INVESTIGAÇÃO OPERACIONAL, vol. 17 (Junho 1997) 71-86

A STANDARD GENETIC ALGORITHM FOR CLUSTERING WITH PRECEDENCE CONSTRAINTS

Margarida Vaz Pato*

Instituto Superior de Economia e Gestão
Universidade Técnica de Lisboa
Rua Miguel Lupi, 20
1200 Lisboa - Portugal

Lídia Lampreia Lourenço

Faculdade de Ciências e Tecnologia
Universidade Nova de Lisboa
Quinta da Torre
2825 Monte da Caparica - Portugal

Resumo

O problema tratado neste artigo refere-se à classificação de N elementos num número máximo de M grupos disjuntos, satisfazendo as restrições de capacidade destes e as de precedência no agrupamento dos elementos. Como critério de agregação usa-se a minimização da dissemelhança total entre elementos colocados no mesmo grupo. Este problema de classificação pode ser aplicado, por exemplo, ao desenho de software.

É apresentada uma heurística genética com base numa codificação dos agrupamentos, caracterizada pela identificação do índice do grupo em que cada elemento é colocado. Os resultados da experiência computacional envolvendo a comparação da heurística genética com uma heurística melhorativa, e uma híbrida, indicam um melhor comportamento da heurística genética para problemas de pequena dimensão e para os problemas sem restrições de capacidade. Em relação ao tempo computacional a genética monstrou-se mais desfavorável do que a melhorativa.

Abstract

Our paper reports on the clustering of N items into a maximum of M non-overlapping groups subject to capacity and precedence constraints when grouping the items. The clustering criterion employed is that of total dissimilarity of items grouped together. This classification problem can, for instance, be applied to the clustering of tasks in software production projects.

The authors developed a genetic heuristic, based on a specific encoding to identify the group in which each element is inserted. Results of the computational experiments, involving comparison of the genetic heuristic with another improvement heuristic and a hybrid heuristic, indicate a favourable behaviour of the basic genetic for the smaller problems, as well as for the uncapacitated problems, in terms of the quality of the solution. However, for problems with a larger number of items, the genetic and the hybrid heuristics did not perform so well as the standard improvement heuristic. Although, in terms of computing time, the genetic heuristic is more expensive compared with the standard improvement heuristic, these experiements will encourage us to redefine the genetic procedure.

Keywords

clustering, process organization, precedence constraints, genetic heuristics.

* This research has been partly supported by the Centro de Investigação Operacional (FC-Universidade de Lisboa) within the PRAXIS XXI (NJICT) Project nr 2/2.1/MAT/139/94 and by the Centro de Matemática Aplicada (FCT-Universidade Nova de Lisboa).

1. Introduction

When dealing with a huge number of items identified by specific characteristics, it is often necessary to cluster them in order to study each group separately, or even to take each group as a single entity. Determination of the number of groups required for clustering, together with selection of the criterion used for the grouping are the first steps of cluster analysis. This is followed by the task of clustering the items on the basis of the criterion chosen and according to the specific nature of the situation.

Cluster analysis problems arise in many scientific areas such as biology, medicine, psychology, economics, computer science and even literature. A survey of methods and applications may be found, for instance, in Duran and Odell[5] or in Gnanadesikan and Kettenring[8].

In this paper we study a specific cluster analysis problem - the Clustering with Precedence Constraints Problem - where a maximum number of groups is pre-defined, as well as subsidiary constraints on group capacities and precedence constraints in the grouping of items. The clustering criterion calls for total dissimilarities among the items clustered in the same groups. The major applications for this problem are related to the assignment of tasks to work groups in large software production projects.

Some authors have formulated the problem as a mixed integer linear problem, and have used heuristics and branch-and-bound methods to tackle it. The problems solved are of small and medium size, and one may speculate that if the number of items ranges from some tens to hundreds, the methods, as they stand, would not be applicable, and enhanced heuristic versions should be developed. As constructive heuristics and standard exchange scheme heuristics could not, apparently, be significantly perfected, we decided to explore the evolutionary approach. Though the problem is highly constrained, the genetic algorithm may be used if the encoding and genetic operators incorporate all the problem constraints reasonably well.

The Clustering with Precedence Constraints Problem is described in Section 2. Section 3 presents a genetic algorithm for that problem, while Section 4 gives the results of the computational experiments for small and medium-sized instances, partly taken from literature. Section 5, the final section, points to modifications that could lead us to handle larger problems through genetic search.

2. The Clustering with Precedence Constraints Problem

Let us consider N items and real numbers d_{ij} (i=1,..., N-1; j=i+1,..., N), which stand for the dissimilarities between each pair of items. Known data include the maximum number of groups, M, the weight of each item (p_i, i=1,..., N), the upper bound on the number of items per group (B_k, k=1,..., M), and on the weight of each group (C_k, k=1,..., M), besides several precedence constraints that prevent items from being clustered in a group if their preceding items have not, as yet, been placed in the current group, or in any previous group.

The Clustering with Precedence Constraints Problem (CPCP) addresses the partition of the set of items into a maximum of M disjoint subsets, according to the criterion of minimization of the total amount of the dissimilarities among items placed together, while satisfying the capacity and precedence constraints explained earlier.

