Statistics and Computing **14**: 181–198, 2004

Node deletion sequences in influence diagrams using genetic algorithms

M. GÓMEZ* and C. BIELZA†

**Department of Computer Science and Artificial Intelligence, University of Granada, Spain*
mgomez@decsai.ugr.es
†*Department of Artificial Intelligence, Technical University of Madrid, Spain*
mcbielza@fi.upm.es

Received August 2002 and accepted November 2003

Influence diagrams are powerful tools for representing and solving complex inference and decision-making problems under uncertainty. They are directed acyclic graphs with nodes and arcs that have a precise meaning. The algorithm for evaluating an influence diagram deletes nodes from the graph in a particular order given by the position of each node and its arcs with respect to the value node. In many cases, however, there is more than one possible node deletion sequence. They all lead to the optimal solution of the problem, but may involve different computational efforts, which is a primary issue when facing real-size models. Finding the optimal deletion sequence is a NP-hard problem. The proposals given in the literature have proven to require complex transformations of the influence diagram. In this paper, we present a genetic algorithm-based approach, which merely has to be added to the influence diagram evaluation algorithm we use, and whose codification is straightforward. The experiments, varying parameters like crossover and mutation operators, population sizes and mutation rates, are analysed statistically, showing favourable results over existing heuristics.

Keywords: decision-making under uncertainty, influence diagrams, genetic algorithms, NP-hard problems, node deletion sequence, statistical analysis

1. Introduction

An influence diagram (Howard and Matheson 1984) is a directed acyclic graph that is very commonly used in decision-making and inference problems. It has three types of nodes: (1) decision nodes (rectangular) representing decisions to be made; (2) chance nodes (circular) representing uncertainties modelled by probability distributions; and (3) a value node (diamond-shaped) with no successors, representing the (expected) utilities that model decision-maker's preferences. The arcs have different meanings depending upon which node they are directed to: the arcs to chance nodes or the value node indicate probabilistic dependence and functional dependence, respectively, while the arcs pointing at a decision node indicate the information known at the time of making that decision. The former are called *conditional* arcs and are what we are interested in here. The latter are only *informational*.

The basic operations of the Shachter's evaluation algorithm (Shachter 1986) are *chance node removal* (by computing the expected utility) and *decision node removal* (by maximizing the expected utility), carried out provided that the chance/decision nodes are predecessors of the value node. A chance node can be removed if its only successor is the value node. As a consequence, the latter inherits all the removed node predecessors. To eliminate a decision node that precedes the value node, it is assumed that there are no barren (or sink) nodes and that all other conditional predecessors of the value node are informational predecessors of the decision node. The value node inherits no new conditional predecessors. When it is possible to remove neither a chance nor a decision node, the algorithm finds an arc between two chance nodes that can be *reverted* –via Bayes' formula–, each node inheriting the predecessors of the other one. This transformation will make it possible to finally find (perhaps after various arc reversals) a chance node to remove. All these operations are combined sequentially to give the standard algorithm.

Let $G = (N, A)$ denote the influence diagram with nodes N and arcs A. Let D and C be the sets in N of decision and chance nodes, respectively, and v be the value node. Given a node i, the sets $C(i)$, $I(i)$, $S(i)$ denote its conditional predecessors, its informational predecessors and its direct successors,

respectively. With this notation, the pseudo-code for Shachter's algorithm is as follows:

```
1. Check for regular influence diagram and add
   ''no forgetting'' arcs
2. Eliminate all barren nodes
3. While C(v) ≠ ∅ do
     If ∃i ∈ C ∩ C(v) : S(i) = {v}, then remove chance
       node i
         Else if ∃i ∈ D ∩ C(v) : C(v)\{i} ⊂ I(i)
             remove decision node i
             eliminate new barren nodes
         Else find i ∈ C ∩ C(v) : D ∩ S(i) = ∅
             While C ∩ S(i) ≠ ∅ do
              find j ∈ C ∩ S(i) : there is no other
                directed (i, j)-path
              reverse arc (i, j)
             remove chance node i
```

This procedure sequentially removes nodes from the diagram until only the value node remains. The order of these nodes, from the first to the last node removed by the algorithm, is called a *node deletion sequence* (Shenoy 1992). This sequence is not completely determined. There is no problem when choosing between two decision nodes or between a decision and a chance node for removal, because there is only one way to do it. This is because of the way the algorithm operates. First, it will never be possible to have two candidates decision nodes to be removed because of the regularity of the influence diagram. Second, a decision node and a chance node will never tie for removal because this is guaranteed by the definition of the influence diagram and the conditions required for removal. So, with decision node d and chance node x, where both $d, x \in C(v)$, we have three possibilities: (1) there is no arc between d and x; (2) there exists arc (d, x); and (3) there exists arc (x, d). In cases (1) and (2), x is removed before d, while in (3) d is removed before x.

The problem is when the influence diagram has two or more chance nodes ready to be removed. This stems from the information constraints (represented by the informational arcs) that may only be specified up to a partial order, i.e., the transitive closure of the precedence binary relation induced by the informational arcs is a partial order on the set of all decision and chance nodes (see e.g. Nielsen and Jensen 1999). From a semantic viewpoint, it means that the decision-maker may not impose constraints regarding which of two or more chance nodes must precede the others in the decision-making process. For example, after deciding the admission of a patient into a hospital, there will be no constraints concerning which of the uncertainties *cost of stay* and *risks of being admitted* (infections, contagions, ...) is the first to occur. Another problematic case is when two or more arcs are candidates for reversal, which may yield even more different node deletion sequences.

All the sequences chosen to evaluate the diagram lead to the optimal solution of the problem, but may involve different computational efforts. The inheritance of predecessors in Shachter's algorithm may dramatically increase the size of some tables: the v table (in which the expected utilities are recorded) when a chance node is removed, and the two conditional probability tables involved in reversing an arc.

Computational effort is a critical issue in real problems. In Bielza *et al.* (2000), an influence diagram that models neonatal jaundice management to be used in a large hospital in Madrid consisted of 62 nodes and 169 arcs. During the problem-solving process, it required a maximum storage capacity (for the operation that brings about the highest increase) of 3.03×10^{13} memory positions to record all probabilities and expected utilities. This size exceeds the capacity of any personal computer, which renders such an appealing tool like influence diagrams almost useless in complex problems, revealing the huge gap between the theoretical and practical developments of techniques of this kind (Henrion 1989).

Finding an optimal deletion sequence (which minimises the maximum storage capacity) is an optimisation problem that has been shown to be NP-hard (Arnborg, Corneil and Proskurowski 1987). So, this is a difficulty inherent to influence diagrams, regardless of the algorithm employed for their evaluation. When the problem has a separable utility function, i.e., the joint utility function decomposes into several functions defined on smaller domains, the algorithm (Tatman and Shachter 1990) uses local computations at chance node removals, alleviating the computational burden somewhat. Otherwise, we can resort to heuristics for finding good deletion sequences. Kong (1986), Ezawa (1986), Mellouli (1987) and Zhang (1993) provide general heuristics, some of them briefly explained below.

Ezawa (1986) tackles the problem by transforming it into a problem of finding the maximum flow over a network constructed from the original influence diagram, relying on a precedence tree among the nodes. The process is too complicated, detached from the influence diagram framework, and intractable for complex influence diagrams. Kong (1986) proposes the heuristic called one-step-look-ahead: the next of the qualifying variables to be deleted is the one that leads to computations over the smallest domain. However, it is easy to check that choosing the node at random is very likely to lead to a better solution than yielded using Kong's heuristic, as usually found in randomised algorithms.

In addition to Shachter's algorithm, other algorithms may be used to solve influence diagrams. Rather than on arc-reversals, these algorithms are based on node-deletion strategies or on clique-trees approaches (see e.g. Cooper 1988, Shenoy 1992, Jensen, Jensen and Dittmer 1994, Shachter and Ndilikilikesha 1993, Zhang 1998, Madsen and Jensen 1999). The equivalence of all the algorithms is established in Shachter, Andersen and Poh 1990; Shachter, Andersen and Szolovits 1994. These other methods artificially convert the influence diagram into a Bayesian network and use Bayesian network propagation. The efficiency achieved in propagation algorithms comes from the factorization of the potentials used to quantify uncertainty (and preferences for decision problems). This factorization becomes explicit with cluster trees, on which computations are carried out. The key

point in the construction of the cluster tree is the *triangulation* of the (moral) undirected graph expressing the independencies of the problem.

It turns out that determining node deletion sequences amounts to finding elimination orders in triangulating those undirected graphs. Therefore, we could use different heuristics proposed for the triangulation problem, see e.g. Kjaerulff (1992), Larrañaga *et al.* (1997), Cano and Moral (1994), Shoikhet and Geiger (1997) and Amir (2001). Actually, unlike Bayesian networks, we cannot choose any elimination order for the triangulation; we must use strong elimination orders, imposed by the information constraints (see e.g. Jensen 2001), whenever they comply with valid orders according to Shachter's algorithm. At this stage, it appears that adaptations of these previous heuristics would work well.

However, our viewpoint will be as follows. First, Bayesian network-based algorithms only need to eliminate nodes in one direction (sometimes called COLLECTEVIDENCE), to compute the optimal strategy and the maximum expected utility of the problem. Yet, Shachter's algorithm also computes the posterior distributions of all variables (just before removing each chance node). This information is really valuable during the model construction and validation. Influence diagrams, usually embedded in Decision Support Systems, provide decision makers with these distributions, which are used, for example, to obtain diagnoses or result explanations in the medical field. This is not directly supplied by Bayesian networks algorithms.

This difference is outlined in Shachter and Ndilikilikesha (1993), when they explain the operations required to solve potential influence diagrams. The arc reversal operation is replaced by potential reversal. The product of the two potentials created is now maintained without normalizing divisions. Thus, only one of the resulting tables after an arc reversal is needed and less numbers are stored. But this is done at the expense of not computing posterior distributions. This situation in stronger for the other algorithms, that keep potentials as local as possible, demanding even more computations to obtain posterior distributions. In contrast, Shachter's algorithm considers the effect of the arc reversal operations required for it. Therefore, we select Shachter's algorithm to tackle this problem. There are two sequences to determine here: one for node deletion and another one for arc reversals (only when required), making the problem harder than merely searching for a certain elimination order in triangulating undirected graphs.

Another reason for driving this way our research is the users involved in decision making. A lot of decision makers or decision analysts have a decision theory education, often not necessarily trained in Bayesian networks. Skilled tasks like triangulation, propagation, conversion into a Bayesian network, etc. might overwhelm people used to manage classic tools in Decision Analysis, like decision trees and influence diagrams, see e.g. Clemen (1996) and Kirkwood (1997).

