Belgian Journal of Operations Research, Statistics and Computer Science Vol. 33 (1,2)

The Grouping Genetic Algorithms : widening the scope of the GAs

E. Falkenauer

CRIF – Research Centre for Belgian Metalworking Industry

Department of Industrial Automation

CP 106 – P4

50, Av. F. D. Roosevelt

B-1050 Brussels, Belgium

e_mail: PIERRE_LECOCQ_CRIF@eurokom.ie

Abstract

An important class of computational problems are grouping problems, where the aim is to group members of a set, i.e. to find a good partitioning of the set. We show why both the classic and the ordering GAs fare poorly in this domain by pointing out their inherent difficulty to capture the regularities of the "functional landscape" of grouping problems. We then propose a new encoding scheme and genetic operators adapted to these problems, yielding the Grouping Genetic Algorithm (GGA) paradigm. We illustrate the approach with three examples of important grouping problems successfully treated with the GGA: the problems of Bin Packing and Line Balancing, Economies of Scale, and Conjunctive Conceptual Clustering applied to the problem of creation of part families.

Keywords : Genetic algorithm, grouping, partitioning, solution encoding

1. Introduction - the Grouping Problems

Many problems naturally arising in practice consist in *partitioning* a set **U** of objects 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 objects, where each object is in exactly one group, i.e. to find a *grouping* of those objects.

In most 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 object cannot be grouped with all possible subsets of the remaining objects.

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 optimize:

Problem	*Hard Constraint*	*Cost Function*
Bin Packing	Sum of sizes of objects in any group<C	Number of groups, Min
Workshop Layouting	Number of machines in any group<C	Intercell Traffic, Min
Graph Coloring	No connected nodes in any group	Number of groups, Min

As can be seen, the grouping problems are characterized by cost functions which depend on the *composition of the groups*, that is, where one *object* taken isolately has no meaning.

Often the cost function improves as the size (in terms of number of objects) of the groups grows or, conversely, as the number of the groups decreases. Of course, the hard constraints forbid the trivial 'solution' consisting of putting all the objects into the same unique group.

Many grouping problems (including all the examples above) are NP-hard ([Garey and Johnson,79]), which implies that any known *exact* algorithm (i.e. one which would be guaranteed to find the global optimum) will run in time *exponential* in the size of the problem instance. Such an algorithm is thus in most cases unusable for real-world size problems.

Since their introduction some 15 years ago by [Holland,75], the GAs have been extensively studied and applied to a large variety of problems, including machine learning and NP-hard optimization problems (e.g. [Davis, 87], [Goldberg,89], [Holland et al.,86], [Grefenstette,85], [Grefenstette,87], [Schaffer,89], [Belew and Booker, 91], [Männer and Manderick, 92], etc.). Their encouraging success with these latter problems makes the GAs a candidate for solving the grouping problems.

For space reasons, we will assume in the sequel that the reader is familiar with the basics of the GA paradigm. The literature on the topic is already quite extensive and the article by Marc Pirlot in the first volume of this issue can be consulted if necessary.

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. The application of a straightforward encoding scheme (see below)

---

[1] There are many more grouping problems of great practical importance, but their description would be too lengthy for the purpose of this paper. Three more will be described in the sequel together with the corresponding GGA.

together with classic 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], [Laszewski, 91]). We will show why we think this approach cannot yield conclusive results on these problems.

2.1 The Crossover

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

2.1.1 Schema Disruption

Let's assume the most straightforward encoding scheme, namely one gene per object. For example, the chromosome

ADBCEB

would encode the solution where the first object 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. Note that the same allele was selected for the third and the sixth objects - the objects are thus *grouped* in the same group.

Since, by definition of the problem, the cost function improves with growing size of groups, the grouping of the two objects into one group constitutes a gain and should be thus transmitted to the next generations. However, the two genes are positioned too far from each other on the chromosome to be safe against disruption during the crossover. Indeed, the probability that a crossing site will fall between the two of them grows with their distance.

We must thus find a way to shorten this kind of schema. The standard way to do this is through *inversion*. By storing the loci together with the alleles, the same chromosome, written this time

123456
ADBCEB,

could be arranged by (possibly successive) inversion(s) into

123654
ADBBEC.

This chromosome has the interesting genes tightly together. This time the group of Bs has good chances to survive a crossover.

The problems begin when the groups become larger, that is, incidentally, when the optimization process starts to develop a good solution of the grouping problem. Indeed, in a chromosome like

123654
ADBBBB,

the probability of disruption of the very promising group of four Bs is as high as it was for the group of two Bs without inversion. We thus see that inversion cannot help in assuring good survival rates of schemata relevant to the grouping problem. This is because the *good* schemata for this problem are, by definition, *long* schemata (of similar objects).

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. The result is, of course, an algorithm stagnating on poor, never improving solutions.

The proverbial smart reader could now object that if the size of the groups is bounded above by a sufficiently tight hard constraint (e.g. a sufficiently small constant C in the Workshop Layouting Problem above), the argument of *long* schemata doesn't hold. It should be remembered, however, that we are interested in algorithms for *arbitrary* instances of the grouping problems, that

is including those where the hard constraints are loose.

2.1.2 *Encoding Redundancy and Context Insensitivity*

Among the six *design principles* for constructing useful representations (see e.g. [Radcliffe, 92]) figures the principle of Minimal Redundancy - each member of the search space (here the space of all valid groupings) should be represented by exactly one chromosome, in order to reduce the size of the space the GA has to search.

The classic encoding above is highly redundant. Indeed, the cost function of a grouping problem depends only on the grouping of the objects, rather than the numbering of the group. For instance, in the Graph Coloring Problem, only the resulting number of colors counts, 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 (i.e. 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).

Not only such an encoding increases tremendously the space the GA must search but, even worse, it leads to the highly undesirable effect of casting context-dependent information *out of context* under the standard crossover. Indeed, in the first chromosome above, 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 the crossover, has disastrous consequences. To see this, let us apply the standard two-point crossover (with the crossing sites at the first and third position) 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. Indeed, in an ideal crossover, a part of the genetic contents of a child is inherited from one parent, and the rest from the other parent (i.e. no mutation takes place during the recombination). Consequently, if the parents are identical, *all* of the contents is inherited from that unique parent, i.e. the child is a copy of the parent[2]. 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[3]!

In other words, while the schemata are well transmitted with respect to the *chromosomes* under the standard 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 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, in the case of grouping.

Consider for example the following chromosome :

---

[2] [Radcliffe, 92] calls *respect* the transmission of the part of genetic contents *both* parents agree on.

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

ABDBAC.

A mutation of this individual could yield

ABDEAC,

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

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

AAABBB

would lead, for example, to

AAEBBB.

On the one hand, the allele E 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 objects 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. Indeed, the search is based on (statistical) sampling of schemata, with transmission of well-scoring schemata to the next generations (implying increased sampling rate of those schemata, according to the best sampling strategy under uncertainty, established by [Holland, 75]). However, with a life expectancy of just one generation, the schemata represented by the individuals mutated by the classic mutation have no chance of being correctly sampled and their eventual apport is lost.

