Available online at www.sciencedirect.com

SCIENCE DIRECT®

European Journal of Operational Research 160 (2005) 638–662

EUROPEAN JOURNAL OF OPERATIONAL RESEARCH

www.elsevier.com/locate/dsw

A list-based compact representation for large decision tables management

J.A. Fernández del Pozo [a,*], C. Bielza [a], M. Gómez [b]

[a] Decision Analysis Group, Artificial Intelligence Department, Technical University of Madrid, Campus de Montegancedo, Boadilla del Monte, 28660 Madrid, Spain

[b] Computer Science and Artificial Intelligence Department, University of Granada, Daniel Saucedo Aranda, 18071 Granada, Spain

Received 23 January 2002; accepted 15 June 2003
Available online 15 December 2003

Abstract

Due to the huge size of the tables we manage when dealing with real decision-making problems under uncertainty, we propose turning them into minimum storage space multidimensional matrices. The process involves searching for the best order of the matrix dimensions, which is a NP-hard problem. Moreover, during the search, the computation of the new storage space that each order requires and copying the table with respect to the new order may be too time consuming or even intractable if we want a process to work in a reasonable time on an ordinary PC. In this paper, we provide efficient heuristics to solve all these problems. The optimal table includes the same knowledge as the original table, but it is compacted, which is very valuable for knowledge retrieval, learning and expert reasoning explanation purposes.

Keywords: Combinatorial optimisation; Decision analysis; Decision support systems; Heuristics; Learning and explanation

1. Introduction

The Decision Support Systems (DSS) now in demand are very complex knowledge-based systems. Based on the Decision Analysis discipline, see, e.g., Raiffa (1968), their construction involves structuring the decision-making problem (using modern graphical models like influence diagrams, see Shachter, 1986), eliciting uncertainty and preferences (using probability and utility models, see Keeney and Raiffa, 1993), and solving the problem. Although all these tasks are difficult, we shall go one step further.

Once we have solved the problem, we have one decision table per decision variable, containing its optimal alternatives, i.e. the alternatives of maximum expected utility. In general, a decision table can be considered as a set of attributes or variables that determine an action, alternative or policy. The table grows exponentially with the number of attributes. In real problems, each table may have up

[*] Corresponding author. Tel.: +34-91-3367433; fax: +34-91-3524819.

E-mail addresses: jafernandez@fi.upm.es (J.A. Fernández del Pozo), mcbielza@fi.upm.es (C. Bielza), mgomez@decsai.ugr.es (M. Gómez).

to millions of rows (each attribute configuration) and typically more than twenty columns (attributes), the results of the problem thereby relying on a combinatorial knowledge representation space.

Storing and managing so much information is not the only problem that arises. Decision-makers use the decision tables to query which is the best recommendation for a certain case or attribute configuration. Indeed, these tables are very important for this purpose. However, decision-makers demand DSSs that provide clear, concise, consistent and complete explanations that translate the reasoning mechanism (underlying the table content) into their domain to justify the decisions proposed. The explanation should give a description of why the proposed decision is optimal and new insights into the problem solution. This kind of knowledge synthesis will also serve for validating the system.

In fact, a system providing good explanations is very hard to build (Henrion et al., 1991). The main reasons are: explanations should be presented from all the possible points of view in a structured and hierarchical way, with different levels for users and analysts; they should only employ knowledge from the user domain; they should be as general as possible and emphasise the evidence of the presence (absence) of arguments in favour of (against) the proposal.

Despite these difficulties, in this paper we show how they can be addressed, resulting in useful systems for real problems.

So-called *learning from data* is a general goal pursued by a number of disciplines for extracting important patterns and trends and understanding what the data say. Therefore, it is easy to imagine that our proposal in this paper may bear some resemblance to some such techniques. For example, the aim of (supervised) classification from Machine Learning is to learn a mapping from a vector of attributes to a class variable. In our case, this variable is the optimal alternative from the decision-making problem. Tree-based classifiers, such as CART (Breiman et al., 1993), ID3 (Quinlan, 1986), C4.5 (Quinlan, 1993), have proved to be particularly useful with non-metric data and without prior information about the appropriate form of classifier (Duda et al., 2001).

However, our approach is quite different. Our method is based on a list-based structure rather than on a tree-based structure. The search for good candidates is global, involving the whole attribute set, while trees use a greedy local search over structures. Avoiding the hierarchism of the tree-building process, we overcome the typical instability found in the trees with regard to small changes (see Hastie et al., 2001). On the other hand, tree-classifiers are very flexible and can be used with every data type (metric, non-metric, or in combination). Our methodology will be limited to finite data.

In this paper, the central idea is that the table content is not knowledge unless it is organised somehow, like a torn book is knowledge only when it has been properly stuck together and repaired. Unlike classification trees, our list does not have to be built, it has to be reorganised. Trees aim at maximising a score of class purity, which does not make any sense for our "classifier", because it does not yield misclassifications. All the cases are already correctly classified, and we want to explain why they are classified like this. We translate this problem into finding the shortest list. In a sense, we work with full trees, which are later "pruned" when we find sets of cases that share certain information values (see Section 2.3 below), providing just the sought-after explanations.

Also, the tables were originally collected for a purpose other than the explanations we seek, i.e. they are the influence diagram evaluation output. Thus, data were not collected using efficient strategies to answer specific questions; they are observational data as opposed to experimental data. Moreover, the rows of our tables range over all the attribute configurations. It means that they cannot be repeated, obviously not being the classical carefully selected laboratory training sample found in Machine Learning and Statistics. Hence, missing values arise only in the class variable, unlike tree-based methods which have missing values in the attributes. This occurs when the decision tables only include a subset of the whole problem solution due to computational problems, leading to unknown policies. We will see below how to deal with these values.

This discussion suggests setting our framework in the Data Mining field, see, e.g., Hand et al. (2001).

Data mining is a rapidly growing interdisciplinary area of tools in which Machine Learning, Pattern Recognition, Database Technology, Statistics, among others, play a role. Moreover, it can be seen as part of the overall process of knowledge discovery in databases (Fayyad et al., 1996). Although rarely documented outside the database literature, the typical massive data sets in Data Mining demand explicit specification of the data management strategies (the way in which the data are stored, indexed, accessed,...). This paper will account for this as well.

We find another possible connection between our approach and rough sets (Pawlak, 1991, 1997). Rough sets have been used for rules induction (Machine Learning) or as a technique for eliminating redundant information (data analysis). The rough set criterion has been shown to be a special case of the entropy criterion used by ID3 (Wong et al., 1986). The rough set algorithm is based on set theory and topology and it is an inductive learning algorithm. We are more interested in a deductive algorithm. Also, our approach groups cases with the same class (like the class relation in rough sets) and with the same attribute value (like the equivalence relation in rough sets), although differently. Our groups are pure; all the cases belong to the same class, whereas rough sets approximate the classes partly via the upper subsets, which mix cases from various classes.

The paper is organised as follows. Imitating the way computers manage multidimensional matrices, Section 2 defines a new list-based structure for storing the decision tables. The list basically searches for an organisation of the decision table columns that minimises storage space. Section 3 offers two heuristics to guide the search for the optimum organisation: a variable neighbourhood algorithm and a genetic algorithm. The performance of each one is tested via some illustrative experiments. Regardless of the chosen searching process, the information needs to be copied to a new table with differently ordered columns although this is not feasible if there are a lot of columns. Section 4 deals with this issue and it also provides a test to quickly check whether the new table is better than the old one. As an added value, the final compact table will also explain the DSS proposals. Obviously, the methodology is not only applicable to decision tables, as suggested in the last section.

2. Synthesised tables using *KBM2L* lists

2.1. *KBM2L lists*

In an attempt to prevent some decision tables exceeding the storage capacity of any personal computer, we express their contents by means of another representation.

The set of attributes of a table will be called *schema*. If we impose an order on the components of the schema (attributes), a *base* is a vector whose elements are these attributes. A change in this order modifies the position of the variables in the schema, but not the proposals of the DSS. Also, since we work in discrete attribute domains, we assume an order, natural or conventional, in the values of every domain. So, an *index* is a vector whose elements are the values of the attributes of the base. The index could be interpreted as the coordinates with respect to the base.

By defining an order in the schema attributes and in the domains, we can consider the decision tables as multidimensional matrices (MM). So, the content of the table stored in the cell with coordinates $\vec{c} = (c_0, c_1, \ldots, c_n)$ will be assigned to the position $\mathrm{MM}[c_0, c_1, \ldots, c_n]$. For the ith attribute ($i = 0, 1, \ldots, n$), let D_i denote the cardinal of its domain and $\prod_{j=i+1}^{n} D_j$ its *weight* w_i.

Computers manage multidimensional matrices as lists (Knuth, 1968), where each position is a function of the order chosen for the matrix dimensions. Namely, the values are ordered by means of an application like $f : R^{n+1} \to R$, where

$$f(c_0, c_1, \ldots, c_n) = c_0 \prod_{i=1}^{n} D_i + c_1 \prod_{i=2}^{n} D_i + \cdots + c_n = q \qquad (1)$$

provides the offset q of a value with respect to the first element of the table in a given base. The MM values will be stored successively in a computer, where only the memory address of the first one is known and the offset depends on the base.

The vector of weights is $\vec{w} = (w_0, w_1, \ldots, w_n)$ with $w_k = w_{k+1}D_{k+1}$, $w_n = 1$ and $k = 0, 1, \ldots, n-1$. The weights are the coefficients that multiply the coordinates in (1), like the radix powers in the number value expression (algebraic interpretation). From the point of view of memory positions allocation and access, weight w_i is the number of cells separating two values that only differ one unit in the ith coordinate (geometric interpretation), i.e. the values

$$(c_0, c_1, \ldots, c_i, \ldots, c_n) \quad \text{and}$$

$$(c_0, c_1, \ldots, c_i + 1, \ldots, c_n).$$

We can use relationship (1) to access all the values. Given a cell $\vec{c}$, we can compute its memory position and retrieve its value. Conversely, the index of the cell can be built from the position, provided the base (the order of the attributes) is known.

Let us take the adjacent cells of the list for which the DSS proposes the same action. The resulting *grains* of knowledge or sets of cases with the same optimal policy will be called *items*. The basic idea is that if the content of the table presents some level of granularity, we can store only one value for each group of cases, as sparse matrices do.

With another base, we have the same knowledge in the table but we change the granularity and, hence, the memory requirements to store the final list of items. The objective is to get a base that minimises the number of items, bringing up the grains of knowledge. These will also serve as a means of explanation, finding relationships between groups of attributes and the proposals of the DSS.

For example, let the content of a decision table be represented and stored as the list

$$< (0, \sharp) < (1, \sharp) < \cdots < (p-1, y),$$

$$< (p, x) < \cdots < (q, x) < (q+1, z) < \cdots$$
$$< (w_0 D_0 - 1, \sharp),$$

where each cell of the table is represented as the pair *(offset, policy)*, w_0 is the weight of the first attribute, D_0 is the cardinal of its domain, the sign $<$ shows the order among the memory positions used to store the pair, and $\sharp$ denotes any policy value. Let us suppose the cells between positions p and q, where $p < q$, contain the same policy x, as above.

The new list we propose for this table will include all the cases at the positions between p and q (p and q included) in only one item, saving memory space. Thus, the fragment of the list $< (p-1, y) < (p, x) < \cdots < (q, x)$ collapses as $< p-1, y| < q, x|$. The notation $< offset, policy|$ reflects two ideas. Firstly, the offsets of the items are strictly increasing and, secondly, it summarises a set of adjacent cells with the same policy, which is, at the same time, different from the proposal of the next item.

