were gained during runs of the simulated annealing algorithm whose termination criterion is not affected by the amount of completed trials [1].

Very briefly, the comparison of different packing heuristics includes the following strategies:

- LO: a non-iterated greedy heuristic similar to Coffman et al.'s level algorithms [3]. To fill a certain level, all suitable rectangles are selected according to a first-fit strategy. Placing these rectangles with descending heights produces rectangular areas inside each level. One of these areas can be filled without destroying the guillotineability of a packing scheme. Thus, the LO-algorithm improves a pure level-oriented approach. Several (10) static sequences of the rectangles are tried, finally accepting the one which leads to the best solution quality.
- MM: a non-iterated greedy strategy which generates packing schemes by gluing together pairs of rectangles. Each step of the algorithm consists of performing a maximum matching algorithm which determines pairs of rectangles to be combined. Candidate pairs should ideally fit together (i.e. produce hardly any loss) and should result in an additive width of no more than W. Instead of both rectangles which were glued together, a single rectangle serves as their representative within the problem instance. This new rectangle is of minimal shape, large enough to cover the union of both elementary items. The algorithm terminates if only a single rectangle remains.

Table 1
Maximum number of trials for all iterated strategies

n	Packing schemes
25	2500
35	3000
45	3500
60	4250

Table 2
Average quality of the solutions generated by different algorithms for the GBPP; absolute distance to known best solution.

No./n	LO	MM	LO.it	Bok	SA	GA
1/25	10	8	4.4	4.2	2.7	1.8
2/25	4	5	3.4	3.0	2.5	1.8
3/25	7	5	4.7	4.6	3.2	2.1
4/35	9	7	5.9	5.8	4.9	3.8
5/35	8	8	5.1	4.8	3.6	2.3
6/35	11	8	4.6	4.4	3.6	2.0
7/45	13	11	7.9	7.2	5.1	3.3
8/45	13	10	7.2	6.6	4.4	3.2
9/45	12	10	6.4	6.6	4.1	3.2
10/60	13	10	9.9	8.1	6.5	4.6
11/60	21	17	12.4	11.8	8.6	6.8
12/60	15	20	13.2	12.3	5.8	5.7
avg.	11.3	9.92	7.09	6.62	4.58	3.38
%	100	88	63	58	40	30
%	335	293	210	196	136	100

- LO.it: an iterated version of the LO-strategy. Sequences of rectangles are generated randomly and transformed into packing schemes by applying the LO-algorithm.
- Bok: a random search strategy which randomly generates valid preorder strings.
- SA: a simulated annealing algorithm as indicated above based on a preorder encoding. The initial temperature of 100.0 was reduced every 10–15 trials by a factor of 0.95. These values were determined empirically.
- GA: a genetic algorithm as described above. Each population contains 10 individuals ($|\Pi| = 10$) and the variables max.offs and mutation.prob equal 5 resp. 30. Again, the values stem from some empirical experiments.

In all algorithms each generated packing scheme is optimized by a hillclimbing operator (see Section 2.4).

Table 2 presents the results when applying the proposed algorithms to the 12 instances mentioned above. For each instance the calculated average solution quality is given by its distance (in units of length) to the value of the known best solution. Except for the two deterministic strategies LO and MM, all results are averaged over at least 10 repetitive runs.

The results of Table 2 indicate that merely modest improvements are reached when simple iterated heuristics, like LO.it and Bok, instead of non-iterated algorithms are applied. Their solutions are just 40% closer to the known best quality. Thus, simply iterating greedy algorithms does not provide reasonable solutions for the GBPP (LO.it and Bok try 2500–4500 as many packing schemes as the LO and the MM strategy). More significant improvements can be achieved by the more sophisticated heuristics simulated annealing and genetic algorithm. In total, the genetic algorithm generates the best solutions, which, on the average of all 12 instances, come 1.2 units of length ($\approx 36\%$) closer to the known best solution compared to the simulated annealing strategy.