In other words, the classic mutation is *too destructive* once the GA begins to reach a good solution of the grouping problem.

Apart from the fact that the mutated individuals are of hardly any use for the genetic search as shown above, they represent another nuisance: their presence effectively reduces the number of reasonably good individuals in the population. This in turn impairs the benefit of *implicit parallelism* (which depends on the population size), the most important strength of the GA paradigm. Indeed, only the schemata which can be reasonably sampled during the genetic search can contribute to its success. However, when a part of the population is formed of individuals that are so short-lived that they *cannot* contribute to the search, they reduce the number of individuals which do contribute. This means that even the reasonably good individuals (i.e. the schemata *they* represent) receive a smaller sampling rate, due to the reduced effective size of the population.

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. The 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][4], [Bhuyan et al., 91], [Jones and Beltramo, 91]). We will again show why we think this approach cannot yield the expected results on these problems.

3.1 Encoding Redundancy

Another way of handling grouping problems is to represent permutations of the objects

[4] This reference is especially interesting with respect to this paper. See section 5.1.1 below.

(members of the set **U** in section 1. above), and use a decoding mechanism that reveals the actual assignment of the objects to groups (the resulting partition of the set) corresponding to each chromosome. The decoding mechanism usually proceeds by considering the objects 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 (BPP) in the sequel. Suppose there are ten objects to pack, numbered 0 through 9. A valid chromosome is one where each object appears exactly once, for example

`0123456789.`

The solution of the BPP, i.e. the actual assignment of the ten objects to bins, corresponding to that chromosome, is obtained by the following decoding. Take the objects in the order given by the chromosome (say from the left to the right, i.e. first the object 0, then the object 1, then 2, and so on), and assign each to the first bin that can accomodate it (i.e. where there is enough space left). Each time there is no such bin, request a new one.

As said above, redundancy is a highly undesirable feature of an encoding scheme. The one outlined above is highly redundant. Indeed, suppose that the objects in the above chromosome are partitioned as follows

`0123|45678|9,`

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

`3210|45678|9` or
`0123|87654|9`, or
`87645|1032|9`,

to give just three of them, encodes *the same* solution of the original Bin Packing Problem. In fact, the redundancy of the encoding, i.e. the number of distinct *chromosomes* which encode the same *solution* of the original problem, grows *exponentially* with the size of the instance. That means that the GA spends an exponential time "exploring" the artificially large search space of *identical solutions*, yet different chromosomes. Such a waste of time impairs substantially the power of the algorithm.

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 above chromosome

`0123456789,`

and suppose it is decoded from the left to the right as above. This chromosome encodes a solution where objects 4 through 8 are in the same group. However, this information depends on the genes to the left of the group, i.e. in 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 objects, a bin filled with either the objects 9,1,2 or 9,0,1, respectively, can be so filled that no other object 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 objects 4 through 8 in the same bin.

There is a number of ordering crossovers available in the literature nowadays. For the sake of clarity, let's concentrate on Goldberg's PMX. It is probably the most widely used ordering crossover, thanks to the theoretical treatment of its capacity to transmit order-schemata (*o-schemata*, the ordering equivalent of Holland's schemata) given in [Goldberg, 89]. The PMX transmits well the *absolute* position of genes on the chromosome. Now the only difference, w.r.t. the position of the group of genes 45678, between the chromosomes

0123456789 and

9123456780

consists in a permutation of the first and last genes (objects) - the information the PMX transmits is identical in the two chromosomes. That means the latter chromosome could be a child of the former, even though the two encode *very* different solutions of the BPP. In other words, the PMX transmits information which more often than not gets a different meaning in the new chromosome.

One could still argue that "more often than not" is not *allways*, meaning that the crossover could still be useful. However, the effects of such an operator become essentially random. Thus the "crossover" basically turns into a mutation, implying the loss of the most potent device of the GA.

Other ordering crossovers suffer of similar problems. The reason is that *o-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. Consequently, in grouping problems, sampling the *o-schemata* is of little use for estimating the quality of the corresponding solutions.

3.2.2 Schema Disruption

We have shown in section 2.1.1. that an encoding mapping *objects* onto genes in a chromosome leads to high probabilities of disruption of useful parts of the chromosome. Since the permutation-based encoding also maps objects 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 redundacy of the encoding, or
- being too destructive, if the modification occurs near the head of the chromosome.

Once again, these problems stem from the object-oriented encoding.

4. The Grouping Genetic Algorithm

The Grouping Genetic Algorithm (GGA) differs from the classic GA in two 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.

4.1 The Encoding

As we have seen, neither the standard nor the ordering genetic operators are suitable for grouping problems. Unlike with the deceptive problems of [Goldberg, 87], the strong schemata are not misleading. Rather, they do not survive the very genetic search supposed to improve them.

The main reason is that the structure of the simple chromosomes (which the above operators work with) is *object* 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. That is a serious drawback: the fact that the *object* i is in the group j is meaningless - it is the fact that the *group* j contains certain object(s) (including i) that is important. Of course, given the distribution of the objects to the groups, one can always compute the state of the groups, but such an information is far *too indirect* for the GA (i.e. the operators) to be taken into account efficiently.

Note that these remarks are nothing more than a call for compliance with the *Thesis of Good Building Blocks*, central to the GA paradigm.

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. For example, the first and the next to last chromosomes from section 2 would be encoded as follows:

`ADBCEB:BECDA`
`AAABBB:AB,`

with the group part written after the semicolon.

More concretely, let us consider the first of these two chromosomes and the Bin Packing Problem. Numbering the objects from `0` through `5`, the *object* part of the chromosome can be explicitely written

`012345`
`ADBCEB: ...,`

meaning the object `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* (i.e. bins). 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* (and the only practical use) of the object part. Indeed, by a lookup there, we can establish what the meaningless names stand for. Namely,

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

In fact, the chromosome could also be written in a less visual way as

`{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 object part of the chromosomes merely serving to identify which objects 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 of a solution, 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, through 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 ***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.

4.2 The Crossover

As pointed out in the previous section, a crossover for a grouping problem will have to work

with variable length chromosomes, as the number of groups (i.e. the number of genes in the chromosomes) cannot be fixed in advance, being (directly or indirectly) the object of the optimization.

The variable length of chromosomes is not the only particularity of the GGA as compared to the classic GA. In the latter, during a crossover, genes are combined *independently* each of the others, in a cut-and-concatenate manner[5]. Since the GGA has one gene per group, rather than object, it is almost never the case that the chromosomes can be combined by simply joining together some genes from one parent and some genes from the other without altering any of them. Indeed, especially when the groups contain many objects, it is extremely rare that the objects contained in the groups coming from the first parent form, *all* and *alone*, groups in the second parent (there is a *partial* overlap among the groups in the two parents).

