A Hybrid Grouping Genetic Algorithm for Bin Packing

Emanuel Falkenauer

CRIF - Research Centre for Belgian Metalworking Industry
Industrial Management and Automation

CP 106 - P4
50, av. F.D.Roosevelt
B-1050 Brussels
Belgium

Phone: ++32 2 650 42 70
Fax: ++32 2 646 25 69
E-mail: efalkena@ulb.ac.be

Abstract

The Grouping Genetic Algorithm (GGA) is a Genetic Algorithm heavily modified to suit the structure of grouping problems. Those are the problems where the aim is to find a good partition of a set, or to *group* together the members of the set. The Bin Packing Problem (BPP) is a well known NP-hard grouping problem - items of various sizes have to be grouped inside bins of fixed capacity. On the other hand, the Reduction Method of Martello and Toth, based on their Dominance Criterion, constitutes one of the best OR techniques for optimization of the BPP to date.

In this paper, we first describe the GGA paradigm as compared to the classic Holland-style GA and the ordering GA. We then show how the Bin Packing GGA can be enhanced with a local optimization inspired by the Dominance Criterion. An extensive experimental comparison shows that the combination yields an algorithm superior to either of its components.

Key words: Grouping, partitioning, bin packing, genetic algorithm, solution encoding, dominance, reduction.

1. Introduction

0.1 The Bin Packing Problem

The Bin Packing Problem (BPP) is defined as follows ([Garey and Johnson, 79]): given a finite set O of numbers (the item sizes) and two constants C (the bin's capacity) and N (the number of bins), is it possible to 'pack' all the items into N bins, i.e. does there exist a partition of O into N or less subsets, such that the sum of elements in any of the subsets doesn't exceed C?

This NP-complete decision problem naturally gives rise to the associated NP-hard

optimization problem, the subject of this paper: what is the *best* packing, i.e. what is the *minimum* number of subsets in the above mentioned partition?

Being NP-hard, there is no known optimal algorithm for BPP running in polynomial time. However, [Garey and Johnson, 79] cite simple heuristics which can be shown to be no worse (but also no better) than a rather small multiplying factor above the optimal number of bins. The idea is straightforward: starting with one empty bin, take the items one by one and for each of them first search the bins so far used for a space large enough to accommodate it. If such a bin can be found, put the item there, if not, request a new bin. Putting the item into the first available bin found yields the First Fit (FF) heuristic. Searching for the most filled bin still having enough space for the item yields the Best Fit, a seemingly better heuristic, which can, however, be shown to perform as well (as bad) as the FF, while being slower. In the realm of Operations Research methods, [Martello and Toth, 90a] have recently introduced a more powerful approximation algorithm for BPP, discussed below.

0.2 The Grouping Problems

The Bin Packing Problem is member of a large family of problems, many of them naturally arising in practice, which consist in *partitioning* a set **U** of items into a collection of mutually disjoint subsets $\mathbf{U}_i$ of **U**, i.e. such that

$\cup \mathbf{U}_i = \mathbf{U}$ and

$\mathbf{U}_i \cap \mathbf{U}_j = \emptyset,\ i \neq j$.

One can also see these problems as ones where the aim is to *group* the members of the set **U** into one or more (at most **card(U)**) *groups* of items, with each item in exactly one group, i.e. to find a *grouping* of those items.

In most of these problems, not *all* possible groupings are allowed: a solution of the problem must comply with various *hard constraints*, otherwise the solution is invalid. That is, usually an item cannot be grouped with all possible subsets of the remaining items.

The objective of the grouping is to optimize a *cost function* defined over the set of all valid *groupings*. The following are just three examples[1] of well-known grouping problems, with the hard constraint a solution must comply with (where C is an arbitrary constant), and the cost function to minimize:

Problem	*Hard Constraint*	*Cost Function*
Bin Packing	Sum of sizes of items in any group<C	Number of groups
Workshop Layouting	Number of machines in any group<C	Total intercell traffic
Graph Coloring	No connected nodes in any group	Number of groups

As can be seen, the grouping problems are characterized by cost functions which depend on the *composition of the groups*[2], that is, where one *item* taken isolatedly has little or no meaning. The Grouping Genetic Algorithm (GGA) is a GA heavily modified to suit the particular structure of grouping problems. It has been presented as a method applicable to many different problems by [Falkenauer, 93] and [Falkenauer, 94]. A 'raw' GGA applied to the Bin Packing Problem (BPP) was proposed by [Falkenauer and Delchambre, 92].

In this paper, we report on a *hybridization* of the Bin Packing GGA of [Falkenauer and

[1] There are many more grouping problems of great practical importance, but their description would be too lengthy for the purpose of this paper.

[2] This is why we talk about *grouping* problems, rather than *partitioning* - we thus emphasize the importance of the groups, rather than the 'cuts'.

Delchambre, 92] with a recent OR technique for the BPP, the Dominance Criterion of [Martello and Toth, 90a].

The rest of the paper is organized as follows. In sections 2 and 3 we point out the respective weaknesses of standard and ordering GAs when applied to grouping problems. In section 4, we present the Grouping GA (GGA) which addresses those drawbacks. The ideas behind the Dominance Criterion of Martello and Toth are presented in section 5. Section 6 introduces the Hybrid GGA (HGGA) for BPP resulting from a 'marriage' of the two techniques. An extensive experimental comparison of the HGGA with a sophisticated branch-and-bound algorithm, the MTP procedure of [Martello and Toth, 90b], is carried out in section 7. Conclusions are drawn in section 8.

2. Standard GA Operators and Grouping Problems

In this section we examine the effects of the classic genetic operators on the structures relevant to the grouping problems. Application of a straightforward encoding scheme together with classic ([Holland, 75] type) genetic operators is the first route that has been taken in the GA literature treating grouping problems (e.g. [Van Driessche and Piessens, 92], [Ding et al., 92], [Jones and Beltramo, 91], [Von Laszewski, 91]). We will show why we think this is not the best GA approach for these problems.

2.1 The Encoding and Redundancy

Let us consider the most straightforward encoding scheme, namely one gene per item. For example, the chromosome

`ADBCEB`

would encode the solution where the first item is in the group `A`, the second in the group `D`, third in `B`, fourth in `C`, fifth in `E` and sixth in the group `B`.

Among the six *design principles* for constructing useful representations (see e.g. [Radcliffe, 91]) figures the principle of Minimal Redundancy - each member of the search space (here the space of all valid groupings) should be represented by as few distinct chromosomes as possible (ideally exactly one), in order to reduce the size of the space the GA has to search.

The straightforward encoding above is highly redundant. Indeed, the cost function of a grouping problem depends only on the grouping of the items, rather than the numbering of the group. For instance, in the Graph Coloring Problem, only the distribution of colors over the nodes counts[3], whatever the actual colors (their *names*) used. Thus with `A` standing for Amber, `B` for Blue, `C` for Crimson and `D` for DarkRed,

`ABCADD` and
`CADCBB`

both encode the same solution of the problem, namely the one where the first and fourth nodes of the graph are assigned one color, the fifth and sixth nodes a second color, and the second and third nodes a third and fourth color respectively.

The degree of redundancy (i.e. the number of distinct chromosomes encoding the same solution of the original problem) of this scheme grows *exponentially* with the number of groups, that is, indirectly, with the size of the problem. Thus the size of the space the GA has to search is *much* larger than the original space of solutions. Consequently, the power of the GA is *seriously*

[3] The resulting *number* of colors, i.e. the value of the cost function, results from a given color distribution.

impaired.

2.2 The Crossover

Let us see how the significant (strong) schemata relevant to the problem of grouping are transferred from parents to offspring under the standard crossover.

2.2.1 Context Insensitivity

The straightforward encoding leads to the highly undesirable effect of casting context-dependent information *out of context* under the standard crossover. Indeed, in the first chromosome above (`ABCADD`), the `C` affected to the third gene only has sense in the context of that particular chromosome, where it means that the third node is not grouped with any other node. Taking that `C` out of the context during crossover has disastrous consequences. To see this, let us apply the standard two-point crossover to the two chromosomes above:

`A|BC|ADD` crossed with
`C|AD|CBB` would yield
`CBCCBB` as one of the two children.

In absence of mutation, a recombination of two *identical* individuals must produce progeny which is again identical to the parents. The two parents above *are* identical with respect to the problem being solved by the GA, because they both encode the same solution of the problem. Hence a correct recombination operator should produce an individual which again encodes the same solution. However, the resulting child above encodes a 'solution' which has *nothing* in common with the solution its parents encode: there are two groups instead of four[4]!