The keystone of this paper comes from a comment pointed out by Shachter (1986) in one of the pioneering papers on influence diagrams:

> An important improvement in the algorithm would be determining the optimal choice when breaking "ties" [. . .] This choice affects the time and memory requirements for future iterations. In general, even an expensive procedure to determine the optimal sequence of node reductions may be worthwhile.

The paper is organised as follows. Section 2 thoroughly explains the problem of node deletion sequences. Section 3 discusses our first attempts at dealing with this problem. Section 4 introduces the genetic algorithm we have developed. It requires only the qualitative evaluation of the influence diagram and has a natural and straightforward codification. In Section 5, experiments varying parameters like crossover and mutation operators, population sizes and mutation rates are carried out and analysed statistically. The results are a substantial improvement on those obtained with other heuristics. Finally, Section 6 states some conclusions and lines of future research.

2. Node deletion sequences and computational effort

Let us introduce a notation for our main concern, that is, the computational effort of solving an influence diagram.

Let S_i be the size of the table associated with node i, i.e., the total number of entries it requires. S_i will vary according to the type of node:

$$S_i = \begin{cases} |i| \times \prod_{j \in C(i)} |j| & \text{if } i \in C \\ \prod_{j \in C(i)} |j| & \text{if } i = v \end{cases} \quad (1)$$

where $|i|$ denotes the domain cardinal of node (or variable) i, i.e. the number of possible values of i. These values will be outcomes for chance nodes and (expected) utilities for the value node.

The sum of the sizes of all the tables in the diagram is the total storage capacity TS (or memory requirements) of the diagram: $TS = \sum_{i \in C} S_i + S_v$. Note that the tables for decision nodes always retain the same elements throughout the whole process of influence diagram evaluation, namely, their alternatives. For this reason, we will not include the sizes of decision node tables in TS.

As explained above, there are two situations that lead to different node deletion sequences.

2.1. *Chance nodes tying for removal*

The first case arises when the influence diagram has two or more chance nodes ready to be removed.

Example 1. Figure 1 illustrates this idea. All the variables are assumed to be binary except B, which has four possible outcomes and $A1$ with five states. The number below each node i is S_i. The isolated number inside a triangle is TS.

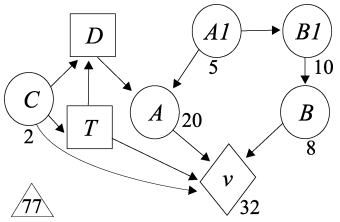

Fig. 1. *Two candidates chance nodes, A and B, for removal*

Only chance nodes A or B can be removed in the first step. Whichever you choose, v will inherit its predecessors. The computational effort will differ depending on the domain of these predecessors $A1$, $B1$, D and the subsequent stages of the algorithm, although the optimal solution will always be reached.

Figure 2 shows the three different possibilities for solving the influence diagram in Fig. 1. It shows how the diagram, the table sizes and the total storage capacity change as the diagram is evaluated. Let us suppose we choose to start by removing node A. The node deletion sequence is $\langle ABB_1A_1DTC\rangle$ and the maximum storage capacity is 185 (see *ID*2 in Fig. 2). Note the fluctuation of storage capacities *TS*: 77, 185, 97, 47, 10, 6, 4, 1.

For this example, there are two more node deletion sequences: $\langle BB_1AA_1DTC\rangle$ and $\langle BAB_1A_1DTC\rangle$ (see Fig. 2). For sequence $\langle BB_1AA_1DTC\rangle$, the maximum storage capacity is 77. For the other sequence $\langle BAB_1A_1DTC\rangle$, the maximum storage capacity is 97. Therefore, the second sequence is the optimal sequence for this problem. Note that the maximum storage capacity for sequences 1 and 3 is reached when removing A. For the second sequence, it is reached in the initial diagram (where no diagram reduction has been performed yet), but the second highest value is when removing A as well.

Luckily, Kong's heuristic finds the same sequence. For instance, in Fig. 1, we should pick B over A, because its deletion involves computations over $\{C, T, A, B, B1\}$, whereas removal of A involves computations over $\{C, T, A, B, D, A1\}$.

Note that we are following Shachter's algorithm. Otherwise, we might have even more deletion sequences. For example, in *ID*4 in Fig. 2, Shachter's algorithm only permits removal of $B1$, though $A1$ could be removed after arc $(A1, B1)$ reversal.

In general, we can infer when the inheritance of predecessors exponentially increases the size of the v table after a chance node removal. A chance node removal bears on the value node table and on its own table (which vanishes). The following proposition states the results.

Proposition 1. *Let ID^1 be an influence diagram with an associated S_v^1 and TS^1. Denote $C^1(v)$ as the conditional predecessors of v in this diagram. Let i be a chance node with S_i^1, $C^1(i)$. Let ID^2 be the influence diagram obtained from ID^1 after i removal, with sizes S_v^2 and TS^2 ($S_i^2 = 0$). Since the conditional predecessors of the value node in ID^2 become*

$$C^2(v) \leftarrow C^1(v) \cup C^1(i)\backslash\{i\}$$

then the following recursive formulas hold

$$S_v^2 = \frac{S_v^1}{|i|} \prod_{j\in C^1(i)\backslash C^1(v)} |j|$$

$$TS^2 = TS^1 - S_i^1 + S_v^2 - S_v^1$$

Hence,

(a) $S_v^2 \geq S_v^1$ iff $\prod_{j\in C^1(i)\backslash C^1(v)} |j| \geq |i|$.
(b) $TS^2 \geq TS^1$ iff $S_v^2 - S_v^1 \geq S_i^1$.

If $C^1(i) \subset C^1(v)$ $(\Rightarrow C^1(i)\backslash C^1(v) = \emptyset)$, then $\prod_{j\in C^1(i)\backslash C^1(v)} |j|$ is obviously considered 1.

Proof: It is straightforward from (1). □

S_v increases when the domain cardinal of the removed node is less than the domain cardinal of the product space of the predecessors inherited by v. See e.g. the step from *ID*3 to *ID*5 in Fig. 2, which is the worst case since v inherits *all* the predecessors of the removed node (since $C^1(i) \cap C^1(v) = \emptyset$).

As far as *TS* is concerned, its growth from one diagram to the next depends not only on the value node inheritances but also on the table size of the removed node. In other words, *TS* increases when this table size is less than the difference between the v table size after and before the chance node removal. For example, node B and the 8 elements of its table are to be removed from *ID*2 in Fig. 2 with 185 memory positions required. v, with a table of size 160, inherits node $B1$ (see *ID*4 in Fig. 2). The new storage requirements can be computed as $185 - 8 + 160(\frac{2}{4} - 1) = 97$.

2.2. *Arcs tying for reversal*

The second case that introduces variability in the transformations to the influence diagram arises when two or more arcs are candidates for reversal. These arcs may or may not have the same origin.

Example 2. Figure 3 illustrates this idea. All the variables are assumed to be binary except A which has three possible outcomes.

Neither A nor B can be removed until v is their only successor. Therefore, the only possibility is to reverse arc (A, B) or (B, C) with mutual inheritance of predecessors. The arc chosen may have an influence on the size of the probability distributions stored at the chance nodes involved and, consequently, on the computational burden.

Figure 4 shows the four different possibilities for solving the influence diagram in Fig. 3. Note that although an arc reversal only immediately affects the sizes of the conditional probability tables of both nodes defining the arc, it later has a bearing on the inheritances that v will receive. In the example, influence diagram *ID*2 requires 12 entries in the v table, and it only requires 8 entries later at *ID*4. The reversal of arc (A, B) causes A to

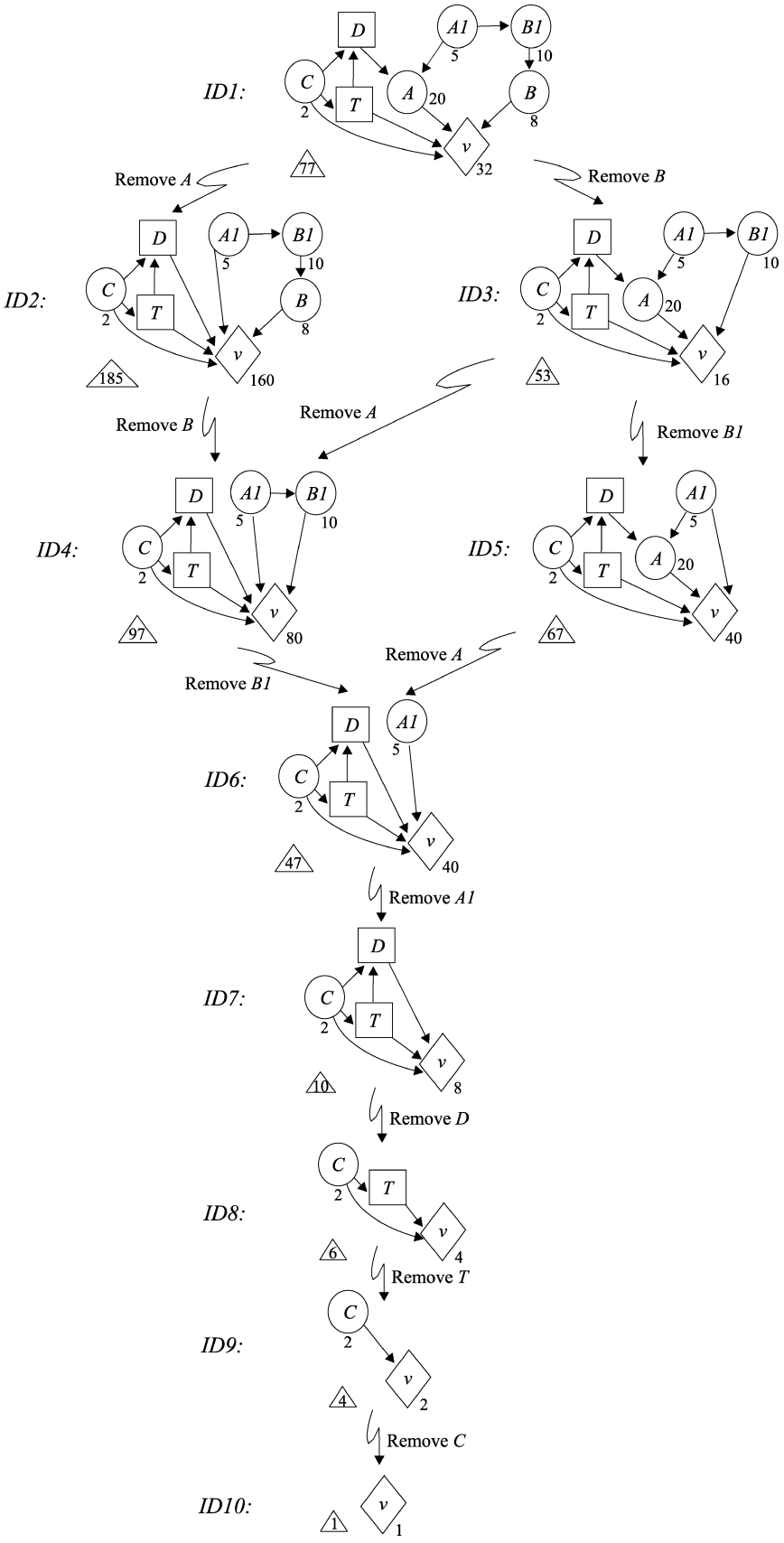

Fig. 2. *All the possible ways of evaluating the influence diagram in Fig. 1*

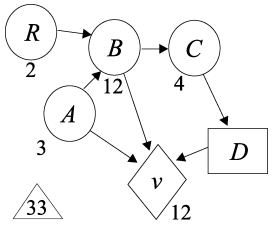

Fig. 3. *Two candidates arcs,* (A, B) *and* (B, C)*, for reversal*

inherit R as a new predecessor, which is inherited by v after A removal. However, diagram *ID*3 also requires 12 entries in the v table, but it requires 24 entries later at *ID*5, because the reversal of arc (B, C) implies that B inherits C as a new predecessor, and then v inherits C (B new predecessor) and R (B old predecessor) as new predecessors after B removal.

