TWO EVOLUTIONARY APPROACHES TO CROSS-CLUSTERING PROBLEMS

Henri Luchian
Faculty of Computer Science,
"Al.I.Cuza" University of Iasi,
Romania
hluchian@infoiasi.ro

Ben Paechter
School of Computing,
Napier University,
Edinburgh, U.K.
benp@dcs.napier.ac.uk

Vlad Radulescu
Faculty of Computer Science,
"Al.I.Cuza" University of Iasi,
Romania

Silvia Luchian
Faculty of Computer Science,
"Al.I.Cuza" University of Iasi,
Romania

Abstract. Cross-clustering asks for a Boolean matrix to be brought to a quasi-canonical form. The problem has many applications in image processing, circuit design, archaeology, ecology etc. The heuristics currently used to solve it rely on either topological sorting or quasi-random search. We present here two evolutionary approaches to this problem: a permutation-based solution and a clustering one. The results on both real data and randomly generated, scalable, test data show very good convergence and encouraging efficiency properties, mainly for our second approach.

1 Introduction

Cross-clustering – "classification croisée" in French, [DID82] – asks for a (sparse) Boolean matrix to be brought to an "optimal" quasi-canonical form (QCF), only by repeatedly interchanging two rows or two columns. The optimal QCF, however, is loosely defined: the 1's have to be compacted into (not necessarily "rectangular") blocks, usually along one of the diagonals of the matrix. Figure 1 gives an example of an input matrix and a corresponding QCF solution (dark pixels represent the 1's).

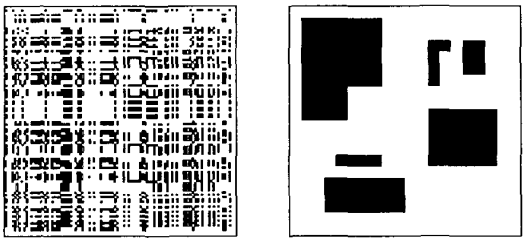

Figure 1. Pixel-level representation of a 100×100 input matrix and of a corresponding optimal QCF.

On one hand, this non-rigorous statement of the problem does not encourage the use of hard-computing techniques (linear programming, normal-forms transformations); on the other hand, it can lead to an optimizational view on cross-clustering.

There is no exact deterministic algorithm for cross-clustering. Currently used heuristics are inspired from topological sorting, from algorithms for algebraic normal forms or even random iterative procedures ([DID82]). For example, the topological sorting approach aims at defining a partial ordering on the set of rows/columns and then interchanging rows or columns according to this order. All of the currently used heuristics hardly achieve the required optimal quasi-normal forms, even for matrices with only hundreds of elements.

We illustrate in the next section the interpretation of cross-clustering in Archaeology; other well-known applications are in circuit design, image processing, Ecology etc.

Cross-clustering can be seen in two complementary ways. One obvious point of view comes from its particular, algebraic flavor: cross-clustering is akin to bringing a matrix to a quasi-canonical form. The other view comes from the observation that cross-clustering asks for a solution which can be seen as the Cartesian product of the solutions to two instances of the usual clustering problem. These two views on cross-clustering inspired two approaches for solving the problem in an evolutionary setting: a permutation-based approach and a "block" approach.

The rest of the paper is organized as follows: section **2.** describes a domain interpretation of cross-clustering, namely in Archaeology; section **3.** describes our permutation-based approach; section **4.** deals with the blocks approach; section **5.** discusses experiments and further work.

2 Cross-clustering – the archaeological view

A well-known application of cross-clustering is data sequencing in Archaeology. The idea of using Boolean matrices for data sequencing dates back at the beginning of the century; credit for this idea is given to one of the greatest Egyptologists, Sir W.M. Flinders Petrie. We describe below the version of cross-clustering which is considered in data sequencing.

n objects are described by means of **a** attributes; each object displays a small number of these attributes and usually each attribute appears for relatively few objects. A matrix **M** can be built, with object identifiers as row labels and attribute identifiers as column labels. A 1 is placed in position **M**(i,j) if object i has the attribute j; otherwise, a **M**(i,j)=0. Usually, **M** is a sparse matrix. The data sequencing problem asks for the matrix **M** to be brought to a form where the 1's are compacted in blocks which "contain"

0-7803-5536-9/99/$10.00 ©1999 IEEE

as few 0's as possible; the only operations to be used are rows or columns swap. Figure 2 shows a small-size example: the input matrix **M** and a quasi-normal form of it.