This list will be called *KBM2L* list. The name stands for a Multidimensional Matrix whose content is a Knowledge Base (which is a more general structure than a decision table), that is transformed into a List (*KBMMtoL* $\sim$ *KBM2L*), i.e. a list of a KB MM. This list must be consistent in the following sense: all the offsets are non-negative and smaller than the maximum $w_0 D_0 - 1$, they are in strictly increasing order, and the policy of adjacent contexts is different (if the policies were identical, the cases would be joined into a single item).

Example 1. Let us take the matrix in the base $[c_0, c_1]$:

		c_1				
		0	1	2	3	4
c_0	0	0	0	0	0	1
	1	1	1	1	1	1
	2	2	1	1	2	2

Its linear storage in memory can be represented by offsets:

```
<(0,0)   <(1,0)   <(2,0)   <(3,0)   <(4,1),
<(5,1)   <(6,1)   <(7,1)   <(8,1)   <(9,1),
<(10,2)  <(11,1)  <(12,1)  <(13,2)  <(14,2),
```

The derived *KBM2L* list has 5 items:

$$< 3, 0| < 9, 1| < 10, 2| < 12, 1| < 14, 2|$$

in offset notation, or

$$< (0,3),0| < (1,4),1| < (2,0),2| < (2,2),1| < (2,4),2|$$

in indices notation. Note that the other base $[c_1, c_0]$ has 12 items.

2.2. Constructing the KBM2L associated with a decision table

The construction process of a *KBM2L* list from a decision table starts with an empty list, i.e. with an item representing the complete absence of knowledge, denoted as −1. Thus, the list is $< w_0D_0 - 1, -1|$.

Each state of the list is different as new cases are added to it. Thus, we have developed software that employs up to 26 rules for item management. Each rule studies what the *KBM2L* list of items is like before and after applying the rule.

For example, R24 is one of the rules addressing an insertion into contexts in the generic list: if the list is

$$\cdots < q_{i+1}, d_{i+1}| < q_{i+2}, d_{i+2}| < q_{i+3}, d_{i+3}| \cdots$$

and a new case $< x, d|$ is to be inserted, with

$$q_{i+2} < x < q_{i+3}, \quad x - 1 = q_{i+2}, \quad d \neq d_{i+2}, \quad d \neq d_{i+3}$$

then the new list is

$$< q_{i+1}, d_{i+1}| < q_{i+2}, d_{i+2}| < x, d| < q_{i+3}, d_{i+3}|.$$

This rule increases the list size, although this is not true for every rule.

Once we have built the *KBM2L* list and we know how to manage its items by following the rules, the list must be optimised. Namely, we look for the order of the schema of attributes or base that minimises the number of items stored for a given matrix. It would imply a change of base, possibly involving copying all the information, and a procedure to guide the search for the best base. Sections 3 and 4 will look into this subject in more depth.

But the process has more complications. The decision tables yielded by solving the decision-making problem usually come from the solution of a subproblem, i.e. they are *partial* tables. The complete evaluation may be so costly (in terms of time and memory requirements) that we actually solve a complete set of subproblems, each one as the result of instantiating some attributes (Ezawa, 1998). Therefore, these tables do not have all the combinations of the attributes.

Thus, the complete process consists of: (1) all the subproblems are evaluated (sequentially or in parallel) and the partial tables are stored in files; (2) the first partial table is translated into a *KBM2L* list; (3) the second partial table is added to the list built beforehand and by means of a learning mechanism. This means that it is advisable to optimise the list while new knowledge is being introduced into the list (i.e. before reading the next table). Each stage in the addition process involves an organisation of items (and thereby of attributes) that facilitates future additions. (4) We continue with the following partial table, collecting all the partial results at the end to shape a real decision table. Lastly and obviously, the final list is optimised.

Example 2. By way of an illustration, let us take a simple decision problem with its knowledge stored in two decision tables. The table schema has three attributes $\{X_0, X_1, X_2\}$, the base is [0,1,2] and all the domains are binary (values 0 and 1). $\{A, B, C\}$ is the set of alternatives. The first table, see Table 1, is the evaluation output of the instance with $X_0 = 0$ and the second one, see Table 2, with $X_0 = 1$.

The initial empty list is $< 7, -1|$. The procedure begins with the translation of one table to the *KBM2L*. Let us take Table 1. We read each case of the table and insert the cases one by one into the *KBM2L*, see Table 3. Note how fragmented the list is, with 5 items. Next we optimise this list. Since it is a very small example, we can test *all* the possible

Table 1
Partial decision table ($X_0 = 0$)

X_0	X_1	X_2	Offset	Policy
0	0	0	0	*B*
0	1	0	2	*B*
0	1	1	3	*A*
0	0	1	1	*A*

Table 2
Partial decision table ($X_0 = 1$)

X_0	X_1	X_2	Offset	Policy
1	0	1	5	C
1	1	0	6	C
1	0	0	4	A
1	1	1	7	C

solutions ($3! = 6$). The best base is [0,2,1], with an associated list of 3 items: $< 1, B| < 3, A| < 7, -1|_{[0,2,1]}$. Note the base is shown as a subscript for clarity.

Table 4 illustrates how the other partial results (the second table) are added to the optimised list of three items.

The last step is the optimisation of the final list. Since there are three alternatives and we have three items, the last list is the best one. Note the differences between the first list and the final list:

Initial list
$< 0, B| < 1, A| < 2, B| < 4, A| < 7, C|_{[0,1,2]}$

Final list
$< 1, B| < 4, A| < 7, C|_{[0,2,1]}$

Table 3
From a table to a *KBM2L*

Case in offset $q \to KBM2L$
$((0,0,0), B)$ in offset $0 \to < 0, B| < 7, -1|$
$((0,1,0), B)$ in offset $2 \to < 0, B| < 1, -1| < 2, B| < 7, -1|$
$((0,1,1), A)$ in offset $3 \to < 0, B| < 1, -1| < 2, B| < 3, A| < 7, -1|$
$((0,0,1), A)$ in offset $1 \to < 0, B| < 1, A| < 2, B| < 3, A| < 7, -1|$

Table 4
Adding the second table to the list

Case in offset $q \to KBM2L$
$((1,0,1), C)$ in offset $5_{[0,1,2]}$ and $6_{[0,2,1]} \to < 1, B| < 3, A| < 5, -1| < 6, C| < 7, -1|_{[0,2,1]}$
$((1,1,0), C)$ in offset $6_{[0,1,2]}$ and $5_{[0,2,1]} \to < 1, B| < 3, A| < 4, -1| < 6, C| < 7, -1|_{[0,2,1]}$
$((1,0,0), A)$ in offset $4_{[0,1,2]}$ and $4_{[0,2,1]} \to < 1, B| < 4, A| < 6, C| < 7, -1|_{[0,2,1]}$
$((1,1,1), C)$ in offset $7_{[0,1,2]}$ and $7_{[0,2,1]} \to < 1, B| < 4, A| < 7, C|_{[0,2,1]}$

2.3. Index parts for explanation purposes

We have shown how the size of a decision table can be compacted using *KBM2L* lists, still including the same knowledge as the original table. Moreover, these lists provide a way of explaining the DSS proposals, as we shall see now. The decision-maker will not only know what to do at each stage of the problem, but also what the implicit rules of the protocol modelled and evaluated by the system are. This certainly makes the final DSS more usable, adding important details of validation, affinity and relevance of the attributes of the original table. Hence, optimising the storage of the decision table and finding explanations are to some extent the same problem.

Let us take an item of the *KBM2L* list. Remember each item comprises adjacent cases of the table with the same policy. The adjacency was relative to the order imposed by the offset, which was associated with a scheme. Now we wonder whether there is also a similarity among the attributes of those adjacent cases. Thus, we look at the other part of the item: their indices. Since the offset does not explicitly show the indices, it is preferable to use indices notation-based *KBM2L*.

Let us define two clearly different parts of the indices an item comprises. The first part is the *fixed* part of the indices: the common components of all the item cases. The fact that values of the respective attributes of these components are the same somehow *explains* why the policy is also the same. Therefore, the set of attributes of the fixed part can be interpreted as the policy explanation.

The second part, which is complementary to the first part, is the *variable* part: the cases do not share the same values and, therefore, the attributes are not relevant in deciding the best action to be taken.

Fortunately, we shall not have to look at the whole set of indices an item includes to find the fixed and variable parts. It is enough to look at the indices $I_{\inf}$ and $I_{\sup}$ associated with its extreme cases. The fixed part is the result of the logical-AND: $I_{\inf} \wedge I_{\sup}$.

Example 2 (*continued*). As an illustration of the parts of an item, let us take the final *KBM2L* with 3 items (Table 4). For item $< 1, B|$, $I_{\inf} = (c_0 = 0, c_2 = 0, c_1 = 0)$ and $I_{\sup} = (c_0 = 0, c_2 = 0, c_1 = 1)$. The item contains 2 cases. Then, (c_0, c_2) is the fixed part and c_1 the variable part of the indices. $(c_0 = 0, c_2 = 0)$ explains to some extent the fact that the policy is B. For item $< 4, A|$, $I_{\inf} = (c_0 = 0, c_2 = 1, c_1 = 0)$ and $I_{\sup} = (c_0 = 1, c_2 = 0, c_1 = 0)$. The item includes 3 cases. The fixed part is empty; the variable part is the whole index. For item $< 7, C|$, $I_{\inf} = (c_0 = 1, c_2 = 0, c_1 = 1)$ and $I_{\sup} = (c_0 = 1, c_2 = 1, c_1 = 1)$. The item contains 3 cases and the fixed part is c_0.

It is worth pointing out some possible cases. The whole index of an item that is a single case is fixed $(I_{\inf} = I_{\sup})$, whereas its variable part is empty. On the contrary, as in the above example, an item may have an empty fixed part. This is likely if the item gathers all or most of the possible cases of a table. In this case, none of the attributes are important in the policy. The description of the item is more complicated for explanation purposes and requires the analysis of the domain of the attributes of greater weight. The rationale behind these situations is that the more cases an item includes, the more difficult it is to find common indices. The extreme situation of having only one item (same policy for every case) would lead to an empty fixed part: every context leads to the same policy, so, we cannot explain such a diversity of cases covering all the attribute domains with a few fixed attributes.

3. Optimising the search for good bases

A *KBM2L* of a decision table is associated with a certain storage space and reorganisation of the information. We would like to have the same information stored but in a minimum space and with maximum organisation. Since the problem semantics must be preserved, i.e. the set of attributes should be unchanged, we are not looking for *any possible* way of storing the information of the table. Our search will be constrained to the possible permutations of attributes. In tables of δ attributes, we must consider $\delta!$ possible solutions (the domains order being fixed). Because the attribute domains are discrete, the domain order might also be permuted, increasing the search space of solutions to $\delta! \prod_{i=0}^{\delta-1} D_i!$. We have not yet implemented this second possibility, and we have $\delta!$ solutions or bases.

Therefore, the problem of finding the *KBM2L* list with the least number of items is one of combinatorial optimisation. The general scheme of searching for a good base will try to improve the current base generating another base according to some strategy. If this new list is more optimal than the old one (i.e. if it requires a shorter list), it is kept and the process iterates to improve it again.

There is still a problem: the improvement of the solutions cannot be verified in polynomial time if the complete lists are compared—an exponential problem at each step that will be tackled in Section 4.