Table 1 shows all the sequences and their maximum storage capacities. (i, j) denotes the reversal of this arc. The required reversals are inserted in the sequence. The first two sequences $\langle ARBDC\rangle$, $\langle ABRDC\rangle$ are optimal.

Note that the four transformation orders have the same length or number of required transformations (8), 3 arc reversals at the same position, and lead to different node deletion sequences. Nevertheless, this is not always true.

Example 3. Figure 5 shows another influence diagram.

Among the different transformation orders, we find

$$(A, C)(A, B)A(B, E)B(C, E)CDE$$

and

$$(A, B)(A, C)A(B, C)(B, E)B(C, E)CDE$$

which are of different length, because they have a different number of reversals, and both lead to the same node deletion sequence $\langle ABCDE\rangle$. The second order requires one more reversal because two arcs must be reversed –(B, C), (B, E)– after A removal and before B removal. The former is created as a consequence of (A, C) reversal. However, after (A, B) reversal, the first order creates arc (C, B) instead, i.e. the opposite arc, which does not need be reversed in this first transformation order.

In general, the following proposition infers when an arc reversal increases size *TS*.

Table 1. *Results for the influence diagram of Fig. 3*

Transformation order	Max stor capac
$(A, B)A(R, B)R(B, C)BDC$	34
$(A, B)A(B, C)B(R, C)RDC$	34
$(B, C)B(A, C)A(R, C)RDC$	53
$(B, C)B(R, C)R(A, C)ADC$	53

Proposition 2. *Let* ID^1 *be an influence diagram with an associated* TS^1*. Denote* $C^1(i)$ *and* $C^1(j)$ *as the conditional predecessors of chance nodes* i *and* j*, respectively, in* ID^1*, with* S_i^1, S_j^1*. Let* ID^2 *be the influence diagram obtained from* ID^1 *after* (i, j) *reversal, with* TS^2*. Since there is mutual inheritance of predecessors in* ID^2*, then the following recursive formulas hold*

$$S_i^2 = S_i^1 |j| \prod_{k\in C^1(j)\setminus C^1(i)} |k|$$

$$S_j^2 = \frac{S_j^1}{|i|} \prod_{k\in C^1(i)\setminus C^1(j)} |k|$$

$$TS^2 = TS^1 + S_i^2 + S_j^2 - S_i^1 - S_j^1$$

Hence,

(a) $S_i^2 \geq S_i^1$.
(b) $S_j^2 \geq S_j^1$ iff $\prod_{k\in C^1(i)\setminus C^1(j)} |k| \geq |i|$.
(c) $TS^2 \geq TS^1$ iff $S_i^2 + S_j^2 \geq S_i^1 + S_j^1$.

Once again, $\prod_{k\in\emptyset} |k| = 1$ by definition.

Proof: Straightforward from (1). □

When reversing arc (i, j), the size of the i table always increases. The size of the conditional probability table stored at j depends on the i domain cardinal (that S_j loses) compared to the domain cardinal of the product space of its inherited nodes (that S_j gains). It is similar to Proposition 1(a), node v there playing the role of j here. Moreover, in order to have an increase in *TS*, the global behavior of $S_i + S_j$ has to be checked (new global size greater than old).

Figure 4 is a possible illustration of Proposition 2. After (B, C) reversal (see *ID*3), S_C increases since $3 \cdot 2 > 2$ ($S_C^1 = 4$, $S_C^2 = S_C^1 \frac{4}{2} 3 \cdot 2 = 12$). Therefore, *TS* obviously increases. However, after (A, B) reversal (see *ID*2), S_B decreases since $1 < 3$ ($S_B^1 = 12$, $S_B^2 = S_C^1 \frac{1}{3} = 4$). Thus, we have to check condition (c): $12 + 4 > 3 + 12$ to see that *TS* increases. Finally, note that the removal of a decision node D never increases *TS*: $TS^2 = TS^1 + S_v^1(\frac{1}{|D|} - 1) \leq TS^1$.

In summary, each transformation to an influence diagram has a bearing on many steps ahead. We have derived the change in its storage requirements one-step-ahead. This depends on many factors such as the number of inherited nodes and their cardinals, the size of the deleted tables, the domain cardinal of the node at the origin of the reversed arc, etc. Yet this greedy (myopic) knowledge of the change does not allow us to find the best node deletion sequence (otherwise, Kong's heuristic would be the ideal strategy), although it will be used to compute recursively the cost of each transformation and to get a starting point for our initial proposals, see Section 3.

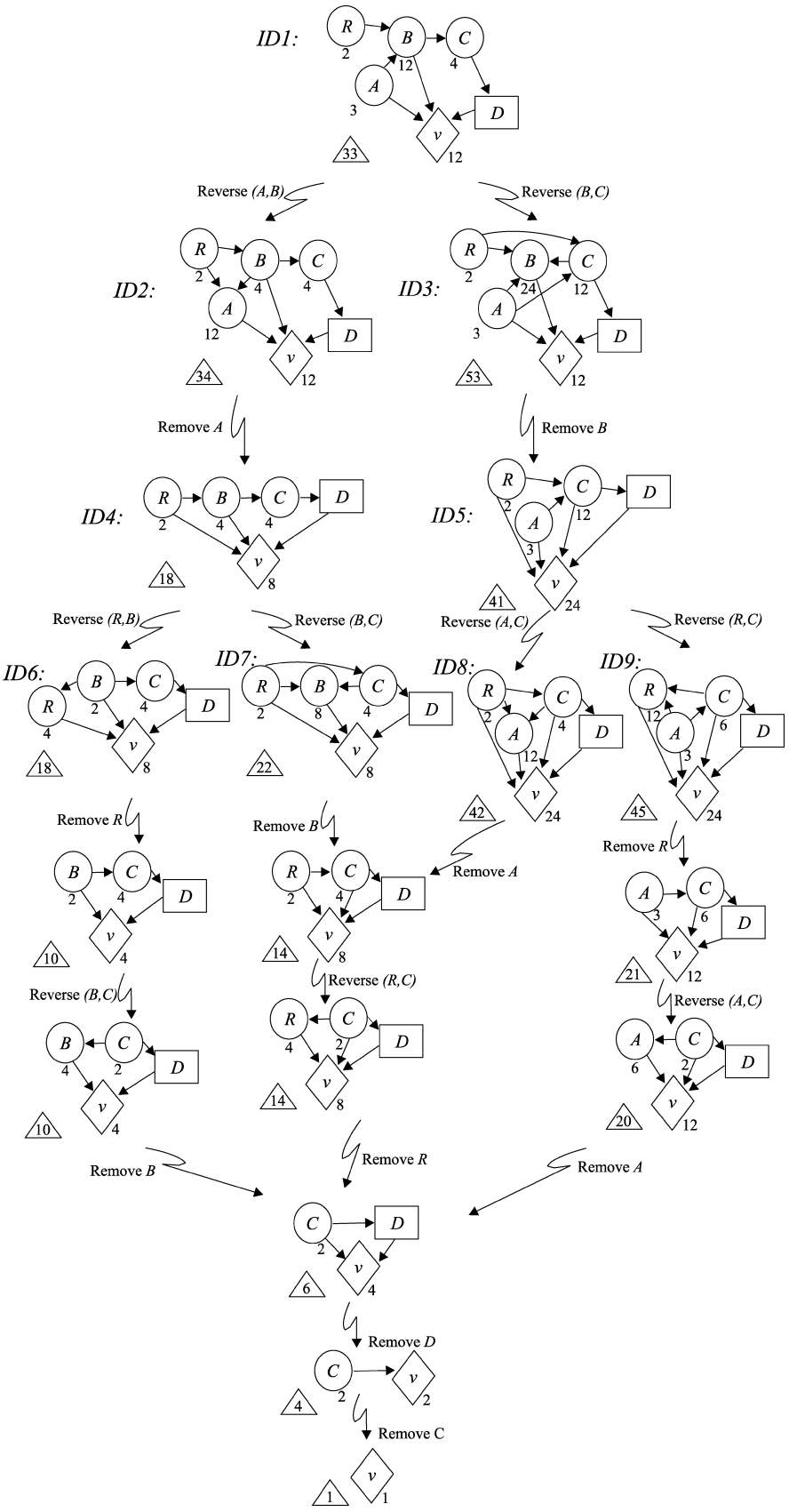

Fig. 4. *All the possible ways of evaluating the influence diagram in Fig. 3*

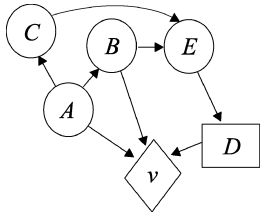

Fig. 5. *Illustrating the same sequences but different transformation orders*

3. Our initial proposals

Our initial idea is to develop a qualitative implementation of the standard algorithm, in which the basic operations of node removals and arc reversals are carried out structurally rather than numerically (much in the way of figures of Section 2). By means of this tool we can find out the structure of the influence diagram at any time, thereby giving memory requirements for probabilities and expected utilities. It is easy to add Kong's heuristic to this algorithm, although we should remember its unreliable results, see an example in Gómez (2002).