In other words, while the schemata are well transmitted with respect to the *chromosomes* under the standard encoding/crossover, their meaning with respect to the *problem* to solve (i.e. the cost function to optimize) is lost in the process of recombination.

2.2.2 Schema Disruption

Under any encoding mapping items to genes, grouping problems of practical relevance normally have *long schemata*. However, under the standard crossover, the probability of disruption of a schema grows with its defining length. In other words, while the classic crossover (fitted with inversion or not) might converge to a better solution in the beginning of the genetic search, once a good candidate is found, instead of improving this solution, it works against its own progress towards destruction of the good schemata.

2.3 The Mutation

Let's again consider the standard encoding and see the effects of the standard mutation, i.e. a random modification of a randomly chosen allele.

Consider for example the following chromosome :

`ABDBAC`.

A mutation of this individual could yield

`ABDEAC`,

[4] Depending on the problem's hard constraints, the resulting child might be valid or invalid.

which could be beneficial, for the allele `E`, perhaps missing from the population, appears in the genetic pool.

The troubles begin as the algorithm approaches a good solution, developing large groups of identical alleles. The standard mutation of

`AAABBB`

would lead, for example, to

`AACBBB`.

On the one hand, the allele `C` appears in the population - a possibly beneficial effect. On the other hand, the new chromosome contains a 'group' of just one element. Since grouping of items usually accounts for a gain, this mutated individual will most probably show a *steep loss* of fitness in comparison with the other non-mutated individuals. Consequently, this individual will be eliminated with high probability from the population on the very next step of the algorithm, yielding hardly any benefit for the genetic search. In other words, the classic mutation is *too destructive* once the GA begins to reach a good solution of the grouping problem.

One solution to these problems would be to abandon the concept of mutation as a random modification of a *small* (minimal) part of the chromosome, in profit of a more sophisticated operator acting on several, possibly many genes simultaneously. However, that would yield an operator that basically disregards the genes in the chromosome. We show in section 4.2 below that it is possible to take a more usual route.

3. Ordering GA Operators and Grouping Problems

In this section we examine the effects of *ordering* genetic operators on the structures relevant to the grouping problems. Application of an encoding scheme representing *permutations* of the members of the set, together with ordering genetic operators is the other route that has been taken in the GA literature treating grouping problems (e.g. [Smith, 85], [Bhuyan et al., 91], [Jones and Beltramo, 91], [Reeves, 94]). We will again show why we think this approach is not the best one for these problems.

3.1 The Encoding and Redundancy

Another way of handling grouping problems is to represent permutations of the items (members of the set **U** in section 1 above), and use a decoding mechanism that reveals the actual assignment of the items to groups (the resulting partition of the set) corresponding to each chromosome. The decoding mechanism usually proceeds by considering the items one by one in the *order* given by the chromosome, and assigning them to the first group available.

For the sake of clarity, let's consider the Bin Packing Problem. Suppose there are ten items to pack, numbered `0` through `9`. A valid chromosome is one where each item appears exactly once, for example

`0123456789`.

This encoding is highly redundant. Indeed, suppose that the items in the above chromosome are partitioned as follows

`0123|45678|9`,

i.e. there are three bins, one containing the items `0` through `3`, the second containing the items `4` through `8`, and the third containing the item `9`. Now any permutation of the items having the same bin contents, such as

`3210|45678|9` or

87645|1032|9,

encodes *the same* solution of the original Bin Packing Problem. As for the straightforward encoding in section 2, the degree of redundancy of this encoding grows *exponentially* with the size of the instance. Again, the power of the GA is seriously diminished.

3.2 The Crossover

3.2.1 Context Insensitivity

Like the classic crossovers operating on chromosomes using the encoding from section 2, most ordering crossovers working with the permutation encoding cast context-dependent information *out of context* during recombination.

Indeed, given the mechanism of decoding the chromosome, it is clear that the meaning of a gene in the solution the chromosome encodes, depends heavily on all the genes that *precede* it on the chromosome. For instance, consider again the BPP and the chromosome

0123456789,

and suppose it is decoded from the left to the right as above. This chromosome encodes a solution where items 4 through 8 are in the same group. However, this information depends on the genes to the left of the group, i.e. on the head of the chromosome. For instance, the chromosomes

9123456780 or
9012345678

most probably encode different solutions of the Bin Packing Problem. Indeed, depending on the sizes of the items, a bin filled with either the items 9,1,2 or 9,0,1, respectively, can be so filled that no other item would fit in the remaining space. That would yield, say, the solutions

912|34567|80 and
901|234|5678, respectively,

none of which has the items 4 through 8 in the same bin.

There is a number of ordering crossovers available in the literature nowadays (see e.g. [Davis, 91] for several of them). Let's concentrate on Goldberg's PMX ([Goldberg, 89]), probably the best known ordering crossover. The PMX transmits well the *absolute* positions of genes on the chromosome. It works as follows. Given two parents, a crossing section is selected at random, e.g.

0123|4567|89 and
9173|5482|60.

The corresponding genes in the crossing section define a mapping, in this case 4↔5, 5↔4, 6↔8 and 7↔2. The first child is constructed by copying the crossing section from the first parent and all the genes from the second parent which do not appear in the crossing section of the first, yielding in this case

91 3|4567| 0.

Finally, the remaining places are filled by the mapping's images of the genes that previously occupied those positions. That yields

9123|4567|80.

The second child is obtained by permuting the roles of the parents.
Now the only difference between the chromosomes

0123456789 and
9123456780

consists of a swap of the first and last genes. Hence with a crossing section anywhere between the

two genes, the information the PMX transmits is identical in the two chromosomes w.r.t. the position of the group `45678`. That means the latter chromosome could be a child of the former (as illustrated by the example), even though the two encode *very* different solutions of the BPP, one having the group `45678` together, the other not. In other words, the PMX transmits information which more often than not gets a different meaning in the new chromosome.

Other ordering crossovers suffer similar problems. The reason is that *o-schemata* (the ordering equivalent of Holland's schemata) have little meaning in a grouping problem - they are *not* building blocs capable of conveying useful information on the solution they're part of. Indeed, the only information useful in grouping problems concerns the *groups* and these are obtained from a permutation of *items* in a far too indirect way. Consequently, as illustrated by [Falkenauer, 94] on a simple example, sampling *o-schemata* in grouping problems is of little use for estimating the quality of the corresponding solutions.

The inadequacy of ordering crossovers for grouping problems is not only backed by the theoretical arguments given above - it has been confirmed by experimental evaluation. Indeed, testing various approaches to the Cutting Stock Problem[5], [Hinterding and Khan, 94] have observed a *degradation* of performance of an ordering GA with increasing crossover rate. What their results mean is that the information inherited by progeny from their parents (under the ordering crossover) is mostly *detrimental* to the quality of the progeny. That goes in line with our claim that the information passed on is *not* useful, since the best ordering 'GA' for a grouping problem would be one with *no* crossover!

3.2.2 Schema Disruption

We have indicated in section 2.2.2 that an encoding mapping *items* onto genes in a chromosome leads to high probabilities of disruption of useful parts of the chromosome. Since the permutation-based encoding also maps items onto genes, the ordering GA suffers of the same drawback.

3.3 The Mutation

In the ordering GA, the mutation operator modifies the order of the genes on the chromosome. However, given the above, such an operator has a high probability of either not having any effect at all because of the high redundancy of the encoding, or being *too destructive* because of its impact on *all* the items that map onto genes following the modified site. Once again, these problems stem from the item oriented (rather than *group* oriented) encoding.

4. The Grouping Genetic Algorithm

The Grouping Genetic Algorithm (GGA) differs from the classic GA in two important aspects. First, a special encoding scheme is used in order to make the relevant structures of grouping problems become genes in chromosomes, i.e. the building blocks the GA works with. Second, given the encoding, special genetic operators are used, suitable for the chromosomes.

---

[5] The Cutting Stock Problem is essentially a Bin Packing where the bins are of various capacities.

4.1 The Encoding

As we have seen, neither the standard nor the ordering genetic operators are suitable for grouping problems. The reason is that the structure of the simple chromosomes (which the above operators work with) is *item* oriented, instead of being *group* oriented. In short, the above encodings are not adapted to the cost function to optimize. Indeed, the cost function of a grouping problem depends on the *groups*, but there is no structural counterpart for them in the chromosomes above.