This problem has been proposed by Karimi[13] and Klein, Beck and Konsynski[14] to process organization in an information system. The processes of a system and their connections may reach a high level of complexity and in that case clustering into modules with minimum interconnections is a way of organizing the overall system.

Further possible applications suggested concern support for CAD/CAM software production, production of sofware for industrial process control and robotics. The problem of assigning jobs to independent processors of a computer, referred to in Sofianopoulou[18], with a slightly different clustering criterion (minimizing costs of communication between jobs, as well as the execution costs of the jobs) may share the same kind of constraints as the CPCP, besides a similar objective function.

Aronson and Klein[1] formulated the CPCP as an extension of another formulation[14] for a CPCP without capacity constraints. In the abovementioned CPCP formulation[1], the following parameters are required:

M - maximum number of groups, $M \geq 2$ and integer;

N - number of items, $N > M$ and integer;

d_{ij} - dissimilarity between item i and item j, $d_{ij} > 0$ for $i = 1,\ldots, N-1$ and $j=i+1,\ldots, N$;

B_k - maximum number of items for group k, $B_k \geq 0$ and integer for $k=1,\ldots, M$;

C_k - maximum weight for group k, $C_k \geq 0$ for $k = 1,\ldots, M$;

p_i - weight of item i, $p_i \geq 0$ for $i = 1,\ldots, N$.

Moreover, the precedence relations for some pairs of items, say (i,j), require that item j may only be grouped if item i has already been placed in a previous group. This assumes that the groups are ranked according to a specific order.

Let us now present the objective function and the model constraints:

$$\text{minimize} \qquad z = \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} d_{ij}\, y_{ij} \qquad (0)$$

subject to

$$x_{ik} + x_{jk} - y_{ij} \leq 1 \qquad (i=1,\ldots, N-1;\ j=i+1,\ldots, N;\ k=1,\ldots, M) \qquad (1)$$

$$\sum_{k=1}^{M} x_{ik} = 1 \qquad (i=1,\ldots, N) \qquad (2)$$

$$\sum_{i=1}^{N} x_{ik} \leq B_k \qquad (k=1,\ldots, M) \qquad (3)$$

$$\sum_{i=1}^{N} p_i x_{ik} \leq C_k \qquad (k=1,\ldots, M) \qquad (4)$$

$$\sum_{k=1}^{M} k\, x_{ik} \leq \sum_{k=1}^{M} k\, x_{jk} \qquad \text{(for each pair (i,j) such that i precedes j)} \qquad (5)$$

$$y_{ij} \geq 0 \qquad (i=1,\ldots, N-1;\ j=i+1,\ldots, N) \qquad (6)$$

$$x_{ik} = 0, 1 \qquad (i=1,\ldots, N;\ k=1,\ldots, M) \qquad (7)$$

The variable x_{ik} (for all i and k) takes the value 1 if item i is clustered into group k, or 0 otherwise. As for the variable y_{ij} (also defined for all i and all j greater than i), when it is equal to 1 it means that items i and j belong to the same group, being 0 otherwise. Constraints (6) on the variables y_{ij} are simply of the non-negativity type, because minimization of the objective function (0) and constraints (1) and (7) force such variables to be binary in the optimum, without the need to impose the binary conditions.

As for the objective function (0), it represents the total sum of the dissimilarities for the items grouped together.

The first set of constraints (1) establishes the relation between the x_{ik} and the y_{ij} variables. The following set (2) enforces each item to be assigned to a unique group, whilst constraints (3) and (4) are defined to limit the number of elements and the total weight of each group. Finally, the precedence constraints are expressed in terms of inequalities (5).

In this formulation the size of the instances tends to be very high, even for small-sized problems. For instance, if the CPCP is used to classify 13 items into 8 groups with 12 precedence relations, the formulation has 182 variables and 685 constraints, without considering the constraints for definition of the variables ((6) and (7)).

Moreover, the CPCP is a generalization of a clustering problem classified as NP-hard in Garey and Johnson[7], thus placing the CPCP itself in the NP-hard class.

Therefore LP based techniques are not advisable and of course a complete enumeration of solutions is impracticable. Hence heuristics arise as natural approaches, at least for the larger-sized problems that may not be solved by exact methods within a reasonable amount of computing time.

Let us now refer to other similar clustering problems that could lead to different formulations and solution-finding methods.

It should be remembered that in formulation (0) to (7) for the CPCP, the y_{ij} variables are used to establish the objective criterion, whereas the other variables, the x_{ik}s, are used to impose the clustering restrictions.

If neither capacity nor precedence restrictions were imposed on the CPCP, the problem could be viewed as a quadratic semi-assignment problem, as shown in Gondran and Minoux[10]

(page 467). In their formulation, the unique set of (binary) variables translates the objective criterion and determines where each element should be placed. But if we consider precedence constraints, this quadratic model cannot be adapted to the situation. However, even if it were possible to model the CPCP as a binary quadratic problem, it would not be advisable to do so in view of the computing difficulties known for that kind of problem.