3.1. General ideas

In problems where the set of attributes is very large, it may be necessary to initially set the weight or position of some attributes in the base and to proceed by learning the subproblem. This is equivalent to fixing the role of an attribute, like relevant (high weight) or irrelevant (low weight). This is a reasonable strategy to start with.

The probability of obtaining the optimal base in a random test is $\frac{1}{\delta!} = p_\delta$, without any information about attribute positions. If we know prior to the study that there are v relevant attributes in all items, the above probability is $\frac{v!}{\delta!} = p_{\delta-v}$. Note that $p_{\delta-v} > p_\delta$ if $v > 0$, i.e. if we have some knowledge about the domain.

Let us suppose we are going to move from base B to base B'. At that moment, we know the current best list length and the length related to B. We also know a lower bound of the list length with respect to base B' (see a method in Section 4). If we want a change of base to reduce the size of the *KBM2L* list, we should extract the new base B' from a subset that contains the optimal base and bases that are better than B.

One clearly has to reorganise the table into bases that allow the items to be joined and place equal items adjacently. Fortunately, the process of base generation learns how to generate better lists, with a greater probability of union of contexts. With this information, we will impose constraints on the order we generate to prevent item fragmentation and promote unions.

Example 3. Let us take the following *KBM2L*. The initial base is $B = [0,1,2,3,4]$ with binary attributes. The representation space has 32 cases. Suppose none of the attributes has been set as relevant/irrelevant. The initial list is in offset notation:

$$< 3,x| < 7,y| < 9,x| < 31,y|$$

and in indices notation:

$$< (0,0,0,1,1),x| < (0,0,1,1,1),y|,$$

$$< (0,1,0,0,1),x| < (1,1,1,1,1),y|.$$

Note the two separate items with policy x at offsets 3 and 9. The first item has indices $c_0 = c_1 = c_2 = 0$, which are the fixed part. However, the item at offset 9 has $c_0 = 0,\ c_1 = 1,\ c_2 = 0,\ c_3 = 0$ as the fixed part. As a consequence: (1) attribute 4 is always irrelevant, covering its whole domain and will remain at the same position for the next base; (2) attribute 1 is fixed in both items, though it is 0 in the first one and 1 in the second one, covering its whole domain (irrelevant as a whole). Thus, a movement of attribute 1 to a position of lesser weight is suggested. If we try base $B' = [0,2,1,3,4]$, the new *KBM2L* list is

$$< 5,x| < 31,y|$$

in offset notation. In indices notation:

$$< (0,0,1,0,1),x| < (1,1,1,1,1),y|.$$

It results in the union of both items. Moreover, if various grains of equal policy give rise to one grain, others may join as well (items with policy y in the example).

Now there are only two items. The whole list would be, in offset notation:

$$< 0,x| < 1,x| < 2,x| < 3,x| < 4,x|,$$

$$< 5,x| < 6,y| < 7,y| < 8,y| \cdots < 30,y| < 31,y|.$$

And in indices notation:

$$< (0,0,0,0,0),x| \cdots < (0,0,1,0,1),x|,$$

$$< (0,0,1,1,0),y| \cdots < (1,1,1,1,1),y|.$$

Note the vectors I_{inf} and I_{sup}. The fixed part for the first item consists of the first two indices (c_0, c_2) and the variable part is (c_1, c_3, c_4). For the second item, its fixed part is empty and its variable part is $(c_0, c_1, c_2, c_3, c_4)$.

As another general rule for guiding the search for the optimal base, we have just given the idea of transposing indices that cover their whole domains to positions of lesser weights. The respective attribute is irrelevant for the fragmented policy and the items can be joined.

We generate more information relative to the search space below. We suggest two techniques for minimising the number of items. The first one uses a variable neighbourhood (VN) methodology (Hansen and Mladenović, 2001) and the second one is a genetic algorithm (GA), both adapted to our specific problem.

At each iteration the algorithms maintain only one *KBM2L* and a set of bases. The optimisation process works on the candidate base using the fitness of each base (in the GA) or the proximity between two bases (in the VN approach). Obviously, we do not store either a population of lists or the neighbour lists with all cases on all the bases.

3.2. Variable neighbourhood algorithm

In a first stage, we tried a combination of local and global search. We understood "local" to mean changing a few elements from one base to the following base. Thus, if the base was [0,1,2,3] and we changed only two attributes, we moved through its 2-neighbourhood: {[1,0,2,3], [0,2,1,3], [0,1,3,2], [2,1,0,3], [0,3,2,1], [3,1,2,0]}. This amounts to moving towards all the elements of the Hamming distance H equal to 2.

In general, a 2-neighbourhood contains $\delta(\delta-1)/2$ neighbours, whereas there are $(\delta-1)!$ neighbours at a distance H equal to δ. For the above base ($\delta=4$), they are: {[1,2,3,0], [2,3,0,1], [3,0,1,2], [3,2,1,0], [2,0,3,1], [3,2,1,0]}.

The strategy was to mix a local search by enumeration of bases in 2-neighbourhoods (all the bases) with some extra random jumps into δ-neighbourhoods (far from the provisional solution). We computed statistics (minimum, maximum and average item size per policy, etc.) that described the items of the *KBM2L* list. The use of just the local search was often enough to get good results.

We later discovered that our idea of "locality" or "neighbourhood" should be changed. A key observation was that the effect of changing two attributes is completely different depending on where they are located, although the Hamming distance is 2 anyway.

In order to study this remark in further detail, let us create a graphical representation of the *KBM2L* list that shows how the cases are grouped when the base is changed. This representation shows a spectrum of the storage list: the sets of cases (X-axis) with the same optimal decision policy (same colour). Fig. 1 shows a table in a representation space with 2^8 cases (eight binary attributes). Note how the change of base has a bearing on the union/fragmentation of items. The lists associated with each spectrum have the same information (256 cases), but the optimal list (the last one) requires less memory space to store these cases, see Table 5.

Let B and B' be the initial and the new base, respectively. Let $B \to B'$ denote a generic change of base. Then, we suggest a simple classification of the base changes:

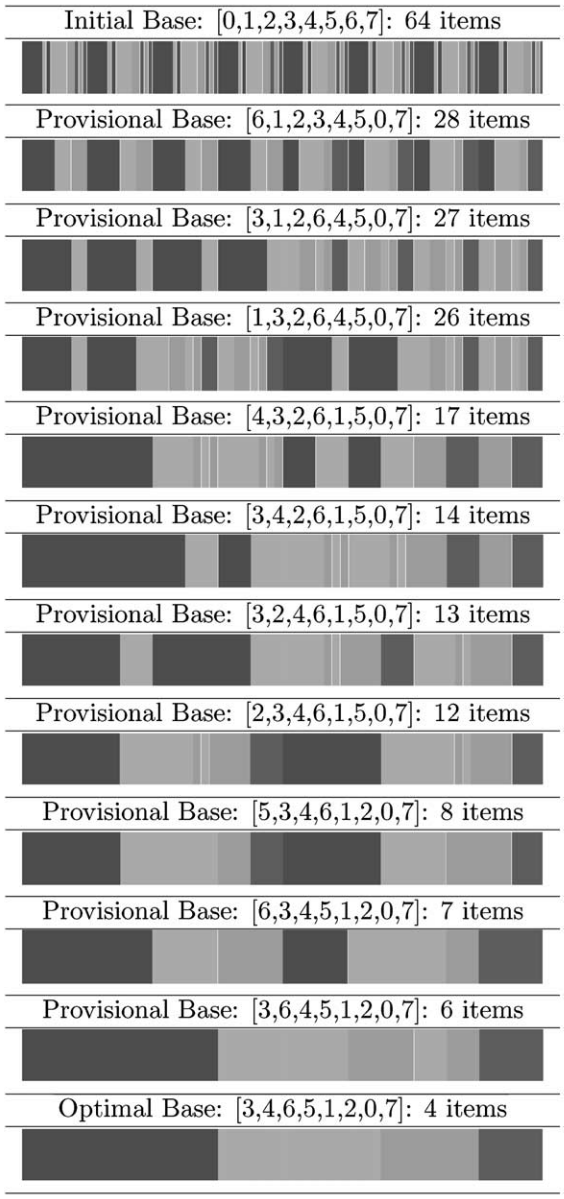


Fig. 1. *KBM2L* spectra.

1. Changes among few attributes, all of them with *high* weight in both bases. The effect on the initial list is its transformation by means of global movements of big blocks of information.
2. Changes among few attributes, all of them with *low* weight in both bases. The effect is a transformation by means of local movements of tiny blocks of information.
3. Changes among few attributes, all of them with *medium* weight in both bases. The initial list is transformed by means of global movements of small blocks.
4. Overlap of cases 1, 2 and 3.

Table 5
Specific information about Fig. 1

Initial base	(lower, upper)	64 Items
[0,1,2,3,4,5,6,7] < ((index), policy)|	Offsets	Cases
<(0,0,0,0,1,0,0,1), 1|	(0, 9)	10
<(0,0,0,0,1,0,1,1), 2|	(10, 11)	2
<(0,0,0,0,1,1,0,1), 1|	(12, 130)	2
<(0,0,0,1,0,1,0,1), 2|	(14, 21)	8
<(0,0,0,1,1,0,0,1), 3|	(22, 25)	4
<(0,0,0,1,1,0,1,1), 4|	(26, 27)	2
...	...	...
<(1,1,1,1,1,1,1,1), 4|	(254, 255)	2
Optimal base	(lower, upper)	4 Items
[3,4,6,5,1,2,0,7] < ((index), policy)|	Offsets	Cases
<(0,1,0,1,1,1,1,1), 1|	(0, 95)	96
<(1,0,1,0,1,1,1,1), 2|	(96, 175)	80
<(1,1,0,1,1,1,1,1), 3|	(176, 223)	48
<(1,1,1,1,1,1,1,1), 4|	(224, 255)	32

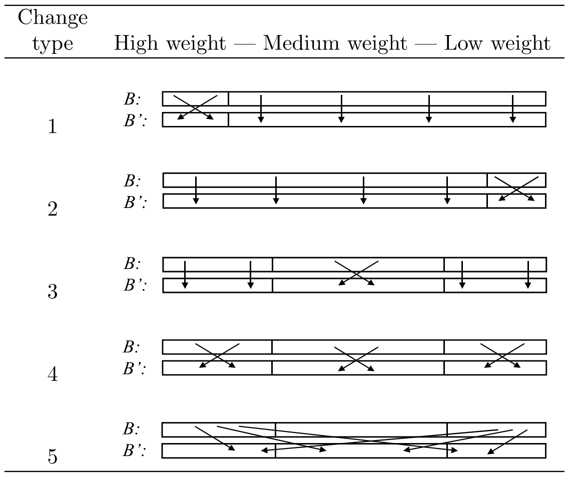


Fig. 2. Classification of base changes.

5. *General* changes of base, where many attributes change their weights of any magnitude, transform the list chaotically.

In addition, any change of base has a set of *cases* that do not move with the transformation, i.e. fixed points. For example, for binary attributes, the cases $000\cdots0$ and $111\cdots1$ have the same offset in all possible bases, though there are many more cases. See more comments on this in Section 4.

Fig. 2 summarises the types of base changes. Obviously, there is no direct relationship between the base change types and the *KBM2L* length. Otherwise, we would be able to choose the best change of base and obtain an optimal *KBM2L* easily, and this is not possible.

This important qualitative description of the different base changes allows us to propose a number of heuristics to guide the search based on an alternative idea of neighbourhood, that is more accurate and closer to our optimisation problem.