Note that these remarks are nothing more than a call for compliance with the *Thesis of Good Building Blocks* (see e.g. [Goldberg, 89][6]), central to the GA paradigm. Others issued similar calls in the past, most notably Radcliffe (e.g. [Radcliffe, 92]), Vose and Liepins (e.g. [Vose and Liepins, 91]). Still, the references in sections 2 and 3 seem to indicate that their calls were not heard to the extent they deserved, at least as far as grouping problems are concerned.

To remedy the above problems, we have chosen the following encoding scheme: the standard chromosome from section 2 is augmented with a *group part*, encoding the groups on a *one gene for one group* basis. More concretely, let us consider the first chromosome in section 2 and the Bin Packing Problem. Numbering the items from `0` through `5`, the *item* part of the chromosome can be explicitly written

`012345`
`ADBCEB`: ...,

meaning the item `0` is in the bin *labeled* (named) `A`, `1` in the bin `D`, `2` and `5` in `B`, `3` in `C`, and `4` in `E`. This is the straightforward chromosome from section 2. The *group* part of the chromosome represents *only the groups* (bins in BPP). Thus

... :`BECDA`

expresses the fact that there are five bins in the solution. Of course, what *names* are used for each of the bins is irrelevant in the BPP: only the *contents* of each bin counts in this problem. We thus come to the *raison d'être* of the item part. Indeed, by a lookup there, we can establish what the names stand for. Namely,

`A={0}`, `B={2,5}`, `C={3}`, `D={1}` and `E={4}`.

In fact, the chromosome could also be written

`{0}{2,5}{3}{1}{4}`.

The important point is that the genetic operators will **work with the group part** of the chromosomes, the standard item part of the chromosomes merely serving to identify which items actually form which group. Note in particular that this implies that the operators will have to handle chromosomes of *variable length*.

In short, the encoding scheme we adopted makes the *genes* represent the *groups*. The rationale is that in grouping problems it is the groups which are the meaningful building blocks, i.e. the smallest piece of a solution which can convey information on the expected quality of the solution they are part of. This is crucial: indeed, the very idea behind the GA paradigm is to perform an *exploration* of the search space, so that promising regions are identified, together with an *exploitation* of the information thus gathered, by an increased search effort in those regions. If, on the contrary, the encoding scheme does not allow the building blocks to be exploited (i.e. transmitted from parents to offspring, thus allowing a continuous search in their surrounding) and

[6] Goldberg calls it the Building Blocks Hypothesis.

simultaneously serve as estimators of quality of the regions of the search space they occupy, then the GA strategy inevitably fails and the algorithm performs in fact little more than a random search or *naïve evolution*.

Note finally that the order of the groups in the chromosome is irrelevant in the GGA (this is obvious under the inversion operator).

4.2 The Crossover

As pointed out in the previous section, a GGA crossover[7] will work with variable length chromosomes with genes representing the groups.

Given the fact that the hard constraints and the cost function vary among different grouping problems, the ways groups can be combined without producing invalid or too bad individuals are not the same for all those problems. Thus the crossover used will *not* be the same for all of them. However, it will fit the following pattern:

1. Select at random two crossing sites, delimiting the *crossing section*, in each of the two parents.
2. *Inject* the contents of the crossing section of the first parent at the first crossing site of the second parent. Recall that the crossover works with the group part of the chromosome, so this means injecting some of the *groups* from the first parent into the second.
3. Eliminate all *items* now occurring twice from the groups they were members of in the second parent, so that the 'old' membership of these items gives way to the membership specified by the 'new' injected groups. Consequently, *some* of the 'old' groups coming from the second parent are altered: they do not contain all the same items anymore, since some of those items had to be eliminated.[8]
4. If necessary, *adapt* the resulting groups, according to the hard constraints and the cost function to optimize. At this stage, local problem-dependent heuristics can be applied.
5. Apply the points 2. through 4. to the two parents with their roles permuted in order to generate the second child.

As can easily be seen, the idea behind the GGA crossover is to promote promising *groups* by inheritance. In this respect, it is interesting to compare the GGA to the algorithm proposed by [Reeves, 94]. He has recently tackled the Bin Packing Problem with an ordering GA coupled with a decoder of the chromosomes (permutations of items) using the First Fit and Best Fit heuristics. The experimental results obtained with the ordering GA were inferior to the ones reported in [Falkenauer and Delchambre, 92], which means that they are substantially inferior to the performance of the algorithm presented here.

However, more interesting than this comparison is the method Reeves uses to enhance the performance of the ordering GA. In a hybrid algorithm, he performs a *reduction*[9] of the problem:

---

[7] The *recombination operator* presented here turns out to be rather more sophisticated than the classic *crossover*. Nevertheless, we kept the latter denomination because of the intended role of the operator.

[8] The injection operation has some of the flavor of the *cut & splice* recombination operator of Goldberg's messy GA (e.g. [Goldberg et al., 91]). The fundamental difference lies with the fact that the GGA's crossover always ends up with each *item* present exactly once, through a possible modification of genes coming from the second parent.

[9] This 'reduction' is not to be confused with the Reduction Method of Martello and Toth, the latter being based on a criterion of *dominance* of one bin over others.

each time a sufficiently filled bin is found in a solution (i.e. an individual decoded with the FF or BF), the bin is 'fixed' and the objects it contains are eliminated from *all* individuals in the population, yielding a smaller problem.

Recall that the purpose of the GGA crossover operator is to transfer *groups* (bins in BPP) from parents to offspring. Supposing a bin remains consistently above-average[10], this mechanism will eventually propagate the bin throughout the whole population. Clearly, at that point the bin will be 'fixed' (because *all* individuals in the population will then contain it), and could be removed from the problem.

Thus the reduction of Reeves does the same thing as the GGA's crossover: it propagates promising bins. However, as we show in [Falkenauer, 94], the former gets very easily stuck in a local optimum, because it violates the search strategy of the GA. Indeed, whenever the GGA discovers a promising bin, it is propagated throughout the population, which means that the sampling rate of the solutions containing that bin increases in accordance with the optimal sampling strategy of [Holland, 75]. Yet that strategy also requires to *continue* the sampling of the other solutions (i.e. those that do *not* contain the bin), albeit at a reduced rate. When the reduction of Reeves eliminates a bin, the sampling of the solutions that do not contain it *ceases immediately* and is never resumed, which violates the optimal sampling strategy.

Although it is not the objective of this paper to compare experimentally the GGA to the classic GA approaches to grouping problems, let us note the comparison made by [Falkenauer, 95]. In that paper, a GGA is compared to the results obtained by [Jones and Beltramo, 91] with nine different standard GAs applied to the problem of Equal Piles.

Equal Piles is a grouping problem extremely similar to Bin Packing: a set of items of given sizes must be distributed over a given number of bins, the objective being to *equalize* as much as possible the contents of the bins. Among the nine GAs they studied, an ordering GA (see section 3) combined with a greedy heuristic was found to be significantly better than the best standard GA (section 2). Nevertheless, the former algorithm was itself outperformed by a wide margin by the GGA.

4.3 The Mutation

A mutation operator for grouping problems must work with groups rather than items. As for the crossover, the implementation details of the operator depend on the particular grouping problem on hand. Nevertheless, three general strategies can be outlined: *create* a new group, *eliminate* an existing group, or *shuffle* a small number of items among groups.

4.4 The Inversion

The inversion operator serves to shorten good schemata in order to facilitate their transmission from parents to offspring, thus ensuring an increased rate of sampling of the above-average ones ([Holland, 75]). In a Grouping GA, it is applied to the *group* part of the chromosome. Thus for instance, the chromosome

`ADBCEB:BECDA`

could be inverted into

[10] That is, the individuals containing that bin consistently score better than those which do not contain it. Recall that under the GGA encoding, a group represents a *group-schema* of order one.

ADBCEB:CEBDA.

Note that the item part of the chromosome stays unchanged. Indeed, the groups are still composed of the same items.

The example illustrates the utility of this operator: should B and D be promising genes (i.e. well-performing groups), the probability of transmitting *both* of them during the next crossover is improved after the inversion, since they are now closer together, i.e. safer against disruption. That in turn makes the proliferation of the good *group-schemata* easier.