On the other hand, Jensen[12] presented a dynamic programming formulation for clustering N items into M groups without constraints, where the dissimilarities among the items placed in a group are penalized through the inverse of the cardinality of that group. Such a dynamic model may be applied, with minor modifications, to our CPCP, as the capacity and precedence constraints could easily be imposed on the model. But unfortunately dynamic programming calculations remain very hard to handle. Even in the case of small-sized problems, it would be difficult to solve this problem within a reasonable amount of computing time.

3. The Genetic Heuristic

Genetic heuristics are search procedures that rely to a large extent on basic biological rules for genetics. Holland[11] is considered to be the father of genetic algorithms as applied within the field of combinatorial optimization. Goldberg[9], Davis[4] and Michalewics[16] present the main features of genetic heuristics and the corresponding algorithms, besides many bibliographical references concerning their applications. Although theoretical support for genetics is, to date, mainly restricted to conventional binary codes, which should be applied only to particular problems (see for instance Reeves[17]), the genetic-based methodology has proved to be successful in practice, even for highly constrained combinatorial problems.

Literature has already referred to several uses of genetic algorithms in clustering problems (Michalewics[16] and Faulkenauer[6]), though they are simpler than our Clustering with Precedence Constraints Problem, as they do not embed precedence constraints.

On the other hand, Reeves[17] refers to a successful application of such algorithms to a problem with precedence constraints which, broadly speaking, is also a clustering problem - the instruction scheduling problem (Beaty[2]).

We shall now begin by explaining the genetic heuristic developed for the CPCP (Lourenço[15]). The algorithm is given in Subsection 3.1, while the four subsections that follow are devoted to the chromosome encoding and to the selection, crossover and mutation operators.

3.1 The Algorithm

The genetic algorithm, called GENET, is summarized in Figure 1.

Each individual of the population is identified by a chromosome encoding a feasible solution for the CPCP. The initial population has P individuals (here, the dimension of the population P was set to 20). Three of these 20 feasible solutions are created by a constructive

heuristic[15] briefly presented in Subsection 4.2, whereas the remaining 17 feasible solutions are randomly generated, following a procedure also given in that work[15].

```
procedure GENET
generate the population with P individuals
compute the fitness value for each individual
identify the fittest individual
gener = 1
while gener ≤ maxgener do
        act with selection operator
        act with crossover operator and update the fittest individual
        act with mutation operator and update the fittest individual
        gener = gener + 1
end GENET
```

Figure 1 - The Genetic Algorithm GENET

In each iteration of the algorithm or generation a new population is created. On 20 occasions, in each generation, the selection operator chooses one chromosome from the population, based on its fitness. Then the crossover, as well as the mutation operator, acts on the 20 chromosomes selected, thus creating the population for the subsequent generation.

When the maximum number of *maxgener* (here, equal to 100) generations is reached, the algorithm stops and produces the best individual (chromosome) found so far and the corresponding solution.

The features used in the algorithm were suggested by the abovementioned literature, as well as our own computing experience. Other versions were tried out but abandoned in view of their poorer behaviour, when compared to GENET, such as those below:

- a direct binary encoding scheme;
- the selection operator, acting only once at the initialization step;
- a crossover, where offsprings always replace parents if feasible.

3.2 The Encoding

Each individual is associated with one chromosome representing a feasible solution for the problem. The chromosome consists of a string of N genes, each standing for an item and indicating, through its allele, the group in which the item is placed (Figure 2).

Figure 2 - A Chromosome

Let us consider a small instance of the CPCP, where 13 items are to be clustered into 8 groups. Other data for this case are:

$[d_{ij}]$ i=1,..., N-1; j=2,..., N =

$$\begin{bmatrix}
0.3 & 0.4 & 0.8 & 0.6 & 0.2 & 0.1 & 0.5 & 0.9 & 0.6 & 0.4 & 0.1 & 0.6 \\
- & 0.2 & 0.7 & 0.8 & 0.3 & 0.7 & 0.6 & 0.6 & 0.3 & 0.5 & 0.9 & 0.5 \\
- & - & 0.8 & 0.1 & 0.7 & 0.6 & 0.5 & 0.5 & 0.8 & 0.1 & 0.4 & 0.5 \\
- & - & - & 0.9 & 0.2 & 0.6 & 0.9 & 0.5 & 0.6 & 0.3 & 0.3 & 0.4 \\
- & - & - & - & 0.4 & 0.5 & 0.4 & 0.2 & 0.7 & 0.4 & 0.7 & 0.7 \\
- & - & - & - & - & 0.6 & 0.9 & 0.3 & 0.1 & 0.8 & 0.3 & 0.2 \\
- & - & - & - & - & - & 0.1 & 0.4 & 0.7 & 0.5 & 0.5 & 0.8 \\
- & - & - & - & - & - & - & 0.8 & 0.3 & 0.6 & 0.6 & 0.8 \\
- & - & - & - & - & - & - & - & 0.8 & 0.5 & 0.5 & 0.6 \\
- & - & - & - & - & - & - & - & - & 0.4 & 0.4 & 0.1 \\
- & - & - & - & - & - & - & - & - & - & 0.9 & 0.7 \\
- & - & - & - & - & - & - & - & - & - & - & 0.8
\end{bmatrix}$$

$[B_k]$ k=1,..., M = [5 6 5 4 3 4 8 8]

$[C_k]$ k=1,..., M = [9.0 15.0 14.0 4.0 9.0 3.0 10.0 12.0]

$[p_i]$ i=1,..., N = [1.4 2.6 3.1 2.8 2.0 1.2 2.2 2.2 1.9 2.4 2.2 1.0 1.8]

Finally, precedence constraints for this instance are given in Figure 3.