To reveal the overall quality of the generated solutions Table 3 presents a different interpretation of the average (mean of all instances and their repetitions) values from Table 2. The absolute distances presented in the 'avg.' row of Table 2 now are recalculated into a relative distance depending on the quality of the known best solutions.

Nevertheless, it is just a pure comparison of solution qualities which is presented in Tables 2 and 3. The different runtimes needed to calculate these solutions are not taken into account. A combined measure, reflecting both, the runtime of a heuristic H as well as the quality of H's solutions, can be defined as follows:

$$\text{cost}(H) := (\text{time}(H) * c_{\text{cpu}}) + (\text{waste}(H) * c_{\text{mat}} * f).$$

Here, time(H) denotes the average runtime which was needed by algorithm H to solve the 12 instances and waste(H) represents the average amount of waste spent by H's packing schemes. The costs c_{mat} per unit area of waste and the repetition frequency f for a scheme, influence the proportion which a single packing scheme's waste contributes to the overall costs. The frequency f denotes the amount of realizations which are intended for a calculated packing scheme.

Table 3
Average quality of the solutions generated by different algorithms for the GBPP; relative distance to known best solution

avg.	LO	MM	LO.it	Bok	SA	GA
rel. dist. (%)	6.70	5.88	4.20	3.92	2.71	2.00

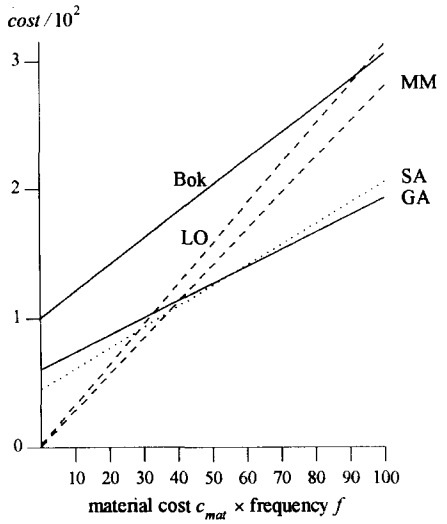

Fig. 7. cost(H) courses for different heuristics H.

Since all runtimes were measured on identical computers, the costs c_{cpu} for CPU time are normalized to 1. As a consequence, the courses of cost(H) values can be depicted by straight lines as c_{mat} can be measured as a multiple of c_{cpu}. For the different heuristics from Table 2, Fig. 7 shows the resulting costs versus the material costs c_{mat} times frequency f.

When looking at Fig. 7, especially the intersections of lines are of major interest. For instance, if f equals one and a packing scheme is realized just once, the ratio $c_{\text{mat}}/c_{\text{cpu}}$ must exceed 40 until simulated annealing and genetic algorithm produce less expensive solutions than all other heuristics. For cheaper materials the non-iterated heuristics should be applied. However, if the frequency f increases to a value of x, this break-even point is reduced to $40/x$. As a consequence, the use of sophisticated heuristics becomes rapidly profitable if the calculated packing schemes are realized a number of times.

3. Parallel genetic algorithm

A major motivation for the use of genetic algorithms actually is their inherent parallelism. In contrast to simulated annealing, the implementation of genetic algorithms on multiprocessor computers is quite simple and straightforward. Either such a parallelization scheme aims at speeding up the calculations or it can be applied in order to achieve a vigorous improvement of the generated solutions. This latter aspect is mainly dealt with during the current section.

When seeking for an example, how to parallelize genetic algorithms, again nature itself provides it. The (spatially) separated development of species in nature can be simulated when several sequential algorithms are executed in parallel, each working on a local, discrete sub-population. In irregular intervals single individuals explicitly or implicitly migrate between sub-populations. For this, a logical neighborhood has to be defined among them inside which migrations may occur. Overlapping neighborhoods of neighbored processors enable the propagation of genetic material even to all distinct populations. The degree of overlap then determines the degree of isolation of local populations.

Two different models of a nature-like parallelization of genetic algorithms are proposed in literature: *migration* and *diffusion*. Both strategies have been realized in several variations, yet.