We improve this approach by developing a new tool that tries to construct a tree with all the possible influence diagrams reached by the different combinations of transformations. The original influence diagram is located at the root of the tree. Given an influence diagram, qualitative evaluation allows us to find out which the next possible influence diagrams are, i.e. the diagrams obtained by one transformation to a copy of the diagram. Each one is located at each branch that sprouts from the previous diagram. Figures 2 and 4 are examples of this tree.

In order to rule out a combinatorial explosion of the search tree, not all the possible diagrams are inserted into the tree. Equal diagrams are not allowed. Thus, if several paths through the tree lead to identical influence diagrams, the method chooses the lowest cost path (i.e., the least computational effort) and prunes the other paths. Note that the tree has only one possibility when removing a decision node, i.e., one branch from each branch of the last level.

However, although it is a structural approach, the construction of the tree is obviously intractable for large influence diagrams. For the case of Fig. 4, the search is exhaustive since the diagram is small.

We proceed as follows to get a smaller tree:

1. *Initial bound using Kong.* Qualitatively solve the influence diagram using Kong's heuristic and get a bound of the maximum *TS* achieved.
2. *Improve that bound.* Qualitatively solve the influence diagram many times, deciding the deletion sequence at random, and get another bound of the maximum *TS* achieved. This is usually better (lower) than the above and is, therefore, more restrictive. Hence, a greater number of possible deletion sequences will be able to be discarded during the search stage (step 3 below).
3. *Search.* Search the parts of the tree that lead to sizes below that bound, i.e., any branch yielding a problem size greater than the established bound is discarded. Therefore, the associated influence diagram is not considered and all the subsequent influence diagrams obtained later from it are pruned. This is much in the typical branch-and-bound spirit of Operations Research.
4. Steps 2 and 3 can be iterated several times in order to refine the bound.

Even in this case, there are too many possible deletion sequences in complex influence diagrams, and we can expect to have to eventually prune the tree further (randomly for example).

The complexity of arc reversals is somehow alleviated by considering only a small percentage of possible reversals, then deciding the arc at random several times, and saving the best orders until that fixed percentage is reached.

Although it is only a semi-exhaustive search, this method we have implemented improves the solution obtained by means of Kong's heuristic and merely calls for the qualitative evaluation of the influence diagram.

4. A genetic algorithm

In this section, we introduce a genetic algorithm, a kind of probabilistic search algorithm that mimics biological evolution processes (Holland 1975) and that has been used recently in many NP-hard optimization problems. As we mentioned in Section 1, these algorithms have been applied to the triangulation problem in Bayesian networks. In contrast to that framework, we find constraints on the possible elimination orders for the triangulation, making harder the movement through the problem search space. Moreover, arc reversals are indeed needed by decision makers, who wish to have posterior probabilities for the variables. To the best of our knowledge, it does not exist yet a genetic algorithm adapted to Shachter's algorithm, fulfilling these requirements.

Although there is no guarantee that a genetic algorithm can find the optimal solution of the problem, there is empirical evidence that acceptable solutions are found in a reasonable time, as compared to other combinatorial optimization algorithms (Mitchell 1998).

In genetic algorithms, the individuals represent the search space points of the problem, and each one is characterized by genetic material. The target is to find the individual with the best material from the search space, measured by an objective or evaluation function or *fitness*. The part of the search space to be examined —*population*— changes at each iteration: each generation retains the best individuals to produce children, while the worst individuals tend to vanish.

In our case, the individuals are different ways of solving the influence diagram, that is, node deletion sequences. Thus, an individual is given by a set of parameters (genes), each representing a node removal. The objective function is the maximum *TS* of the tables needed to store probabilities and expected

utilities. The following subsections focus on each step of the above scheme.

4.1. *Codification of individuals*

The codification of individuals is a crucial issue. It must represent all the individuals of the search space and accommodate the distinguishing features of the problem under consideration. Genes represent only node removals but they internally store information about the required arc reversals. Arc reversals performed to remove a node induce a different influence diagram structure and this is captured with the objective function.

Thus, the algorithm will manage transformation orders. The sequence of genes $\langle ARBDC\rangle$ (see Table 1) contains information about arc reversal for nodes A, R and B and it would correspond to order $(A, B)A(R, B)R(B, C)BDC$. The objective of arc reversals is to prepare a chance node removal. So, when gene A is generated first, it also contains information about the arc reversal(s) required to be removed. All the arcs coming from A (other than to v) must be reversed, in this case, only arc (A, B).

At each iteration, we will not generate repeated individuals (see below), and therefore, no distinction will be made between two transformation orders that lead to the same sequence. Only one of them will be generated with its fitness computed from the respective transformation order. This loss of individuals –at each iteration— is the price we have to pay for having an easy codification of individuals.

After this discussion, let us introduce some more notation. Given an influence diagram with n (decision and chance) nodes to be removed, the individual will consist of n genes. Let $\mathcal{S}$ be the set of possible node deletion sequences $s = \langle t_1, t_2, \ldots, t_n\rangle$, where t_i is the i-th removed node in the sequence. Then, TS^i is the total storage capacity of the influence diagram that results when node t_i is removed, maybe, after some arc reversal(s) when t_i is a chance node. We define the fitness of an individual s as $f(s) = \max_{1\le i\le n} TS^i$. The genetic algorithm searches for the individual $s \in \mathcal{S}$ with minimum $f(s)$.

Often node removals cannot be performed in any order, because there are constraints between them. Some nodes cannot be removed until others have been removed previously (see Section 1). These constraints or dependencies are very important, since they influence the set of operations to be performed on the individuals. For this reason, we take the following steps to determine the relationships between the different node removal operations:

1. Once the algorithm has generated a population (see Section 4.2), it observes which nodes (genes) appear at position i ($i = 1, \ldots, n$) in every individual's genetic material.
2. A list of length n is then formed, indicating all the nodes that appear at position i of every individual in the population in its position i. Let N_i be the set of those nodes.
3. The nodes are grouped according to this list: if two or more positions of the list, e.g., i, j, are such that $N_i \cap N_j \neq \emptyset$, then $N_i \cup N_j$ form a group. Actually, what we have just done is to form groups of nodes with internal dependencies with respect to their removal; nodes that lead to partial deletion orders. If a removal is always carried out at the same position, it will lead to a singleton (as e.g. for decision nodes removals). The groups represent stages in the solution of the influence diagram, which are related to the order imposed by the decision nodes. Therefore, the nodes of one group have to be removed before those of the following group.
4. The last step is to determine the rules that govern each group, i.e. rules of precedence in the group node removal process. They are stored in a square matrix, whose dimension is equal to the number of nodes in the group and where the elements $a_{ij} = 1$ if node j can only be removed once node i has been deleted, with i, j nodes in the group. Otherwise, $a_{ij} = 0$.

This process creates and controls a search space, with the special individuals of our problem. Genetic operators will later produce more individuals, also controlled by these rules and groups to guarantee valid deletion sequences. Groups and rules information will be used in the processes of genetic material exchange among individuals, see Section 4.3. This information will be constantly updated, whenever necessary, as long as new individuals join the population.

Example 4. To illustrate groups and rules, let us take the influence diagram of Fig. 6.

Suppose that after an initial population has been generated, its individuals are

$$\langle XZYACBD\rangle, \langle ABCXZYD\rangle, \langle XACZBYD\rangle, \langle ABCXYZD\rangle.$$

Then, the list is

$$N_1 = \{X, A\};$$
$$N_2 = \{Z, B, A\};$$
$$N_3 = \{Y, C\};$$
$$N_4 = \{A, X, Z\};$$
$$N_5 = \{C, Y, Z, B\};$$
$$N_6 = \{B, Y, Z\};$$
$$N_7 = \{D\}$$

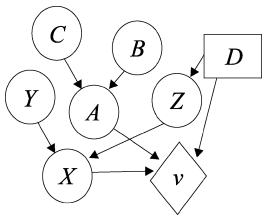

Fig. 6. *Illustrating groups and rules.*

There are two groups: $\bigcup_{i=1}^{6} N_i = \{A, B, C, X, Y, Z\}$ is the first group and $\{D\}$ is the second group. Decision nodes will always form unitary groups. The rules for the first group will be given by the matrix

$$\begin{array}{c} \\ A \\ B \\ C \\ X \\ Y \\ Z \end{array} \begin{pmatrix} A & B & C & X & Y & Z \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

In this case, we have 80 possible node deletion sequences. Note that since all the individuals are obviously valid deletion sequences, the first step has output the rules that force Y and Z to be deleted after X and that force B and C to be deleted after A. At this step, there are no more rules.

As the population starts to evolve producing new individuals, the crossover and mutation operators will be closed operations, i.e., groups of the offspring will respect the rules. Therefore, the ones will never disappear from the rule matrices. Conversely, some zeroes may become ones during the evolution process, as long as we approach certain (good) populations with more constraints among the relative positions of their genes, namely, as long as the algorithm converges to a certain deletion sequence. Eventually, new groups may be formed. For example, if the above population produced the offspring $\langle XZACYBD \rangle$, and the extended population was reduced by dropping its first individual, then the updated matrix would have a 1 at positions (1, 5) and (3, 5), because Y is always deleted after A and after C. No new groups are formed in this case. For this reason, the way we choose the initial population and how we make it evolve is essential to ensure that we are not trapped too early in a population that has a constraint imposed by chance rather than by an improvement in the search for the optimum.

4.2. *Initial population*

Our initial population will be a set of influence diagram deletion sequences, for which the next node to be removed, if there is more than one qualifying candidate, is selected at random. It is well-known that a *non*-random initial population may increase the speed of convergence of the genetic algorithm, at the expense, however, of being trapped in local optima.

As regards the cardinal of this population, it should be taken into account that: (1) too many individuals may slow down the search process enormously, because the selection, crossover and mutation operators and the computation of the objective function would be hard to manage; (2) too few individuals do not adequately cover the search space and may yield incomplete group information. Based on empirical evidence, Alander (1992) suggests a cardinal between l and $2l$, where l is the number of genes of an individual. In our case, $l = n$ and we introduce this choice as parameter λ of our algorithm.

4.3. *Crossover and mutation operators*

4.3.1. Selection of parents

Once the initial population has been chosen and its quality determined via the objective function, parents are selected from it in every iteration to produce children that will be added to the population. We propose the selection of $\lambda/2$ individuals to produce children, i.e., we have $\lambda/4$ pairs for crossover. While fewer than $\lambda/2$ individuals are selected, individual k is selected according to a probability p_k. Since a probability proportional to the objective function has shown to lead to premature convergence (local optima), we choose a probability based on a non-linear ranking: all the individuals are sorted from best (ranking 1) to worst (ranking λ) according to their fitness, and an individual with ranking k is selected with probability p_k proportional to $q(1-q)^{k-1}$ where $q \in (0, 1)$ is a fixed parameter (Mitchell 1998). Hence, this selection process provides the best individuals (i.e., those with smaller f) with a greater probability of passing on their genes to future generations. The smaller f is, the greater the probability of the individual being selected as a parent and transmitting its genes, avoiding, however, that individuals with very good fitness dominate the population and the selection mechanism (super-individuals).