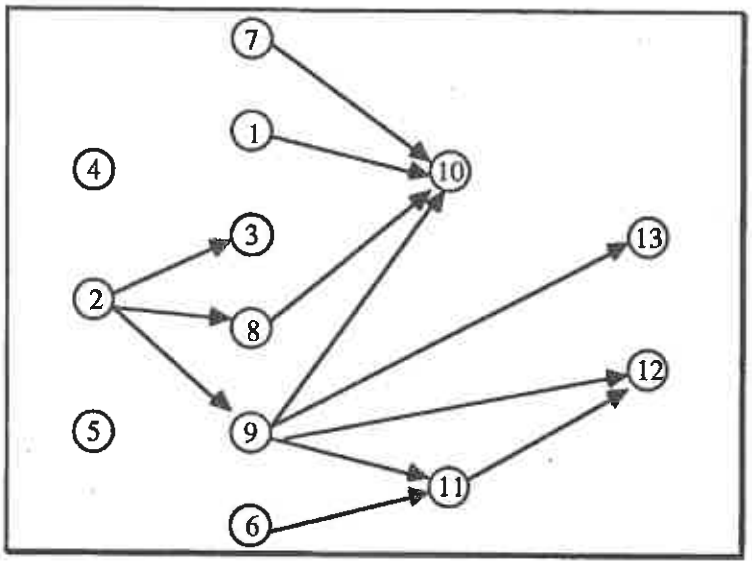


Figure 3 - Precedence Constraints for the Example

A clustering that fulfils all constraints for this case is illustrated in Figure 4. Here groups have the following composition: g_1={2}, g_2={5,9}, g_3={13}, g_4={4}, g_5={1,6}, g_6={11}, g_7={3,12} and g_8={7,8,10}.

The given encoding represents the specific clustering for the CPCP instance by the chromosome:

5	1	7	4	2	5	8	8	2	8	6	7	3

Here, item 13 is clustered in group 3 (the last gene with its allele in bold print) and both items 5 and 9 fall into group 2 (the fifth and ninth genes, respectively).

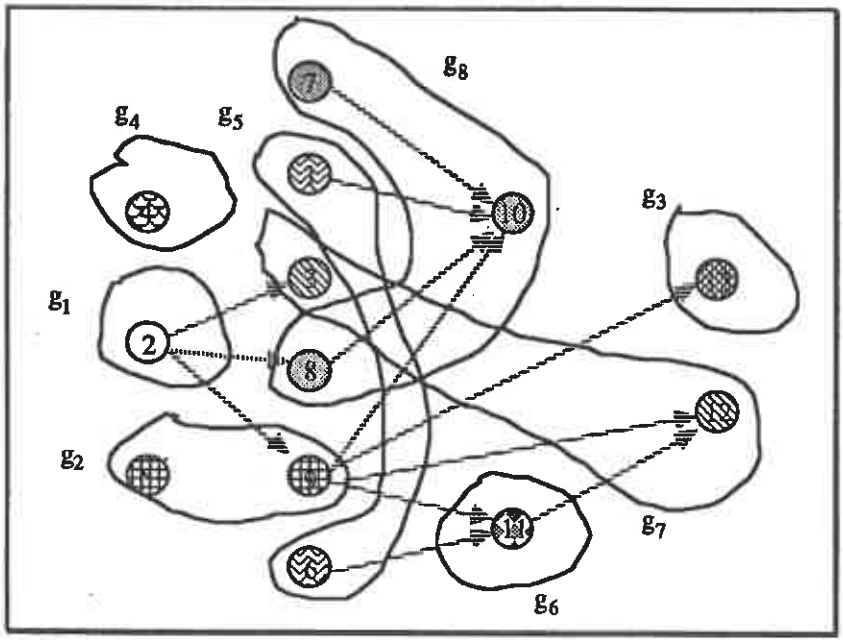


Figure 4 - A Clustering for the Example

We now have the corresponding solution for the CPCP formulation of Section 2: $x_{15} = x_{21} = x_{37} = x_{44} = x_{52} = x_{65} = x_{78} = x_{88} = x_{92} = x_{10,8} = x_{11,6} = x_{12,7} = x_{13,3} = 1$; $y_{59} = y_{3,12} = y_{16} = y_{78} = y_{7,10} = y_{8,10} = 1$; and all others equal to 0.

3.3 The Selection Operator

The selection of chromosomes to remain in the population for the current generation is based on their fitness values, that is, the probability of selection of an individual chromosome is proportional to its fitness. The fitness of each chromosome is supposed to reflect the objective value of our problem, which represents total dissimilarity of the corresponding feasible solution. Thus, the fitness value must be better for the lower dissimilarity chromosomes and always positive.