In general, migration models [9,12,15] are based on a quite loose coupling of their component algorithms. Just subsequent to each lapse of a fixed amount of generations, single selected individuals may migrate to neighbored populations. In the meantime, a conventional genetic algorithm is performed. Incoming migrants are included into the local populations. Either copies of current individuals (*immigration*-model) or original individuals (*emigration*-model), which might have partially spread their genetic material before, are allowed to migrate to neighbors.

In the diffusion models [10–12] there is no need for an explicit migration of individuals. All individuals are considered to move freely within their neighborhood. Thus, the selection operator is no longer limited to the local individuals, but gets access to all neighbored populations, too. Furthermore, neighbored individuals can be accessed at any time without any limitations. Different to the migration models, only local offsprings of neighbored individuals (and no migrants) are included into the local populations.

To implement these parallelization schemes, some kind of interprocess communication (e.g. a mail box) has to be established. As a major guideline idle times due to communication delays should be prevented. Our realization of the proposed schemes fully preserve the asynchronisity of each component algorithm, as no synchronized exchange of individuals is provided. Instead, the receipt of migrants is treated like an unforeseen event, for which no waiting time is spent. For details concerning the implementations of the parallel genetic algorithms, again see [8].

3.1. Results of distributed algorithms

Several variants proposed in the literature were applied to distribute the genetic bin packing algorithm. Due to space limitations only results for the most successful emigration, immigration and diffusion models will be presented here.

All implementations were done on a unique hardware platform, a transputer-based multiprocessor system containing up to 64 computing nodes (processors). This parallel computer belongs to the MIMD (Multiple Instructions Multiple Data) machines with distributed memory. A single network topology was used to interconnect all processors for all of our experiments. This topology was proposed by Mühlenbein et al. [11] and looks like a circular ladder. Across two subsequent rungs a further diagonal connection is established. Then, the four processors which are physically adjacent to each processor P make up P's logical neighborhood for the exchange of individuals. On each processor exactly one nodal genetic algorithm is resident.

The problem instances from Section 2.5 again serve for test purposes in the parallel case. Now, each instance was solved in five repetitive runs.

Table 4 compares four parallel genetic algorithms to solve the GBPP with 32 and 64 proces-

Table 4
Average quality of the solutions generated by the different parallel genetic algorithms

Strategy	Processors	Distance	Speedup
isolation	32	2.03	7.16
migration_A	32	1.35	6.95
migration_B	32	1.45	6.33
diffusion	32	1.70	6.36
isolation	64	1.68	9.19
migration_A	64	1.02	11.04
migration_B	64	1.17	7.21
diffusion	64	1.60	8.02

sors. Each local population still includes 10 individuals. All other parameters which are not explicitly mentioned in the following keep their values from the sequential version. Just individuals with an above-average fitness are allowed to migrate in both variants, migration_A and migration_B. Like the selection of parents, all migrants are randomly chosen from all suitable individuals within the local populations. With the migration_A strategy an exchange of individuals takes place every 5 generations and each time 2 individuals are transferred (as a copy) to a single (random) neighbor. This immigration model was proposed by Tanese [16].

A more seldom exchange of individuals at a frequency of 10 generations is realized in the migration_B algorithm. Again two migrants leave at a time, but now they are (randomly) spread amongst all neighbors. As migrants may not be duplicated but have to be original individuals, the migration_B strategy becomes an emigration algorithm.

For the diffusion model, the enlarged selection mechanism is only applied when determining the *mom*-individual. Each *dad*-string still is selected from the local population.

An 'isolated' algorithm is added for comparative purposes, too. As indicated by its name, this strategy describes a special parallel algorithm with isolated local populations. Neither an exchange of individuals nor the access to non-local individuals is provided. The algorithm just controls a set of non-interacting populations. Thus, it enables to evaluate the effects of communication within parallel genetic algorithms.