	Attr 1	Attr 2	Attr 3	Attr 4	Attr 5	Attr 6	Attr 7	Attr 8	Attr 9	Attr 10
Obj1	1	0	1	1	1	0	1	0	0	1
Obj2	0	1	0	0	0	0	0	1	1	0
Obj3	1	0	0	1	1	1	1	0	0	1
Obj4	0	1	0	0	0	0	0	1	1	0
Obj5	0	0	0	0	1	0	0	1	1	1
Obj6	1	1	0	0	1	0	0	0	0	0
Obj7	0	1	0	0	0	0	0	1	0	1
Obj8	0	1	1	1	1	0	1	0	0	1
Obj9	1	0	0	0	0	0	0	0	0	1
Obj10	1	0	0	0	1	0	0	1	0	1

2 a.

	Attr 6	Attr 4	Attr 7	Attr 3	Attr 1	Attr 10	Attr 5	Attr 2	Attr 8	Attr 9
Obj4	0	0	0	0	0	0	0	1	1	1
Obj2	0	0	0	0	0	0	0	1	1	1
Obj5	0	0	0	0	0	1	1	0	1	1
Obj7	0	0	0	0	0	1	0	1	1	0
Obj10	0	0	0	0	1	1	1	0	1	0
Obj6	0	0	0	0	1	0	1	1	0	0
Obj9	0	0	0	0	1	1	0	0	0	0
Obj8	0	1	1	1	0	1	1	1	0	0
Obj1	0	1	1	1	1	1	1	0	0	0
Obj3	1	1	1	0	1	1	1	0	0	0

2 b.

Figure 2. An instance of the cross-clustering problem: a–the input matrix M; b–quasi-normal form of M.

If **M** coded for archaeological data as described above, then the quasi-normal form in figure 2.b would suggest that the 10 objects belong to 4 stages of the culture under research; the object sets A={4,2}, B={5,7,10}, C={6,9}and D={8,1,3} would represent the four stages. Using the set notations A, B, C, D for the stages, then one can consider that the attribute set {2,8,9} is characteristic for stage A, the attribute set {10,5,2,8} characterizes stage B etc.

The problem at hand can be seen as a special case of clustering by partition: given a set of **n** objects and their characterization by means of **a** categorial (Boolean) attributes, find a partition of the set of objects and (simultaneously) select the significant attributes for each class of the partition; the optimality criterion is given by the accuracy of the quasi-normal form of the matrix (e.g., minimize the number of 0's inside each block of 1's). Note that the number of stages (cardinalities of the partitions) is not known beforehand, so this is an unsupervised clustering problem. Since the optimal partition would always be the one of cardinality **n**, an additional optimality criterion is the maximality of the blocks of 1's.

3 A permutation-based approach

Our first approach has been an "optimal permutation" one: given **m**, a Boolean (sparse) matrix, find the permutation of rows, *row_perm*, and the permutation of columns, *col_perm*, which, when applied to the given matrix, bring **M** to an "optimal" quasi-normal form. Under this approach, the solution in figure 2.b above is: *row_perm* = (4,2,5,7,10,6,9,8,1,3), *col_perm* = (6,4,7,3,1,10,5,2,8,9).

The criterion for QCF optimality is not obvious. Indeed, an attractive criterion is given by the condition that as few 0's as possible exist "inside" the blocks of 1's and as few as possible 1's lie "outside" these blocks. However, if the automated procedure would only use this criterion, then the optimal QCF may consist of blocks of single 1's; a restriction which would enforce the decrease of the number of blocks of 1's has to be considered – see the discussion below on the evaluation.

Representation. Under this approach, the representation is straightforward: a chromosome encodes a pair of candidate permutations – one for the rows and one for the columns[1]. A permutation is represented by a sequence of either **n** (for rows) or **a** (for columns) natural numbers between 1 and **n**, respectively 1 and **a**, with no repetition. A major problem which arises with this representation is epistasis ([REW95]): not only the well-known interdependence of the genes in any permutation encoding has to be taken into account, but also the effect on convergence of the simultaneous search of two correlated permutations. Indeed, any change in the row permutation may alter dramatically the corresponding optimal column permutation and vice-versa.