Note that once the ranking is known, the probabilities are always the same and do not change from generation to generation (static method), unlike the probabilities proportional to the fitness (dynamic method).

4.3.2. Crossover

The parents selected previously are coupled at random. Each couple will be forced, in principle, to produce two children. An exception is the VR operator, for which parents are not joined in couples (see below).

Crossover should increase the quality of the population, at least on average. We have implemented a number of crossover operators. The first one, designed for our problem, will be called the *group exchange* operator (GE). Each offspring inherits complete groups of genes from its parents. Which parent is selected for each group is decided at random, although it might be done taking into account the objective function: the progenitor with a smaller f would have a greater probability of transmitting its group of genes. The inheritance of complete groups of genes assures that the newly created individuals represent valid evaluations of the influence diagram. However, offspring might be equal to their parents or equal to each other, in which case we reject the descendant and try again, because repeated individuals are not allowed in the same generation. This is unlikely to occur when we have enough groups.

Other crossover operators come from the well-known travelling salesman problem (TSP). Although also an ordering problem, the TSP is rather different than ours: TSP is usually assumed

to be symmetric (e.g., sequence $\langle ABC \rangle$ represents the same individual as $\langle CBA \rangle$) and only the relative order matters (e.g., sequence $\langle ABC \rangle$ represents the same individual as $\langle BCA \rangle$). Furthermore, in our problem, not all the permutations are legal individuals and this is why we defined the groups and rules above.

Therefore, we have chosen the operators that can be used here, having in many cases to adapt them to our specific problem. The operators are: partially-mapped crossover operator (PMX), cycle crossover operator (CX), order crossover operator (OX1), order-based crossover operator (OX2), alternating-position crossover operator (AP), and voting recombination operator (VR), see their description in Appendix A.

Two main adaptations are carried out: (1) some operators have to be applied at a group level due to the impossibility of mixing genes of different groups; (2) some operators have been changed a little in order to avoid too much computational cost when rejecting either invalid or repeated individuals.

The following operators are applied at a group level: GE, OX1, VR. To obtain more population diversity, the operator is applied over *several* groups. The remaining operators, PMX, CX, OX2 and AP, can be applied directly to the whole individual without taking into account the groups, because, due to their nature, genes belonging to one group in the parents will never appear in a different group in the offspring.

The following operators have been slightly modified: CX, OX2, VR. As far as CX is concerned, when a cycle from one of the parents (say parent 1) has finished, and we find that the first element we can take from the other parent (say parent 2) is equal in both parents (same gene at the same position), the newly initiated cycle ends and we have to go back to parent 1. Since it may produce offspring identical to one of its parents (here to parent 1), we modify the original CX operator by forcing it to continue taking genes from parent 2 in this situation. For instance, if parent 1 is $\langle HCFEDGBA \rangle$ and parent 2 is $\langle ACEGHFDB \rangle$, since gene C is the beginning of the second cycle and occupies the same position in both parents, then an offspring created with the modification is $\langle HCEGDFBA \rangle$. Otherwise, this offspring would be identical to parent 1. Also, we observe that, unlike other crossover operators, the same couple of parents always produce the same couple of offspring, and if some offspring does not meet the rules of some group(s), it is rejected and not included in the next generation.

As far as OX2 is concerned, the number of random positions in a parent string is fixed and equal to (the closest integer to) 40% of the number of genes in the individuals to avoid offspring identical to their parents. However, we allow different positions to be used to create both offspring rather than the usual same positions. This will be a resort when the same positions lead to offspring that do not meet the rules of the group. As an extreme case, after trying all the possible combinations of the number of positions chosen, no two valid offspring may be created from a couple. In this case, only the successful offspring are included in the next generation.

As far as VR is concerned, we maintain neither a fixed number of parents p nor a fixed threshold t. Our proposal consists of dynamically updating p and t through the generations to improve its results: lower p and t in the initial generations, where it is not very demanding to inherit genes, since initial populations that are far from convergence have hardly many genes in common; greater p, and t closer to p afterwards, as long as the population is more uniform.

Apart from CX and OX2, the problem of rejecting invalid individuals also occurs with the PMX, OX1 and VR operators, although it is rather unlikely with VR. For PMX, we alleviate the problem somewhat by allowing the two cut points that define the substring or mapping section in the parents to be varied. For OX1, we allow the length and position of the selected string segments to be varied.

4.3.3. Mutation

Offspring are added to the population and there is a probability close to zero of them mutating, i.e., changing their genetic material somewhat. With mutation, new states are explored and local optima are avoided. In our implementation, we introduce a new parameter ρ that will be the probability of mutation.

We have implemented a number of mutation operators. We have borrowed operators from the TSP again (see e.g., Larrañaga *et al.* 1999): displacement mutation operator (DM), exchange mutation operator (EM), insertion mutation operator (ISM), simple-inversion mutation operator (SIM), inversion mutation operator (IVM), and scramble mutation operator (SM), see their description in Appendix A.

Once again, our specific problem demands *all* the operators to be applied at a group level, because, otherwise, genes might escape from the group they belong to. The resulting mutated group is guaranteed to meet the rules. For example, with the EM operator, it is necessary to check whether the selected exchange between two genes i, j is allowable. This is done by checking whether the element a_{ij} of the associated matrix of this group is equal to 0. Otherwise, two new random selections i', j' will be drawn until two unconstrained positions are found. If a group cannot mutate into a valid group under any circumstances (this has happened with the DM, SIM and IVM operators), mutation is not carried out and the group remains unchanged.

The EM and ISM operators require mutation of genes belonging to groups of length greater than 1. However, the DM, IVM, SIM and SM operators require groups of length greater than 2 to obtain reasonable mutations.

4.4. *Reduction of the population*

After crossover, the population has more individuals and some of them are removed in order to reduce the population to its original size λ. Since sometimes less than $\lambda/2$ new offspring are created, we always remove as many individuals as new offspring have been created.

Now we want the best individuals to be left in the population. The best individual found is always kept and introduced in the

next generation, which is a somewhat elitist strategy (*queen bee*). The remaining individuals are discarded according to a probability based on a non-linear ranking, where this time the individuals are sorted from worst to best depending on their fitness. Therefore, individuals with a greater f will have a greater probability of being removed from the population. This idea of generation replacement is related to the Modified Genetic Algorithm introduced by Michalewicz (1992).

4.5. *Stopping rule*

We have defined above the fitness f of each individual. We introduce here another measure that complements f. Note in Example 2, that the first and second sequences are optimal and $f(\langle ARBDC\rangle) = f(\langle ABRDC\rangle) = 34$ (see Table 1). However, the mean storage capacity required during influence diagram evaluation by following the first sequence is 14.38 ($= 115/8$), which is lower than 18.12, the mean size required by the second sequence.

The mean size or mean storage capacity records the average requirements a computer will need when evaluating the influence diagram. Hence, we are interested not only in the worst moment during the evaluation of the influence diagram (given by f) but also in an average behaviour throughout the process. This is rather informative and descriptive in relation to a node deletion sequence. Therefore, faced with two sequences with the same f value, we will prefer the sequence with the lowest mean storage requirements.

There are a number of stopping rules for genetic algorithms. We use the following scheme to decide when the algorithm cannot find a better solution: stop the algorithm if (a) after a fixed number of iterations I, there is no improvement; or (b) the population has converged at level β.

For (a), we consider that there is an improvement in a new iteration (i.e., a new generation) if there is some new individual such that: (1) its f is better than that of the best individual of the previous generation; or (2) if f's are equal, its associated mean storage capacity is less than that of the best individual of the previous generation. With this convergence criterion, the termination of the algorithm is guaranteed.

For (b), following De Jong (1975), a population converges at level β if at least β% of the genes have converged. A gene has converged at level α if it has the same value in at least α% of the individuals in the population. This criterion takes into account the uniformity of the population.

5. Experiments

In this section, we carry out experiments to find the optimal node deletion sequence of a real-size influence diagram related to neonatal jaundice management. It will be called *Jaundice*, see Bielza *et al.* (2000) for an extended version of this diagram. The *Jaundice* graph has 46 nodes plus the value node (2 decision nodes), 97 arcs and 10,936 initial memory positions. See Appendix B for its description. We also look for the best parametrisation of the genetic algorithm, i.e. the tuning of the strategic parameters, to the node deletion sequence problem.

5.1. *Genetic algorithm set-up*

The number of genes is $n = 40$ (6 nodes were found to be barren nodes), 7 groups are always formed with respective sizes: 8, 12, 1, 1, 2, 1, 15 genes. The following parameters are fixed as: $q = 1/40$ (for the selection of parents); crossover and mutation operators that work at a group level are applied to (the closest integer to) 40% of the groups; $\alpha = \beta = 95$; $I = 400$ iterations (waiting for an improvement). Dynamic updating of p and t in the VR operator is as follows: $t = 3$, $p = 6$, for less than 400 generations; $t = 4$, $p = 7$, for between 400 and 800 generations; $t = 5$, $p = 7$, for more than 800 generations.

For each experiment, 20 executions of the algorithm are carried out with the same genetic operators, probability of mutation ρ, and population size λ. The best evaluation found at each execution is recorded. Thus, we have $f_1, \ldots, f_{20}$ for each experiment. The results shown in the following section consist of the best f_i, their average and standard deviation, the average percentage of converged bits and the average number of iterations performed before convergence.

5.2. *Results*

The first batch of experiments comprises all the possible combinations of genetic operators ($42 = 7 \times 6$), with a fixed population size and mutation rate: $\lambda = 30$, $\rho = 0.01$ (see Table 2). This is 840 (42×20) experiments altogether. The figures correspond to the results explained above. Each one is computed from 20 executions of the algorithm.

Firstly, we analyse the best f-value found. It is 179,186,784 and it was only achieved 3 times out of the 840 executions run. Only crossover operators GE (twice) and AP found it. Only mutation operators IVM (twice) and ISM found it. A very close value (179,186,816) was found by pairs OX1-DM, OX1-ISM and GE-DM. Conversely, the worst f-value, yielded as the best one by 30 executions, was greater than 716,746,751.