For 32 and 64 processors being involved, the results of Table 4 show the average distance (in units of length) to the known best solution and an average speedup, both values averaged over the 12 instances per parallel algorithm. In this context, the speedup defines the factor between the amount of generations needed by the sequential and by the distributed algorithm to produce the same solution quality as the sequential algorithm. This value gives a measure for speeding up the algorithms by parallelization.

The results of Table 4 reveal that each variant profits from the use of additional processing resources (sub-populations). With 32 as well as with 64 local populations all algorithms can improve their solutions which were reached on a smaller number of processors. However, the scale of improvement strongly depends on the parallelization scheme which is applied. Whilst both migration strategies already with 32 processors reduce the average height of their – sequentially generated – packing schemes for about 2 units of length, the diffusion algorithm hardly improves the quality which is generated by the non-communicating isolation model. As the speedup for all models is roughly equal, the weakness of the diffusion strategy is caused due to a lack of improvements found in the later part of the algorithm.

In total, the migration_A version with 64 populations reaches the best qualities as it generates solutions that differ not more than one unit from the known best solutions. This corresponds to an average waste of approximately 2%.

All speedups do hardly differ and fail to come close to the amount of processors being involved. However, linear speedups cannot be expected when calculating them in the way mentioned above. For example, in order to achieve a linear speedup just 13.28 ($= 850/64$) generations remain for a parallel algorithm with 64 processors to calculate the recommended solutions, for which the sequential algorithm may spent 850 generations (if n equals 60). Note that both, the sequential and the parallel genetic algorithm produce 5 offsprings per component population within each generation (cf. Table 1).

Reasons for the observed behavior of the algo-

rithms, especially for the minor quality of the diffusion model, become obvious if the genetic manifold which is represented by the individuals within a population is analyzed [9]. With the diffusion model the genetic manifold of neighbored populations hardly decreases in the course of the generations. This effect is caused by the transmission policy which is applied: In the diffusion model an exchange of genetic material is not realized by an exchange of complete individuals, but happens when single subtrees from neighbored individuals are inherited.

The next section is going to reveal that the efficiency of the crossover operator depends on the genetic similarity of both parents. Due to the transmission policy of both migration models, a controlled and slow reduction of the genetic manifold within the local populations can be achieved. However, a premature and strong convergency is not useful, anyway. In the diffusion model, no reduction of the genetic variability takes place at all. Thus, the crossover operator gets no support for its attempts to create high quality offsprings.

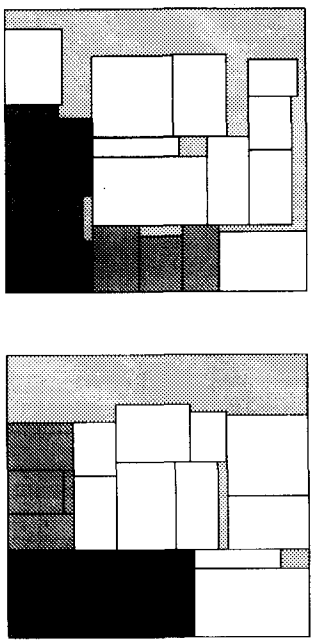

Fig. 8. Packing schemes without (above) and with (below) meta-rectangles.

4. Meta-rectangles

4.1. Principles

The special structure of the solutions in the GBPP finally offers a quite interesting extension. Due to the guillotineability of packing schemes each grouping of neighbored rectangles forms a partial arrangement (a *room*) with still a rectangular shape. This feature enables the creation of so-called meta-rectangles which freeze the layout of a certain room.

Thus, each meta-rectangle lays down the relative arrangement for several rectangles within a room. In the following such a room is represented by a single meta-rectangle of the corresponding size (cf. Section 2.4). A meta-rectangle can be treated like any 'normal' rectangle, it can be replaced, rotated or grouped with other (meta-) rectangles. The relative layout of all inclosed rectangles, however, is not changed by any of these modifications. Fig. 8 illustrates this fact: Two packing schemes are presented, one scheme where no meta-rectangles are composed yet, but two partial packing schemes are marked (above). In the second scheme (below) both marked areas have been frozen to meta-rectangles and rearranged to a new solution. Note that in the packing schemes the ingredients of both emphasized partial layouts are not modified. Just the waste inside the rooms is covered by the meta-rectangles, too.