Figure 3 illustrates the fitness-distance correlation, using Hamming distances, ([JOF95], [SCK96]) for the representation described above and fitness function **f** given below. For calculating and representing the FDC, we used matrices with 10,000 elements and around 30% non-zero values. In order to keep the figure readable, we give relatively few points out of the 5,000 considered; these points however describe accurately the overall shape of the distribution FDC distribution. The FDC value is around 0.17, which gives little hope for a good convergence of the algorithm.

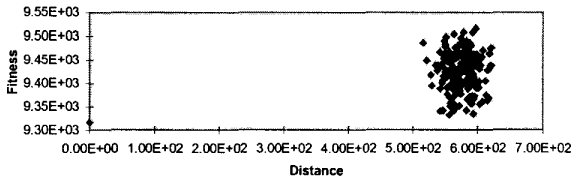

Figure 3. Fitness-distance correlation for the permutation-based representation.

Experiments confirmed that simultaneous evolution of the two permutations does not lead to convergence to a good solution: for all tests below, no acceptable QCF has been obtained in 1000 generations. Our solution to this has been an alternate evolution: we designed a fitness function, $\mathbf{f}_{row}$ and a set of operators, *op-row*, for row-permutation evolution and a

[1] Another possibility would be to encode only one of these permutations and use another heuristic for finding the other one.

fitness function, $\mathbf{f}_{col}$, and a set of operators, *op-col*, for column-permutation evolution. The general scheme of the genetic algorithm is slightly altered: every **g** generations, the algorithm switches between the use of ($\mathbf{f}_{rows}$, *op-row*) and the use of ($\mathbf{f}_{col}$, *op-col*); **g** is a parameter of the algorithm. The relation between **g** and the average fitness of the solutions found over 10 runs of the algorithm, for each value of **g**, is illustrated in figure 4; the average fitness values are scaled. The input matrix had 100 rows and 100 columns, with a percentage of 30% non-zero elements. Experiments with other input matrices were similar, so that we used **g**=10 in all further experiments.

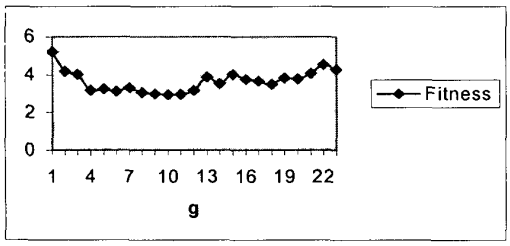

Figure 4. Average best_fitness over 10 runs for each value of **g** between 1 and 23.

Evaluation. The sought solution is a pair (*perm-row-best*, *perm-col-best*) which, when applied to the rows and columns of the input matrix **M**, gives an optimal QCF. A candidate solution *Chr* = (*perm-row*, *perm-col*) is evaluated as follows:

- *build the matrix* $\mathbf{M}_{fin}$ *which results by applying the respective permutation to the rows / columns of* **M**;

- *for i=1..n, find, in* $\mathbf{M}_{fin}$, the *positions* $j_{first}(i)$ *and* $j_{last}(i)$ *of the first and last* 1 *in row i; let* $nr\_1(i)$ *represent the number of 1's in row i;*

- *if* $nr\_1(i)=0$ *(and so,* $j_{last}(i) = j_{first}(i)=0$*), then* $f_{i\_row}(Chr)=0$*, else*

$f_{i\_row}(Chr)=Min((j_{last}(i) - j_{first}(i)), (j_{first}(i) - j_{last}(i) + \mathbf{a})) / nr\_1(i)$;

- *calculate* $\mathbf{f}_{row}(Chr) = \sum_{i=1}^{n} \mathbf{f}_{i\_row}(Chr)$;

- *use analogous steps for calculating* $\mathbf{f}_{col}(Chr)$;

- *for the standard GA scheme, the fitness of Chr is:* $\mathbf{f}(Chr) = f_{row}(Chr) + f_{col}(Chr)$;

- *the goal is to minimize the fitness.*

This fitness function aims at minimizing the number of 0's inside the blocks of 1's; it also tends to minimize the dimensions of such blocks. Its advantage is that it treats rows and columns in a non-discriminatory fashion, so that there is no need to switch fitness functions while alternating the row and column evolution, in the scheme described above.