5. The Dominance Criterion

[Martello and Toth, 90a] have recently introduced a new approximation algorithm for the Bin Packing Problem. Their method is based on a simple yet powerful observation illustrated in 1: given the contents of two bins B_1 and B_2, if there exists a subset $\{i_1, \ldots i_n\}$ of items in B_2 and a partition $\{P_1, \ldots P_n\}$ of items in B_1 such that for each item i_i there is a no bigger[11] corresponding P_i, then B_2 is said to *dominate* B_1, because a solution obtained with B_2 as one of the bins requires no more bins than one with B_1.

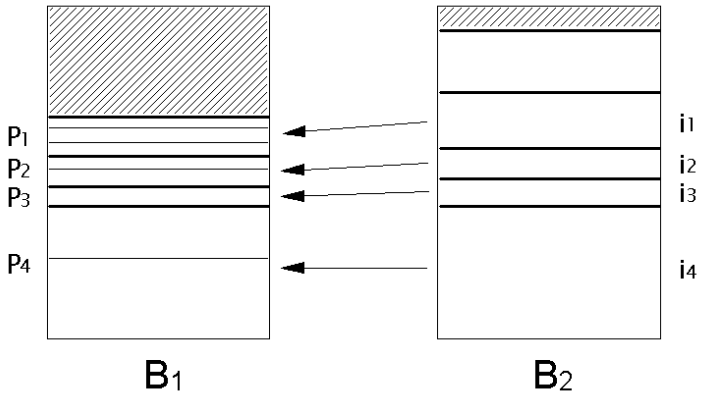

Figure 1 The Dominance Criterion: B_2 dominates B_1

A Bin Packing algorithm would repeatedly find a bin dominating *all* others, add that bin to the solution and *reduce* the problem by removing the items just assigned. However, that procedure would run in exponential time. In order to obtain a procedure of a reasonable complexity, [Martello and Toth, 90a] propose to consider only sets of size three or less.

An effective approximation algorithm can be obtained from the above procedure as follows. When no set dominating all others can be found anymore, the problem is *relaxed* by removing the smallest item among those which are not yet assigned to any bin, and the procedure is repeated. [Martello and Toth, 90a] use this algorithm to compute lower bounds on the number of necessary bins for BPP instances.

We use the concept of dominance in conjunction with the Grouping GA described above. It serves as a *local optimization* producing high-quality bins which are then efficiently processed by the GGA.

6. The Hybrid GGA for Bin Packing

In this section, we first define a suitable cost function for the Bin Packing Problem, and then show how the Grouping Genetic Algorithm can be combined with the concept of dominance.

[11] That is, the total size of items in P_i is less than or equal to the size of i_i.

6.1 The Problem Redefinition

Let's define a suitable cost function for the Bin Packing Problem defined in section 1.1. The objective being to find the minimum number of bins required, the first cost function which comes to mind is simply the number of bins used to 'pack' all the items. This is correct from a strictly mathematical point of view, but is unusable in practice. Indeed, such a cost function leads to an extremely 'unfriendly' landscape of the search space: a very small number of optimal points in the space are lost in an exponential number of points where this purported cost function is just one unit above the optimum. Worse, all those slightly suboptimal points yield the *same* cost. The trouble is that such a cost function lacks any capacity of *guiding* an algorithm in the search, making the problem a 'Needle-in-a-haystack'.

We thus settled in [Falkenauer and Delchambre, 92] for the following cost function for the BPP: maximize

$$f_{BPP} = \frac{\sum_{i=1..N}(F_i / C)^k}{N}$$

with N being the number of bins used in the solution,
F_i the sum of sizes of the items in the bin i (the fill of the bin),
C the bin capacity and
k a constant, k>1.

The constant k expresses our concentration on the 'extremist' bins in comparison to the less filled ones. The larger k is, the more we prefer well-filled 'elite' groups as opposed to a collection of about equally filled bins. We have experimented with several values of k and found out that k=2 gives good results. Larger values of k seem to lead to premature convergence of the algorithm, as the local optima, due to a few well-filled bins, are too hard to escape.

The exact assessment of the influence of k is difficult, due to the extremely large number of possible instances and the corresponding solutions. The question is whether some choice of k can lead to a cost function that has global optima *other* than those of the original one, i.e. such that for a certain value of k,

$$f_{BPP}(P_{N+1}) \geq f_{BPP}(P_N),$$

where N is the minimum number of bins necessary to contain all the items, P_N an optimal solution requiring this minimal number of bins, and P_{N+1} a solution requiring one more.

The f_{BPP} function departs from the original one (i.e. number of bins) as k grows above 1 and the well-filled 'elite' bins are promoted. Examining the most adverse possibility, namely the one where P_{N+1} features an 'elite' of N_F full bins among a total of N+1 bins, while P_N consists of N equally filled bins[12], the above inequality yields, after a conservative development,

$$2^k - 1 \geq 2k.$$

We thus see that for moderate values of k, in particular for $k \leq 2$, the last inequality is *not* satisfied, so

$$f_{BPP}(P_{N+1}) < f_{BPP}(P_N) \qquad \text{for } k \leq 2,$$

and the f_{BPP} function defined above yields the same optima as the original BPP objective.

[12] This is a conservative approach, as an arbitrary instance of the problem does not necessarily admit the two solutions.

6.2 BPRX - the Bin Packing Crossover with Replacement

6.2.1 The Mechanism

A crossover's job consists of producing offspring out of two parents in such a way that the children inherit as much as possible of the meaningful information from *both* parents. Since it is the bin[13] that conveys important information in BPP, we must find a way to transmit bins from the parents onto the children. This is done as follows.

Consider the following *group parts* of the chromosomes (recall that there is one gene per bin):

`ABCDEF` (first parent)
`abcd` (second parent).

First, copies are made of the two parents (in order not to destroy the parents) and two crossing sites are chosen at random in each of them, yielding for example

`A|BCD|EF` and
`ab|cd|.`

Next, the bins between the crossing sites in the second chromosome are *injected* into the first, at the first crossing site, yielding

`AcdBCDEF.`

Now some of the items appear twice in the solution and must be thus eliminated. Suppose the items injected with the bins `c` and `d` also appear in the bins `C`, `E` and `F`. We eliminate those bins, leaving

`AcdBD.`

With the elimination of those three bins we have, however, most probably eliminated items which were not injected with the bins `c` and `d`. Those items are thus missing from the solution.

To fix this last problem, in our previous work we used the First Fit Descending (FFD) heuristic to reinsert again the missing objects ([Falkenauer and Delchambre, 92]). That is, they were sorted in descending order of their sizes and then put one by one into the first sufficiently empty bin.

In our new algorithm, prior to the application of FFD, we first perform a *local optimization* inspired by the ideas of [Martello and Toth, 90a] in the following way. Taking one by one the bins so far in the solution[14], we check whether it is possible to **replace** up to three items in the bin by one or two items from those currently missing in the bins, in such a way that the total size of the items in the bin increases without overflowing the bin. If so, we perform the replacement, which means that some of the previously unassigned items are assigned to the bin, while some of the items previously assigned to the bin become 'unassigned'.

Note that the replacement has two important consequences. First, it *fills better* the target bin than it was before. Since a good packing has well-filled bins, this improves the quality of the solution on the one hand. On the other hand, since the total size of the items is a constant, it also leaves more space in the other bins (indirectly), which makes them more capable of accommodating additional items, and that leads ultimately to the desired reduction of the number of necessary bins. Second, the exchange makes available (i.e. 'unassigns') *smaller items* than before the exchange, which makes easier the task of adding those items to bins already in the solution.

---

[13] More precisely, a *coadapted subset* of the bins in the solution.

[14] These are the bins coming from the other parent, as well as those which were not affected by the injection.

This process is repeated as long as such exchanges are possible. When none is possible anymore and there are still items to be reinserted, the usual FFD heuristic is applied to complete the solution (i.e. assign all items to bins).

6.2.2 The Rationale

The method of [Martello and Toth, 90a] aims at identifying bins which *dominate* the others, so that they can be fixed, thus reducing the effective size of the problem. However, because of the size of the search space, it is impossible to verify in a reasonable time that a given bin dominates *all* the others. The approximation described in section 5 above solves the problem of complexity of the algorithm, but introduces a new one: it happens that the simplified method does *not* find a bin which dominates the others. In order to continue towards *some* solution, rather than stopping the algorithm with a "do not know" answer, the problem is *relaxed* in some way[15] and the reduction procedure is reapplied.

