Triangulation of Bayesian networks with recursive estimation of distribution algorithms

Txomin Romero\textsuperscript{a,b,}\ast, Pedro Larrañaga\textsuperscript{c}

\textsuperscript{a} Department of Computer Science and Artificial Intelligence, University of the Basque Country, Spain
\textsuperscript{b} Donostia International Physics Center, Donostia – San Sebastian, Paseo Manuel de Lardizabal 4, 20018 Guipuzcoa, Spain
\textsuperscript{c} Departamento de Inteligencia Artificial, Universidad Politécnica de Madrid, Spain

\section*{Abstract}

Bayesian networks can be used as a model to make inferences in domains with intrinsic uncertainty, that is, to determine the probability distribution of a set of variables given the instantiation of another set. The inference is an NP-hard problem. There are several algorithms to make exact and approximate inference. One of the most popular, and that is also an exact method, is the evidence propagation algorithm of Lauritzen and Spiegelhalter [S.L. Lauritzen, D.J. Spiegelhalter, Local computations with probabilities on graphical structures and their application on expert systems, Journal of the Royal Statistical Society B 50 (2) (1988) 157–224], improved later by Jensen et al. [F.V. Jensen, S.L. Lauritzen, K.G. Olesen, Bayesian updating in causal probabilistic networks by local computations, In Computational Statistics Quarterly 4 (1990) 269–282]. This algorithm needs an ordering of the variables in order to make the triangulation of the moral graph associated with the original Bayesian network structure. The effectiveness of the inference depends on the variable ordering. In this paper, we will use a new paradigm for evolutionary computation, the estimation of distribution algorithms (EDAs), to get the optimal ordering of the variables to obtain the most efficient triangulation. We will also present a new type of evolutionary algorithm, the recursive EDAs (REDAs). We will prove that REDAs improve the behaviour of EDAs in this particular problem, and that their results are competitive with other triangulation techniques.

\section*{1. Introduction}

Let $X = (X_1, X_2, \ldots, X_n)$ be an $n$-dimensional random variable, where $x_i$ is an instantiation of $X_i$. A PGM or Probabilistic Graphical Model ($S, \theta$) is a graphical structure $S = (X, \mathcal{A})$ and a set of local parameters $\theta$, $X$ a set of nodes, represents the system variables and $\mathcal{A}$, a set of arcs, the conditional dependence/independence relationships among the variables of the structure. A Bayesian network (BN) is a PGM where the graphical structure is a directed acyclic graph (DAG), $X_i$ are random discrete variables (called nodes) and the set of parameters $\theta_k = (\theta_{ijk})$, where $k$ goes from 1 to $r_k$, $j$ from 1 to $q_i$, and $i$ from 1 to $n$, represents the local probability distributions over $X$, i.e., $\theta_{ijk}$ is the conditional probability of $X_i$ being in its $k$th value given that the set of its parents variables is in its $j$th-value. Finally, $r_i$ is the number of different values of $i$th variable and $q_i$ represents the number of possible instantiations for the set of parents of $i$th variable.
The BN paradigm can be used as a model to make inferences in domains with intrinsic uncertainty. The factorisation of the joint distribution that a BN represents allows an efficient reasoning inside the model. Introductions and classic textbooks about BN include [10,18] and [30]. As a model to make inferences, we will try to obtain their best triangulation, but we will also use them in the estimation of distribution algorithms themselves.

The most direct way to make inference in a BN is to compute the marginalization over the not instantiated variables. The number of terms involved in the marginalization grows exponentially with the number of variables. Lauritzen and Spigelhalter show [26] that, instead of calculating each joint probability separately in order to add them in a posterior step, we can group the common factors of all of them in order to make the calculation more efficient. For example, for the Asia network (Fig. 1), we have:

\[
P(A, X, D) = \sum_{T, E, L, S} P(A, T, X, E, D, L, B, S)
\]

\[
= P(A) \sum_T P(T | A) \left[ \sum_E P(X | E) \left[ \sum_L P(D | E, B) \left[ \sum_S P(L | S) P(B | S) P(S) \right] \right] \right]
\]

We can rewrite Eq. (1) as

\[
P(A, X, D) = \sum_{T, E, L, S} \psi(A) \psi(T, A) \psi(X, E) \psi(E, T, L) \psi(D, E, B) \psi(L, S) \psi(B, S) \psi(S)
\]