An alternative way of evaluating the candidate QCF of matrix **M** is to express the compactness of existing groups of 1's, by means of fitness function $\mathbf{f}_1$, with row and column components as below:

$$\mathbf{f}^1_{row}(Chr) = \sum_{i=1}^{n} \sum_{\substack{j \\ \mathbf{M}(i,j)=1}} \left| j - \frac{\mathbf{f}_{i-row} \cdot nl\_i}{2} \right|; \qquad \mathbf{f}^1_{col}(Chr) = \sum_{j=1}^{n} \sum_{\substack{i \\ \mathbf{M}(i,j)=1}} \left| i - \frac{\mathbf{f}_{j-col} \cdot nl\_j}{2} \right|$$

where $\mathbf{f}_{j\text{-col}}$ and nl_j are analogous to their row counterparts.

Operators. Each mutation operator has two versions: a row operator and a column one. The only difference between the two versions of the same operator is that one is used for genes of the row permutation and the other one – for the column permutation.

- *swap mutation*: randomly choose two genes in the row (column) permutation and interchange them;

- *partial permutation*: randomly choose a subset of the set of row (column) genes in the chromosome and randomly interchange them;

- *sequential permutation*: same as before, but the selected genes have to be successive;

- *circular permutation*: a randomly chosen sequence of row (column) genes is rotated.

The crossover operator is also typed – it has a row version and a column one, which only differ in the part of the chromosome where each one can be applied. We used *cycle crossover* ([OLI87]).

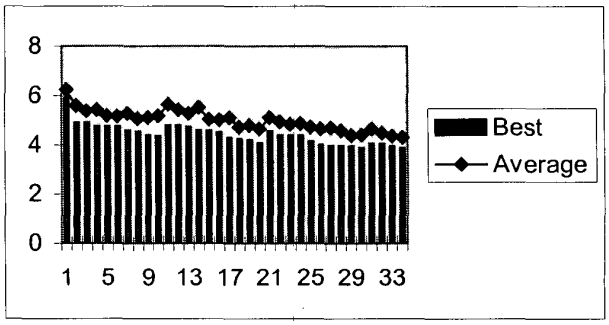

Figure 5. Typical evolution of best and average fitness for **g**=10.

As mentioned above, in order to cope with epistasis, every **g** generations the algorithm switches the evaluation procedure from $\mathbf{f}_{row}$ to $\mathbf{f}_{col}$ and the set of operators from row ones to column ones. We used rank-based, non-elitist selection ([MIC96]). Figure 5 shows a typical evolution of the best_so_far fitness value and of average fitness, for **g**=10. Note the "saw-teeth" shape: the evolution cycle starts with a constant (row) fitness decrease until the switching generation; next, the (column) fitness increases abruptly compared to the last row value, then constantly decreases until the next switching generation etc. A smooth, but less effective, evolution occurs if $\mathbf{f}(Chr) = f_{row}(Chr) + f_{col}(Chr)$ is used and only the set of operators is changed at switching points.

Our permutation-based approach has an obvious weak point: the blocks of 1's, which are the target of the optimization process, cannot be defined, hence recognized and taken into consideration during the evaluation process; therefore, the fitness function loosely interprets the quality of the candidate solution. Indeed, the blocks of 1's are only

vaguely approximated by the positions of the first and last 1 in a row/column. Fitness-distance correlation confirms this observation: the fitness-distance correlation is weak (values between 0.088 and 0.17 – see [LUC99]), which does not suggest ([JOF95]) a good convergence of the genetic algorithm. On the bright side, this approach is more sensitive to rows/columns interchange inside each found block of 1's or 0's – a feature that will disappear in our second, more robust, approach.

4 A block-based approach

Our second algorithm, which we call BLOCKS, is inspired from an existing random-search cross-clustering technique presented in [DID82]. Roughly, that procedure clusters the rows/columns of the initial matrix at random; subsequently, the elements at the intersections of each row cluster with each column cluster are treated as *blocks* (either "1" blocks or "0" blocks). The iterative part of the procedure changes the row/column clusters with the aim of improving the quasi-canonical form (a secondary goal is to decrease the number of clusters).