Depending on the hard constraints, in some grouping problems, a cut-and-concatenate crossover would produce invalid individuals. In others, the progeny obtained could be valid, but of very poor quality.

Dealing with these problems implies an *adaptation* of some groups in one parent to those in the other. Thus the progeny will, in most cases, contain some groups (i.e. genes) which are not present in exactly the same form in either parent.

Given the fact that the hard constraints and the cost function vary among different grouping problems, 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. Recall that the crossover works with the group part of the chromosome, so that this means selecting some of the *groups* in each parent.
2. *Inject* the contents of the crossing section of the first parent at the first crossing site of the second parent.
3. Eliminate all *objects* now occurring twice from the groups they were members of in the second parent (thus the 'old' membership of these objects gives way to the membership specified by the 'new' injected groups). Thus *some* of the 'old' groups coming from the second parent are altered: they do not contain the same objects anymore, since some of those objects had to be eliminated.
4. If necessary, *adapt* the resulting groups, according to the hard constraints and the cost function to optimize.
5. Apply the points 2 through 4 to the two parents with their roles permuted in order to generate the second child.

We will give below more detailed examples of crossovers adapted to three different grouping problems.

4.3 The Mutation

As pointed out in section 3.2, the classic mutation operator (applied to chromosomes obtained with the straightforward encoding) is too destructive for grouping problems. As for the crossover, this is due to the fact that the classic operator is much too *object* oriented.

A mutation operator for grouping problems must work with groups rather than objects. As

---

[5] In many cases the resulting chromosome is subject to en extensive decoding before the phenotype (i.e. the value of the cost function of the solution it represents) can be obtained. Such an indirect relation between a solution and its genetic representation is a nuisance: the GA only sees the genotype (i.e. chromosomes), and a simple crossover cannot take into account the intricacies of solution coding/decoding.

for the crossover, the implementation details of the operator depend on the particular grouping problem on hand. Nevertheless, two general strategies can be outlined: either *create* a new group or *eliminate* an existing group.

4.4 The Inversion

The inversion operator serves to shorten good schemata in order to facilitate their transmission from parents to offspring (thus ensuring the required high rate of sampling of the above-average schemete by the GA).

Since the classic inversion operator is insensible to the length of the chromosome and does not modify the genetic information represented by the chromosome (i.e. an inverted chromosome still denotes the same phenotype), it can be applied without modification to the chromosomes used in a Grouping GA. However, it will be applied to the *group* part of the chromosome, as the genes represent the groups.

Thus for instance, the chromosome

ADBCEB : BECDA

could be inverted into

ADBCEB : CEBDA.

Note that the object part of the chromosome stays unchanged. Indeed, the groups are still composed of the same objects - only the position on the chromosome of the representation of the groups has changed.

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 schemata easier.

5. Three GGA Applications

In this section, we will describe three concrete grouping problems which the Grouping GA has been successfully applied to. In all three of them, the encoding scheme outlined in the section 4.1 above and the inversion operator from section 4.3 have been used. We will thus concentrate in the sequel on the details of the crossover and mutation operators, after a definition of each of the problems.

5.1 The Bin Packing and Line Balancing

This section is made up of excerpts from [Falkenauer and Delchambre, 92a]. This GGA has been implemented as a module of the ROBCAD robotics-oriented CAM software (see [Delchambre et al, 92] and [Falkenauer and Delchambre, 92b]), where it serves for optimization of (robotized) assembly lines.

5.1.1 The Problem Definition

The bin packing problem (BPP) is defined as follows ([Garey and Johnson, 79]): given a finite set O of numbers (the object sizes) and two constants B (the bin size) and N (the number of bins), is it possible to 'pack' all the objects 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 B?

This NP-complete decision problem naturally gives rise to the associated NP-hard optimization problem, the first subject of this section what is the *best* packing, i.e. how many bins are necessary to pack all the objects into (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 (and there will most probably never be one). 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 objects 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 object there, if not, request a new bin. Putting the object into the first available bin found yields the First Fit (FF) heuristic. Searching for the most filled bin still having enough space for the object 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.

Note that the problem treated by [Smith, 85], in spite of being referred to in the reference as Bin Packing, differs from the BPP defined above in two aspects. First, [Smith, 85] considers *two-dimensional* objects, whereas we consider the *one-dimensional* problem of [Garey and Johnson, 79]. Second, more importantly, [Smith, 85] tries to pack *as many as possible* of the objects into *one* "bin" (actually a rectangular area), whereas we seek to pack *all* the objects into as few *bins* as possible. Clearly, these are very different problems. Indeed, Figure 1, where Δ is an arbitrarily small positive value, constitutes a proof that there are instances of the BPP where the optimal solution contains only sub-optimally packed bins, even though at least one optimally packed bin can be found. Hence a method using the algorithm of [Smith, 85] will find the optimal bin and miss the BPP optimum.

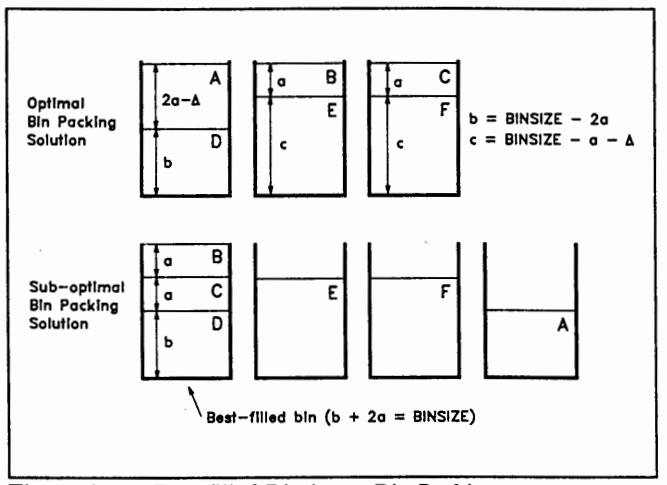

Figure 1 Best-filled Bin is not Bin Packing

To the best of our knowledge, the BPP above has not been treated elsewhere by a GA.

The line balancing problem (LBP) can be described as follows: given a set of tasks of various lengths, subject to precedence constraints (i.e. some tasks have as prerequisite the completion of one or more other tasks, see [Sacerdoti,77]), and a time constant called *cycle time*, how should be the tasks distributed over workstations along a production (assembly) line, so that (1) no workstation takes longer than the cycle time to complete all the tasks assigned to it and (2) the precedence constraints are complied with?

In more formal terms, we define the LBP as the following decision problem:

Given a directed acyclic graph G=(T,P) (the nodes T representing the tasks and the arrows P representing the precedence constraints) with a constant L_i (task length) assigned to each node T_i, a constant C (the cycle time) and a constant N, can the nodes T be partitioned into N or less subsets S_j (the j-th station's tasks) in such a way that (1) for each of the subsets, the sum of L_is associated with the nodes in the subset doesn't exceed C, and (2) there exists an ordering of the subsets such that whenever two nodes in distinct subsets are joined by an arrow in G, the arrow goes from a higher-ordered (earlier) to a lower-ordered (later) subset?