Secondly, we observe the remaining values at the 20 executions, summarised as the second and third values of each combination of operators in the table. It allows us to statistically analyse the differences between these combinations. The Kruskal-Wallis test gave a highly significant p-value for the differences among means of all the crossover operators ($p = 0.0004$). For mutation operators, p is 0.029. For a thorough study, we also perform an analysis of multiple comparisons of the operator effects (function `multicomp` of S-Plus, Venables and Ripley 1994) to see where any differences exist and how large the differences are. It computes simultaneous 95% confidence intervals for all pairwise differences between f means, based on the operator types (the so-called MCA comparisons, Hsu 1996). Figure 7 shows it for crossover operators. The labeling states that Sidak's

Table 2. *Results for the* Jaundice *graph. First batch*

	GE	OX1	OX2	AP	PMX	CX	VR
DM	179,186,816	179,186,816	179,190,240	179,190,240	179,191,984	179,193,216	179,191,968
	220,010,770	186,164,595	188,155,685	251,867,379	213,043,132	206,076,227	224,989,033
	124,500,930	31,165,243	40,069,863	130,841,273	122,579,939	65,648,376	72,538,455
	24.5	42.75	50.375	40.875	56.5	37.375	53.75
	647	625	591	644	565	785	1,037
EM	179,191,776	179,191,088	179,189,088	179,189,856	179,190,240	179,190,240	179,190,816
	197,116,040	228,970,343	206,073,694	268,790,363	294,672,984	216,030,194	277,751,298
	55,155,784	127,812,486	120,199,965	169,494,165	191,577,696	65,964,610	142,619,991
	35.25	48.875	47.5	34.5	59.875	37.25	62.875
	962	661	684	691	568	809	1,290
ISM	179,186,784	179,186,816	179,191,392	179,190,624	179,190,816	179,190,816	179,190,432
	197,114,859	179,195,772	179,196,078	277,750,248	215,034,949	224,990,282	213,041,627
	55,156,105	2,584	1,916	169,245,119	124,691,129	112,091,030	122,576,010
	22.5	35.625	43	26.375	54.125	31.5	54.5
	647	670	592	768	569	726	1,354
SIM	179,190,240	179,190,240	179,190,240	179,190,240	179,190,240	179,190,240	179,191,392
	267,796,192	211,052,084	188,155,632	272,773,616	224,989,234	264,808,434	240,918,021
	156,302,408	107,975,093	40,069,875	144,889,418	72,539,486	134,121,766	116,079,492
	41.75	53.75	55.375	40	57.375	47.375	67.5
	563	621	597	522	545	572	1,083
IVM	179,186,784	179,190,816	179,193,696	179,186,784	179,192,544	179,191,280	179,193,696
	255,847,845	186,164,279	179,196,663	244,898,490	246,892,046	275,759,565	287,704,853
	133,522,276	31,162,802	770	130,814,889	121,091,254	168,479,063	164,873,574
	44	57.75	60.125	45.625	69.875	51.375	62.75
	616	632	601	541	601	631	761
SM	179,189,856	179,190,240	179,189,856	179,190,240	179,191,968	179,190,240	179,190,240
	417,122,766	195,125,038	179,196,307	281,730,541	281,732,429	255,850,094	293,678,942
	206,656,468	49,450,943	1,850	175,988,849	195,639,128	158,530,897	184,625,643
	42.625	57.5	60.125	36.125	63	44.875	59.625
	542	695	634	600	562	653	908

method has been chosen by S-plus for critical point computation. Intervals that do not intersect the vertical line identify statistically significant comparisons of pairs of means declared as different.

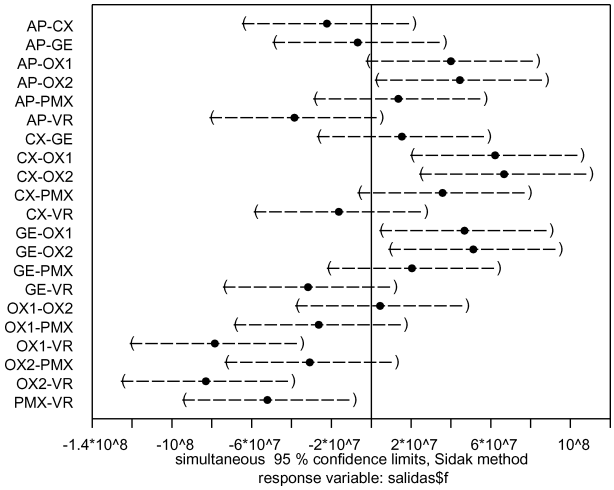

Fig. 7. *Multiple comparisons for crossover operators*

The analysis of mutation operators revealed no statistically significant differences among them, except for pairs DM-SM and ISM-SM both to the left of the vertical line.

The best crossover operators are OX1 and OX2 (without significant differences between them). The worst is AP. VR and GE are also bad but not as poor. We could say that the best mutation operators are ISM and DM, and the worst is SM, although the differences among them are not very significant.

It seems that the minimum f-value is rather hidden, judging by the times it was found in the first experiments. Thus, the second batch of experiments increases ρ up to 0.2, to introduce diversity (see Table 3).

Now the (same) best f-value 179,186,784 was achieved 6 times, and the best value was greater than 716,746,751 fewer times (26 rather than 30). Moreover, all crossover operators but OX1 and CX found 179,186,784 as the best value. All mutation operators but ISM found the same value.

The statistical analysis showed highly significant differences between crossovers with the same ranking of operators, although the worst is VR rather than AP. The only pairs of mutation operators with statistically significant differences were ISM-SIM and ISM-SM, both to the left of the vertical line. The best mutation

Table 3. *Results for the* Jaundice *graph. Second batch*

	GE	OX1	OX2	AP	PMX	CX	VR
DM	179,186,784	179,192,544	179,193,104	179,191,776	179,190,624	179,191,136	179,190,240
	197,115,804	179,196,475	179,196,706	251,868,430	197,115,883	235,940,893	273,767,354
	55,155,864	1,296	848	130,841,952	55,155,389	80,235,524	134,083,303
	22.125	28	35.25	39.75	38.25	34.375	38.5
	795	825	747	605	619	686	1,053
EM	179,196,288	179,196,896	179,186,784	179,193,120	179,193,696	179,189,088	179,190,560
	277,751,333	188,156,734	179,196,106	246,891,688	197,116,264	259,828,426	226,980,483
	196,936,857	41,110,558	2,536	132,289,820	55,155,198	169,245,789	125,987,623
	23	36.75	37.125	29	43.875	28.125	43
	967	932	693	794	780	521	1,490
ISM	179,189,856	179,191,392	179,190,240	179,192,928	179,190,816	179,190,240	179,192,832
	179,195,834	179,196,621	188,155,894	188,156,527	197,115,474	206,075,619	257,841,986
	2,234	1,231	40,069,813	40,069,664	55,154,007	65,648,638	168,259,491
	15.75	27.5	31.125	26.375	31.625	24.25	31.875
	955	834	677	858	643	723	1,364
SIM	179,191,968	179,190,240	179,190,240	179,186,784	179,196,896	179,190,816	179,186,784
	206,074,587	206,074,798	188,154,923	218,020,803	233,948,006	291,686,390	310,601,750
	120,203,009	120,204,405	40,067,308	69,717,615	115,346,759	169,894,952	168,259,491
	35.375	44.25	41.75	34.625	49.125	41.5	49.75
	630	703	610	560	653	616	1,068
IVM	179,186,784	179,190,624	179,190,240	179,190,240	179,190,240	179,191,968	179,186,816
	248,880,898	179,196,090	179,195,673	222,002,611	230,962,537	257,840,840	230,962,622
	87,980,500	1,846	2,226	126,462,361	129,580,130	168,257,759	129,579,674
	40.625	45.75	45	48.875	54.75	44.625	54.625
	719	734	626	610	542	671	1,128
SM	179,189,088	179,190,240	179,190,624	179,191,968	179,186,784	179,190,240	179,190,816
	291,688,061	188,155,120	179,196,071	232,954,368	222,001,408	241,913,920	290,690,693
	186,803,451	40,067,676	1,882	131,292,245	126,464,551	133,526,119	143,191,788
	38.125	45.375	44.5	40.125	48.125	44.75	52.125
	636	663	579	605	487	659	1,012

operator is ISM (followed by DM) and the worst is SIM or SM.

Figure 8 plots the (typical) pattern found with respect to the evolution of the population for an execution of the algorithm. For operators VR and SM, $\rho = 0.2$, $\lambda = 30$, the population is represented via the average of the f-values of all its individuals (Y axis) against the number of generations (X axis).

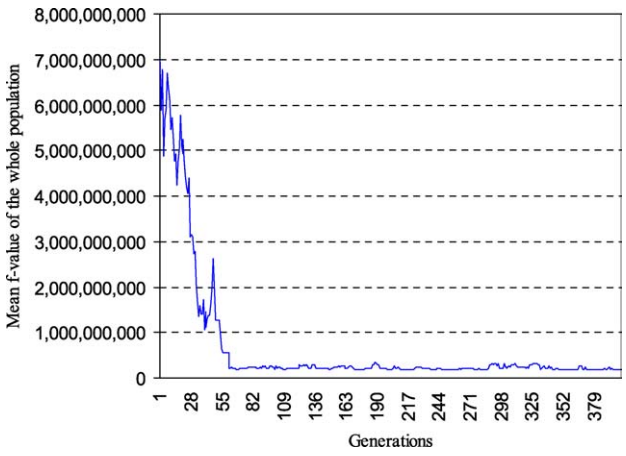

Fig. 8. *Evolution of the population. Parameters VR, SM, $\rho = 0.2$, $\lambda = 30$*

The following experiments vary the population size and the mutation probability to derive their influence. We consider $\lambda = 30, 50, 70$ and $\rho = 0.01, 0.1, 0.2$ for pairs of fixed genetic operators. Table 4 shows the results for two pairs: the first pair is a good pair as deduced above (therefore, with better results), while the other pair was randomly chosen.

The results improve as long as the population size grows and the differences are statistically significant, especially between pairs (30,50) and (30,70). For $\lambda = 30$, the average f-value was 225,486,970; for $\lambda = 50$, it was 190,312,013, and for $\lambda = 70$, it was 187,158,842.

The results also improve for greater mutation probabilities (see also Tables 2 and 3): for $\rho = 0.01$, the average f-value was 231,332,940; for $\rho = 0.1$, it was 205,853,280; and for $\rho = 0.2$, it was 218,118,269. The differences were statistically significant (p-value=0.0012). Note that the actual probability of mutation is lower than ρ for some mutation operators (especially DM and IVM) because of the rejected trials.