In this approach, a solution to the cross-clustering problem consists of a partition of the set of rows and a partition of the set of columns; these two partitions give an optimal QCF of the input matrix, by means of the sub-matrices defined by their intersections. That is, all the compact blocks of 1's of an optimal QCF have to be among those sub-matrices. In order to obtain the optimal QCF from such a solution, one has to interchange rows/columns until the identifiers in each class of the two partitions are grouped in successive positions. The possible requirement to further improve the solution by placing the found blocks of 1's in specific positions (e.g., along a diagonal) can be met by a procedure described in section 5. Under the block-based approach, the solution in figure 2.b above is expressed as: k=3, l=3, row_partition {(8,1,3), (10,6,9), (4,2,5,7)}, column_partition {(6,4,7,3), (1,10,5), (2,8,9,)}. The resulting blocks are considered either as 1_blocks (e.g., the block at the intersection {8,1,3} × {6,4,7,3}) or 0_blocks (e.g., {4,2,5,7}×{6,4,7,3}), depending on which Boolean value appears more often in that block. Note that, under this approach, the compactness of the blocks of 1's is the optimality criterion, since the order of the rows/columns in a class of the respective partition is irrelevant; in other words, the block-based approach is not sensitive to rows / columns interchanges inside each block of 1's or 0's.

More formally, let $\mathbf{M_{m*n}}$ be the input matrix, $\mathbf{I}=\{\mathbf{i_1},...,\mathbf{i_m}\}$, the set of row identifiers and $\mathbf{J}=\{\mathbf{j_1},...,\mathbf{j_n}\}$, the set of column identifiers. We have to find the optimal pair (row_partition, column_partition), where the row_partition is $(\mathbf{S}_1, \ldots, \mathbf{S}_k) = (\{i^1_{s_1},...,i^{m_1}_{s_1}\}, ..., \{i^1_{s_k},...,i^{m_k}_{s_k}\})$ and the column_partition is $(\mathbf{Q}_1,..., \mathbf{Q}_l)=(\{j^1_{q_1},...,j^{n_1}_{q_1}\},...,\{j^1_{q_l},...,i^{n_l}_{q_l}\})$. Since the cardinalities of the two partitions, **k** and **l**, are not known beforehand, we are dealing with an unsupervised cross-clustering problem.

In order to assess the dimension of the search space, suffice it to say that, for a 50×50 input matrix, the Cartesian product of the set of all row partitions and the set of all column partitions has the order of magnitude 10^{94}, while for a 100×100 input matrix, the number of different pairs (row_partition, column_partition), only for k=l=5, has the order of magnitude 10^{136}.

Halting condition. Since blocks are clearly defined in this approach, we can use a convenient halting condition for the algorithm. We chose a condition that expresses the level of accuracy of the found solution: the algorithm stores the first candidate solution for which all the blocks have under **p**% "minority elements" in any block, where **p** is a parameter of the algorithm; the algorithm continues to run for 10 generations and then stops – the solution given by the algorithm is the best between the stored solution and the last_generation_best. The solution in figure 2.b has **p**=33.3, which is likely to be acceptable for a matrix with 40% non-zero values.

Selection and replacement strategy. Since the search space is huge and also because the likely interval containing all fitness values is comparable to $[0;10^6]$, we tried to prevent the selection pressure caused by occasionally-found local optima. To this end, we used *rank-based non-elitist selection*. Further improvement of the convergence has been obtained by means of a combined replacement strategy: all descendants obtained after recombination are placed in the next generation (the above-mentioned selection mechanism being used for applying the crossover), while the rest of the new generation is selected at random from the current population. This leads to an even lower selection pressure.

Representation. A chromosome consists of an encoding of the two partitions: each row_ gene represents a subset of the set of rows and each column_gene represents a subset of the set of columns; obviously, the reunion of all subsets represented by row_(column_) genes equals the entire set of rows (columns). Obviously, chromosomes may have different lengths, which allows for performing unsupervised processing.

A gene represents an unordered subset of either the set of rows or columns; each row/column is represented exactly once in a chromosome. Blocks are defined by the Cartesian product of the row genes and column genes.

The Fitness-Distance Correlation is more encouraging than the one for the permutation-based representation: the FDC values are around 0.35 for all samples (sample sizes around 5,000 points). Figure 6. illustrates the FDC distribution, preserving only the overall shape and displaying only relatively few points.