However necessary to keep the algorithm practically useful, the relaxation step introduces an extraneous element into the procedure: there is little guarantee that the relaxation applied will preserve the global optimality of the solution under construction. Since there is no backtracking allowed (for efficiency reasons), once a wrong relaxation is performed, the global optimum is out of reach and the method will settle to a local one.

The replacement stage in the BPRX was inspired by the method of Martello and Toth. Indeed, what is done during that stage is nothing else than a local search for dominant bins, using an approximative criterion very similar to the one used by Martello and Toth. However, the Grouping GA does not suffer the drawback of convergence to a local optimum. Since *whole bins* are transmitted during a crossover, each improvement is usefully propagated throughout the population, by means of transmission from parents to offspring. That makes it possible to 'test' the quality of each bin in numerous contexts, which is analogous to testing the dominance of each bin under many different relaxations.

Indeed, consider a bin in one individual in the population and recall that in the Grouping GA, the bins correspond to genes in the chromosomes. If the gene is transmitted without modification over the succesive generations of the GGA, then it probably dominates most of the other bins that could be made with the items it contains. On the other hand, if it *is* modified, then it is because there is a bin which dominates it.

Finally, the recombinating power of the BPRX crossover takes still further advantage of an approximated dominance criterion. Consider two bins having no item in common and suppose each is part of one individual in the population. Since both individuals have survived the evolutionary competition, the bins they are made of dominate, under the approximated criterion, all other bins which could be formed with the items the bins contain[16]. Now since the BPRX crossover combines whole bins and the bins have no item in common, they can be *both* inherited in a child, without a need of new verification of their dominance. In short, the crossover constructs solutions that *automatically* contain bins dominating the others.

---

[15] [Martello and Toth, 90a] propose three sensible ways of relaxing a problem, but there are numerous other ways of relaxing a given instance.

[16] This does not mean that they dominate *all* the others though, because a more thorough (less approximated) criterion could reveal that they are themselves dominated.

6.3 The Mutation

The mutation operator is simple. Given a chromosome, we select at random a few bins (i.e. *genes*) and eliminate them. The items which composed those bins are thus missing from the solution and must be inserted back. To do so, we again use the ideas applied for the crossover. Namely, the eliminated items first serve as the basis for eventual improvement of the bins left unchanged, during a stage of replacement. When no replacement is possible anymore, the FFD heuristic is used to complete the solution.

7. Experimental Results

7.1 The Setup

In order to assess the merit of the algorithm, we compared the hybrid GGA (HGGA in the sequel) to the MTP procedure of [Martello and Toth, 90b], i.e. an enumerative (branch-and-bound) method which has the Dominance Criterion embedded. We chose the MTP as the benchmark because it is considered by many to be one of the best methods for the Bin Packing Problem to date.

The GGA used in the experiments is a steady-state[17] order-based GA using a population of 100 individuals and a tournament of size 2 as the selection strategy. On each generation, 50 individuals were replaced by progeny produced by crossover of the 50 best individuals, 33 individuals (selected at random) were mutated and 25 individuals underwent inversion. The full details of the general procedure can be found in [Falkenauer, 94].

The initial population was generated by running the First Fit heuristic on 100 random permutations of the items. Note that the heuristic is extremely fast, yielding a run time of 0.0 CPU seconds whenever an optimal solution appeared already in the initial population.

Two sets of experiments were performed. One was designed to see how the HGGA fares on the MTP's "turf", i.e. we generated instances of the kind considered in [Martello and Toth, 90a] (see section 7.2 below). The second set of experiments was designed to see the practical limits of the two algorithms. To do so, we generated what seems to be the *most difficult* BPP instances (see section 7.3).

Each instance was submitted both to the MTP procedure and to the Hybrid GGA[18]. This way, the performances of the two algorithms were compared on the *same* instances of BPP. The HGGA was coded in C++ and run on a R4000 Silicon Graphics workstation under IRIX 5.1. The MTP procedure was coded in FORTRAN (we used the code of Martello and Toth) and run on a Control Data CD4000 (also an R4000 machine) under EP/IX 2.1.

Note that the enumerative nature of the MTP procedure makes it take an excessive time when confronted with difficult instances, and an artificial shutoff is required in order to obtain *some* result. We thus allowed the MTP to perform at most 1500000 (one and a half million) backtracks on any of the instances. However, in some cases, this shutoff aborted the MTP sooner than after the amount of time (in CPU seconds) we gave to the HGGA, which could be perceived as an unfair comparison. Consequently, whenever this was the case, we increased the backtrack

[17] See e.g. [Syswerda, 89]. Note however that according to [Davis, 91], most steady-state GAs in use create and insert just one or two children in a generation, whereas we replace 50% of the population.

[18] We run the GGA once on each of 160 instances, rather than running it several times on a more limited set.

limit accordingly, and rerun the MTP.

7.2 Uniform Item Size Distribution

[Martello and Toth, 90a] give the performance of their method on instances of the BPP constructed by drawing integer item sizes uniformly random from a given range. A look at their results reveals that among the various classes of problems they considered (different bin capacities and different ranges of sizes of the items), the following setup proved the most difficult for their method: bin capacity 150 and integer item sizes uniformly random between 20 and 100.

We generated instances of this kind (i.e. items of sizes uniformly distributed between 20 and 100 to be packed into bins of capacity 150) with the number of items 120, 250, 500 and 1000, respectively. We generated 20 instances of each.

The results of the comparison are summarised below in tables 1 through 4. Each of the tables shows, for each of the 20 instances, the number of the instance (`Run`), the theoretical minimum number of bins (`Theo` = $\lceil$totsize/binsize$\rceil$)[19], and the respective results of the Hybrid GGA (**`HGGA`**) and the MTP procedure (**`MTP`**). For the HGGA, the tables indicate the number of bins obtained (`Bins`), the number of cost function evaluations performed (`Evals`)[20], and the time spent in CPU seconds (`Time`). For the MTP procedure, the tables indicate the number of bins obtained (`Bins`), the absolute and relative difference with respect to the number of bins obtained with the HGGA (`Loss` and `Loss%`), the number of backtracks the MTP performed (`Backs`, where the value of `1500000` or more indicates that the procedure was stopped before termination), and the time required in CPU seconds (`Time`).

For the first two sets of data (120 and 250 items), we imposed a maximum of 134000 cost function evaluations (2000 generations) for the HGGA. For the other two (500 and 1000 items), that limit was set to 335000 (5000 generations).

As observed by Martello and Toth, this class of problems proves difficult for their method. Already for the relatively small instances of 120 items, it was unable to finish for two of the twenty data sets. Only nine instances were completely solved with 250 items, and none was solved with 500 or 1000 items. Nevertheless, the MTP fared well on the smallest instances, where it was very fast most of the time.

Among the instances of 120 items (Table 1), in two cases (runs 9 and 20) neither algorithm came up with a solution having the theoretical number of bins, so we conjecture that there is no such solution. However, the MTP had to be aborted prematurely, so we have no proof. The interesting point is that in both cases, a solution in *Theo*+1 bins (MTP's best) appeared already in the initial population of the HGGA, which seems to indicate that for these two instances, an optimal solution is extremely easy to find, yet very hard to prove.

Apart from those two instances, the 120-item data proved to be easy for the MTP, which was faster than the HGGA. Nevertheless, the difference was only marginal.

From 250 items up, the 'explosive' nature of MTP starts to show, and the results show a superiority of the HGGA in two respects: the HGGA supplies *better* solutions, and it does so

[19] *Theo* is the minimum number of bins that can accommodate the total size of the items. Consequently, a solution in *Theo* bins is globally optimal.

[20] There were on average 67 new individuals created per generation of the HGGA.

faster than the MTP procedure.

For seventeen out of the twenty instances of 250 items (Table 2), the HGGA has discovered a solution in *Theo* bins, i.e. a global optimum. For the remaining three (runs 8, 13 and 14), the HGGA's solution had the same number of bins as the MTP's, so we again conjecture that these were globally optimal as well, although we again have no proof, as the MTP had to be aborted in all three cases.

The MTP procedure was less successful in finding the global optima for these instances: in nine out of the twenty cases, it was unable to find a solution as good as the HGGA's, the difference (*Loss*) being two bins in one case (run 6).

The results on instances of 500 items (Table 3) and 1000 items (Table 4) confirm the superiority of the HGGA in comparison to the MTP procedure. In *all* cases, the HGGA found a globally optimal solution in *Theo* bins[21], while the MTP *never* did (see the *Loss* column).