Here, each factor is not a conditional probability, but a potential function. The potential functions are, at first, the conditional probabilities, and their parameters are the variables that are connected in the graph. An useful procedure to discover which variables are related to each potential function is to moralize the graph, that is, to add an arc between the parent nodes of each node (see Fig. 2). Nevertheless, when we sum over \(S\) in expression \(\psi(L, S) \psi(B, S) \psi(S)\) for all the values of \(S\), we obtain a new factor (a new auxiliary potential function) depending on \(L\) and \(B\), and between these nodes there is no arc. This is a problem due to the prior definition of potential function. We can solve this problem with the triangulation of the graph (see [26] for a more extensive explanation). A graph is triangulated if it has no cycles with a length greater than three without a cord. In the Asia network, it is enough to add \(L \rightarrow B\) arc in order to triangulate it (see Fig. 3), but in more complex networks it is not so easy. A good triangulation could make it possible to obtain a solution to some problems related to graphs in polynomial time instead of exponential time [3].
In our experiments, the algorithm that searches for the best triangulation will use the known method of the vertex elimination. Basically, when a vertex is selected to be removed from the graph, we add all the arcs needed to make the subgraph of their adjacent nodes complete. Then, we remove the vertex and all the arcs related to it. The graph that incorporates all the new arcs to its set of original arcs is a triangulated graph. The sequence of removed vertices determines the efficiency of the resulting triangulated graph, given by the size of its cliques.1 There are several criteria to evaluate the quality of a triangulation [34]. One of the most known criterions is the minimum weight, used in the experiments as the evaluation function of the EDAs. This method associates a weight, $w(C_i)$, to each clique:

$$w(C_i) = \sum_{X_j \in C_i} \log_2(r_j)$$  \hspace{1cm} (3)$$

where $r_j$ is the number of possible states of node $X_j$ that belongs to clique $C_i$. So, we can define a weight of the triangulated graph:

$$w(G_t) = \log_2 \sum_{C_i \in C} \prod_{X_j \in C_i} r_j$$  \hspace{1cm} (4)$$

So, our aim is to minimize $w(G_t)$.

Obtaining the best triangulation for a Bayesian network is an NP-hard problem [2,36]. In this work, we will use the univariate marginal distribution algorithm (UMDA) [28] and the mutual information maximization for input clustering (MIMIC) [12], both of them estimation of distribution algorithms (EDAs), to find the best possible triangulation. EDAs are a new meta-heuristic approach belonging to evolutionary computation that are founded in probability theory.

The rest of the paper is organized as follows: in Section 2 we introduce the estimation of distribution algorithms and describe the individual representation. In Section 3 we present the recursive estimation of distribution algorithm (REDA) paradigm. In Section 4 we present the results of the EDA and REDA experiments. In Section 5 we review the early work on the search of the best triangulation, and Section 6 presents the concluding remarks.

2. Estimation of distribution algorithms

EDAs [23,25,27,29] are population-based stochastic heuristics that replace the classic crossover and mutation operators of genetic algorithms by learning the probability distribution from a database and its later simulation. Initially, a random set of individuals is generated. Each individual is evaluated using an objective function. The new population of individuals is sampled from the probability distribution estimated using a subset of individuals selected from the previous generation. Individuals with better function values have a higher chance to be in the subset of individuals selected. The process is iterated until some stop criterion is fulfilled. Thus we can capture all the relationships among the variables explicitly. EDAs have been applied to several problems such as in the graph matching [6], the partial abductive inference in BNs [13] or BNs structure learning [7,31,33].

2.1. General form of an EDA algorithm

We shall establish some definitions before showing the general pseudocode of an EDA:

- $Z = (Z_1, \ldots, Z_m)$ represents the $m$-dimensional variable of $m$ components and $z = (z_1, \ldots, z_m)$ will denote an instantiation of the $m$-dimensional variable (an individual).