It is easy to show that the LBP is NP-complete: it can be reduced to the NP-complete BPP, which it contains as a special case (namely, the set of precedence constraints, the arrow set P, is empty for BPP). Needless to say, the associated optimization problem, where we ask what is the

minimum number of stations required, the second subject of this section, is NP-hard.

Let us concentrate on the BPP and try to define a suitable cost function to optimize. 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 objects. This is correct from the 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 a sea of points where this purported cost function is just one unit above the optimum. More importantly, 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.

Consider the extreme case where just one arrangement of the objects yields the optimum of, say, N bins. The number of possible arrangements yielding N+1 bins grows exponentially with N and is thus very large even for small problem sizes. Nevertheless, all these points in the search space yield the same cost of N+1 and thus appear to be absolutely equal in terms of merit for searching their surroundings. In other words, an algorithm would have to run into the optimal solution by mere chance. That would be impractical, to say the least.

We thus have to find a cost function which assigns similar (but not equal) values to similar solutions, while having the same optima as the function above. In other words, we have to identify the smallest natural piece of a solution which is meaningful enough to convey information about the expected quality of the solution it's part of.

Fortunately, this is easy to do in the case of BPP: the bin points itself out as the natural 'information quantum'. We simply realize that the better *each* of the bins is used, the fewer bins one needs to pack all the objects in. Or, conversely, a bad use of bins' capacity leads to the necessity of supplementary bins, in order to contain the objects not packed into the wasted space.

In order to champion the bin, rather than the overall performance of all the bins together, we also have to account for the following: if we take two bins and shuffle their contents among them, the situation where one of the bins is nearly full (leaving the other one nearly empty) is better than when the two bins are about equally filled. This is because the nearly empty bin will more easily accommodate additional objects which could otherwise be too big to fit into either of the half-filled bins.

We thus settled for the following cost function for the BPP: maximize

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

with N being the number of bins used,
$fill_i$ the sum of sizes of the objects in the bin i,
C the bin capacity and
k a constant, k>1.

In other words, the cost function to maximize is the average, over all bins, of the k-th power of 'bin efficiency' measuring the exploitation of a bin's capacity.

The constant k expresses our concentration on the well-filled 'elite' bins in comparison to the less filled ones. Should k=1, only the total number of bins used would matter, contrary to the remark above. The larger k is, the more we prefer the 'extremists', 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.

5.1.2 *BPCX - the Bin Packing Crossover*

A crossover's job consists in producing offspring out of two parents in such a way that the children inherit as much of the meaningful information from *both* parents as possible. Since it is the bin 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 to cross (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 objects appear twice in the solution and must be thus eliminated. Suppose the objects 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 objects which were not injected with the bins c and d. Those objects are thus missing from the solution. To fix this last problem, we apply the FFD heuristic to reinsert them, yielding, say

`AcdBD`*x*,

where *x* are one or more bins formed of the reinserted objects.

As can easily be seen, the child just constructed indeed inherited important information from *both* parents, namely the bins `A`, `B` and `D` from the first and `c` and `d` from the second. Note, however, that the bins `A`, `B` and `D` might not be exactly the original ones found in the first parent, because the FFD might have filled them up with some of the objects reinserted in the last stage of the BPCX. Nonetheless, this is actually beneficial, since it leads to bins even better filled than in the parent.

5.1.3 *The Mutation*

The mutation operator for the BPP is very simple: given a chromosome, we select at random a few bins (i.e. *groups*) and eliminate them. The objects which composed those bins are thus missing from the solution and we use the FF to insert them back in a *random order*.

In order to improve the chances of the mutation to improve the current solution, we follow two rules: the emptiest bin is always among the eliminated ones, and we always eliminate and subsequently reinsert at least three bins (the number of used bins cannot be improved with less).

Note that, as with the BPCX, even some of the bins not selected for elimination might be filled up with objects from the eliminated bins.

5.1.4 *Experimental Results*

Since the First Fit Descending can be shown to be a good heuristic for the BPP, it constituted a benchmark in tests of performance of our GA approach.

We constructed the test data as follows: we first generated objects of random sizes admitting a *perfect packing* (i.e. f_{BPP}=1), and then subtracted from randomly chosen objects a total of

LEEWAY percentage of the size of one bin. For example, with LEEWAY set to 3% and the bin size of 255, the total size of the objects was 7.65 less than the total capacity of the bins in the perfect packing. Thus the test reflected the ability of the algorithm to get as *close* as possible to the optimum, rather than its eventual ability to find *the* optimum perfect packing. Indeed, the latter test would be a test of optimality, which would be equivalent to asking whether we can solve in a reasonable time an NP-complete decision problem - something the algorithm wasn't and couldn't be designed to.

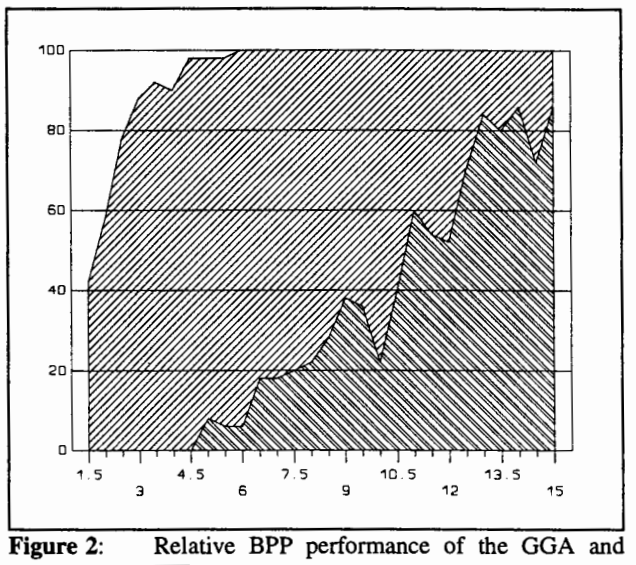

Figure 2: Relative BPP performance of the GGA and FFD.

Since the total size of the objects was only a fraction of the bin size less than the total capacity of the bins in the perfect packing, the optimum number of bins hasn't changed. Hence we observed the ability of the algorithm to reach that number of bins, compared to the FFD heuristic. However, in order to account for a practical use of the GA, we required it to reach the optimum number of bins in at most 5000 generations.

The results are summarized in Figure 1. It shows the average proportion, over 50 successive runs, of test instances of 64 objects successfully optimized by the GA and FFD respectively, function of the LEEWAY (1.5 through 15% of the bin size). The chart shows the net superiority of the GA in 'tough' conditions, i.e. when the space for the objects to pack is tight.

The running times of the GA were of the order of a minute on a 4D35 Silicon Graphics (33 MIPS).