We introduce a new proximity measure between bases that takes into account the effect of the base change on the new *KBM2L* list. For a type-1 change of base, the new *KBM2L* list will be similar to the old list, and the measure value should be low. For a type-5 change, the new list will be quite different, and the measure value should be high.

Let G be a function defined as follows. For all bases B, B' such that $B \neq B'$

$$G(B,B') = r_{\text{left}} + \left(\sum_{i=0}^{\delta-1} r_i + H(B,B') - 1 \right)$$

and $G(B,B) = 0$, where $H(B,B')$ is the Hamming distance between B and B', r_i is the number of attributes between the initial position and the final position of each permuted attribute i, and r_{left} is the number of adjacent non-permuted attributes to the left of the permuted attribute with the highest weight.

G is subordinated to the Hamming distance. G counts, via H, how many attributes have been permuted. But H is only a distance between codes, and the role a base plays in the problem is not revealed. Hence, G counts, via r_i's, how far along the transformation the attributes that have been permuted are, taking into account the change in the weight of the attributes. If an attribute does not jump or jumps to an adjacent position, then it does not contribute to r_i $(r_i = 0)$. This is a way of roughly identifying the changes of base typified above.

Moreover, in order to enhance the classification, G captures the idea that changes among a few attributes are different depending on where the attributes are. For example, an exchange of the first and second attributes (type-1 change) will lead to few changes in the *KBM2L*. However, an exchange of the last and penultimate attributes (type-2 change) will lead to many changes in the *KBM2L*. The term r_{left} in G captures this idea: the latter change would yield a longer G-value than the former.

Table 6 illustrates some examples of function G. The first and third cases have short G-values because the weights of the attributes have changed a little bit. They would be type-3 changes of base. The second and fourth examples have longer G-values because the weights of the changed attributes are rather different. They are type-5 changes of base.

Although desirable, it is not easy to define a *distance* in the whole search space that captures the effect of the attribute movements. In fact, G is not a distance since the triangular inequality, $G(B,B') \leqslant G(B,B'') + G(B'',B') \forall B,B',B''$, does not hold.

Counterexample. If B is [0,1,2,3,4,5], B' is [0,1,2,3,5,4] and B'' is [1,0,2,3,4,5], then $G(B,B'') = 1, G(B',B'') = 3$, but $G(B,B') = 4 + 2 - 1 = 5$.

Nevertheless, we use G constrained to a subset of bases as follows (even though it is not a distance). Let $N_{H2}(B_0)$ be the set of bases B such that $H(B,B_0) = 2$ for a given base B_0. For all $B \in N_{H2}(B_0)$, we denote G as

$$G_{H2}(B_0,B) = r_{\text{left}} + \sum_{i=0}^{\delta-1} r_i + 1$$

and $G_{H2}(B_0,B_0) = 0$.

Let us examine the differences between the Hamming distance and function G (and G_{H2}) in more depth. Given a base B_0, its neighbourhood $N_{H2}(B_0)$ is easy to compute. For $B_0 = [0,1,2,3,4]$, $N_{H2}(B_0)$ includes the following 10 bases:

$$\begin{aligned}
&B_0 = [0,1,2,3,4], \\
&B_1 = [1,0,2,3,4] \quad B_6 = [0,3,2,1,4], \\
&B_2 = [2,1,0,3,4] \quad B_7 = [0,4,2,3,1], \\
&B_3 = [3,1,2,0,4] \quad B_8 = [0,1,3,2,4], \\
&B_4 = [4,1,2,3,0] \quad B_9 = [0,1,4,3,2], \\
&B_5 = [0,2,1,3,4] \quad B_{10} = [0,1,2,4,3].
\end{aligned}$$

In Table 7, we compute the G-values for each pair of these 10 bases. The first row corresponds with $G_{H2}(B_0,B), \forall B \in N_{H2}(B_0)$.

Note the different values G_{H2} takes along the first row, from 0 to 7. For 5 attributes, G_{H2} is in the discrete range {0,1,2,3,4,5,6,7}, whereas the H distance is in {0,2,3,4,5}. Fig. 3 depicts this idea graphically.

Note that G-values between bases (the other rows) are also rather different, likewise their H distances, not shown in that table. For example, the set of bases in $N_{H2}(B_0)$ such that $G_{H2}(B_0,B_j) \leqslant 3$ is $\{B_0,B_1,B_2,B_5,B_8\}$. Their quantities $G(B_i,B_j)$ can

Table 6
Examples of H and G

Bases	H	G
$\_\_{}^{(q}\_\_XY\_\_\_\_\_$ $\_\_\_\_\_YX\_\_\_\_\_$	2	$q+1$
$\_\_\_\_\_{}^{(q}\_\_X\_\_\_{}^{(r}\_\_Y\_\_\_\_\_$ $\_\_\_\_\_Y\_\_\_{}^{(r}\_\_X\_\_\_\_\_$	2	$q+2r+1$
$\_\_{}^{(q}\_\_XYZ\_\_\_\_\_$ $\_\_\_\_\_ZXY\_\_\_\_\_$	3	$q+3$
$\_{}^{(q}\_\_X\_\_\_{}^{(r}\_\_Y\_\_\_{}^{(s}\_\_Z\_\_\_$ $\_\_\_\_Z\_\_\_{}^{(r}\_\_X\_\_\_{}^{(s}\_\_Y\_\_\_$ $\_\_\_\_\vert\_\_\_\_\_\_\_\_{}^{(t}\_\_\_\_\_\vert\_\_\_$	3	$q+r+s+t+2$

Table 7
G-values for every pair in $N_{H2}(B_0)$ ($\delta = 5$)

G	B_0	B_1	B_2	B_3	B_4	B_5	B_6	B_7	B_8	B_9	B_{10}
$[01234] = B_0$	0	1	2	6	7	2	7	6	3	5	4
$[10234] = B_1$	1	0	3	5	7	3	5	7	3	5	3
$[21034] = B_2$	2	3	0	5	7	3	7	9	5	7	5
$[31204] = B_3$	6	5	5	0	7	7	5	11	5	9	7
$[41230] = B_4$	7	7	7	7	0	9	11	8	9	7	7
$[02134] = B_5$	2	3	3	7	9	0	4	6	4	6	4
$[03214] = B_6$	7	5	7	5	11	4	0	6	4	8	6
$[04231] = B_7$	6	7	9	11	8	6	6	0	8	6	6
$[01324] = B_8$	3	3	5	5	9	4	4	8	0	5	5
$[01432] = B_9$	5	5	7	9	7	6	8	6	5	0	5
$[01243] = B_{10}$	4	3	5	7	7	4	6	6	5	5	0

be greater than 3, see Fig. 4. The Hamming distance between them is always 3 for every pair (B_i, B_j), $i, j \neq 0$ except for (B_1, B_8), which is 4. This is an example of G-proximity within $N_{H2}(B_0)$.

On the contrary, the set of bases in $N_{H2}(B_0)$ such that $5 < G_{H2}(B_0, B_j) \leqslant 7$, i.e. with high values for G_{H2}, is $\{B_3, B_4, B_6, B_7\}$. Values $G(B_i, B_j)$ are also high, see Fig. 5, as an example of non-G-proximity within $N_{H2}(B_0)$. The Hamming distance between them is always 3 for every pair, except for (B_4, B_6) and (B_3, B_7), which is 4. Note again that H discriminates very little compared to G.

Now, our initial good results by locally searching through neighbourhoods of Hamming distance equal to 2 are explained: the search was in fact through variable G_{H2}-neighbourhoods.

The induced measure G_{H2} in $N_{H2}(B_0)$ is, like G, very rich, since it distinguishes between the base changes. This new proximity measure is the guide to control the search. Assuming we are at a base B_0, we try a movement to a base B in $N_{H2}(B_0)$. If B_0 were judged bad, the next movement would be towards a base farther away on the G_{H2} scale (e.g. $B_0 \to B_4$ in Table 7). When it is suspected that the current base B_0 is near the optimal base, or at least its *KBM2L* size is small, the movements will be towards close bases with respect to G_{H2} (e.g. $B_0 \to B_1$). The new base B would be the centre of the neighbourhood $N_{H2}(B)$, where the next search

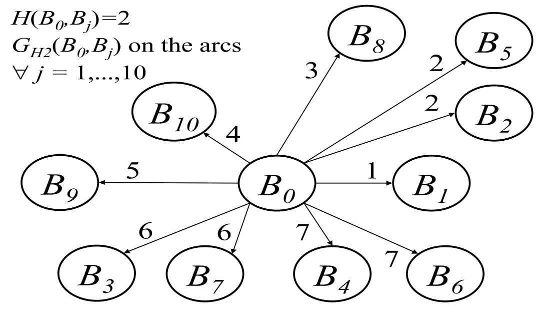


Fig. 3. Values of $G_{H2}(B_0, B_j)$, $j = 1, \ldots, 10$.

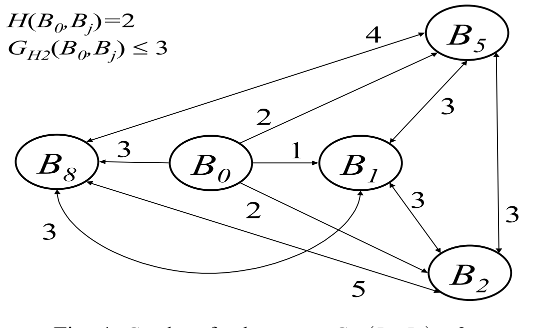


Fig. 4. G-values for bases s.t. $G_{H2}(B_0, B_j) \leqslant 3$.

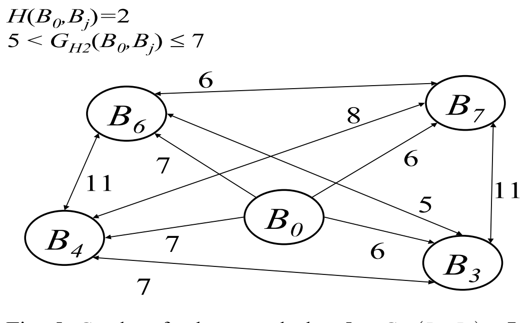


Fig. 5. G-values for bases such that $5 < G_{H2}(B_0, B_j) \leqslant 7$.

Table 8
G-values for every pair in $N_{H2}(B_0)$ ($\delta = 4$)

G	B_0	B_1	B_2	B_3	B_4	B_5	B_6
$[VUXY] = B_0$	0	1	3	5	2	4	3
$[UVXY] = B_1$	1	0	4	4	3	5	3
$[XUVY] = B_2$	3	4	0	5	3	7	5
$[YUXV] = B_3$	5	4	5	0	7	5	5
$[VXUY] = B_4$	2	3	3	7	0	4	4
$[VYXU] = B_5$	4	5	7	5	4	0	4
$[VUYX] = B_6$	3	3	5	5	4	4	0

would start. Fortunately, this strategy covers the whole search space, still following simple permutations.

This scheme bears a resemblance to simulated annealing (Kirkpatrick et al., 1983), which could fit our framework.

Example 4. Let us consider the following example with four binary attributes $\{X, Y, U, V\}$ and a set of 4 possible policies {0,1,2,3}. Let us take $B_0 = [V, U, X, Y]$. The 6 bases in $N_{H2}(B_0)$ are

$$B_1 = [U, V, X, Y] \quad B_2 = [X, U, V, Y],$$
$$B_3 = [Y, U, X, V] \quad B_4 = [V, X, U, Y],$$
$$B_5 = [V, Y, X, U] \quad B_6 = [V, U, Y, X].$$

When moving from B_0 to B_i, the types of base changes are: 1 to B_1, 2 to B_6, 3 to B_4, 5 to B_3, an overlap of 1 and 3 to B_2, and an overlap of 2 and 3 to B_5.