In addition, the meaning of time taken by the MTP becomes especially clear here. For all but four of the 500 item instances, and for all with 1000 items, the difference between the solutions of the two algorithms was two bins or more. This means that even if increased the backtrack limit for the MTP so that it could improve the solution by one bin, which would undoubtedly lead to extremely long execution times, the MTP would *still* end up with a solution worse than the HGGA. Consequently, the HGGA fared better in both quality of the solution *and* the speed of its delivery.

7.3 Triplets

With the second class of tests, we tried to establish the practical limits of the HGGA. We considered problems with item sizes drawn from the range (0.25, 0.50) to be packed into bins of capacity 1. In these problems, a well-filled bin must contain one 'big' item (larger than the third of the bin capacity) and two 'small' ones (smaller than the third of the bin), which is why we term them 'triplets'. What makes these problems difficult is the fact that putting two 'big' or three 'small' items into a bin *is* possible, but inevitably leads to a waste of space (the bin cannot be completely filled by an additional item), implying a suboptimal solution.

[van Vliet, 93][22] points out a similarity between the difficulty of 'triplets' and those of 3SAT[23]. While the 3SAT Problem is NP-complete, the satisfiability of a conjuction of two-variable clauses (2SAT) is solvable in polynomial time ([Garey and Johnson, 79]), just like BPP instances with two items per bin[24]. On the other hand, when the clauses grow longer (kSAT, k>3), each clause is easier to satisfy, because there needs to be only one out of the k variables with the value TRUE. Consequently, with a constant number of literals, kSAT instances are easier with growing k, so 3SAT is the most difficult SAT problem[25]. Similarly, BPP instances are easier to approximate when the number of items per bin grows above three, so 'triplets' are the most difficult BPP instances.

---

[21] The existence of a solution in *Theo* bins for all the instances can be explained by the fact that the large number of items implies a large pool of various sizes from which well-filled bins can be drawn.

[22] André van Vliet suggested to test the algorithm on this kind of instances.

[23] 3SAT is the satisfiability (SAT) problem where all clauses are disjunctions of exactly three Boolean variables ([Garey and Johnson, 79]).

[24] Such instances would have all items larger than the third of the bin size and smaller than its half. They are trivially solvable in quadratic time.

[25] Of course, there are trivial 3SAT instances, but *on average*, randomly generated 3SAT instances are harder than randomly generated kSAT instances with k>3.

In order to preserve the difficulty of the problem, we generated instances of *known optima*[26] as follows. Considering bin capacity of 1000, an item was first generated with a size drawn uniformly from the range [380,490]. That left a space S of between 510 and 620 in the bin of 1000. The size of the second item was drawn uniformly from [250, S/2). The third item completed the bin.

To test the algorithms, we generated 'triplets' with 60, 120, 249 and 501 items, 20 instances of each. The results of the HGGA and the MTP procedure are summarised in Tables 5 through 8 below. For the first two sets of instances (60 and 120 items), we imposed a limit of 67000 evaluations (1000 generations). For the other two, we set that limit to 134000 (2000 generations).

The performance of the MTP procedure shows that 'triplets' are indeed very hard problems. The MTP was able to finish only on 6 of the twenty instances of 60 items, and it never finished on any of the bigger ones.

In two runs of 60 items (runs 8 and 19), the HGGA failed to find the optimal solution. Obviously, the 67000 evaluations limit was too tight. Interestingly, on both these instances, the MTP fared one bin worse yet.

Among the six runs which the MTP was able to optimize completely, in only two cases (runs 11 and 13) was the MTP faster than the HGGA. Except for these, the MTP took longer than the HGGA to find an equally good solution. In sixteen out of twenty cases, it took longer to find a *worse* solution.

From 120 items up, the HGGA fared better than the MTP procedure in all respects. It always found a globally optimal solution, while the MTP never did. The HGGA was also much faster[27].

While still showing a remarkable performance given the size and difficulty of the problems, the 501 item triplets constitute a limit to online performance of the HGGA run on ordinary hardware, and larger instances will probably have to be run overnight.

NOTE: All the 160 BPP instances used in the above experiments were made available to the OR-Library Benchmark Database kept by John Beasley at the Imperial College, U.K. (the best way to first contact OR-Library is to email the message 'info' to o.rlibrary@ic.ac.uk).

8. Conclusions

We have presented an algorithm issued from two techniques for the Bin Packing Problem, the Grouping Genetic Algorithm of Falkenauer and the Reduction Method of Martello and Toth.

The GGA's distinctive features make it *exploit* the very structure of the grouping problems that makes the standard and the ordering GAs fail. That ability, joined with the efficient OR technique for *generating* those blocks, leads to a hybrid GGA performing better than either of its components separately. The superiority of the hybrid GGA over the MTP procedure of Martello and Toth was confirmed by an extensive experimental comparison.

---

[26] As the MTP's performance in Tables 5 to 8 reveals, it was impossible to generate triplets at random and then *compute* the global optima.

[27] Note that the MTP always claimed more than two bins more than the HGGA.

Run	Theo	HGGA			MTP				
		Bins	Evals	Time	Bins	Loss	Loss%	Backs	Time
1	48	48	201	15.2	48	0	0.0	56	0.1
2	49	49	0	0.0	49	0	0.0	0	0.1
3	46	46	67	5.8	46	0	0.0	124935	29.0
4	49	49	804	50.4	49	0	0.0	74	0.0
5	50	50	0	0.0	50	0	0.0	0	0.0
6	48	48	268	19.4	48	0	0.0	43	0.1
7	48	48	268	19.0	48	0	0.0	69	0.0
8	49	49	335	21.7	49	0	0.0	54	0.0
9	**50**	**51**	134000	3668.7	51	0	0.0	10000000	3681.4
10	46	46	603	39.5	46	0	0.0	103	0.1
11	52	52	0	0.0	52	0	0.0	0	0.1
12	49	49	335	23.7	49	0	0.0	64	0.1
13	48	48	402	25.7	48	0	0.0	88	0.0
14	49	49	0	0.0	49	0	0.0	0	0.0
15	50	50	0	0.0	50	0	0.0	0	0.0
16	48	48	134	11.1	48	0	0.0	36	0.1
17	52	52	0	0.0	52	0	0.0	0	0.0
18	52	52	1340	76.1	52	0	0.0	48	0.0
19	49	49	201	14.3	49	0	0.0	24	0.0
20	**49**	**50**	134000	3634.7	50	0	0.0	7500000	3679.4
Averages				*381*		*0*	***0.0***		*370*
				6min					***6min***

Table 1 Uniform, 120 items.

Run	Theo	HGGA			MTP				
		Bins	Evals	Time	Bins	Loss	Loss%	Backs	Time
1	99	99	3082	256.7	100	1	1.0	1500000	1001.6
2	100	100	469	47.4	100	0	0.0	151	0.2
3	102	102	2613	223.8	102	0	0.0	418	0.3
4	100	100	268	27.0	100	0	0.0	128	0.1
5	101	101	1809	163.5	101	0	0.0	10225	4.1
6	101	101	7236	477.6	103	2	2.0	1500000	522.1
7	102	102	134	14.5	102	0	0.0	115	0.1
8	**103**	**104**	134000	6628.8	104	0	0.0	4000000	7411.8
9	105	105	18827	924.4	106	1	1.0	3100000	1049.1
10	101	101	1675	158.3	102	1	1.0	1500000	597.1
11	105	105	1005	95.6	106	1	1.0	1500000	377.2
12	101	101	3082	240.0	102	1	1.0	1500000	1075.5
13	**105**	**106**	134000	5996.7	106	0	0.0	7000000	6100.9
14	**102**	**103**	134000	6346.6	103	0	0.0	3000000	6969.2
15	100	100	804	82.6	100	0	0.0	135	0.1
16	105	105	98289	4440.1	106	1	1.0	3000000	4672.8
17	97	97	3216	254.5	98	1	1.0	1500000	545.4
18	100	100	402	38.5	100	0	0.0	138	0.1
19	100	100	2948	246.8	100	0	0.0	320	0.4
20	102	102	737	68.0	102	0	0.0	109	0.1
Averages			*27430*	*1337*		*0*	***0.4***		*1516*
				22min					***25min***

Table 2 Uniform, 250 items.