Considering that the minimization of the objective function of the CPCP given by (0) is equivalent to

$$\text{maximize} \qquad z' = -\sum_{i=1}^{N-1} \sum_{j=i+1}^{N} d_{ij}\, y_{ij} \tag{8}$$

and also to

$$\text{maximize} \qquad z'' = -\sum_{i=1}^{N-1} \sum_{j=i+1}^{N} d_{ij}\, y_{ij} + \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} d_{ij} \tag{9}$$

the fitness value for each chromosome will be expressed by the value of this last objective function (9) calculated in the respective feasible solution. Therefore, the fitness values are positive, as required, and better for the lower dissimilarity chromosomes.

The selection operator performs like a roulette wheel, where the fittest chromosome is associated with the largest sector of the wheel, thus having the greatest probability of being chosen.

Let us show how the roulette works in a case taken from the example. Take 5 chromosomes from the population and suppose that the corresponding feasible solutions have the following total dissimilarities or values of the objective function (0): 14.5, 3.6, 10.0, 10.6 and 7.0. These dissimilarities amount to 40.3 and, bearing in mind the reformulation of the objective given by (9), we find the following fitness values for the 5 chromosomes to be: 25.8, 36.7, 30.3, 29.7 and 33.3. Figure 5 displays the roulette for this selection process.

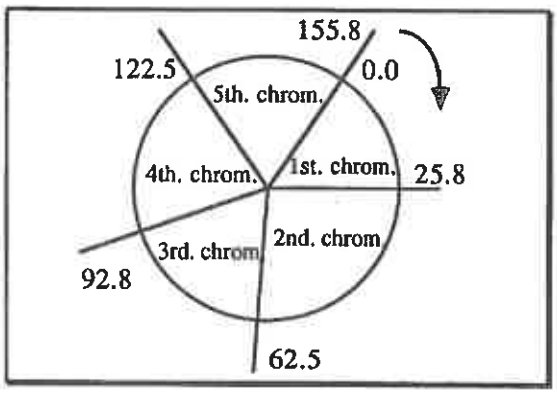


Figure 5 - Roulette for an Illustrative Case

Let us assume that the (uniformly) random generation of a real number between 0.0 and 155.8 produces number 65.0. The chromosome selected is therefore the third one, whose fitness value is 30.3. In fact, the addition of the first two fitness values gives 62.5 (below 65.0) and the addition of the first three values gives 92.8 (above 65.0).

The selection operator is repeated a certain number of times - in GENET on 20 occasions - and the resulting chromosomes build the population for the current generation. Some of the chromosomes may ultimately be identical.

3.4 The Crossover Operator

The number of crossovers in a generation is randomly generated between 1 and 5. These figures were set in accordance with the parameter for the dimension of the population ($P = 20$).

For each crossover a pair of chromosomes (the parents) is chosen by the roulette, thus giving priority to the fittest chromosomes. Then, in order to create the child-chromosomes, approximately 1/3 of the N genes are randomly chosen and their alleles are swapped from one chromosome to the other of the same pair.

As an illustration, let us take two parent-chromosomes of the instance given above:

7	1	7	5	4	5	8	8	2	8	6	7	3

1	2	3	3	1	1	1	2	2	3	2	2	2

Here [13×1/3] = 4, and so the offsprings will have 4 genes whose values result from swapping the corresponding alleles of their parents (all in bold print in the parents):

7	1	3	5	1	5	1	8	2	8	6	7	3

1	2	7	3	4	1	8	2	2	3	2	2	2

These correspond to the following two groupings respectively, and in this case happen to be feasible:

- $g_1=\{2,5,7\}$, $g_2=\{9\}$, $g_3=\{3,13\}$, $g_4=\{\ \}$, $g_5=\{4,6\}$, $g_6=\{11\}$, $g_7=\{1,12\}$ and $g_8=\{8,10\}$;
- $g_1=\{1,6\}$, $g_2=\{2,8,9,11,12,13\}$, $g_3=\{4\}$, $g_4=\{5\}$, $g_5=\{\ \}$, $g_6=\{\ \}$, $g_7=\{3\}$ and $g_8=\{7\}$.

This crossover operator was designed with the single purpose of creating solutions that are partly similar to the original ones, without considering the feasibility.

The two child-chromosomes, if feasible, substitute their parents, but only if they are better. If only one of the descendants is feasible, this feasible chromosome replaces the most ill-fitted parent. The parent-chromosomes may be selected for crossover twice or more often in the same generation.

As the population dimension is 20, once the crossovers occur, more than half of the population is kept identical to the previous generation and the remainder does at least retain some characteristics of their parents.

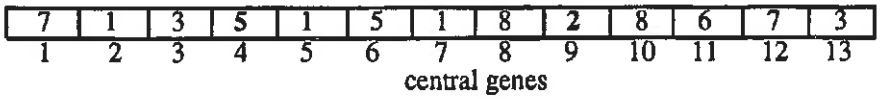
3.5 The Mutation Operator

The mutation operator may act on up to 1/5 of the population's chromosomes. In other words, the number of mutants can be 0, 1, 2, 3 or 4, in the case of the present implementation.