The next table shows the policy vector of all 16 cases according to all the different bases in $N_{H2}(B_0)$. We will see that the optimal bases are not within $N_{H2}(B_0)$.

Base B_0 = [V,U,X, Y] 16 items

0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3

Base B_1 = [U,V,X, Y] 16 items

0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3

Base B_2 = [X, U,V,Y] 16 items

0	1	0	1	0	1	0	1	2	3	2	3	2	3	2	3

Base B_3 = [Y,U,X, V] 8 items

0	0	2	2	0	0	2	2	1	1	3	3	1	1	3	3

Base B_4 = [V,X, U,Y] 16 items

0	1	0	1	2	3	2	3	0	1	0	1	2	3	2	3

Base B_5 = [V,Y,X, U] 8 items

0	0	2	2	1	1	3	3	0	0	2	2	1	1	3	3

Base B_6 = [V,U,Y,X] 16 items

0	2	1	3	0	2	1	3	0	2	1	3	0	2	1	3

With B_1, B_2, B_4 and B_6, the number of items in the *KBM2L* is 16. Therefore, B_0 should be improved by moving towards a base B such that $G_{H2}(B_0, B)$ is large (maybe a type-5 base change). G-values for this set of bases are shown in Table 8. From this table, we choose B_3 as B since G_{H2} is maximum.

Now, with B_3, the *KBM2L* has 8 items and we try another movement. Table 8 does not need to be computed again because although B_3 now becomes B_0, the G-values for $N_{H2}(B_0 = [Y, U, X, V])$ can be obtained by relabelling the attributes. The 6 bases in $N_{H2}(B_0)$ are

$$B_1 = [U, Y, X, V] \quad B_2 = [X, U, Y, V],$$
$$B_3 = [V, U, X, Y] \quad B_4 = [Y, X, U, V],$$
$$B_5 = [Y, V, X, U] \quad B_6 = [Y, U, V, X].$$

The next movement should be towards a base B s.t. $G_{H2}(B_0, B)$ is small, like $B = B_4$ or $B = B_1$. After exploring B_1 and B_4, with $G_{H2}(B_0, B_1) = 1$ and $G_{H2}(B_0, B_4) = 2$, B_4 is better. This is an optimal base, giving rise to 4 items, as shown in the next table.

Base B_0 = [Y,U,X, V] 8 items

0	0	2	2	0	0	2	2	1	1	3	3	1	1	3	3

Base B_1 = [U,Y,X, V] 8 items

0	0	2	2	1	1	3	3	0	0	2	2	1	1	3	3

Base B_2 = [X, U,Y,V] 8 items

0	0	1	1	0	0	1	1	2	2	3	3	2	2	3	3

Base B_3 = [V,U,X, Y] 8 items

0	0	2	2	0	0	2	2	1	1	3	3	1	1	3	3

Base B_4 = [Y,X, U,V] 4 items

0	0	0	0	2	2	2	2	1	1	1	1	3	3	3	3

Base B_5 = [Y,V,X, U] 8 items

0	0	2	2	0	0	2	2	1	1	3	3	1	1	3	3

Base B_6 = [Y,U,V,X] 16 items

0	2	0	2	0	2	0	2	1	3	1	3	1	3	1	3

3.3. Genetic algorithm

The application of genetic algorithms to combinatorial optimisation problems is well documented in several application domains (Mitchell, 1998). In these algorithms, the search space of the problems is represented as a collection of individuals. These individuals codify the solutions with character strings (or arrays of numbers, or matrices,...), which are often referred to as genes. In our problem, an individual will be a base, i.e. an order for the attributes (genes).

The purpose of using a GA is to find the individual in the search space with the best genetic material. The quality of an individual is measured via an evaluation function or fitness, related in our case to the storage space required by the *KBM2L* associated with each base. Since computing the *KBM2L* size for each individual is extremely costly, the fitness will be a lower bound for this size (see "Test of a new base" in Section 4). At each iteration, we will only need to store bases (individuals) together with their fitness, but not the corresponding full lists.

Once an initial population has been chosen, the quality of the individuals is determined and some of them are selected to produce new individuals, which will be added to the population. For all newly created individuals, there exists a probability (near to zero) of mutation, changing their genes. After that, some individuals are removed from the population. One iteration of this process is called a generation.

The operators that define the behaviour of the GA are: selection, crossover and mutation. All of them must be selected to improve the average quality of the population, favouring the maintenance of good genetic material (i.e. good orders for the attributes of the base).

Selection criterion. The selection criterion used to mate and remove individuals tries to avoid strong selective pressure, maintaining population diversity as a way of permitting movement all over the search space.

The selection is carried out according to the following probability (a non-linear function) of being selected

$$prob(rank) = q(1-q)^{rank-1},$$

where *rank* identifies each individual, i.e. it is the order of the individual in the population according to its quality (see the next paragraph) and q is a fixed number in (0,1).

Selection for mating needs a previous arrangement of the individuals from best to worst. So, for the best individual $rank = 1$. This way of selection prevents the best individuals from monopolising the evolution process, while giving them preference.

When this function is used to discard individuals from the population, the opposite arrangement, from worst to best, is required.

Crossover. We have implemented two operators. The one-point crossover is a general operator such that a cut point is selected to divide the individuals to be mated into two parts: f_1 and f_2 for father and m_1 and m_2 for mother. Offsprings are formed by exchanging these parts: $f_1 - m_2$ and $m_1 - f_2$. This operator needs a repair mechanism, because some attributes may be repeated in offspring and this is not possible in a suitable base.

The voting crossover is better suited for our specific problem. This operator is based on the role of each attribute in the individuals. The main idea is to generate a ranking of attributes. Given two bases B_f (father) and B_m (mother), the score proposed by both bases for each attribute a_i at the new base is

$$\frac{j}{L_f} + \frac{k}{L_m} = \frac{j \cdot L_m + k \cdot L_f}{L_f \cdot L_m}, \tag{2}$$

where j is the position of a_i in B_f and k in B_m. Values L_f and L_m are the *KBM2L* lengths (or at least their switch estimations, see Section 4) associated with each base (i.e. the fitness). The smaller the score, the further to the left the attribute is in the base. Let us illustrate the details with an example.

Example 5. Let B_f be [2,1,0,3,5,4] with $L_f = 30$ and B_m be [3,2,1,4,5,0] with $L_m = 45$. For crossover, bases vote to propose a new base B_c. The attributes have the following scores arranged by means of (2):

$$a_0 : \frac{2 \cdot 45 + 5 \cdot 30}{45 \cdot 30} = 0.177,$$

$$a_1 : \frac{1 \cdot 45 + 2 \cdot 30}{45 \cdot 30} = 0.077,$$

$$a_2 : \frac{0 \cdot 45 + 1 \cdot 30}{45 \cdot 30} = 0.022,$$

$$a_3 : \frac{3 \cdot 45 + 0 \cdot 30}{45 \cdot 30} = 0.100,$$

$$a_4 : \frac{5 \cdot 45 + 3 \cdot 30}{45 \cdot 30} = 0.233,$$

$$a_5 : \frac{4 \cdot 45 + 4 \cdot 30}{45 \cdot 30} = 0.222.$$

The minimum is the score of a_2, followed by $a_1 \ldots$, and, finally, the maximum is a_4. So, the new base is $B_c = [2, 1, 3, 0, 5, 4]$. Ties, if any, are resolved at random.

This operator only produces one new child from each mating of father and mother.

Mutation. The mutation operator selects at random two attributes that change their positions in the base.

The GA stops if after a fixed (big) number of generations, the fitness does not improve.

3.4. Experiments

In this section we carry out some experiments to test the algorithms presented above. Our interest will be the CPU time and the solution trace (list size at each iteration) for each *KBM2L* using: (1) the VN algorithm with the H2 metric (VNA-H2); (2) the VN algorithm with the G_{H2} proximity measure (VNA-G_{H2}); (3) the GA with the voting crossover; and (4) a hybrid algorithm based on the VN algorithm with G_{H2} and the GA (GA + VNA-G_{H2}). This fourth algorithm runs a GA until a good solution is judged, launching then a local search, much in the VNA-G_{H2} spirit.

We use a 1-GHz PentiumIII PC, with 512 MB, Java 2 and Windows XP. We consider 10, 90 and 300 attributes A, and domains with cardinal D equal to 2, 4 and 8, giving rise to nine different *KBM2L*s, $k1, k2, \ldots, k9$, with a number of cases as follows:

	$k1$	$k2$	$k3$
$A = 10$	2^{10}	4^{10}	8^{10}
	$k4$	$k5$	$k6$
$A = 90$	2^{90}	4^{90}	8^{90}
	$k7$	$k8$	$k9$
$A = 300$	2^{300}	4^{300}	8^{300}

The list content is generated at random from the offset space. The number of known cases for each table is

	$A = 10$	$A = 90$	$A = 300$
$D = 2$	355	243	259
$D = 4$	267	258	270
$D = 8$	222	258	246

The rest of the table is unknown. For all the lists we knew the optimal base. The number of items for each optimal table was 41. We then changed the base at random and tried to find the optimal. After this, the number of items for each table in the initial iteration is

	$A = 10$	$A = 90$	$A = 300$
$D = 2$	574	486	518
$D = 4$	534	516	540
$D = 8$	444	516	492

Next we show the results. Fig. 6 shows a comparative study of the four algorithms in terms of the (log) CPU time used, measured in milliseconds.

As far as the CPU time is concerned, in general, our experiments show that the GA behaves better than the VN algorithms whenever there are a lot of attributes. All the lists except $k1$, $k4$ and $k9$ support this finding, i.e. the VNA-G_{H2} is better for $k2$, $k3$, $k5$, $k6$ and the GA is better for $k7$, $k8$. The rationale behind this may be as follows. If the table dimension increases, the number of members within the $H2$-neighbourhood where the VN search is performed increases (it has $\delta(\delta - 1)/2$

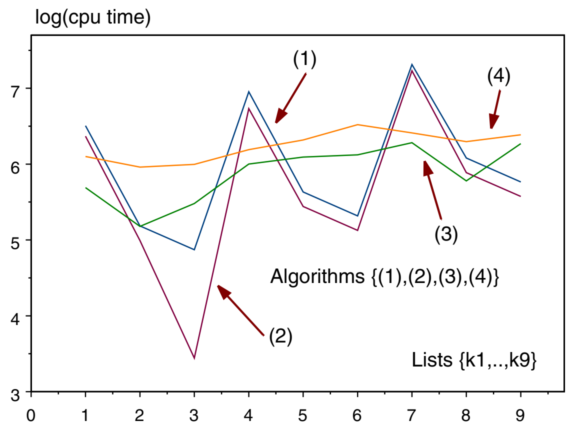


Fig. 6. Comparative results: (1) VNA-$H2$, (2) VNA-G_{H2}, (3) GA and (4) GA + VNA-G_{H2}.

members for δ dimensions). However, the GA population size is often chosen using heuristics with a linear behaviour rather than quadratic, which is less expensive. Thus, based on empirical evidence, Alander (1992) suggests a size between δ and 2δ. Indeed, we have used δ.