Run	Theo	HGGA			MTP				
		Bins	Evals	Time	Bins	Loss	Loss%	Backs	Time
1	198	198	7102	480.5	201	3	1.5	1500000	986.8
2	201	201	2412	177.7	202	1	0.5	1500000	868.5
3	202	202	4422	347.9	204	2	1.0	1500000	910.9
4	204	204	233562	11121.2	206	2	1.0	20000000	11412.1
5	206	206	3283	267.6	209	3	1.5	1500000	844.0
6	206	206	1407	129.7	207	1	0.5	1500000	818.3
7	207	207	28073	1655.5	210	3	1.4	5200000	1854.1
8	204	204	36314	1834.7	207	3	1.5	4000000	2084.5
9	196	196	8911	501.5	198	2	1.0	1500000	1221.8
10	202	202	938	92.5	204	2	1.0	1500000	962.4
11	200	200	1072	106.2	202	2	1.0	1500000	893.6
12	200	200	1876	152.3	202	2	1.0	1500000	793.0
13	199	199	17621	1019.3	202	3	1.5	2200000	1258.2
14	196	196	1541	135.5	197	1	0.5	1500000	860.3
15	204	204	12261	951.7	205	1	0.5	2000000	1202.8
16	201	201	5360	375.2	203	2	1.0	1500000	782.9
17	202	202	1809	162.6	204	2	1.0	1500000	732.7
18	198	198	4556	336.8	201	3	1.5	1500000	754.5
19	202	202	1675	143.9	205	3	1.5	1500000	637.5
20	196	196	4824	306.8	199	3	1.5	1500000	819.2
Averages			*18951*	*1015*		*2*	***1.1***		*1535*
				17min					***26min***

Table 3 Uniform, 500 items.

Run	Theo	HGGA			MTP				
		Bins	Evals	Time	Bins	Loss	Loss%	Backs	Time
1	399	399	2211	2924.7	403	4	1.0	3500000	3279.0
2	406	406	2948	4040.2	410	4	1.0	5000000	4886.6
3	411	411	4958	6262.1	416	5	1.2	8500000	6606.1
4	411	411	35376	32714.3	416	5	1.2	50000000	40285.6
5	397	397	8844	11862.0	401	4	1.0	20000000	20689.8
6	399	399	2948	3774.3	402	3	0.8	5000000	4216.3
7	395	395	2010	3033.2	398	3	0.8	3000000	3449.7
8	404	404	7303	9878.8	406	2	0.5	12500000	12674.4
9	399	399	4355	5585.2	402	3	0.8	4500000	6874.0
10	397	397	6968	8126.2	402	5	1.3	12200000	9568.2
11	400	400	2278	3359.1	404	4	1.0	4000000	3542.8
12	401	401	6700	6782.3	404	3	0.7	8100000	7422.4
13	393	393	1943	2537.4	396	3	0.8	3200000	2714.0
14	396	396	14137	11828.8	401	5	1.3	20000000	23319.4
15	394	394	5762	5838.1	399	5	1.3	5000000	6770.9
16	402	402	13802	12610.8	407	5	1.2	20000000	20458.4
17	404	404	2278	2740.8	407	3	0.7	3000000	3139.6
18	404	404	2077	2379.4	407	3	0.7	3000000	2506.4
19	399	399	1005	1329.7	403	4	1.0	1500000	1353.2
20	400	400	2680	3564.2	405	5	1.3	3000000	4109.6
Averages			*6529*	*7059*		*4*	***1.0***		*9393*
				118min					***157min***

Table 4 Uniform, 1000 items.

Run	Theo	HGGA			MTP				
		Bins	Evals	Time	Bins	Loss	Loss%	Backs	Time
1	20	20	603	4.0	20	0	0.0	36254	9.5
2	20	20	737	5.8	20	0	0.0	28451	12.6
3	20	20	201	1.5	23	3	15.0	1500000	564.2
4	20	20	938	5.9	22	2	10.0	1500000	444.7
5	20	20	67	0.6	22	2	10.0	1500000	404.6
6	20	20	1541	9.0	22	2	10.0	1500000	415.2
7	20	20	63717	284.1	22	2	10.0	1500000	485.7
8	**20**	**21**	67000	295.3	22	1	4.8	1500000	395.9
9	20	20	1139	6.8	22	2	10.0	1500000	451.6
10	20	20	938	6.2	20	0	0.0	26983	9.6
11	20	20	2680	**15.3**	20	0	0.0	1783	**0.9**
12	20	20	67	0.6	20	0	0.0	13325	6.3
13	20	20	335	**2.5**	20	0	0.0	6450	**1.5**
14	20	20	871	4.7	22	2	10.0	1500000	385.0
15	20	20	1005	5.9	22	2	10.0	1500000	400.8
16	20	20	469	3.4	23	3	15.0	1500000	537.4
17	20	20	335	2.2	23	3	15.0	1500000	528.3
18	20	20	1541	9.2	22	2	10.0	1500000	429.9
19	**20**	**21**	67000	281.1	22	1	4.8	1500000	385.6
20	20	20	201	1.6	22	2	10.0	1500000	399.5
Averages		*10569*		*47*		*1*	***7.2***		*313*
				47sec					***5min***

Table 5 Triplets, 60 items.

Run	Theo	HGGA			MTP				
		Bins	Evals	Time	Bins	Loss	Loss%	Backs	Time
1	40	40	6633	120.9	44	4	10.0	1500000	844.3
2	40	40	4824	104.0	43	3	7.5	1500000	823.0
3	40	40	4489	95.9	43	3	7.5	1500000	956.4
4	40	40	1273	39.1	44	4	10.0	1500000	859.3
5	40	40	2948	75.8	45	5	12.5	1500000	1184.4
6	40	40	7839	148.5	45	5	12.5	1500000	1188.7
7	40	40	1541	47.2	45	5	12.5	1500000	1054.3
8	40	40	2680	61.4	43	3	7.5	1500000	777.3
9	40	40	1273	36.9	43	3	7.5	1500000	642.9
10	40	40	15343	255.5	44	4	10.0	1500000	1002.6
11	40	40	4623	102.9	44	4	10.0	1500000	885.7
12	40	40	1742	49.5	45	5	12.5	1500000	979.9
13	40	40	1742	42.5	44	4	10.0	1500000	1013.5
14	40	40	1742	57.3	44	4	10.0	1500000	834.8
15	40	40	1541	40.9	44	4	10.0	1500000	824.3
16	40	40	1541	46.8	44	4	10.0	1500000	873.0
17	40	40	4422	93.0	43	3	7.5	1500000	629.3
18	40	40	2010	51.1	44	4	10.0	1500000	790.2
19	40	40	3015	67.3	46	6	15.0	1500000	1171.1
20	40	40	1541	40.1	45	5	12.5	1500000	1075.5
Averages		*3368*		*79*		*4*	***10.3***		*921*
				1.5min					***15min***

Table 6 Triplets, 120 items.

Run	Theo	HGGA			MTP				
		Bins	Evals	Time	Bins	Loss	Loss%	Backs	Time
1	83	83	6365	322.9	93	10	12.0	1500000	2381.4
2	83	83	3149	226.8	88	5	6.0	1500000	1526.4
3	83	83	2881	217.4	88	5	6.0	1500000	1455.3
4	83	83	21105	723.0	90	7	8.4	1500000	1717.3
5	83	83	7638	382.1	91	8	9.6	1500000	2513.4
6	83	83	46699	1716.6	90	7	8.4	1500000	2176.9
7	83	83	42277	1473.6	90	7	8.4	1500000	2107.5
8	83	83	131923	4399.6	92	9	10.8	1500000	2492.9
9	83	83	15477	614.5	91	8	9.6	1500000	2437.5
10	83	83	4556	318.2	90	7	8.4	1500000	1522.3
11	83	83	18626	776.9	94	11	13.3	1500000	2814.8
12	83	83	2345	191.2	90	7	8.4	1500000	1687.5
13	83	83	4489	261.9	89	6	7.2	1500000	1608.2
14	83	83	7035	360.2	91	8	9.6	1500000	2362.9
15	83	83	3551	203.6	89	6	7.2	1500000	1398.6
16	83	83	871	75.3	91	8	9.6	1500000	2682.7
17	83	83	17420	667.4	90	7	8.4	1500000	2080.8
18	83	83	4288	306.5	90	7	8.4	1500000	2086.3
19	83	83	5159	293.5	91	8	9.6	1500000	2237.3
20	83	83	29011	1024.9	91	8	9.6	1500000	2199.2
Averages			*18743*	*728*		*7*	***9.0***		*2074*
				12min					***35min***

Table 7 Triplets, 249 items.