Each chromosome in which mutation will occur is randomly chosen among the population. This method also applies in the selection of two of its genes. For each of these two genes (items) the corresponding allele (the group wherein the item is clustered) is compared with the central index group (in the case of an even number of items, the central group is considered as the average of the two central index groups):

- if the allele is below the central group, one unit is added to it, i. e. the item shifts to the subsequent group;
- if the allele is above the central group, one unit is subtracted, and the item shifts to the previous group. This mutated chromosome is then tested against all the constraints, and only if it is feasible, does it replace the original.

To illustrate this operator, let us consider a chromosome for the same example problem:



which, after random determination of the two genes - the fourth and the ninth - mutates into:

The second chromosome still represents a feasible solution: $g_1=\{2,5,7\}$, $g_2=\{\ \}$, $g_3=\{3,9,13\}$, $g_4=\{4\}$, $g_5=\{6\}$, $g_6=\{11\}$, $g_7=\{1,12\}$ and $g_8=\{8,10\}$.

This chromosome-mutating operator was designed with two main aims. One is to concentrate the items in the neighbourhood of the central group. Such chromosomes will differ from those created by the constructive and improvement heuristics (see Subsection 4.2) because these procedures concentrate the items in the first and last groups, respectively. The other aim is to ensure a frequent verification of precedence constraints. However, as a lot of mutants are not feasible (often they do not respect group capacities) and are therefore omitted, we mutate frequently (up to 4 times in 20 chromosomes) to bring about some diversification among individuals in the population. In fact this operation is not rare in genetic heuristics, as opposed to natural biological evolution.

4. Computational Experiment

Computational experiments were carried out to test the performance of the genetic heuristic GENET following the algorithm of Figure 1 and two other heuristics: an improvement heuristic based on exchanges of items and a hybrid heuristic, produced by combining the improvement with the genetic procedure.

All the algorithms were coded in PASCAL and ran in a HP 486/33 VL personal computer.

We shall begin by describing the test instances in Subsection 4.1. This is followed by a short presentation of the comparison heuristics, in Subsection 4.2. Results are finally given in Subsection 4.3 and commented upon in Subsection 4.4.

4.1 Instances

The test instances came from a semi-random generation process, following the rules and parameters given in Aronson and Klein[1] (who took partial data for some problems from Tonge[19]), as no complete set of data for this kind of clustering problem - the CPCP - was found. The only exception is an instance to be referred to in Subsection 4.4, belonging to Type C problems.

Parameters of test problems were randomly generated within the bounds stated in Table 1: the number of groups, the maximum number of items, the maximum weight for groups and the weights of the items.

Generation of the dissimilarity matrix was carried out following two different rules:

- rule 1 - each element of the matrix is randomly drawn from the interval [0.1,0.9];
- rule 2 - two spatial coordinates are randomly generated between 1 and 100 for each item. Euclidean distances for each pair of items are then calculated, which leads to the dissimilarity matrix.

The precedence constraints were also set semi-randomly for some instances[15], whilst others were taken from literature - Class 2 problems.

Four different classes of instances were considered - Class 1 to Class 4 - and, for each one, 25 problems were generated by changing one of the parameters within the bounds.

For Class 1 two sets of 13 and 12 problems each were considered: problems Type A, with 2 to 5 groups and problems Type B, with 6 to 8 groups. Similar differences exhibit the following pairs of problem Types created respectively for Classes 2, 3 and 4:

- Type C (12 problems) and Type D (13 problems);
- Type E (7 problems) and Type F (18 problems);
- Type G (7 problems) and Type H (18 problems).

Some problems from Types C and D have a dissimilarity matrix copied from Aronson and Klein[1].

	Class 1	Class 2	Class 3	Class 4
	Types A and B	Types C and D	Types E and F	Types G and H
N	13	23	32	70
M	2 - 8	3 - 8	4 - 9	3 - 30
nº of precedence constraints	12	25	25	86
maximum nº of items per group	3 - 8	6 - 14	6 - 14	3 - 30
maximum weight per group	3.0 - 15.0	8.6 - 25.0	8.6 - 28.0	320.0 -1400.0
weight of items	0.1 - 3.1	0.1 - 6.0	0.1 - 6.0	1.0 - 319.0
dissimilarities	0.1 - 0.9 (rule 1)	0.1 - 0.9 (rule 1)	0.1 - 0.9 (rule 1)	(rule 2)

Table 1 - Bounds for Data of Test Problems

4.2 Other Heuristics

An exchange-based improvement heuristic and a hybrid genetic heuristic were used to assess the behaviour of the genetic heuristic.

Let us begin by briefly presenting the improvement heuristic IMPROVE, suggested in reference[1] and extensively described in the dissertation[15].

It starts from any feasible solution of the CPCP, in particular from one built through a constructive procedure that clusters the items according to increasing weights, filling the groups ranked by increasing products of the parameters B_k and C_k. These two building criteria - increasing weights of items and increasing products of group capacities - are used to cluster the greatest quantity of items in the less favourable groups first. This constructive procedure is also used to build 3 of the individuals of the initial population of GENET, as mentioned earlier.

As such a classification of items tends to leave the last groups empty, the improvement heuristic runs through the groups, starting by taking the last and continuing in decreasing group index order, displacing to the current group all items with their successors in the current group or in the groups analysed before the current group (that is, the groups with a higher index).