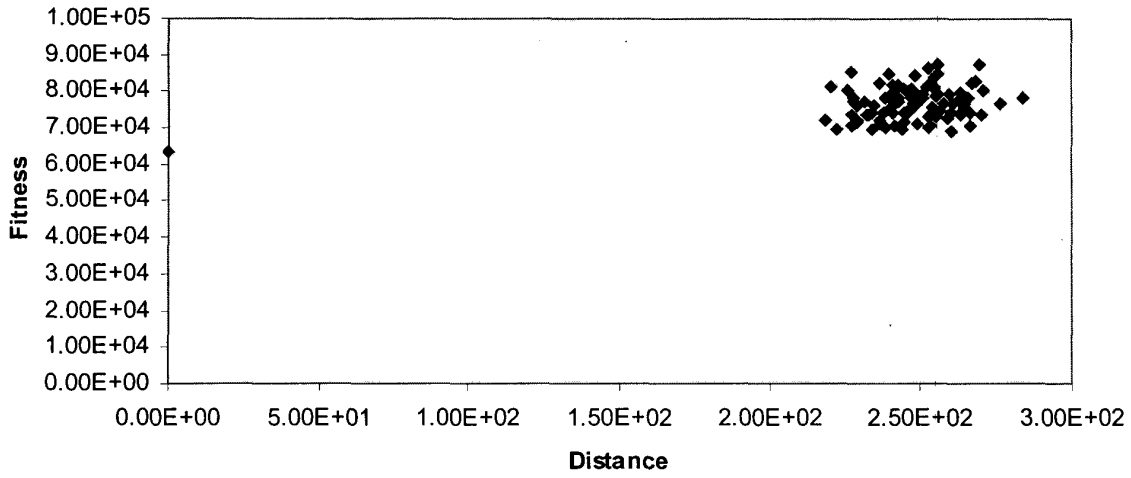

Figure 6. Fitness-distance correlation for the block-based approach.

Evaluation. A pair (row_gene, column_gene) defines a block. A 1_block is a block in which the non-zero values are at least as numerous as the 0's; the other blocks are 0_blocks.

In early experiments, on moderate size matrices (hundreds of elements), evaluation of chromosome *Chr* proceeded as follows:

- penalties of all blocks encoded by *Chr* are summed up;
- a penalty is attached to each block represented in *Chr*: $\mathbf{pen}_{block}=(\min(\#_{block}0, \#_{block}1))^2$;
- in order to control the growth of **k** and **l**, the penalty is adjusted, using the power $\lceil \log_2(\mathbf{k}+\mathbf{l}) \rceil$, where $[x]=\max\{s/\ s\in\mathbf{N},\ s\le x\}$;
- in order to keep track of different x values leading to the same [x] value, the term **k.l** is used: $eval(Chr)=(\sum_{b\cdots\mathbf{in}\cdots ch} pen_b + 2)^{\lceil \log_2(\mathbf{k}+\mathbf{l}) \rceil} + \mathbf{k}\cdot\mathbf{l}$;

-finally, the fitness function to be minimized by the algorithm is obtained by scaling the raw fitness: $f\_bl\ \ (Chr)=(eval(Chr))^{1.3}$.

The scaling exponent has been obtained empirically, as a minimal value which prevents premature convergence.

When experimenting with real-world matrices (up to tens of thousands of elements), the dramatic increase of the search space cardinality made this fitness function perform poorly. Therefore, we used a simplified version of the fitness function, which avoided any increase in selection pressure:

$$f\_block(Chr)=(\sum_{i=1}^{no\_of\_blocks} \frac{\#misplaced\_elements\_in\_block\_i}{\#elements\_in\_block\_i})\times(k+l)$$

Operators. All operators are typed – each one with row and column versions. Row-crossover (applied at a rate of 60%) affects only row genes, the column genes being passed over to descendants without any change. This operator acts as follows:

- randomly choose a row identifier i, $1 \le i \le \mathbf{n}$;
- all rows which precede row i in the first parent are copied, in the same position, into the first descendant; the other rows are copied, in their respective positions, into the second descendant;
- an analogous procedure is applied to rows in the second parent;
- delete any empty row-genes of the descendants.

Three mutation operators, M_1, M_2 and M_3, are used (each one with row and column versions). Genes involved in each mutation operator are from the same chromosome.

M_1 (interchange): a randomly chosen row/column is moved from its original gene to another gene of the same type; if the original gene remains empty, then it is removed from the chromosome.

M_2 (join): two randomly selected non-empty genes of the same type are joined in a single, new gene of the same type.

M_3 (split): a randomly selected gene is split into two new, non-empty genes of the same type.

Each of the six mutations was assigned a rate of 0.05. Mutations are applied to offspring obtained by crossover.