Fig. 7 shows the evolution of the objective function (minimum list size) for the nine lists as the algorithms progress. We only show the results up to the first 10 iterations, for comparison purposes. Note that for the genetic approaches, the fitness is measured in switches (left-hand-side of the vertical line), rather than in items, as explained above. The right-hand-side of the line shows the true list sizes, in items.

All the evolution patterns are exponentially decreasing as a consequence of the fast information reorganisation when the relevant (irrelevant) attributes put on (lose) weight. The $k1$ example shows a slow convergence under VN approaches, a little bit faster under the VNA-G_{H2}. This example might be the typical case that shows the heuristic weakness.

The VNA-G_{H2} globally stands out because of its refined G measure for comparing bases. We must take into account that the iterations cost differently for the GA and the VNA. It is more costly for the former: it initially consumes many resources because of working with a whole population, which includes bad bases (see the high figures in the Y-axis for the genetic methods). Also, it does not compute the true list size but a lower bound. Perhaps the GA would be even better if the operators, specially the crossover, were improved to reveal the features of our problem.

We think the hybrid strategy is not worth it. It firstly reproduces the GA pattern and then, a slow convergence when the VNA is applied.

4. Copying information from one base to another

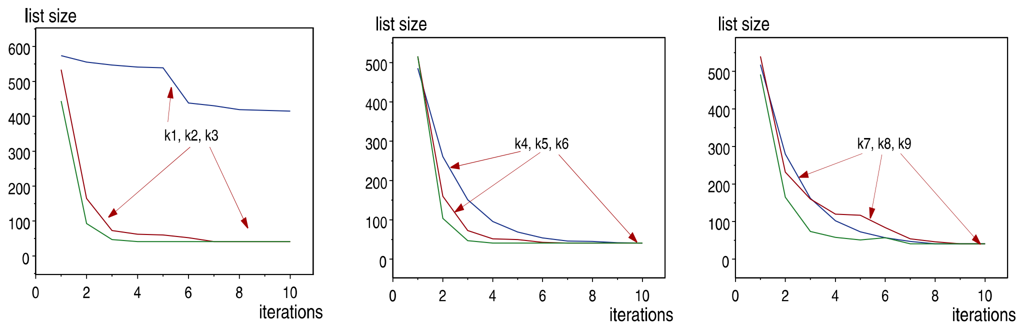
The last section dealt with the combinatory of the search for the optimal base. This section takes into account the combinatory of storage.

The size of real decision tables is a complex problem for coordinate transformation. The change of base is not a trivial problem. All the elements of the table must be copied into the new base according to the new order. The complexity of this copying is $\prod_{i=0}^{\delta-1} D_i$. Taking advantage of data sparseness in the *KBM2L* structure due to storage as items, we propose a number of heuristics to implement an efficient copy procedure.

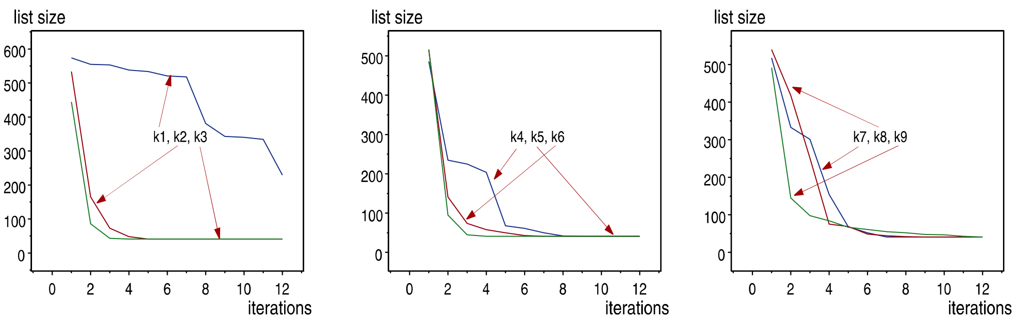
As explained above (see Section 2.2), the method for constructing the *KBM2L* list consists of reading the table inputs with the original initial order $[0, 1, \ldots, \delta - 1]$ and writing these inputs into a structure without information: $< w_0D_0 - 1, -1|$ (in offset notation).

In general, we will face decision tables with unknown data: (1) during the process of constructing the *KBM2L* while the different partial tables are added to the list (see, e.g., Example 2); (2) if the table is very large and it is not possible to solve the whole decision-making problem by solving all the subproblems (instantiating some attributes).

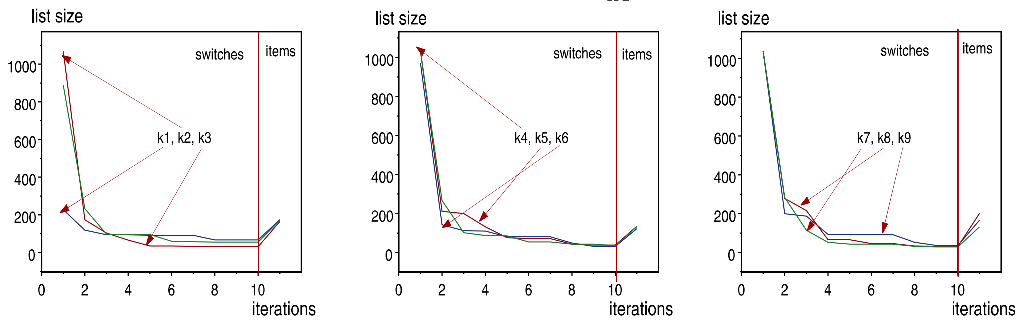
When we change the base, the information, other than -1, is then copied to the new structure, case by case, in a standard way. Therefore, the information copy procedure between *KBM2L* structures in different bases may require up to w_0D_0 elements or cases to be copied. However, we provide some heuristics for discarding many bases, thereby decreasing the search space, avoiding copying all the information, and speeding up



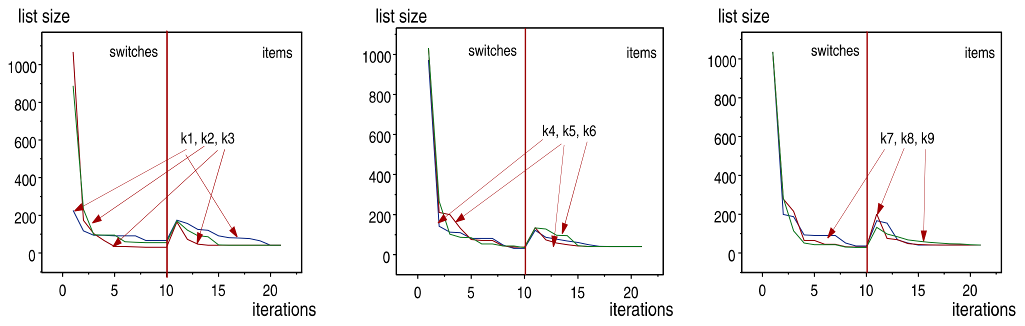
Evolution: VNA-H2



Evolution: VNA-G_{H2}



Evolution: GA



Evolution: GA+VNA-G_{H2}

Fig. 7. Evolution of the four algorithms.

improvement checking with respect to the old base.

Initialisation. When trying to obtain computational savings, we find that the contents are copied only if they *have* information, i.e. it is not necessary to copy the inputs marked as −1 (unknown). Thus, the new information would be initialised with −1. Accordingly, in cases without unknown information, some contents do not have to be copied. For example, if the policy has three outcomes, the new list can be initialised with the first outcome. Thus, this outcome can be ignored in the copy, and only the other two outcomes have to be copied.

Test of a new base. It would be desirable if we could only take a sample of the copy to decide whether the base can be rejected. This means that we would not have to look at the whole copy, which is not an easy task for such massive tables. Remember that extreme indices I_{inf} and I_{sup} determine the length and position of the items. They will be important in this improvement test.

Specifically, our proposal is to copy only every I_{sup} and count the policy *Switches* in the stored sequence of items. We scan the *KBM2L* and record one *Switch* if we see $\cdots < p, A| < r, -1| < q, B| \cdots$ or $\cdots < p, A| < q, B| \cdots$, where policy A is different from B. We will reject the bases with more *Switches* than those of the hitherto minimum *KBM2L*. Note that the complexity of performing this base test depends on the size of the list, which is much smaller than the table size. The experimental results show that this test is very efficient: low computational cost, high rate of base rejection and the optimal base is never rejected.

Fast copy. If the bases have attributes in common at positions of less weight, the items have to be copied out following the storage of the source *KBM2L* list. This base change enables development of an ultra high-speed copy procedure. The initialisation of the target list is a clone of the source list, instead of the empty list, and the −1 items *have* to be copied. Copying stops whenever the new list fragments the grains or contexts and the new *KBM2L* list is bigger. The procedure that follows fits for base changes of type-1 and 3, and when there are not too many unknown cases.

Example 6. Table 9 shows a base change $[A, B, C, D] \to [B, A, C, D]$ where the attributes are binary. Note that this is a type-1 base change. The base change causes a grid within the table, with a resolution of $|A| \times |B| = 4$ (see the horizontal lines).

Once we have cloned the source list, we begin copying. Remember that it is necessary to follow

Table 9
KBM2L fast copy procedure

[*ABCD*] Initial	Offset [*ABCD*]	Policy		[*BACD*] New	Offset [*ABCD*]	Policy
0000	0	d_0	*	0000	0	d_0
0001	1	d_0		0001	1	d_0
0010	2	d_0		0010	2	d_0
0011	3	d_0		0011	3	d_0
0100	4	d_0		0100	8	d_0
0101	5	d_1		0101	9	d_0
0110	6	d_1		0110	10	d_0
0111	7	d_2		0111	11	d_0
1000	8	d_0		1000	4	d_0
1001	9	d_0		1001	5	d_1
1010	10	d_0		1010	6	d_1
1011	11	d_0		1011	7	d_2
1100	12	d_2	*	1100	12	d_2
1101	13	d_2		1101	13	d_2
1110	14	d_2		1110	14	d_2
1111	15	d_2		1111	15	d_2

the steps: (1) read a case; (2) permute the attribute values according to the new base or compute the new offset; (3) compare its policy to the one it is assigned (after cloning); (4) copy if the two are different; and (5) apply the rules of item management to organise the table as a *KBM2L*.

The idea here is to only copy the knowledge that has been moved with respect to the source table and not to touch the cases with fixed offset. If the attribute values involved in the permutation (A and B) are the same sequence of values before and after the permutation, then the offset value does not change given that all the attributes have the same domain cardinalities. In our example: $AB = 00$ and $AB = 11$. The $*$ symbol in the table means that we are at the beginning of one fixed segment of the grid. So, we have avoided copying (and the other tasks numbered above) 8 cases so far.

Moreover, note that the block of cases from offset 4 to 7 in the source table moves to the position of the block of cases from offset 8 to 11 in the target table, and vice versa. Then, when a block is detected, a copy en bloc is launched. It means that the four policies of a block are directly copied, in the original order, avoiding the task of changing the base (step (2) above). Rows 6, 7, 8, 10, 11, 12 in the second table are the only cases we have to update. So, the standard copy copies 16 cases, while the fast copy only needs to copy 6 cases. The savings in a big schema are very sizeable.

Partial copy. A partial information copy procedure follows. It implements a learning procedure between lists. First, the number of cases associated with each policy is counted in the source list. Second, the target list is initialised with −2 if the original list has unknown cases. Note that here −1 is information to be copied, as if it were another policy outcome. Otherwise, the list is initialised in a standard way with −1.