The criterion for choosing items to be displaced at each stage is determined by taking into account the maximum reduction that may be achieved in the total dissimilarity.

The hybrid heuristic, HYBRID, is basically a genetic search, characterized by the fact that, at the end of each generation, it calls the above improvement procedure from the chromosomes (feasible solutions) of the population.

4.3 Results

The measures used to evaluate the heuristics were the solution quality and the computing time in seconds. For each problem Type, average values were calculated, as seen in Table 2.

To assess the quality of the solution, we used the percentual gap between the total dissimilarity of the solution obtained by the genetic-based heuristics (GENET or HYBRID) and the value given by IMPROVE, that is, $100\times(z^{IMP}-z^{genetic})/z^{IMP}$. These values are listed in columns (5) and (8).

For the genetic heuristic GENET the percentage of non-discarded or accepted chromosomes over created chromosomes, in all the 100 generations, is also given. Columns (6) and (7) of Table 2 present these figures for the crossover and for the mutation respectively.

Table 3 presents similar results to those of Table 2 but refers only to the 4 problems with no capacity constraints that were used in our tests.

As for computing times, IMPROVE took up to 2 seconds for each of these 100 instances, whereas GENET on average spent about 14 seconds per instance and HYBRID, on average, took 39 seconds per instance.

problem Types	N	M	nº of prec. constr.	GENET diss.gap (%)	GENET acc.cro. (%)	GENET acc.mut. (%)	HYBRID diss.gap (%)
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
A	13	2 - 5	12	10	64	32	8
B	13	6 - 8	12	12	86	40	5
C	23	3 - 5	25	2	56	28	3
D	23	6 - 8	25	11	73	48	10
E	32	4 - 6	25	-15	53	29	-9
F	32	7 - 9	25	-9	79	35	-4
G	70	3 - 10	86	-1	63	15	-4
H	70	11 - 30	86	-12	70	27	1

Table 2 - Computational Results - Average Values

problem (Type)	N	M	nº of prec. constr.	GENET diss.gap (%)	GENET acc.cro. (%)	GENET acc.mut. (%)	HYBRID diss.gap (%)
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
problem 10 (A)	13	5	12	28	82	49	23
problem 35 (C)	23	3	12	28	47	5	14
problem 60 (F)	32	8	25	21	82	21	26
problem 85 (G)	70	9	86	10	79	27	10

Table 3 - Computational Results for Uncapacitated Problems

4.4 Comments

As Table 2 shows, heuristic GENET performed better, in terms of solution quality, over the shorter instances (problem Types A, B, C and D), whereas the IMPROVE outperformed it in the larger-sized instances (problem Types E, F, G and H). The hybrid process (column (8)) stands between the other two single heuristics - IMPROVE and GENET.

The occurrence of these instances with a poorer behaviour is due to the small number of individuals in the population - fixed and equal to 20 - with respect to the problem size, a direct consequence of the number of items which, for these instances, is N = 32 or 70. Such results are to be expected, as literature suggests higher cardinality populations for larger problems. The enlargement of the dimension of the population has not yet been tried for all the instances of CPCP, as the available computing resources were insufficient. However, in some cases we simulated that situation (enlargement of the population) and found a much better performance of both genetic-based heuristics.

As can be seen in Table 2, columns (6) and (7), the number of discarded chromosomes is very high as compared to the chromosomes included in the population (corresponding to feasible clusterings). We also observed through our experiment - though not noticeable in Tables 2 and 3 - that, in instances with a lower percentage of feasible chromosomes the results were worst.

This leads us to think that we could develop operators to attain feasibility more frequently or, alternatively, procedures to restore feasibility after performing the crossover and mutation operators.

The computing time is higher for the genetic-based heuristics but this measurement of heuristic behaviour is not significant if we assume that the CPCP is a problem whose solution is not called frequently.

It should be stressed that GENET obtained much better solutions than IMPROVE for the problems with no capacity constraints - column (5) of Table 3.

Comparison of the above figures with results found in literature[1,14] is only partially possible, as authors give neither complete problem data nor similar solution quality results.

Moreover, computing times are not comparable, in view of the considerable differences in computing resources.

However, Aronson and Klein[1] do provide some information. Where solution quality is concerned, they present the optimum value (19.7) for a single problem only (which is precisely one of ours belonging to the Type C problems). They also add that their constructive heuristic produced a solution with total dissimilarity equal to approximately seven times the optimum value, and that their improvement scheme (similar to the one described in Subsection 4.2) reduced the value to 25.4. Our results for this problem are: 25.4, 21.9 and 22.3 respectively for the three heuristics IMPROVE, GENET and HYBRID.

According to the authors quoted, the total of 29 problems tested (25 with parameter ranges similar to ours, and 1 with 140 items and 3 groups), the constructive heuristic failed to build up a feasible solution in 4 cases, whereas their specialized branch-and-bound method, embedding the constructive and improvement heuristics, produced the optimal solution in all other cases - though high computing expenses were incurred in some cases. The branch-and-bound is a generalization of Balas's method to embed multi-branches representing the groups, whilst the levels represent the items. Such a method is not suitable for larger problems.