One could now object that the FFD heuristic is not a 'fair' benchmark for assessment of merits of the GGA approach, since much better algorithms for the Bin Packing Problem are available, e.g. the Reduction method of [Martello and Toth, 90]. However, a look at the crossover and mutation operators presented above reveals that the FFD and the simpler FF heuristics are the only 'domain knowledge' or 'expertise' exploited. Thus *all* of the observed difference of performance between the FFD and the GGA is really due to the GGA itself or, more precisely, to the recombinating power of the crossover.

5.2 The Economies of Scale

This section is made up of excerpts from [Falkenauer, 91a]. This GGA has been successfully applied to optimization of weekly steel production in a large belgian forge.

5.2.1 The Problem Definition

The Economies of Scale Problem is defined as follows: given a list of orders, for each of them a list of possibilities of executing it, and a cost function expressing the cost of producing each order in each of the possible ways, select the way of producing each of the orders in order to minimize the total cost of the production.

More formally, we consider the following optimization problem : Consider a set of objects O_i for which there is a set of possible attributes A_{ij}. Select for each object in O exactly one attribute from A_i in order to minimize a cost function f. The problem is grouping when f decreases with the size and increases with the number of *groups* of objects with *equal attributes*.

The problem becomes nontrivial when the cost function decreases with *economies of scale*, i.e. when *grouping* of several orders into batches produced in *the same way* accounts for a gain. Indeed, when the cost of switching from one production method to the other and/or producing too small quantities of goods in a uniform way grows, the fixed costs of producing each of the orders separately lose their importance in comparison with the gains attained by a mass production.

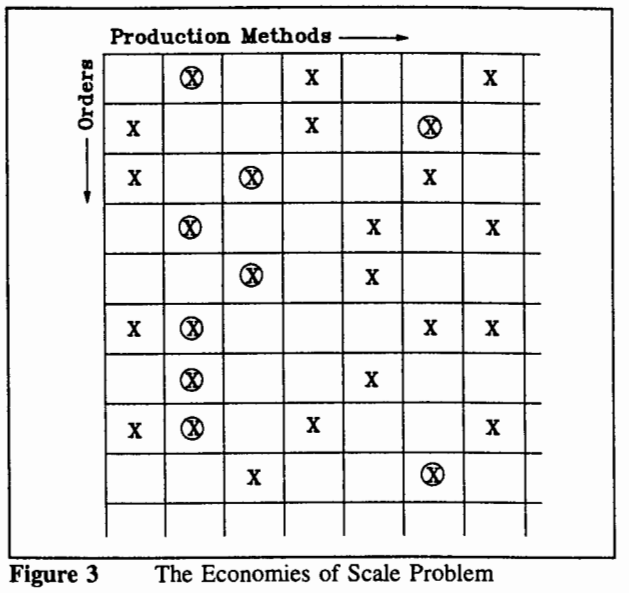

Figure 3 The Economies of Scale Problem

Figure 2 illustrates the problem. In that Figure, each cross indicates a production method (a column) which can be selected for the corresponding order (the line). The circled crosses, representing the methods actually selected, represent a valid solution of the problem, since there is just one cross per line. Note the large number of crosses in the second column: these orders are performed in the same way (in one batch), implying a big economy.

The Figure also illustrates the complexity of the problem when the cost of *non*grouping grows faster than linearly[6]: note that not *all* crosses in the third column were selected. This seemingly implies a higher cost of the production, *unless* one realizes that doing so would leave the cross in the second line separated, incurring an even higher overall cost.

5.2.2 GX - The Grouping Crossover

The grouping crossover proceeds by first copying the first parent into the future child. Then some of the groups of the second parent are injected into this child. The result is a child having some of the groups of the second parent and, as long as they were not destroyed by the injection, the groups of the first parent.

To see the function of the GX in more detail, consider the following group parts of two chromosomes :

`ABCDEFG` and
`abcdef`.

The GX proceeds first by selecting at random two crossing sites in each of the chromosomes, say

`AB|CD|EFG` and
`a|bcde|f`.

To construct the first child, the segment of the second parent delimited by the crossing sites is

---

[6] E.g. when the cost of producing one batch is (C - BATCHSIZE)^k, where k>1 and C>BATCHSIZE are constants.

injected into the copy of the first parent, yielding

AB|bcde|CDEFG.

Since the cost function decreases with the size of the groups, the injection of the groups is followed by an *expansion* of the injected groups, i.e. all the objects not injected with the new groups are scanned and converted, whenever possible, to one of the new (injected) groups.
Eventual double occurrences of groups, or groups eliminated by the expansion, are then eliminated, which gives in the case of the example

AbcdeFG,

the first child. The second child is obtained in the same way, but with the roles of the two parents permuted. This yields successively

a|bcde|f,
aCDbcdef, and
aCDbef.

Note that in this example both children indeed inherited important information from *both* parents. The first child inherited the groups b,c,d and e from the second and, supposing not all of their members were converted by the injection/expansion, the groups A,F and G from the first parent. The second child inherited the groups C and D from the first parent and a,b,e and f from the second.

5.2.3 Two Grouping Mutations

The first operator, the *grouping mutation*, proceeds as follows : it selects at random one of the possible attributes and tries to convert as many as possible of the objects of a chromosome into this group. Most probably, it will be possible to gather into the new group only some of the objects in the chromosome, because not all of them might have the new attribute in their list of possible attributes (A_i). Nevertheless, in comparison to the classic one-object mutation, the chances are much better that the new group will be strong enough not to decrease too much the fitness of the individual.

The second operator, the *eliminating mutation*, selects at random one of the groups present in a chromosome and then tries to eliminate it. This is done by scanning the objects in the group and converting each of them to another group. If the group is not *necessary*, i.e. each of its members has at least two possible attributes, the group will be completely eliminated. This is the other way around to avoid formation of groups too small to survive the comparison with non-mutated individuals.

5.2.4 Experimental Results

5.2.4.1 Artificial Testbed

In order to test the algorithm, we have considered a set of 254 "orders", each of them having an average number of 15 possibilities of being produced, selected at random out of a set of 80. In other words, there were 254 lines and 80 columns, with an average of 15 crosses in each line. However, the objects were evenly divided into 6 groups according to the columns 0 through 5: each object had exactly one cross in these columns (beside the random ones). This constituted our testbed - clearly, the best grouping consisted in selecting all the crosses in the first six columns, leading to the smallest number of largest groups possible (provided the other attributes were indeed uniform and didn't allow a grouping into less than six groups).

In a sample of 10 typical runs, the first random population (47 individuals) selected crosses in 71 columns (i.e., the orders were divided into 71 groups), sensibly near to the maximum of 80. However, an average of 38.2 generations of the GGA were enough to select the first 6 columns as

the best basis for the grouping of the set, eliminating the noise.

5.2.4.2 Industrial Implementation

We have applied the above algorithm to the following problem from metalworking industry. A list of orders for a foundry is given and for each order a list of possible ways of producing the ordered quantity of the metal. By selecting an appropriate method for each of the orders, *economies of scale* can be achieved by *grouping* the orders into lots produced in the same way. This results in a smaller number of production cycles, each of them concerning a larger quantity of metal. The production of large quantities implies smaller inventory relative to the total production.