The creation of meta-rectangles fixes hyperplanes of existent solutions. A reduction of the problem's complexity is entailed, as in the place of all component rectangles now just single meta-rectangles have to be arranged to packing schemes. As a consequence, a speedup of the algorithm should be affected.

Furthermore, even the search is guided with the creation of meta-structures. Subsequent to each composition of meta-rectangles, the recombination operator shall ensure that all offsprings include these frozen hyperplanes. (This can be realized if the substrings of all current meta-rectangles a-priori are selected to be inherited by an offspring.) With respect to the solution space, the

neighborhood of the fixed hyperplanes then is analyzed in more detail, whereas other regions are just covered with single individuals. As each meta-rectangle may establish waste within a packing scheme (cf. Fig. 8), it seems reasonable to freeze only those partial layouts which cause hardly any waste and whose shape is still flexible enough to be grouped with other rectangles.

As stated before, the composition of meta-rectangles varies the problem instance which is to be solved. However, a simple and obvious proof shows that each packing scheme including meta-rectangles still is a valid solution to the original problem, as long as all meta-rectangles are created in the way mentioned above. In this sense a problem formulation including meta-rectangles will be called *abstraction* in the following.

It should be mentioned here that a freezing of partial solutions is not limited to the Bin Packing Problem. For the well-known Traveling Salesman Problem (TSP) a similar approach appears to be suitable. Instead of meta-rectangles then meta-cities are established each of which represents a partial path within a tour. Mühlenbein's solution space analysis for the TSP [12] supports this expectation: Local optimal tours with s cities include just $O(s)$ different edges and the average distance (i.e. amount of distinct edges) between two tours is just $\frac{1}{3}s$. Such a similarity of local optimal solutions can be exploited by the creation of meta-structures which reduce degrees of freedom, but which enable an efficient construction of similar solutions.

4.2. Realization

To convert the proposed concepts into the genetic bin packing algorithm, two additional strategies for the composition and the decomposition of meta-rectangles are required.

For the composition of meta-rectangles first a suitable (cf. Section 4.3) pattern-individual is selected from the population. This individual serves as a model for the forthcoming creation of meta-structures. Then, similar to the crossover operator, all hereditable, high quality subtrees within the pattern-individual are determined. In order to prevent a too strong reduction of the problem's complexity, not each of the selected subtrees actually is transformed into a meta-rectangle. An additional parameter, the *threshold*, is introduced instead, which serves as a lower bound for the problemsize of the current abstraction. Just as long as the problemsize remains above the threshold, more subtrees from a pattern-individual are fixed to meta-structures. In case of conflicts, a random choice determines all meta-rectangles out of the selected subtrees.

The threshold parameter is reduced after each lapse of a fixed amount of generations towards a minimal value. Thus, the portion of meta-rectangles inside an abstraction increases. The structure of the genetic algorithm changes from combining single rectangles (hyperplanes of dimension 1) to the (re-)arrangement of partial solutions without affecting their internal structure. On decreasing threshold values the genetic algorithm then operates on a more abstract level.

The opposite approach is thinkable, too: Starting with a high portion of meta-rectangles and stepwise lowering this level. However, we did not apply this last strategy as it does not reflect the 'natural' approach. It seems more reasonable to combine single rectangles first and to prevent high quality blocks from being destroyed. According to the alternative strategy blocks of several rectangles (how are they created?) would be arranged first and probably destroyed in a later step when the blocks no longer are protected.

To prevent an unbalanced search in the neighbored regions of just single hyperplanes, the composition of meta-rectangles is repeated periodically. Each abstraction is valid only for a fixed number of generations (20 or 40 generations in the below experiments). Subsequent to their validity all current fixations are resolved which implies that all meta-rectangles are dissected into their component rectangles again. Then a renewed fixation of hyperplanes can take place on a newly selected pattern-individual. Thus, different fixations and consequently different abstractions are built at a variable frequency.