As far as the convergence of the algorithm is concerned, VR leads to the slowest algorithms with statistically significant

Table 4. *Results for the* Jaundice *graph. Third batch*

	OX2-ISM			VR-EM		
	$\rho = 0.01$	$\rho = 0.1$	$\rho = 0.20$	$\rho = 0.01$	$\rho = 0.1$	$\rho = 0.20$
$\lambda = 30$	179,192,816	179,190,816	179,186,784	179,190,240	179,191,952	179,191,984
	241,913,880	188,156,358	179,195,834	248,880,666	278,746,572	216,028,511
	167,239,946	40,069,704	2,752	133,713,655	186,435,652	65,960,918
	38.875	35.125	27.5	54.625	50.125	43.875
	614	660	716	1,256	1,218	1,420
$\lambda = 50$	179,192,256	179,189,856	179,186,784	179,186,784	179,186,784	179,191,088
	179,196,346	179,195,647	179,195,854	224,987,213	186,164,083	193,132,938
	1,288	2,407	2,717	112,088,563	31,165,364	42,897,093
	43.875	29.25	24.5	62.125	45.75	42.75
	529	503	582	1,129	1,235	1,376
$\lambda = 70$	179,190,240	179,189,856	179,186,784	179,190,240	179,189,088	179,186,784
	179,195,845	179,194,306	179,194,978	200,100,226	206,072,913	179,194,781
	2,077	2,881	3,120	51,058,156	120,200,149	3,197
	58.375	35.125	26.625	53.375	54.5	46.625
	484	463	494	964	1,122	1,159

differences with respect to the other crossover operators. We could say that the fastest convergence is achieved with PMX, albeit without significant differences and premature convergence. Mutation operators led to not quite conclusive significant differences among the average number of iterations to convergence (p-value $= 0.0251$). The effect of the probability of mutation on the iterations is not statistically significant either. However, as the population size grows, the algorithm reaches its convergence in fewer iterations. The rationale for this may be that since the search space is broader at each iteration, a good individual is reached earlier. Finally, there does not seem to be any relationship between the solution found by the algorithm and the average number of iterations to convergence.

All the executions stopped after the fixed number of iterations without any improvement. Moreover, the average percentage of converged bits does not influence the minimum found (this may be due to the existence of various remote optima). Two conclusions were drawn from these percentages. First, as the probability of mutation increases, this average percentage decreases. It is reasonable since mutation includes diversity in the population, and it will be harder to reach uniformity in the genes. Second, with statistically significant differences, the lowest (highest) percentages of converged genes correspond to crossover operator GE (PMX, VR) and mutation operator ISM (IVM).

Having run our algorithm so many times with different control parameters, we are pretty sure that the minimum found (179,186,784) is the global minimum or close to it. Even so, we ran the algorithm with a large population size of 200 individuals, with the same best minimum.

5.3. *Results versus other techniques*

With our genetic algorithm, the memory requirements of the *Jaundice* diagram has improved three orders of magnitude with respect to the worst case and one order with respect to Kong's heuristic. With the semi-exhaustive method explained in Section 3, we obtained (sometimes) similar results to those of the genetic algorithm, but at the expense of a running time of 4 days. If we solved the influence diagram breaking node deletion ties at random, we would have to solve the diagram a huge number of times without any guarantee that a result similar to that of the genetic algorithm would be achieved.

Moreover, as suggested in the literature, we implemented a local search algorithm to "climb the hills" found by the genetic algorithm and improve the results. Thus, when the genetic algorithm has terminated, we explore all the individuals formed from the best individual by exchanging two genes, whenever it leads to a feasible individual. However, this attempt at refinement has not improved our results. We also tested niching methods (Mahfoud 1995) to maintain a population of diverse individuals and to avoid uniformity, although they have not provided satisfactory results (due to the peculiarities of the problem).

We carried out more experiments to find optimal node deletion sequences in different influence diagrams. Table 5 shows their main features and the results obtained using Kong's heuristic, the semi-exhaustive search being run for a long time, and the genetic algorithm with the pair of operators GE and EM (randomly chosen). As before, 20 executions were carried out for each genetic algorithm, and the best results are shown.

Note that the genetic algorithm always gives the best results and the memory requirements reduction is sometimes sizeable. The semi-exhaustive search turns into a pure random search in large size problems, because of the excessive pruning required.

6. Final remarks

Solving a decision problem is NP-hard (Cooper 1990), and graphical methods developed in the last decade (see e.g., Bielza

Table 5. *Results for different diagrams and methods*

	D_1	D_2	D_3	D_4
Chance nodes	29	38	45	67
Decision nodes	2	2	5	5
Arcs	72	97	123	184
Initial memory positions	7,972	11,150	13,917	19,279
Genetic algorithm	$7.16 * 10^8$	$5.37 * 10^8$	$1.28 * 10^{11}$	$1.02 * 10^{18}$
Semi-exhaustive search	$9.27 * 10^8$	$9.97 * 10^8$	$6.72 * 10^{12}$	$7.78 * 10^{19}$
Kong's heuristic	$5.80 * 10^{10}$	$2.15 * 10^9$	$2.64 * 10^{13}$	$1.51 * 10^{21}$

and Shenoy 1999) try to address complexity by implementing algorithms that take advantage of local computations. Furthermore, some steps of these algorithms lead to more than one possible node deletion sequence, which influences the computational burden, and the choice of the best one is another NP-hard problem.

We have designed a genetic algorithm aimed at alleviating the latter problem. It uses only a qualitative evaluation of the influence diagram and is easy to implement. The selection of adequate individual codification and adequate parameter combinations increases the chance of finding a near-optimal solution in a reasonable number of iterations. Our experiments confirmed this and identified the importance of choosing a good crossover operator rather than a mutation operator, and the relevant effect of the population size and the probability of mutation on the results.

The computational saving is sizeable (depending on the influence diagram characteristics) as compared to other heuristics. Though our parameter study has taken a long time, it does not slow down the usual Decision Support System activity since the study is performed as a preliminary step. The algorithm output will be the input, i.e. the node deletion sequence with optimal storage capacity, to guide the standard evaluation of influence diagrams. Nevertheless, memory requirements of large diagrams may still exceed the capacity of any PC. Fortunately, the influence diagram can be evaluated by combining this genetic algorithm with other techniques (see Bielza et al. 2000).

Likewise, the problem of node deletion sequences is present in other graphical models used in Decision Analysis. For example, valuation networks (Shenoy 1992) have more flexible information constraints and provide more possible deletion sequences, making the problem even worse. Our proposal could also be applied to these other models.

Other lines of future research are directed towards experimenting on an adaptive genetic algorithm (maybe with a variable period of update, Chew, Ong and Lim 2002); on a dynamic variation of some parameters of the algorithm, like ρ (see Fogarty 1989); or on the genetic operators, to change the way parents are selected or the population is reduced; and to propose other crossover and mutation operators. Other modern optimisation techniques, like simulated annealing, might be devised.

Appendix A: Genetic operators

The *partially-mapped* crossover operator (PMX) builds an offspring by choosing a substring from one parent and copying the order and position of as many genes as possible from the other parent. If a gene is already present in the offspring, it is replaced according to the mappings created between the genes from that substring and the analogous one in the other parent, both defined by choosing two random cut points on the parent strings. For example, for parents $p_1 = \langle ABCDEF\rangle$, $p_2 = \langle EDABFC\rangle$, an offspring is $o_1 = \langle EBCDFA\rangle$ if the substring consists of the third and fourth genes. The mappings are $C \leftrightarrow A$, $D \leftrightarrow B$. We first copy the substring and $o_1 = \langle --CD--\rangle$. Then, its first gene would be an E, and the second gene should be a D. But o_1 already has that gene and the second mapping leads to allocate a B as the second gene. Finally, F is copied into the fifth position, and A is the last gene due to the first mapping. By exchanging the parent roles, a second offspring can be built. In the example, it would be $o_2 = \langle CDABEF\rangle$.

The *cycle* crossover operator (CX) builds offspring trying to take each gene and its position from one of their parents. For p_1 and p_2 above, we start taking genes from p_1 and we have $o_1 = \langle A-----\rangle$. Now we look for A in p_2 and it is found in the third position. The third gene from p_1 is C and then we have $o_1 = \langle A-C---\rangle$. This, in turn, implies $o_1 = \langle A-C-EF\rangle$. The following movement would lead to select an A again, completing a cycle. Thus, the remaining genes are taken from p_2 in the same way, to give $o_1 = \langle ADCBEF\rangle$. Similarly, $o_2 = \langle EBADFC\rangle$.

The *order* crossover operator (OX1) copies a substring from one parent, as PMX. Then, it tries to preserve the relative order of genes from the other parent. For p_1 and p_2 above and the same substring than in PMX, we start from $o_1 = \langle --CD--\rangle$. Now, starting from the second cut point (between the fourth and the fifth gene), genes from p_2 are copied in the same order, whenever they are not already present. After reaching the last gene, we continue from the first position. Thus, the sequence to copy from p_2 is $FEAB$, obtaining $o_1 = \langle ABCDFE\rangle$. Similarly, $o_2 = \langle CDABEF\rangle$.

The *order-based* crossover operator (OX2) selects at random several positions in a parent, say in p_2. Next, genes in the selected positions are deleted from p_1. Finally, o_1 is p_1 but its deleted genes are filled in from p_2, following the p_2 order. Thus, in our example, suppose the first, second and fourth positions are selected. The corresponding genes in p_2 are E, D and B, in this order. These genes are located at the second, fourth and fifth positions in p_1. Hence, $o_1 = \langle A-C--F\rangle$. Finally, substring EDB completes the offspring: $o_1 = \langle AECDBF\rangle$. Similarly and using the same selected positions, $o_2 = \langle ADBEFC\rangle$.

The *alternating-position* crossover operator (AP) simply builds an offspring by alternately selecting a gene from each parent, whenever it is not already present in the offspring. In the example, $o_1 = \langle AEBDCF\rangle$ and $o_2 = \langle EADBCF\rangle$.

The *voting* recombination crossover operator (VR) is a $p-$sexual crossover operator ($p \geq 2$), such that a gene is copied into the offspring whenever it occupies the same position in at least t ($t \leq p$) parents. t is the *threshold*. The remaining positions of the offspring are filled randomly with the genes not yet allocated. In our example, also with $p_3 = \langle AEBDCF \rangle$, $p = 3$, $t = 2$, the offspring might be $o_1 = \langle EACBDF \rangle$.

The *displacement* mutation operator (DM) selects a substring at random, removes it from the parent and inserts it in a random place. For example, if substring BCD is chosen in p_1 above, and the random place is after gene E, then $o = \langle AEBCDF \rangle$. The *insertion* mutation operator (ISM) is like DM but with a substring of length 1. Thus, if gene B is randomly chosen in p_1, and the operator randomly inserts it after gene E, then $o = \langle ACDEBF \rangle$.