The interesting point of this application is that for any of the orders, each of the production methods implies a different *fixed cost* due to the method itself, i.e. independent of the eventual grouping with other orders. Hence the algorithm must find the proper way between the grouping which reduces the *variable cost* (mainly the cost of the inventory) and the fixed cost.

Evaluating the performance of the algorithm in this case is more difficult, because the optima are not known in advance. To get an idea anyway, we have compared the GA to a classic enumerative method, the A* [7].

On relatively small sets of data (say 50 orders with 10 possibilities for each of them), the GGA always found the same or a better solution than the A*, while taking less time. Given the fact that on small problems the A* has good chances to come across the optimum, we conclude that the GGA did so as well.

On larger sets of data (150 orders) the use of the A* algorithm becomes difficult, because it is hard to estimate the influence on the total cost of eventual groupings of a large number of orders. On the other hand, with a weak estimator (see footnote 7), the A* algorithm easily overlooks good solutions, because the search space is too large to be searched nearly exhaustively. In any case, the GGA performed substantially better and faster on large problems. More importantly, it enabled the enduser to optimize previously unmanageable productions.

Recently the implementation of the algorithm was enhanced in order to handle lists of orders of up to a thousand lines. The complexity of finding a good grouping for such large instances is daunting and well beyond the human capabilities. Still, the algorithm finds good solutions to those problems in reasonable running times (a few hours on a 4 MIPS workstation).

5.3 Creating Part Families - The Conjunctive Conceptual Clustering

This section is made up of excerpts from [Falkenauer, 91b] and [Falkenauer and Gaspart, 93].

5.3.1 The Problem Definition

In informal terms, we are interested in 'telling the pears from the apples', which translates into the following problem: given a set of objects (parts) with various attributes observed on each of them (the *observations*), how to form *definitions of concepts* over those attributes, fitting the objects in the set. Our aim here is to find out 'what properties *at least* must a pear have to be a pear', i.e. we are looking for *conjunctive* definitions of the concepts.

The task of creation of part families is in fact the one of *conjunctive conceptual clustering*. Indeed, each part can be seen as an *instance* of a *concept*, having all the features (attributes) that

---

[7] A 'blend' of the depth-first and breadth-first tree-searching techniques. An *estimator* of the quality of the solutions at the leaves of the search tree is used to limit the breadth of the search to 'promising' branches.

define that concept[8]. Conversely, a part family can be seen as a concept defined by a conjunction of carefully selected (salient) features, instantiated by the parts which comply with the definition. An example of clustering 'fruits' as 'apples' and 'pears' is given in Table I, where the salient (defining) features are represented by the bold **x**s.

Object								*Attribute and meaning*
O_1	O_2	O_3	O_4	O_5	O_6	O_7	O_8	
x		**x**		**x**	**x**			A_1 Stem
	x		**x**		**x**		**x**	A_2 Soft body
		x				**x**		A_3 Worm hole
	x			**x**	**x**	**x**	**x**	A_4 Spherical body
x	**x**	**x**	**x**			**x**		A_5 Yellow stains
x	**x**	**x**	**x**					A_6 Top bulge
x	**x**	**x**	**x**	**x**	**x**	**x**	**x**	A_7 Size about 10cm
x	**x**	**x**	**x**	**x**	**x**	**x**	**x**	A_8 Main color green
(pears)				(apples)				

Table I Example of a clustering

The conceptual clustering is formally defined as an *optimization* problem, the *cost function* to optimize being a measure of quality of the clustering. Note that the number of clusters is a result of the optimization, i.e. it doesn't have to be specified in advance.

The cost function results from an application of the Occam's Razor and aims at reconciling two antagonist tendencies: (1) describing *as many attributes as possible* of the parts on hand, while (2) using concept definitions as *simple* as possible. Thus the resulting clustering strikes a balance between (1) the *precision* of the description of the parts offered by the part families and (2) the *understandability* of the family descriptions supplied, hence fulfilling the aim of creating a simple yet precise description of the data.

More precisely, we consider the following optimization problem: given the set of objects, select concept-defining attributes which minimize the sum of (1) the number of observations not accounted for by the concepts and (2) the number of attributes necessary to define all the concepts. Note that each of the concepts in a solution needs to be defined just once, no matter by how many objects in the test set it is instantiated, i.e. the sum is not a constant of the set - it decreases as the quality of the concepts improves.

The reasons leading to the above definition of the cost function, beyond the scope of this paper, can be found in [Falkenauer, 91b].

5.3.2 CXM - The Clustering Crossmutation

The following main operator for conjunctive clustering is basically a crossover, in that it constructs the progeny using information from two parents. However, as we will see, a certain amount of modification of the information takes place during the crossover. The genetic contents of the two parents is thus not always used *as is* and that's why the denomination crossmutation seems to be better suited for this recombination operator.

The operator proceeds by first copying the first parent into the future child. Then some of the *concepts* of the second parent are injected into this child. The result is a child having some of the concepts of the second parent and, as long as they were not destroyed by the injection, some of the *concepts* of the first parent.

To see the function of the CXM in more detail, consider the following *group* (concept) parts of two chromosomes:

ABCDEFG and
hijklm.

The CXM proceeds first by selecting at random two crossing sites in each of the chromosomes, say

AB|CD|EFG and
h|ijkl|m.

---

[8] At this stage of the research, we only consider *nominal* attributes, i.e. a part either has or doesn't have the feature.

To construct the first child, the segment of the second parent delimited by the crossing sites is injected into the copy of the first parent, yielding

AB|ijkl|CDEFG.

The injection implies an update of the *object* part of the chromosome in order to reflect the *group* part's modification. This is done by scanning the object part of the child and converting whenever possible the objects into one of the groups being injected, i.e. when an object's observation list allows it, it is declared to be an instance of one of the imported concepts. Eventual double occurrences of concepts or concepts stripped of all their instances are then eliminated from the group part. Suppose, for instance, that B and D already appear in ijkl, and C and G lost all their instances, yielding

AijklEF,

the first child. The second child is obtained in the same way, but with the roles of the two parents permuted. This yields a chromosome with the group part

h|CD|ijklm, becoming after the 'group cleanup' of e.g. j,k and l

hCDim.

Note that in this example both children indeed inherited important information from *both* parents. The first child inherited the concepts i, j, k and l from the second and A, E and F from the first parent. The second child inherited the concepts C and D from the first parent and h, i and m from the second. Note also that the possibility for a concept of being deprived of all its instances and be thus eliminated due to mating sets the CXM apart from other GA operators. Indeed, the recombination operator is no longer the usual concatenation of two or more parts of the parental chromosomes. This is a distinctive feature of the GGA.