The crossover operator also has to be modified in order to enable the recombination of individuals which include meta-rectangles. All current meta-rectangles a-priori are selected to be inher-

ited by an offspring. Further subtrees from both parents may be transmitted as long as they do not include any rectangle which is already covered by a meta-rectangle.

4.3. Results with meta-rectangles

Not only in the sequential but also in all distributed variants of the genetic bin packing algorithm the selection of parents for crossover was done by a ranking strategy. It ranks all individuals according to their fitness values. For simplicity reasons, the application of this fitness ranking makes up a first approach to determine the pattern individuals if a model for creating meta-rectangles is required.

However, it was the major design principle for the crossover operator to combine as many partial solutions from both parents as possible. Since each offspring already includes the partial strings of all current meta-rectangles, a selection strategy appears to be reasonable which ranks the individuals according to their genetic conformity with the meta-structures. Now, those individuals whose genetic encoding shows great similarities to the meta-rectangles are preferably recombined. To measure this correspondence one has to determine the amount of conformable subtrees between the encodings being involved. Besides all meta-rectangles favorable parents very likely contain additional hereditable subtrees including rectangles which are not yet covered by any meta-rectangle.

This variant of a similarity ranking for the selection of pattern-individuals assigns the highest priority to the individual with the largest extent of conformable subtrees. The composition of meta-rectangles according to this criteria ensures the existence of other individuals whose genetic material almost ideally supplies to the current meta-structures.

Table 5 presents the results when one of these selection strategies is applied in a sequential genetic algorithm where a composition of meta-rectangles is performed. The validity of each abstraction lasts 20 generations and the threshold is reduced stepwise from $15n/16$ to $7n/8$ resp. $6n/8$ for both selection variants. For this reduction, the initial problem size $\tilde{n} = 15n/16$ is decremented in equi-distant intervals during the whole run of the algorithm. Here n denotes the initial problem size and not the complexity of any abstraction. As before all values in Table 5 correspond to the average distance to the known best solution, averaged over 12 instances. The results of the former sequential algorithm without a construction of meta-rectangles are repeated in the 'no metas' column.

Table 5
Comparison of two different selection strategies; average solution qualities

	No metas	Fitness ranking		Similarity ranking	
		$7n/8$	$6n/8$	$7n/8$	$6n/8$
avg.	3.38	4.12	4.17	3.61	3.55
%	100	122	123	107	105

Regardless of the minimal threshold value the similarity ranking outperforms the pure fitness-based ranking. With the similarity ranking the mean solution quality comes much closer to the results of the former sequential algorithm without fully reaching these qualities.

The results of Table 6 are capable of explaining the different behavior of the selection strategies. For individual instances no. 1, 6 and 7 the average quality of all mutation-based offsprings ('mutate' row) and all recombination-based offsprings ('Xover' row) are depicted when either no meta-rectangles are used or a fitness ranking is responsible for the selection of parents and the pattern-individuals. Again all these values are presented by the usual distance. In addition, the 'single' row contains the (absolute) number of rectangles which have to be inserted into the

Table 6
Statistical results when solving three instances by two of the algorithms which are responsible for the results in Table 5

	No metas			Fitness ranking		
	No.					
	1	6	7	1	6	7
mutate	4.4	4.8	6.7	4.8	4.4	6.2
Xover	5.6	5.5	9.9	9.6	9.0	11.5
single	4.7	5.2	8.0	9.40	15.3	16.6

offsprings' encodings during the last step of crossover.

Despite the different selection strategies, the quality of all mutation-based offsprings is fairly equal for corresponding instances. As a matter of fact the crossover-based offsprings show quality differences of 2–4 units of length. The reasons for this effect can be taken from the 'single' row. With the fitness ranking the amount of subsequently and singly inserted rectangles at least is doubled compared to the former algorithm without meta-rectangles. Thus, the transmission of complete partial string-encodings plays a minor role within the crossover operator but stochastical influences increase.