Third, a sample from the whole list is copied to the new base, thereby avoiding copying so many cases with respect to the new base. Fourth, an inference of the whole new list is carried out as follows. The gaps between two equal policies (like $-1, -1$ or A, A) are filled in with the policy in question. This is still the same even if there are labels −2 between them (i.e. $\ldots A, -2, A, \ldots$). The gaps between two different policies (like, e.g., A, B or $A, -1$) require more information corresponding to these gaps (other sample or item) to be copied to decide how they should be filled in. At the end, all the −2 items disappear. Finally, a consistency test is performed to check whether the policy counts coincide for both the original and the inferred lists. They should coincide since the copy neither creates nor destroys knowledge.

This procedure seems to behave well, and we use it whenever a base is to be assessed. Once again, we try to avoid copying the full table by copying a sample and using a procedure to infer the rest of the table. Sometimes, the sample has to be augmented because there is not enough information for the inference. The complexity depends on the list, i.e. its length, rather than on the full copy, i.e. the table size.

All these ideas are combined with the algorithms described in Section 3, see Appendix A with their pseudo-codes, to yield a procedure that finds reasonable solutions in a reasonable time.

The following example is fairly realistic in the sense that it has many attributes and more steps are required to achieve the optimum.

Example 7. In this case we have 11 binary attributes, denoted with numbers from 0 to 10. The set of alternatives is {0,1,2}. There are 256 unknown cases out of a total of 2048 cases (i.e. 1792 known cases). Figs. 8 and 9 show all the steps for reaching the optimal base B_{20}.

All the movements are towards bases in $N_{H2}(B)$, where B is the current base at each step. In this example, $1 \leqslant G \leqslant 35$, and $1 \leqslant G_{H2} \leqslant 19$. Since there are too many items at the first three steps, our software only represents the change of item along the X-axis (not the number of cases within each item). From B_4 onwards, the spectrum is as usual. The unknown cases are represented in white.

Note that G_{H2} is high at the beginning, when the current base is far away from the optimal base. $B_5 \rightarrow B_6$ shows that a high G-value ($= 18$) produces a large improvement ($\sim$50%) on the list. $B_{11} \rightarrow B_{12}$ and $B_{14} \rightarrow B_{15}$ show that a low G-value

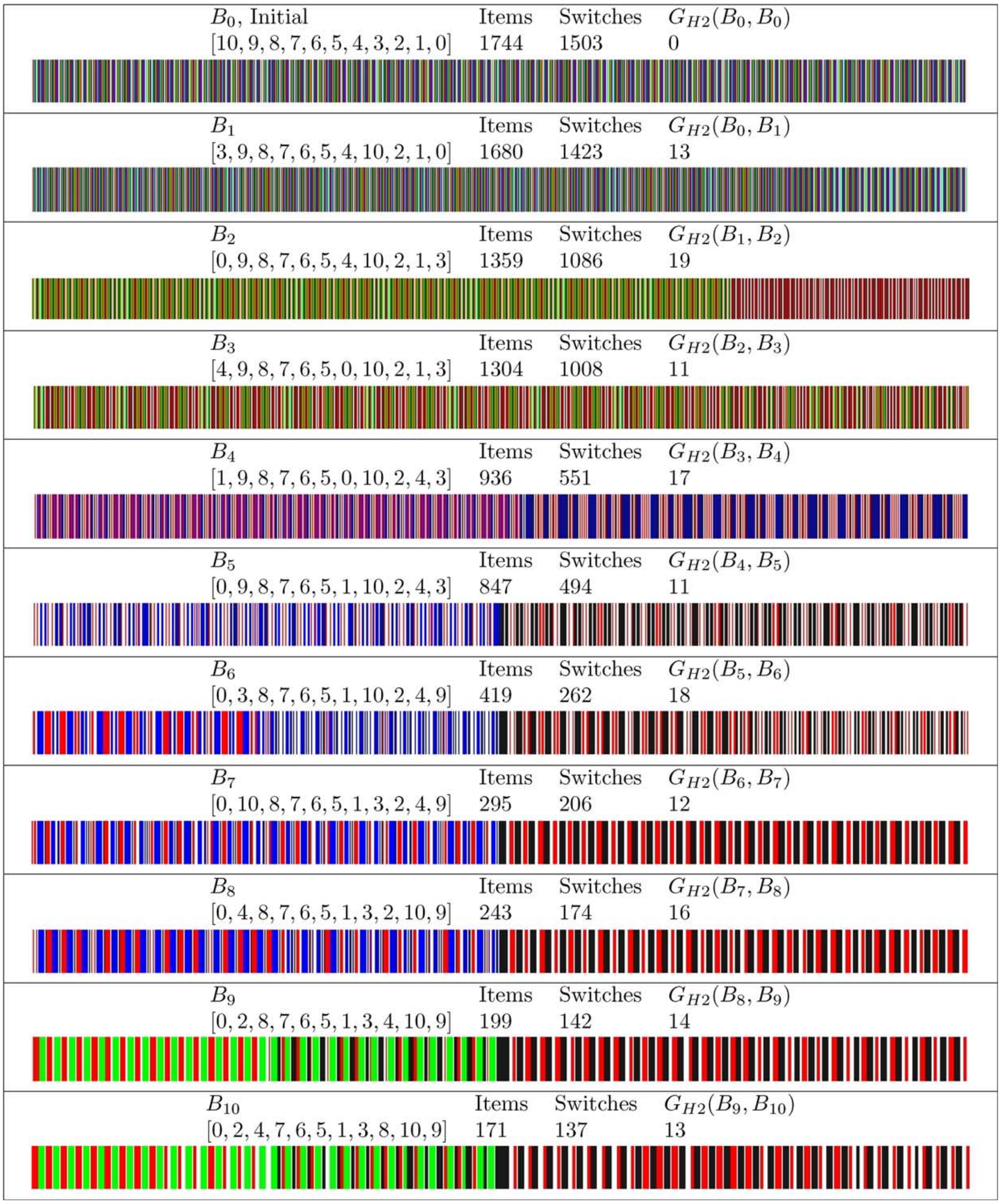


Fig. 8. *KBM2L* spectra for Example 7 (I).

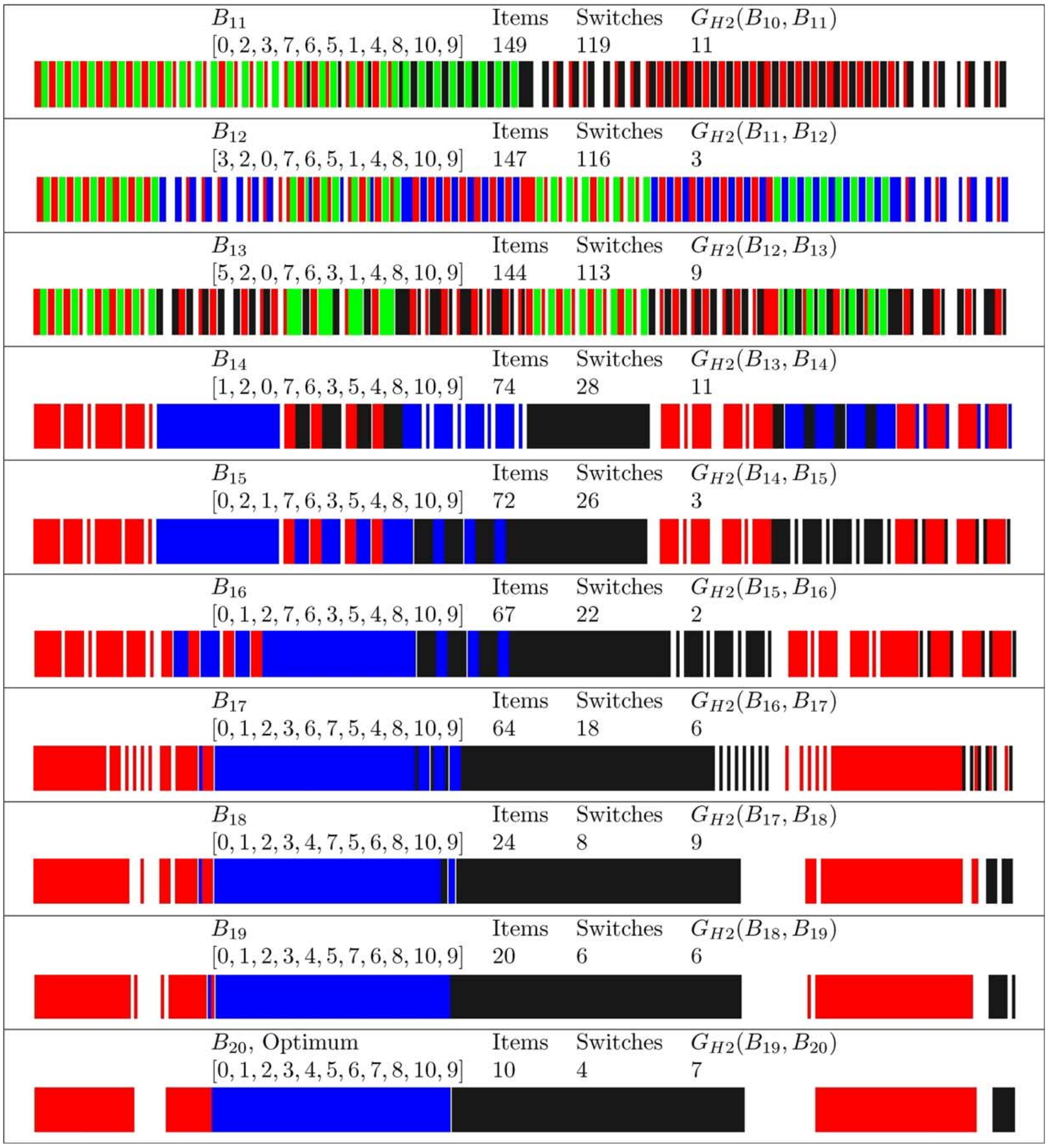


Fig. 9. *KBM2L* spectra for Example 7 (II).

(= 3) yields a small improvement (∼1%). Also, note the base change $B_0 \to B_1$ with $G_{H2}(B_0, B_1) = 13$ and $B_1 \to B_2$ with $G_{H2}(B_1, B_2) = 19$. Some improvement has been achieved (from 1680 items to 1359), but the value $G_{H2}(B_0, B_2) = 12$ shows that B_2 is closer than B_1 to B_0. This suggests an explanation about the low magnitude of the improvement.

As the algorithm progresses, G_{H2} becomes lower. $B_{19} \to B_{20}$ shows that a medium G-value $(=7)$ produces a large improvement (∼50%).

Also, note how the number of switches decreases. It implies that the search becomes more efficient, since it is easier to discard bases. With big *KBM2L* lists, the algorithm may avoid testing many probably bad bases, at the expense of only computing G-values, which is very easy.

The optimal base is B_{20}, with 10 items.

5. Discussion

The knowledge relative to the proposals of a DSS is included in its decision tables. They are accessed and managed as multidimensional matrices. Multidimensional matrices are data structures memorised sequentially by computer systems as lists. The items, being adjacent and identical proposals, mean that the length of the list can be shortened while the same knowledge is retained. Moreover, a good list is able not only to reduce the list storage space but also to extract the attributes that determine each proposal in the contexts. This dual aim strengthens our proposal of *KBM2L* lists to synthesise the knowledge via reorganisation and generalisation of the table information.

From a computational viewpoint, our approach tackles a number of problems that arise when searching for the optimal table organisation. We guide the combinatorial search by means of a genetic algorithm, and a variable neighbourhood strategy with respect to a Hamming distance proxy, introduced to capture the different types of table organisations. We also propose learning heuristics to allow for reorganising complex decision tables: the so-called fast and partial information copying. Finally, to reduce the search space by discarding candidates, we provide a procedure to infer whether a given table organisation is worse than the current one.