As we have pointed out above, the main difficulty in clustering consists of fighting two extreme positions: taking either too many or too few attributes into account while defining the concepts. Since there are (typically) several attributes to be selected for each concept and since we are working with data polluted by noise, the crossover must do more than just importing concepts (i.e. their definitions) from one parent into the other, because the probability is high, especially in the beginning of the genetic search, that the concepts being injected are defined over attributes with little significance (i.e. shared by too few objects). The operator must in fact *adapt* the concepts in one parent to those in the other. Otherwise, the ill-defined concepts would be transmitted to the offspring, lessening its chances for survival. Note that this problem is a side-effect of the very large alphabet used by the chromosomes (its size is equal to the number of all possible concepts over the objects on hand) - the choice of the right 'letters' is guided by the recombination process.

The main idea behind the adaptation during crossover is to view the grouping GA as *integrator of multiple sources of evidence*, a source of evidence being a gene in a chromosome (here a concept definition in the group part). The rationale is that as the genetic search progresses, ever better genes appear in the genetic pool (provided the Thesis of good building blocks is complied with), so it usually pays to create new ones from those already present in the pool[9]. We therefore fitted the crossover with two *mutation* aspects.

When a concept being injected has few instances, suggesting its definition is too 'strong' (too many attributes to be applicable to a large number of objects), some of the attributes from the definition are dropped to accommodate at least one object from the other parent. The object is selected by minimizing the Hamming distance between the original and the reduced concept definition. Such a *definition reduction*, a departure from a pure crossover, acts against too strong concept definitions.

For instance, consider again the fruit example above and suppose the concept [2,4,5,6]

[9] 'Bouncing of old ideas against each other' has been suggested many times as the process leading to creation of good novel ideas.

appears in an individual in the population. Such a concept can only accommodate one object, namely O_2, and without the adaptation property of the CXM only a mutation of the definition could achieve an inclusion of another object into the group. However, suppose this concept is being injected into an individual containing the concept [2,5,6,7]: by dropping A_4 (i.e. the attribute not present in the other gene) from the definition (yielding [2,5,6]), the concept can accommodate two objects (O_2 and O_4). Note that the reduction of [2,4,5,6] to [2,5,6] is likely, since the Hamming distance between them is just 1.

The example illustrates well the integrating aspect of the CXM. The fact that the [2,4,5,6] concept has just one instance is a good clue of it being too restrictive, but doesn't show us *which* of its attributes should be dropped. So, in informal terms, we let the [2,4,5,6] 'idea' 'bounce' against the [2,5,6,7] 'idea', yielding a concept defined over attributes with which they both agree.

Conversely, a low rate of group *definition expansion* takes place during the operator. When a concept being injected is large, suggesting its definition might be too weak, the definition is augmented with an attribute randomly selected from the observed attributes of one of the concept's instances. This mutation aspect of the CXM acts against too weak concept definitions.

As can be seen, the 'mutation' aspects of the CXM are not completely random, i.e. an increased rate of a classic mutation operator wouldn't have the same effect. Indeed, what the CXM does is an *interchromosomal gene adaptation* during mating. To our knowledge, no other GA operator in the literature possesses such a quality. It is another distinctive feature of the GGA paradigm.

5.3.3 Two Clustering Mutations

The mutation operator for this problem is implemented in the form of two distinct operators, as suggested in section 4.2 above. The first operator, the *grouping mutation*, proceeds as follows : it takes the observation list of a randomly selected object, drops at random some of the attributes and tries to convert as many objects as possible into the concept defined by the attribute list thus obtained. Most probably, it will be possible to gather under the new concept only some of the objects in the chromosome, because not all of them might have all the defining attributes on their observation list. Nevertheless, in comparison to the classic one-object mutation, the chances are much better that the new group will be strong enough not to decrease too much the fitness of the individual, while introducing a new group definition into the genetic pool.

The second operator, the *eliminating mutation*, selects at random one of the groups present in a chromosome and then tries to eliminate it. This is achieved by scanning the objects in the group and converting each of them to another group. That is done again by minimizing the Hamming distance between the observation list of the objects and the (eventually reduced) definition of the target groups. This is the other way around to avoid formation of groups too small to survive the comparison with non-mutated individuals.

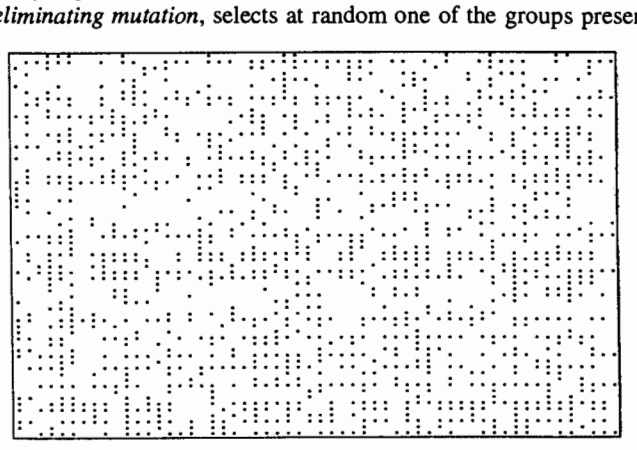

Figure 4 Example of test data shown in random order

5.3.4 Experimental Results

In order to test the performance of the algorithm, we set up the following testbed : for a set of objects with randomly generated attributes, we generated a part of the observations following a

regular pattern, dividing the objects into a small number of groups. The algorithm was then required to classify the objects into clusters defined by the pattern, which amounts to discovering the regularities of the pattern and disregard the randomly generated attributes, thus distinguishing between important (salient) features and noise. The advantage of this approach over tests on 'natural domains' real-world data sets is that here we *know* what kind of behavior should take place. This is important, since not only the ability of the algorithm to optimize the cost function, but also the adequacy of the cost function itself must be verified.

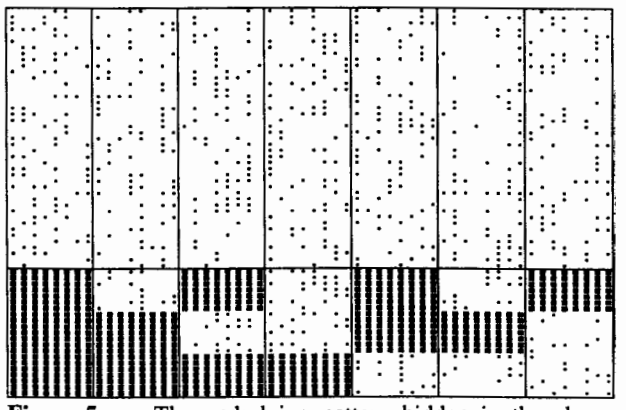

Figure 5 The underlying pattern hidden in the above data

A typical example of such test data is shown in Figure 3, where 56 objects (columns of the matrix) and 64 attributes (lines of the matrix) are in random order, i.e. how they appear to the algorithm. The distribution of observations seems to be completely random, but there is an underlying pattern, as can be seen in Figure 4, where the objects and attributes are in the proper order. In the figure, the group boundaries are represented by vertical lines and the attributes over which the pattern has been imposed (21 of them) are below the horizontal line (the analogous horizontal line in Table I would be drawn between the attributes A_3 and A_4).