The results of Klein, Beck and Konsynski[14] are even more difficult to compare with ours, as their test problems are CPCPs without capacity constraints, involving 10 to 20 items and 3 groups, and they used a standard ILP code to optimally solve the problems.

Though problems of Table 3 are also uncapacitated CPCPs, they cannot be considered for comparison with the above work[14] as the size and data generation processes are different.

On the strength of the above comparisons and computational results, one may conclude that the genetic-based heuristics we developed for the CPCP behaved well in the set of uncapacitated test instances, and poorly at capacitated medium sized test instances.

5. Conclusions and Future Work

This paper presented a clustering problem, where clusters are built up by considering both maximum capacities and precedence constraints when grouping the items, using a total dissimilarity clustering criterion. This is a typical highly constrained problem and the plain version of the genetic heuristic developed in this case performed fairly well for the set of small and medium sized test instances - at least when compared with results obtained from an improvement scheme, also custom-made for the CPCP.

Computing times are higher for the genetic and the hybrid heuristics, though this is not significant in view of the strategic nature of the applications for the CPCP mentioned earlier.

The computational experiments performed point to the interest in trying different genetic heuristic versions, characterized by imposing the feasibility over solutions created by crossover or mutation, or alternatively by an encoding process that naturally conveys most of the problem constraints through the operation of crossover and mutation (Caldas[3]). This should help to

reduce the non-productive computing time resulting from discarding non-feasible chromosomes.

We also suspect that, for the larger problems, an increase in the number of individuals in the population would produce much better results.

We would like to study lower bounds on the optimal total dissimilarity, so as to evaluate the quality of heuristic solution in absolute terms. We consider this to be one of the next most important steps to take in the study of heuristics for the CPCP. Lower bounds could be obtained through exploitation of the dynamic quality of the CPCP. This may involve the study of state space relaxations for dynamic programming formulations similar to or even different from the dynamic programming model referred to in Section 2.

Finally, we look forward to finding real situations to which the CPCP may be applied and therein experiment methodology developed to date. It is commonly known that huge software companies spend a large amount of their budgets on overtime and extra-fees due to delays. If tasks were better organized, resources would be more profitably used. We believe that these would be successful fields of application for the Clustering with Precedence Constraints Problem.

References

[1] J. E. Aronson and G. Klein, "A Clustering Algorithm for Computer-Assisted Process Organization", *Decision Sciences*, vol. 20 (1989), pp. 730-745.

[2] S.Beaty,*Instruction Scheduling Using Genetic Algorithms*,PHD thesis,Colorado State Univ.,USA (1991).

[3] J. C. Caldas, personal communication (1995).

[4] I. Davis, *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, New-York, USA (1991).

[5] B. S. Duran and P. L. Odell, *Cluster Analysis-A Survey*, Springer-Verlag, Berlin, Rep. of Germ. (1974).

[6] E. Faulkenauer, "The Grouping Genetic Algorithms: Widing the Scope of the GAs", *Belgian Journal of Operations Research, Statistics and Computer Science*, vol. 33 (1993) pp. 79-102.

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

[8] R. Gnanadesikan and J. R. Kettenring, "Discriminant Analysis and Clustering", *Statistical Science*, vol. 4 (1989) pp. 34-69.

[9] D. E. Goldberg, *Genetic Algorithms in Search Optimization and Machine Learning*, Addison-Wesley Publishing Company, Reading, USA (1989).

[10] M. Gondran and M. Minoux, *Graphs and Algorithms*, translated by S. Vajda, John Wiley and Sons, Chichester, USA (1986).

[11] J. Holland,*Adaption in Natural and Artificial Systems*, Univ.of Michigan Press, Ann Arbor, USA (1975).

[12] R. J. Jensen, "A Dynamic Programming Algorithm for Cluster Analysis", *Operations Research*, vol.17 (1969) pp. 1034-1057.

[13] J. Karimi, "An Automated Software Design Methodology Using CAPO", *Journal of Management Information Systems*, vol. 3 (1986-87) pp. 71-100.

[14] G. Klein, P. O. Beck and B. Konsynski, "Computer-Aided Process Structuring Via Mixed Integer Programming", *Decision Sciences*, vol. 19 (1988) pp. 750-761.

[15] L. L. Lourenço, *Contributos da Optimização Discreta para a Análise Classificatória. Aplicação de Heurísticas Genéticas a uma Classificação com Precedências*, Master dissertation, Instituto Superior de Economia e Gestão, Universidade Técnica de Lisboa, Portugal (1995).

[16] Z. Michalewics, *Genetic Algorithms + Data Structures = Evolution Programs*, Springer-Verlag, Berlin, Republic of Germany (1992).

[17] C. R. Reeves, *Modern Heuristic Techniques for Combinatorial Problems*, editor, Backwell Scientific Publications, Oxford, England (1993).

[18] S. Sofianopoulou, "The Process Allocation Problem: A Survey of the Application of Graph-Theoretic and Integer Programming Approaches",*Journal of the Operational Research Society*, vol.1 (1992) pp.317-336.

[19] F. M. Tonge, "Assembly Line Balancing Using Probabilistic Combinations of Heuristics", *Management Science*, vol. 11 (1961) pp. 726-735.