From a final user viewpoint, *KBM2L* operation requires the construction of graphical representations (spectra geometrically and symbolically describing the knowledge), query mechanisms that make the dialog with the DSS easier, and explanations of the results to validate the knowledge. Our framework copes with all these issues.

The results of our tests appear to be excellent. When applied to a real medical problem for neonatal jaundice management, a sizeable reduction in the memory space (several orders of magnitude) was achieved, together with explicit and synthesised decision-making rules (Fernández del Pozo et al., 2001).

In this medical problem, the model includes the processes of admission, treatment and discharge of a patient, all organised in an influence diagram with 5 out of 61 nodes being decision nodes. For example, the second decision variable depends on 21 variables, which is tantamount to a table with 4,013,162,496 entries. The fifth decision table is even worse: it considers up to 5,309,410,000,000 different combinations! We had to partially evaluate the whole diagram by instantiating the evidence on 17 nodes, which amounts to having 21,233,664 instances. But only 1000 instances, the ones in which doctors were more interested in, were instantiated. The result was five final decision tables of a size of 18 MB. These results were later composed incrementally in a *KBM2L* structure, as explained above. Obviously, unknown items were found in the *KBM2L*, because we did not evaluate the whole problem.

The *KBM2L* for e.g. the second decision initially had 260 grains of knowledge covering the whole set of 189,000 evaluated cases. After around 300 base changes, we got a representation of the knowledge with only 16 items, which can be read as 16 rules indicating the optimal policy as a function of the key attributes, the fixed part of the items. See Fernández del Pozo et al. (2001) for details.

The use of our methodology for this kind of problems is completely justified because of its innumerable computational difficulties (Bielza et al., 2000). Indeed, in the recent literature on influence diagrams, there is a growing interest in determining the variables that are relevant for each decision, since the standard algorithms for solving influence diagrams tend to construct policy functions with too large domains (see Vomvelová and Jensen, 2002). In contrast to our approach, which is carried out after the diagram has been solved, their concern is to determine these variables beforehand, by analysing the structure of the diagram (see also Fagiuoli and Zaffalon, 1998; Nielsen and Jensen, 1999; Lauritzen and Nilsson, 2001, among others.)

Applications are not confined to decision tables. Our approach is also applicable to data coming from databases, mathematical programs, data acquisition systems, Internet, etc. Likewise in the decision tables, the knowledge in any domain can be represented through relationships among the relevant attribute values and the propositions of interest (decision alternatives, uncertainty measures, clauses, system states...). For example, the synthesis of conditional probability tables (e.g., posterior distributions) would serve for performing diagnosis in medical decision-making problems. Some work has been put into ways of capturing regularities, that is, independencies held only in certain contexts of these probability tables, the so-called *context-specific independence* (Boutilier et al., 1996).

Other interesting ongoing research is to consider permutations of orders in the attribute (discrete) domains, at the expense of enlarging the search space a lot. We might also implement a practical method for performing sensitivity analysis by considering some parameters (probabilities, utilities, dependencies, etc.) as attributes. The resultant *KBM2L* would reveal whether or not each parameter, depending on its weight, keeps the response constant when it is varied. A high weight would show high sensitivity. Another research issue is to deal with querying, including unspecified attribute values (Fernández del Pozo and Bielza, 2002).

Mannila (2000) points out the lack of a data mining theory. Bearing in mind the commonality among data mining and our context, as regards being able to model typical tasks like clustering, classification, rule discovery, data compression, comprehensibility of the discovered knowledge, we hope our approach becomes a promising seed to fill this gap.

Acknowledgements

Research supported by Autonomous Community of Madrid, Project CAM 07T/0027/2000, and by Ministry of Science and Technology, Project DPI2001-3731. The authors would like to thank two anonymous referees for their valuable comments and suggestions.

Appendix A

Here we show the pseudo-code for the VN algorithm based on G_{H2}.

```
set small, set large, set SMALL, set LARGE
working list $\rightarrow$ WL
WL.length $\rightarrow$ Items, WL.Base $\rightarrow B_0$
  Repeat /*iteration*/
For all B s.t. HammingDistance($B_0, B$) $= 2$
(Att$_i$ and Att$_j$ are exchanged), do
0. if (Items is LARGE) $\wedge$ ($G_{H2}(B_0, B)$ is small) $\vee$ (Items is SMALL) $\wedge$ ($G_{H2}(B_0, B)$ is large)
     next(B)
1. Else, if (Att$_i$ is relevant $\vee$ Att$_j$ is irrelevant)
     next(B)
2. NewBaseTest($B$, Items) $\rightarrow$ Switches
3. If (Switches > Items) next($B$)
4. Else, B $\rightarrow$ newL.Base /* Target list */
5. Copy(WL, newL) /* Base change */
6. if (newL.length < Items)
    newL $\rightarrow$ WL and WL.Base $\rightarrow B_0$
```

Step 0 explores bases far from (close to) the current solution when we are far from (close to) the best current solution. This step is specific of the G_{H2} heuristic and if it is not used, the search is performed using $H2$ (VNA-$H2$), which does not take into account the attribute weights. The instruction "next(B)" means the rejection of this base. At Step 1, the relevance of the attributes is exploited in terms of response explanation ability (see Section 3.1). One relevant attribute should not lose much weight and one irrelevant attribute should not put on much weight. Step 2 computes a lower bound, Switches, associated with base B (see Section 4, "Test of a new base"). At Step 3, if the new list has more switches as a consequence of a partial copy of cases, this new base is rejected. Otherwise, Step 4 defines a target list with the new base. Step 5 performs the information copy or base change. At Step 6, the search continues within the new neighbourhood centred on the new base. The algorithm stops when all neighbours are worse.

The GA pseudo-code is a standard one, since its main operators (selection, crossover, mutation) have already been specified within the text (see Section 3.3).

```
Generate initial population
Compute the objective function for each individual
Output the best individual
While stopping rule does not hold
  Select parents from the population
  Produce children from selected parents
  Mutate the individuals
  Extend the population with the children
  Reduce the extended population to its initial size
Output the best individual found
```

References

Alander, J.T., 1992. On optimal population size of genetic algorithms. In: Dewilde, P., Vandewalle, J. (Eds.), CompEuro 1992 Proceedings, Computer Systems and Software Engineering, 6th Annual European Computer Conference. IEEE Computer Society Press, pp. 65–70.

Bielza, C., Gómez, M., Ríos-Insua, S., Fernández del Pozo, J.A., 2000. Structural, elicitation and computational issues faced when solving complex decision making problems with influence diagrams. Computers & Operations Research 27 (7–8), 725–740.

Boutilier, C., Friedman, N., Goldszmidt, M., Koller, D., 1996. Context-specific independence in Bayesian networks. In: Horvitz, E., Jensen, F. (Eds.), Uncertainty in Artificial Intelligence: Proceedings of the 12th Conference. Morgan Kaufmann, San Francisco, CA, pp. 115–123.

Breiman, L., Friedman, J.H., Olshen, R.A., Stone, C.J., 1993. Classification and Regression Trees. Chapman & Hall, New York.

Duda, R.O., Hart, P.E., Stork, D.G., 2001. Pattern Classification, 2nd ed. Wiley, New York.

Ezawa, K., 1998. Evidence propagation and value of evidence on influence diagrams. Operations Research 46 (1), 73–83.

Fagiuoli, E., Zaffalon, M., 1998. A note about redundancy in influence diagrams. International Journal of Approximate Reasoning 19 (3–4), 351–365.

Fayyad, U.M., Piatetsky-Shapiro, G., Smyth, P., 1996. From data mining to knowledge discovery: An overview. In: Fayyad, U.M., Piatetsky-Shapiro, G., Smyth, P., Uthurusami, R. (Eds.), Advances in Knowledge Discovery and Data Mining. AAAI Press, pp. 1–34.

Fernández del Pozo, J.A., Bielza, C., 2002. An interactive framework for open queries in decision support systems. In: Garijo, F.J., Riquelme, J.C., Toro, M. (Eds.), Advances in Artificial Intelligence. In: Lecture Notes in Artificial Intelligence, vol. 2527. Springer, Berlin, pp. 254–264.

Fernández del Pozo, J.A., Bielza, C., Gómez, M., 2001. Knowledge organisation in a neonatal jaundice decision support system. In: Crespo, J., Maojo, V., Martín, F. (Eds.), Medical Data Analysis. In: Lecture Notes in Computer Science, vol. 2199. Springer, Berlin, pp. 88–94.

Hand, D., Mannila, M., Smyth, P., 2001. Principles of Data Mining. The MIT Press, Cambridge, Mass.

Hansen, P., Mladenović, N., 2001. Variable neighbourhood search: Principles and applications. European Journal of Operational Research 130, 449–467.

Hastie, T., Tibshirani, R., Friedman, J., 2001. The Elements of Statistical Learning. Springer, New York.

Henrion, M., Breese, J.S., Horvitz, E.J., 1991. Decision analysis and expert systems. Artificial Intelligence Magazine 12 (4), 64–91.

Keeney, R.L., Raiffa, H., 1993. Decisions with Multiple Objectives: Preferences and Value Tradeoffs, 2nd ed. Cambridge University Press, Cambridge, GB.

Kirkpatrick, S., Gelett, C.D., Vecchi, M.P., 1983. Optimization by simulated annealing. Science 220, 621–630.

Knuth, D.E., 1968. In: The Art of Computer Programming: Fundamental Algorithms, vol. 1. Addison-Wesley, Reading, MA.

Lauritzen, S., Nilsson, D., 2001. Representing and solving decision problems with limited information. Management Science 47 (9), 1235–1251.

Mannila, H., 2000. Theoretical frameworks for data mining. SIGKDD Explorations 1 (2), 30–32.

Mitchell, M., 1998. An Introduction to Genetic Algorithms. MIT Press, Cambridge, MA.

Nielsen, T.D., Jensen, F.V., 1999. Welldefined decision scenarios. In: Laskey, K.B., Prade, H. (Eds.), Uncertainty in Artificial Intelligence: Proceedings of the 15th Conference. Morgan Kaufmann, San Francisco, CA, pp. 502–511.

Pawlak, Z., 1991. Rough Sets. Theoretical Aspects of Reasoning about Data. Kluwer, Dordrecht.

Pawlak, Z., 1997. Rough set approach to knowledge-based decision support. European Journal of Operational Research 99, 48–57.

Quinlan, J.R., 1986. Induction of decision trees. Machine Learning 1, 81–106.

Quinlan, J.R., 1993. C4.5: Programs for Machine Learning. Morgan Kaufmann, San Francisco, CA.

Raiffa, H., 1968. Decision Analysis. Addison-Wesley, Reading, MA.

Shachter, R.D., 1986. Evaluating influence diagrams. Operations Research 34 (6), 871–882.

Vomvelová, M., Jensen, F.V., 2002. An extension of lazy evaluation for influence diagrams avoiding redundant variables in the potentials. In: Gámez, J.A., Salmerón, A. (Eds.), Proceedings of the First European Workshop on Probabilistic Graphical Models -PGM'02-, University of Castilla-LaMancha, Spain, pp. 186–193.

Wong, S.K., Ziarko, W., Li Ye, R., 1986. Comparison of rough set and statistical methods in inductive learning. International Journal of Man–Machine Studies 24, 53–72.