Run	Theo	HGGA			MTP				
		Bins	Evals	Time	Bins	Loss	Loss%	Backs	Time
1	167	167	3752	1806.7	184	17	10.2	1500000	5828.9
2	167	167	3551	1582.2	181	14	8.4	1500000	3437.4
3	167	167	1809	1234.5	177	10	6.0	1500000	2358.7
4	167	167	3082	1821.9	180	13	7.8	1500000	3398.0
5	167	167	5360	2355.2	181	14	8.4	1500000	3709.8
6	167	167	2881	1424.0	183	16	9.6	1500000	10624.4
7	167	167	1809	1161.4	183	16	9.6	1500000	5788.5
8	167	167	2613	1503.7	183	16	9.6	1500000	5798.9
9	167	167	3685	2138.4	177	10	6.0	1500000	2991.3
10	167	167	3082	1550.1	185	18	10.8	1500000	5626.3
11	167	167	2010	1052.9	179	12	7.2	1500000	3771.4
12	167	167	2814	1334.9	178	11	6.6	1500000	3063.7
13	167	167	3216	1502.2	187	20	12.0	1500000	5787.1
14	167	167	5293	1951.0	181	14	8.4	1500000	4494.9
15	167	167	3216	1473.9	183	16	9.6	1500000	5929.5
16	167	167	4623	2350.6	181	14	8.4	1500000	5306.9
17	167	167	2613	1178.8	183	16	9.6	1500000	5522.0
18	167	167	3551	1754.2	183	16	9.6	1500000	6277.2
19	167	167	3484	1775.5	180	13	7.8	1500000	4164.2
20	167	167	4288	2307.2	188	21	12.6	1500000	6519.4
Averages			*3337*	*1663*		*15*	***8.9***		*5020*
				28min					***84min***

Table 8 Triplets, 501 items.

The respective performances of the HGGA and the MTP procedure inspire an important conclusion. Both methods use the same *local* techniques (both FFD and the Dominance Criterion are embedded in MTP), but they use different *global* mechanisms for generating candidate solutions (like all GAs, the HGGA uses crossover, while the MTP follows a branch-and-bound search tree). Given the superiority of the HGGA demonstrated above, we hope that the search mechanism of the GA will be recognized as a *very* viable instrument in searching the vast search spaces of difficult problems. Note however, that the success of the HGGA is due to two key components: an encoding and operators that fit the structure of the problem, and a sophisticated local optimization. We believe that *both* are necessary in any high-performance GA.

Besides the Bin Packing Problem, the GGA holds a promise for many other grouping problems. If the concept of dominance of Martello and Toth can be shown to carry over to other domains, then the marriage of the two paradigms should prove useful for other grouping problems as well.

Acknowledgments

Many thanks to André van Vliet for the original FORTRAN code of the MTP procedure, as well as for suggesting the triplets instances. Thanks to John Beasley from Imperial College for help with including the test instances into the OR-Library.

Thanks to the referees for suggestions that improved the presentation.

The work described in this paper was partly supported by the ESPRIT III project SCOPES.

References

[Belew and Booker, 91] Belew Richard K. and Booker Lashon B. (Eds) *Proceedings of the Fourth International Conference on Genetic Algorithms*, University of California, San Diego, July 13-16, 1991, Morgan Kaufmann Publishers, Inc., San Mateo, CA.

[Bhuyan et al., 91] Bhuyan Jay N., Raghavan Vijay V. and Elayavalli Venkatesh K. *Genetic Algorithm for Clustering with an Ordered Representation* in [Belew and Booker, 91].

[Davis, 91] Davis Lawrence (Ed.) *Handbook of Genetic Algorithms* Van Nostrand Reinhold, New York.

[Ding et al., 92] Ding H., El-Keib A.A. and Smith R.E. *Optimal Clustering of Power Networks Using Genetic Algorithms* TCGA Report No. 92001, March 5, 1992, University of Alabama, Tuscaloosa, AL.

[Falkenauer and Delchambre, 92] Falkenauer Emanuel and Delchambre Alain *A Genetic Algorithm for Bin Packing and Line Balancing*, in "Proc. of the IEEE 1992 Int. Conference on Robotics and Automation (RA92)", May 10-15, 1992, Nice, France.

[Falkenauer, 93] Falkenauer Emanuel *The Grouping Genetic Algorithms - Widening the Scope of the GAs* in JORBEL - Belgian Journal of Operations Research, Statistics and Computer Science, vol. 33 (1, 2), pp.79-102.

[Falkenauer, 94] Falkenauer Emanuel *New Representation and Operators for GAs Applied to Grouping Problems*, in Evolutionary Computation, Vol.2, N.2, pp. 123-144.

[Falkenauer, 95] Falkenauer Emanuel *Solving Equal Piles with a Grouping Genetic Algorithm*, in Eshelman L.J. (Ed.) Proceedings of the Sixth International Conference on Genetic

Algorithms, Morgan Kaufmann Publishers, San Francisco, pp. 492-497.

[Garey and Johnson, 79] Garey Michael R. and Johnson David S., *Computers and Intractability - A Guide to the Theory of NP-completeness*, W.H.Freeman Co., San Francisco, USA.

[Goldberg, 89] Goldberg David E., *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wessley.

[Goldberg et al., 91] Goldberg David E., Deb Kalyanmoy and Korb Bradley *Don't Worry, Be Messy* in [Belew and Booker, 91].

[Grefenstette, 85] Grefenstette John J. (Ed) *Proceedings of the First International Conference on Genetic Algorithms and their Applications*, Carnegie-Mellon University, Pittsburgh, PA, July 24-26, 1985, Lawrence Erlbaum Associates, Publishers, Hillsdale, NJ.

[Hinterding and Khan, 94] Hinterding Robert and Khan Lutfar *Genetic Algorithms for Cutting Stock Problems: with and without Contiguity*, accepted to the Australian AI'94 Workshop on Evolutionary Computation.

[Holland, 75] Holland John H. *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor.

[Jones and Beltramo, 91] Jones Donald R. and Beltramo Mark A. *Solving Partitioning Problems with Genetic Algorithms* in [Belew and Booker, 91].

[Martello and Toth, 90a] Martello Silvano and Toth Paolo *Lower Bounds and Reduction Procedures for the Bin Packing Problem*, in Discrete Applied Mathematics, vol. 22, North-Holland, Elsevier Science Publishers B.V., pp.59-70.

[Martello and Toth, 90b] Martello Silvano and Toth Paolo *Bin-packing problem*, Chapter 8 in Knapsack Problems, Algorithms and Computer Implementations, John Wiley & Sons Ltd., England, pp.221-245.

[Männer and Manderick, 92] Männer Reinhard and Manderick Bernard (Eds) *Parallel Problem Solving from Nature, 2*, Proceedings of the Second Conference on Parallel Problem Solving from Nature (PPSN2), Brussels, Belgium, September 28-30, 1992, North-Holland, Elsevier Science Publishers B.V., Amsterdam, The Netherlands.

[Radcliffe, 91] Radcliffe Nicholas J. *Forma Analysis and Random Respectful Recombination* in [Belew and Booker, 91].

[Radcliffe, 92] Radcliffe Nicholas J. *Non-linear Genetic Representations* in [Männer and Manderick, 92].

[Reeves, 94] Reeves Colin *Hybrid Genetic Algorithms for Bin-packing and Related Problems*, working paper, submited to Annals of OR "Metaheuristics in Combinatorial Optimization, G. Laporte and I.H. Osman (Eds), Baltzer, Bazel, 1995.

[Smith, 85] Smith Derek *Bin Packing with Adaptive Search* in [Grefenstette, 85].

[Syswerda, 89] Syswerda Gilbert *Uniform Crossover in Genetic Algorithms* in Schaffer D.J. (Ed.), Proceedings of the Third International Conference on Genetic Algorithms, Morgan Kaufmann Publishers, San Mateo, CA.

[Van Driessche and Piessens, 92] Van Driessche Raf and Piessens Robert *Load Balancing with Genetic Algorithms* in [Männer and Manderick, 92].

[van Vliet, 93] van Vliet André, Econometric Institute, Erasmus University Rotterdam, private communication.

[Von Laszewski, 91] Von Laszewski Gregor *Intelligent Structural Operators for the k-way Graph Partitioning Problem* in [Belew and Booker, 91].

[Vose and Liepins, 91] Vose Michael D. and Liepins Gunar E. *Schema Disruption* in [Belew and Booker, 91].