To what extent can the complexity of a problem be reduced by a composition of meta-rectangles without affecting the quality of the generated solutions? And does the tradeoff between the solution quality being obtained and the reduction of runtime due to an increased construction of meta-rectangles actually occur?

In order to clarify these questions several sequential genetic algorithms applying the composition of meta-rectangles have to tackle the 12 test instances from Section 2.5 again. Common to all strategies each abstraction is valid for 20 generations and the initial threshold equals $15n/16$. The final threshold values differ from $7n/8$ to $4n/8$. For these different algorithms Fig. 9 shows their average runtimes as well as the average

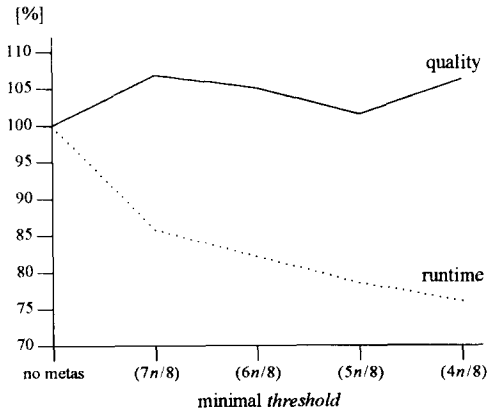

Fig. 9. Average runtime and solution quality for several genetic algorithms with a composition of meta-rectangles.

Table 7
Best solution qualities for algorithms with and without creating metarectangles. 1, 32 and 64 processors being applied

	No metas			With metas		
	dist.					
	1	32	64	1	32	64
abs.	3.38	1.35	1.02	3.29	1.27	0.93
rel.	2.00	1.01	0.60	1.95	0.75	0.55

quality of the generated solutions. All values are given in percent of the corresponding results of an algorithm which disclaims the creation of meta-rectangles.

As expected, the runtimes continuously decrease when the minimal threshold values are lessened, i.e. if the final problemsize of the abstractions is reduced. Despite these different runtimes the mean quality of the generated solutions is hardly effected, anyway. All variants achieve solutions which merely slightly differ from the comparative value and the $5n/8$-strategy almost reaches that result.

The use of the meta-rectangle concept even in parallel, migration-based versions demands two algorithmic extensions: First, each sub-population generates local meta-structures of its own. Thus, the full asynchronisity of all populations is preserved and each local population performs a search in almost different parts of the solution space. Second, to enable the inclusion of migrants (which probably contain meta-structures which differ from the local ones) all meta-rectangles within a migrant's genetic encoding must be dissected into their component rectangles before a migrant leaves.

For the results of Table 7 migration_A algorithms of this kind were applied. The validity of each abstraction was enlarged to 40 generations and the threshold was reduced from its initial value of $15n/16$ to a final value of $6n/8$. Then, independent of the number of processors, all algorithms which follow the concept of meta-rectangles even improve the former best solutions (cf. Tables 2–4) which were generated by a non-meta-rectangle variant. Again, all results denote the average distance (relative and absolute) to the known best solution, averaged over 12 instances.

It is worth to be noted that the mean quality which is generated by the 64-processor version now differs for less than one unit of length from the known best solution.

In addition to these modest quality improvements the reductions of the runtime from Figure 9 still remain valid for all parallel versions of the algorithm, too.

5. Summary and conclusions

This paper introduces a quite effective parallel genetic algorithm for a constrained Bin Packing Problem, which demands a guillotine restriction for valid packing schemes.

Starting from a problem-specific encoding which represents the essential structure of a packing scheme by a binary tree, mutation and crossover operators are formulated to manipulate these structures. The major motivation for the crossover operator is to combine as many partial solutions (subtrees) from both parents as possible, thus preserving the main characteristic from both ancestors and providing a systematic continuation of the search.

Even in its sequential version the proposed algorithm was empirically proven to produce solution qualities which are superior to the packing schemes of several other heuristics. Further significant quality improvements were gained by applying distributed versions of the algorithm. Several parallelization schemes were implemented on a transputer-based MIMD machine with distributed memory.