5 Experiments and further work

Experiments described below have been performed using randomly generated test matrices; this makes the problem more difficult, since many of the real-world applications ask for bringing to an optimal QCF matrices that originally *were* in an optimal QCF. Indeed, two examples illustrate the situation:

- data sequencing: objects had been put in what eventually became an archaeological site, *in an order that itself indicates the optimal QCF*. It is this original order that the archaeological studies aim to discover;

- image processing: blocks would represent (moving) objects[2].

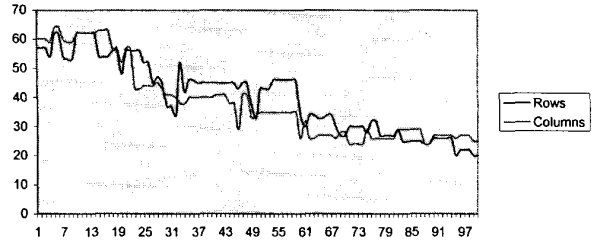

Figure 7. Epistasis: evolution of the number of row groups and column groups.

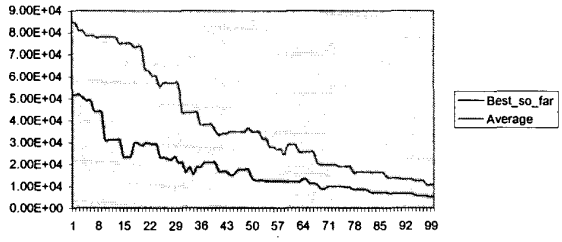

Figure 8. Best_ fitness and average_fitness evolution (block-based approach).

All the existing algorithms (e.g., CROBIN [DID82]) or heuristics (topological sorting, quasi-random search) perform supervised cross-clustering: **k** and **l** are not changed during the run. We implemented them for comparison purposes, but no such implementation led to acceptable results for matrices with more than 2,500 elements. In order to make the comparisons more significant, for each application we adapted the problem definition for the evolutionary approach to match the other techniques as much as possible. Since other methods did not prove useful, we focused the experiments on the convergence of our evolutionary algorithms. We give here a few comments on the figures above; all figures and comments refer to 100×100 input matrices.

Figure 7 illustrates the evolution of the number of groups (partition cardinality) for rows and columns. Epistasis is clearly illustrated by the fact that relatively small variations in one dimensions correspond to relatively large variations in (the local optimum for) the other dimensions.

Figure 8 shows a typical evolution of the best_so_far fitness and that of average_fitness (recall that non-elitist selection has been used). Situations when the decrease of best_so_far corresponds to an evolution to the worst of the average fitness (generations around 15, 25 and 45) may illustrate cases when a (much) better individual is obtained from relatively poor parents. Mutation is not likely to produce such situations, since moving one row/column to another block or joining/splitting existing groups cannot, by itself, improve very much a candidate solution; therefore, we think that crossover is a powerful operator for this problem and is the main responsible for the convergence of the algorithm.

Figure 9 gives a view of typical evolutions of the best_so_far fitness for two settings of the (global) mutation and crossover rates. We have empirically set p_c to 60% and p_m to 5% for each mutation; running a supervisor GA (GRE86) would have been too resource consuming, even though we took advantage of sparse matrices implementations[3].

We aim at improving the fitness function and the set of operators, so as to take into consideration not only the homogeneity of each block, but also the order of rows/columns in each block and the relative positions of the blocks in a candidate solution. We are considering two new mutation operators, which change the order of either rows/columns in a gene, or genes of the same type in a chromosome; also, the fitness function will contain as penalties the number of 0's placed between 1's, both at the level of the elements of the matrix and at the block level.

We are currently implementing a two-step strategy: in a way similar to Δ-coding ([SCB90]) or to Dynamic Parameter Encoding ([WMF91]), we vary the level of optimization from one step to the other. The two steps are:

1. Run BLOCKS with input matrix **M** and let MAT_BEST be the found solution;
2. Build MAT, the "kernel" matrix ([DID82]) of MAT_BEST. MAT is a **k**×**l** binary matrix, the elements of which are 1 for 1_blocks and 0 for 0_blocks. Run BLOCKS with MAT as input matrix.

We also envisage a combination of the two approaches: a run of BLOCKS, followed by the permutation-based algorithm, could further improve the solution. We are currently carrying out experiments with these ideas.