We considered that the algorithm successfully classified the data if it found a solution which cost was equal to or smaller than the cost of 'our solution', i.e. the one consisting of selecting all the observations in the pattern and none among the random ones, as in the Figure 4, where the salient observations are represented by boxes. The largest data sets we treated so far (not shown here for space reasons) had 150 objects with 96 attributes, 30 of which followed a pattern similar to the one shown in Figure 4.

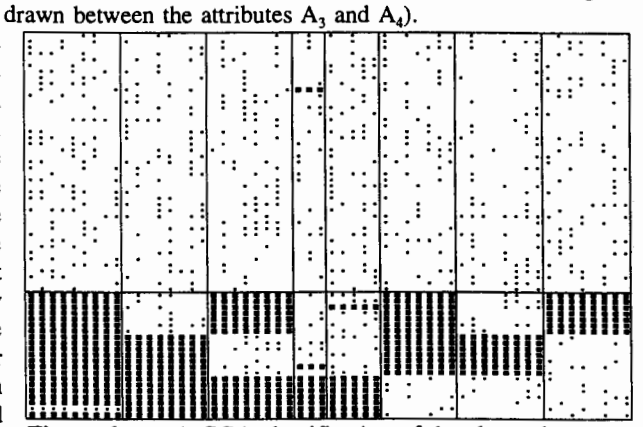

Figure 6 A GGA classification of the above data

The performance of the algorithm on these problems is summarized in Table II showing the number of generations (*Gens* columns) and the time on a 486 PC at 25 MHz (*Secs* columns) required to classify the data in 10 successive runs on the problems with 150 objects, 96 attributes, 30 attributes inside the pattern and an average of 25, 35 and 60 observations per object generated at random outside the pattern (*AttOut* pairs of columns).

Note that generating test data for this problem is not trivial. With too few random attributes, the algorithm quickly figures out the pattern, showing that the problem is too easy. On the other hand, with too many of them, the algorithm discovers other equally good or better solutions (in terms of our cost function), pointing out the difficulty to generate data which do *not* admit good solutions different from the ours. Indeed, all of the solutions found for the problems with 60 attributes outside the pattern were different from (and better than) the one suggested by the pattern.

Figure 6 shows an example of such a 'creative' solution (the intermediate stages are not shown here due to space constraints): in classifying the data from Figure 3 and Figure 4, the algorithm correctly identified the pattern. In addition, it found similarities among objects in the fourth group from Figure 4 and so the group was split in two. It also found an object in the fifth group to be so similar to those in the first group that it relaxed the definition of the latter group to include that object as well.

AttOut →	25		35		60	
Run ↓	Gens	Secs	Gens	Secs	Gens	Secs
1	16	206	18	178	26	420
2	9	118	2	71	16	259
3	3	82	38	312	14	227
4	11	130	15	196	13	229
5	8	111	7	130	24	320
6	11	152	14	143	5	159
7	7	90	8	145	11	221
8	9	102	7	125	12	267
9	9	126	13	212	6	180
10	10	149	9	152	7	210
Mean	9.3	126.6	13.1	166.5	13.4	249.2
StdDev	3.1	34.2	9.4	61.3	6.7	71.2

Table II Summary of performance tests of the algorithm

6. Conclusions

Despite their praised (and well documented) robustness, the classic or ordering GAs are not well adapted to grouping problems. This is because their object-to-gene encodings do not capture the structure of those problems.

Nevertheless, we have shown that a new encoding can lead to efficient GA operators for these problems. The successful applications described suggest that the Grouping GA paradigm holds a promise for many other important problems.

7. References

[Belew and Booker, 91] Belew Richard K. and Booker Lashon B. (Eds) *Proc. of the Fourth Int. Conference on Genetic Algorithms*, University of California, San Diego, July 13-16, 1991, Morgan Kaufmann Publishers, 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, 87] Davis Lawrence (Ed) *Genetic Algorithms and Simulated Annealing*, Pitman Publishing, London.

[Delchambre et al, 92] Delchambre Alain, Falkenauer Emanuel and Hamers Pierre, *Design for Assembly, Resource & Motion Planning in ROBCAD/Rose*, in *Proceedings of the 3rd International ROBCAD User Meeting*, June 23-26, 1992, Toulouse, France.

[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, 91a] Falkenauer Emanuel *A Genetic Algorithm for Grouping*, in "Proc. of the Fifth Int. Symposium on Applied Stochastic Models and Data Analysis", Granada, Spain, April 23-26, 1991, R.Gutiérrez and M.J.Valderrama (Eds), World Scientific, Singapore.

[Falkenauer, 91b] Falkenauer Emanuel, *A Genetic Algorithm for Conceptual Clustering*, Report FMS39, CRIF Industrial Automation, Brussels, December 1991.

[Falkenauer and Delchambre, 92a] 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 and Delchambre, 92b] Falkenauer Emanuel and Delchambre Alain, *Resource Planning in the SCOPES Project* in "Proc. of the 26th Int. Symposium on Automotive Technology and Automation, dedicated conference on Lean Manufacturing in the Automotive Industries", September 13-17, 1993, Aachen, Germany. Automotive Automation Limited, England. pp 295-302.

[Falkenauer and Gaspart, 93] Falkenauer Emanuel and Gaspart Pierre, *Creating Part Families with a Grouping Genetic Algorithm*, in "Proc. of ISIR '93 - Int. Symposium on Intelligent Robotics", Bangalore, India, January 7-9, 1993, M.Vidyasagar (Ed), Tata McGraw-Hill, New Delhi, India.

[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, San Francisco.

[Goldberg, 87] Goldberg David E. *Simple Genetic Algorithms and the Minimal, Deceptive Problem* in [Davis,87].

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

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

[Grefenstette, 87] Grefenstette John J. (Ed) *Genetic Algorithms and their Applications: Proc. of the Second Int. Conference on Genetic Algorithms*, MIT, Cambridge, MA, July 28-31, 1987, Lawrence Erlbaum Associates, Hillsdale, NJ.

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

[Holland, 86] Holland John H. *Escaping Brittleness: The Possibilities of General-Purpose Learning Algorithms Applied to Parallel Rule-Based Systems* in "Machine Learning: An Artificial Intelligence Approach, Vol 2", Michalski et al. (Eds), Morgan Kaufmann, Los Altos, CA.

[Holland et al., 86] Holland John H., Holyoak Keith J., Nisbett Richard E. and Thagard Paul A. *Induction: Processes of Inference, Learning and Discovery*, The MIT Press, Cambridge.

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

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

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

[Schaffer, 89] Schaffer David H. (Ed) *Proc. of the Third Int. Conference on Genetic Algorithms*, George Mason University, June 4-7, 1989, Morgan Kaufmann, San Mateo, CA.

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

[Radcliffe, 92] Radcliffe Nicholas J., *Forma Analysis and Random Respectful Recombination* in [Belew and Booker, 91].

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