The innovative concept of meta-rectangles was proposed in the last part of this paper. Each meta-rectangle temporarily freezes a hyperplane of an existent solution. Thus, the complexity of a problem is reduced and the algorithm's search can be guided into the most promising parts of the solution space. Not only this extension leads to an significant reduction of runtimes, but it also enables an improvement of the average best solution qualities which were generated by algorithms without a composition of these meta-structures.

As the concept of meta-rectangles is not limited to the Bin Packing Problem, further investigations are required to clear the benefit of the proposed strategies in the context of other optimization problems, for instance the Traveling Salesman Problem.

Acknowledgments

I am indebted to my Ph.D. supervisor Prof. Dr. Oliver Vomberger, for his fruitful comments and suggestions and to Peter Schwenderling who did a very excellent job in programming. This research was financially supported by the German Research Community (DFG) under grant number Vo 424/4-1.

References

[1] Aarts, E.H.L., and Korst, J., *Simulated Annealing and Boltzmann Machines*, Wiley, Chichester, 1989.

[2] De Cani, P., "Packing problems in theory and practice", Ph.D. Thesis, University of Birmingham, UK, 1979.

[3] Coffman, E.G., Garey, M.R., Johnson, D.S., and Tarjan, R.E., "Performance bounds for level-oriented two-dimensional packing algorithms", *SIAM Journal on Computing* 9/4 (1980) 808–826.

[4] Davis, L., *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, New York, 1991.

[5] Dowsland, K.A., "Some experiments with simulated annealing techniques for packing problems", Technical Report, University of Wales, Swansea, UK, 1990.

[6] Dyckhoff, H., "A typology of cutting and packing problems", *European Journal of Operational Research* 44/2 (1990) 145–160.

[7] Garey, M.R., and Johnson, D.S., *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, San Francisco, CA, 1979.

[8] Goldberg, D.E., *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, Reading, MA, 1988.

[9] Kröger, B., "Parallele genetische Algorithmen zur Lösung eines zweidimensionalen Bin Packing Problems" (in german), Reihe I, Technical Report No. 33/1993, Department of Mathematics and Computer Science, University of Osnabrück, 1993.

[10] Kröger, B., Schwenderling, P., and Vornberger, O., "Parallel genetic packing on transputers", in: J. Stender (ed.), *Parallel Genetic Algorithms: Theory and Applications*, IOS Press, Amsterdam, 1993, 151–185.

[11] Mühlenbein, H., "Parallel genetic algorithms, population genetics and combinatorial optimization", in: J.D. Schaffer (ed.), *Proceedings of the Third International Confer-*

ence on Genetic Algorithms, Morgan Kaufmann, Arlington, VA, 1989, 416–421.

[12] Mühlenbein, H., "Parallel genetic algorithms and combinatorial optimization", to be published in: *SIAM Journal on Optimization*.

[13] Macfarlane, D., and East, I., "An investigation of several parallel genetic algorithms", in S.J. Turner (ed.), *OUG-12: Tools and Techniques for Transputer Applications*, IOS Press, Amsterdam, 1990, 60–67.

[14] Olivera, J.F., and Ferreira, J.S., "An application of Simulated Annealing to the Nesting Problem", Paper presented at the 34th ORSA/TIMS Joint National Meeting, San Francisco, CA, 1992.

[15] Smith, D., "Bin packing with adaptive Search", in: J.J. Grefenstette (ed.), *Proceedings of an International Conference on Genetic Algorithms and their Applications*, Lawrence Erlbaum, Pittsburgh, PA, 1985, 202–206.

[16] Tanese, R., "Parallel genetic algorithms for a hypercube", in: J.J. Grefenstette (ed.), *Genetic Algorithms and their Applications: Proceedings of the 2nd International Conference on Genetic Algorithms*, Lawrence Erlbaum, Cambridge, MA, 1987, 177–183.

[17] Wong, D.F., Leong, H.W., and Liu, C.L., *Simulated Annealing for VLSI Design*, Kluwer Academic, Boston, MA, 1988.