6 Conclusions

We presented two evolutionary approaches to the general unsupervised cross-clustering problem. Our first solution is a two dimensional optimal permutation search, while the other one, inspired from an existing heuristic, searches for optimal row and column partitions. Each approach has its own advantage, not shared with the other one: the permutation-based solution is more sensitive to detail-level improvements of a candidate solution, while the block-based approach gives a better interpretation to the blocks of 1's and 0's in the candidate solution. Our experiments showed – as expected - that a proper interpretation of the candidate solution is more important than detail-level improvements: the second algorithm converges much quicker and does not display premature convergence.

[2] Note that the matrix may not be boolean in such applications. Our algorithm can be easily adapted to such matrices.

[3] Using sparse-matrix implementations, running time has been cut by two thirds.

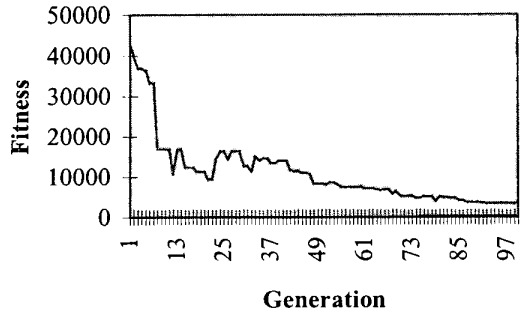

Figure 9a. Best_so_far fitness evolution for a 100*100 input matrix with 20% non-zero values; p_c=50%, p_m=3% for each mutation operator.

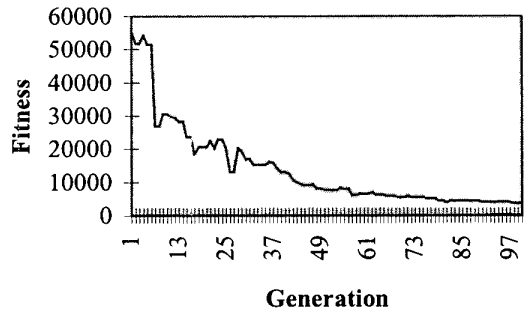

Figure 9b. Best_so_far fitness evolution for a 100*100 input matrix with 30% non-zero values; p_c=60%, p_m =5% for each mutation operator.

Acknowledgements

Henri Luchian gratefully acknowledges the sponsorship from Lockheed Martin International, Microsoft – Romania and The School of Computing at Napier University, Edinburgh.

The authors wish to acknowledge the contribution of M. Petriuc (Microsoft, Redmond, U.S.A.); he carried out part of the computer work and has contributed to the permutation-based approach.

References

[DID82] – Diday, E., Lemaire, J., Pouget, J., Testu, F.: "Elements d'analyse de donnees", Dunod, Paris, 1982.

[GRE86] – Grefenstette, J.J.: "Optimization of Control Parameters for Genetic Algorithms", in IEEE Transactions on Systems, Man and Cybernetics, vol. 16, no.1, 1986, pp.122-128.

[JOF95] – Jones, T., Forrest, St.: "Fitness Distance Correlation as a Measure of Problem Difficulty for Genetic Algorithms", in Proc. of the 6th ICGA, 1995, Morgan Kaufman, pp.184-192.

[LUC99] – Luchian, H.: "Three Evolutionary Approaches to Clustering", to appear in the volume EUROGEN'99, Jyvaskila, Wiley&Sons,1999.

[MIC96] - Michalewicz, Z.: "Genetic Algorithms + Data Structures = Evolution Programs", 3rd edition, Springer Verlag, 1996.

[OLI87] – Oliver, I.M., Smith, D.J., Holland, J.R.C.: "A Study of Permutation Crossover Operators on the Traveling Salesman Problem", in Proc. of the 2nd ICGA (J.J.Grefenstette - editor), Lawrence Erlbaum Associates, 1987, pp.224-230.

[REW95] – Reeves, C.R., Wright, C.C.: "Epistasis in Genetic Algorithms: An Experimental Design Perspective", in Proc. of the 6th ICGA (L.Eshelman-editor), Morgan Kaufman Publishers, 1995, pp.217-224.

[SCK96] – Schoenauer, M., Kallel, L.: "Fitness-Distance Correlation for Variable Length Representations", Raport de recherche, Ecole Polytechnique de Palaiseau, CNRS URA756, 1996.

[WMF91] – Whitley, D., Mathias, K., Fitzhorn, P.: "Delta-coding – An Iterative Search Strategy for Genetic Algorithms", in Proc. of the 4th ICGA, (Belew and Booker-editors), Morgan Kaufman Publishers, 1991, pp.77-84.