The *exchange* mutation operator (EM) exchanges two randomly selected genes. If these are the second and fourth genes in our example, it results in $o = \langle ADCBEF \rangle$. The *simple-inversion* mutation operator (SIM) reverses the substring between two randomly selected cut points. For example, if the cut points are selected between gene 2 and 3 and between gene 5 and 6, the result is $o = \langle ABEDCF \rangle$.

The *inversion* mutation operator (IVM) is like DM but the substring is inserted after being reversed. The same example used to illustrate DM would result in $o = \langle AEDCBF \rangle$. The *scramble* mutation operator (SM) selects a substring at random and scrambles the genes in it. Thus, if substring BCD is chosen in p_1 above, a possible result would be $o = \langle ACBDEF \rangle$.

Appendix B: The *Jaundice* graph

We describe here the *Jaundice* influence diagram with notation: C_i is a chance node, D_i is a decision node, v is the value node, $N\,\#\,\{C(N)\}$ denotes node N with domain cardinal $\#$ and predecessors $C(N)$.

D_1 5 $\{C_{19}, C_3, C_7, C_{21}, C_{29}, C_{37}, C_{22}, C_{39}, C_{43}, C_{44}, C_{26}, C_{40}, C_{41}, C_{42}, C_1, C_{33}, C_9, C_6, C_{36}, C_{18}, C_{34}\}$;
D_2 3 $\{C_{20}, C_4, C_8, D_1\}$; C_1 2 $\emptyset$; C_2 2 $\emptyset$; C_3 2 $\{C_7, C_{19}\}$; C_4 2 $\{C_8, C_{20}\}$; C_5 5 $\emptyset$; C_6 3 $\emptyset$;
C_7 3 $\{C_{30}, C_{31}, C_2, C_{38}, C_{16}, C_{17}, C_{27}, C_{28}, C_{35}\}$; C_8 3 $\{C_{30}, C_{31}, C_2, C_{38}, C_{16}, C_{17}, C_{27}, C_{28}, C_{35}\}$;
C_9 3 $\emptyset$; C_{10} 6 $\{C_{11}, C_{12}\}$; C_{11} 6 $\{C_3, C_{32}, C_{19}\}$; C_{12} 6 $\{C_3, C_{32}, C_{19}\}$; C_{13} 5 $\{C_{14}, C_{15}\}$;
C_{14} 5 $\{D_1, C_3, C_{19}\}$; C_{15} 5 $\{D_2, C_4, C_{20}\}$; C_{16} 2 $\emptyset$; C_{17} 2 $\emptyset$; C_{18} 3 $\emptyset$; C_{19} 2 $\emptyset$; C_{20} 2 $\emptyset$; C_{21} 3 $\emptyset$;
C_{22} 2 $\emptyset$; C_{23} 2 $\{C_{24}\}$; C_{24} 2 $\emptyset$; C_{25} 4 $\emptyset$; C_{26} 4 $\emptyset$; C_{27} 2 $\emptyset$; C_{28} 2 $\emptyset$; C_{29} 4 $\{C_{37}, C_3\}$; C_{30} 2 $\emptyset$;
C_{31} 2 $\emptyset$; C_{32} 3 $\{C_{30}, C_{31}, C_2, C_{38}, C_{16}, C_{17}, C_{27}, C_{28}, C_{35}\}$; C_{33} 2 $\emptyset$; C_{34} 2 $\{C_{18}\}$; C_{35} 2 $\emptyset$; C_{36} 4 $\emptyset$;
C_{37} 4 $\emptyset$; C_{38} 2 $\emptyset$; C_{39} 3 $\{C_2\}$; C_{40} 2 $\{C_{30}\}$; C_{41} 2 $\{C_{42}, C_{31}\}$; C_{42} 2 $\{C_{31}\}$; C_{43} 2 $\{C_2\}$; C_{44} 3 $\{C_2\}$;
v $\{C_5, C_6, C_9, C_{10}, C_{13}, C_{36}\}$;

Acknowledgments

We are very grateful for discussions with P. Larrañaga, J. Martín and J.A. Fernández del Pozo. We wish to especially thank E. Córcoles for the software implementation and his valuable help.

References

Alander J.T. 1992. On optimal population size of genetic algorithms. In: Proc. CompEuro 1992, Comput. Sys. and Soft. Engineer., 6th Annual European Computer Conf., pp. 65–70.

Amir E. 2001. Efficient approximation for triangulation of minimum treewidth, In: Koller D. and Breese J. (Eds.), Proc. 17th Conf. on Uncertainty in Artificial Intelligence, San Francisco, CA, Morgan Kaufmann, pp. 7–15.

Arnborg S., Corneil D., and Proskurowski, A. 1987. Complexity of finding embeddings in a k-tree. SIAM Journal Algebraic Discrete Methods 8:277–284.

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

Bielza C. and Shenoy P.P. 1999. A comparison of graphical techniques for asymmetric decision problems. Management Science 45(11): 1552–1569.

Cano A. and Moral S. 1994. Heuristic algorithms for the triangulation of graphs. In: Proc. 5th Int. Conf. on Information Processing and Management of Uncertainty in Knowledge Based Systems (IPMU), Vol. 1, pp. 166–171.

Chew E.P., Ong C.J., and Lim K.H. 2002. Variable period adaptive genetic algorithm. Computers & Industrial Engineering 42(2–4):353–360.

Clemen R.T. 1996. Making Hard Decisions: An Introduction to Decision Analysis, 2nd edition. Duxbury, Belmont, CA.

Cooper G.F. 1988. A method for using belief networks as influence diagrams. In: Proc. Workshop on Uncertainty in Artificial Intelligence, Minneapolis, Minnesota, pp. 55–63.

Cooper G.F. 1990. The computational complexity of probabilistic inference using Bayesian belief networks. Artificial Intelligence 42:393–405.

De Jong K.A. 1975. An analysis of the behavior of a class of genetic adaptive systems. PhD. Diss., University of Michigan.

Ezawa K.J. 1986. Efficient evaluation of influence diagrams. PhD. Diss., Dept. of Engineering-Economic Systems, Stanford University, Stanford, CA.

Fogarty T.C. 1989. Varying the probability of mutation in the genetic algorithm. In: Proc. Third Int. Conf. on Genetic Algorithms, pp. 104–109.

Gómez. M. 2002. *IctNeo*: A decision support system for neonatal jaundice treatment. PhD. Diss., Dept. of Artificial Intelligence, Technical University of Madrid, Spain.

Henrion M. 1989. Some practical issues in constructing belief networks. Uncertainty in Artificial Intelligence 3:161–173.

Holland J. 1975. Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor.

Howard R.A. and Matheson J.E. 1984. Influence diagrams. In: Howard R. and Matheson J. (Eds.), The Principles and Applications of

Decision Analysis, Vol. II, Menlo Park, CA, Strategic Decisions Group, pp. 719–762.

Hsu J.C. 1996. Multiple Comparisons: Theory and Methods. Chapman & Hall, London.

Jensen F.V. 2001. Bayesian Networks and Decision Graphs. Springer, New York.

Jensen F., Jensen F.V., and Dittmer S.L. 1994. From influence diagrams to junction trees. In: Mantaras R.L. and Poole D. (Eds.), Proc. 10th Conf. on Uncertainty in Artificial Intelligence, San Mateo, CA, Morgan Kaufmann, pp. 367–373.

Kirkwood C.W. 1997. Strategic Decision Making. Multiobjective Decision Analysis with Spreadsheets, Duxbury, Belmont, CA.

Kjaerulff U. 1992. Optimal decomposition of probabilistic networks by simulated annealing. Statistics and Computing 2:7–17.

Kong A. 1986. Multivariate belief functions and graphical models. Ph.D. Diss., Dept. of Statistics, Harvard University, Cambridge, MA.

Larrañaga P., Kuijpers C.M.H., Poza M., and Murga R.H. 1997. Decomposing Bayesian networks: triangulation of the moral graph with genetic algorithms. Statistics and Computing 7(1):19–34.

Larrañaga P., Kuijpers C.M.H., Murga R.H., Inza I., and Dizdarevic S. 1999. Genetic algorithms for the travelling salesman problem: A review of representations and operators. Artificial Intelligence Review 13:129–170.

Madsen A.L. and Jensen F.V. 1999. Lazy evaluation of symmetric Bayesian decision problems. In: Laskey K.B. and Prade H. (Eds.), Proc. 15th Conf. on Uncertainty in Artificial Intelligence. Morgan Kaufmann, San Francisco, CA, pp. 382–390.

Mahfoud S.W. 1995. Niching methods for genetic algorithms. Ph.D. Diss., University of Illinois, Champaigne, IL.

Mellouli K. 1987. On the propagation of beliefs in networks using the Dempster-Shafer theory of evidence. PhD. Diss., School of Business, University of Kansas, Lawrence, KS.

Michalewicz Z. 1992. Genetic Algorithms + Data Structures = Evolution Programs. Springer, Berlin.

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

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

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

Shachter R.D., Andersen S.K., and Poh K.L. 1990. Directed reduction algorithms and decomposable graphs. In: Bonissone P., Henrion M., Kanal L., and Lemmer J. (Eds.), Proc. 6th Conf. on Uncertainty in Artificial Intelligence. Elsevier, New York, NJ, pp. 237–244.

Shachter R.D., Andersen S.K., and Szolovits P. 1994. Global conditioning for probabilistic inference in belief networks. In: López de Mántaras R. and Poole D. (Eds.), Proc. 10th Conf. on Uncertainty in Artificial Intelligence. Morgan Kaufmann, San Francisco, CA, pp. 514–522.

Shachter R.D. and Ndilikilikesha P. 1993. Using potential influence diagrams for probabilistic inference and decision making. In: Heckerman D. and Mamdani A. (Eds.), Proc. 9th Conf. on Uncertainty in Artificial Intelligence, pp. 383–390.

Shenoy P.P. 1992. Valuation-based systems for Bayesian decision analysis. Operations Research 40(3):463–484.

Shoikhet K. and Geiger D. 1997. A practical algorithm for finding optimal triangulations. In: Proc. 14th Nat. Conf. on Artificial Intelligence. AAAI Press, Menlo Park, CA, pp. 185–190.

Tatman J.A. and Shachter R.D. 1990. Dynamic programming and influence diagrams. IEEE Transactions on Systems, Man and Cybernetics 20(2):365–379.

Venables W. and Ripley B. 1994. Modern Applied Statistics with S-Plus. Springer, New York.

Zhang N.L. 1993. Studies on hypergraphs I: hyperforests. Discrete Applied Mathematics 42:95–112.

Zhang N.L. 1998. Probabilistic inference in influence diagrams. In: Cooper G.F. and Moral S. (Eds.), Proc. 14th Conf. on Uncertainty in Artificial Intelligence. Morgan Kaufmann, San Francisco, CA, pp. 514–522.