---

1 A clique is a subset of nodes which is complete (i.e. there is an edge between every pair of nodes) and maximal.
• \( D_l = \{ z^1, \ldots, z^M \} \) will denote the population of \( M \) individuals in the \( l \)th generation.
• \( D^*_l \) will denote the population that makes up the set of \( N \) individuals selected from \( D_l \).
• \( \rho_l(Z) = \rho_l(Z|D^*_l-1) \) will denote the joint generalised probability distribution of \( Z \) at the \( l \)th generation, given \( D^*_l-1 \). The symbol \( \rho \) is used for both discrete and continuous variables.

The general form of the EDA algorithm proposed in this work for searching for the best triangulation can be seen in Fig. 4.

To estimate \( \rho_l(Z) \) (the main problem) and then sample the new generation, EDAs construct and use a PGM from the pool of selected individuals \( D^*_l-1 \). If the variables that the individuals are made of are discrete, the PGM selected is a BN, and if they are continuous, the EDA constructs a gaussian network (GN) [35].

2.2. Individual representation

The EDAs, throughout their evolution, simulate different individuals of \( m \) components \( \{(z_1, z_2, \ldots, z_m)\} \), where \( z_i \) is an integer or real number that have to be univocally associated with a specific ordering of the \( n \) variables. With continuous EDAs, the representation of the individuals represents no problem, because the components of the individual just need to be ordered and each component instantiation associated with its respective index to obtain a valid ordering. That is, in this case \( m = n \). The disadvantage consists of the high redundancy of this representation because different continuous individuals can generate the same ordering. For instance, the continuous individual \((0.5, 1.6, 0.2, 0.1)\) will be associated with the ordering \((3/4, 2/3)\), but so does the individual \((0.3, 2.0, 0.2, 0.0)\).

In the case of discrete domains, the direct individual representation could be a problem. If we have 4 variables and 4! possible permutations or orderings, we cannot use an individual of 4 components whose variables can take 4 instantiations at the most, because we could obtain, for example, the instantiation \((2 − 3 − 4 − 4)\), which is not a valid ordering. But there is a solution, first used in [33] by Romero et al. that makes the individual-ordering association bijective. We can determine a particular ordering from the \( n! \) possible permutations with the factor decomposition of \( n! \). For example, if we have four variables, the possible 4! orderings can be generated in a systematic way, shown in Table 1. The decomposition of 4! is \( 4 \cdot 3 \cdot 2 \cdot 1 \). If we represent the individual with three components \( Z_1, Z_2, Z_3 \) with \( r_1 = 4 \), \( r_2 = 3 \) and \( r_3 = 2 \), it is easy to obtain a particular order of the systematic list from an individual. But this representation also has some disadvantages: there are components with a very high number of possible states, and this makes the efficiency of the EDA worse. A partial solution, used in this paper, is the prime factor decomposition of \( n! \) (for four variables, the prime factor decomposition \( 2 \cdot 2 \cdot 3 \cdot 2 \cdot 1 \) determines an individual of four components). But if we have a high number of variables, the individual can be very long, and some components can continue having a very high number of possible states. In networks of, for instance, 50 variables, we find that 47 is a prime number: so we will have at least a component with 47 possible states.

---

**EDA**

\[
D_0 \leftarrow \text{Generate } M \text{ individuals randomly (the initial population)} \\
l \leftarrow 1 \\
\text{Repeat} \\
\text{For each individual } z^i \in z^1, z^2, \ldots, z^M \text{ do} \\
\text{Calculate the score } s^i \text{ of the individual} \\
\text{If } s^i \text{ is the best score found then } b^i \leftarrow z^i \text{ End if} \\
\text{End for} \\
D^*_l-1 \leftarrow \text{Select } N \leq M \text{ individuals from } D_{l-1} \text{ using a fixed selection method} \\
\rho_l(Z) \leftarrow \text{Estimate the probability distribution of } Z \text{ given } D^*_l-1 \\
D_l \leftarrow \text{Sample } M \text{ individuals (the new population) from } \rho_l(Z) \\
l \leftarrow l + 1 \\
\text{Until a stopping criterion is verified} \\
\text{Return } b^i
\]

**Fig. 4.** Main scheme of the EDA approach.

---

**Table 1**

<table>
<thead>
<tr>
<th>Possible orderings of 4 nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 – 1234</td>
</tr>
<tr>
<td>2 – 1243</td>
</tr>
<tr>
<td>3 – 1324</td>
</tr>
<tr>
<td>4 – 1342</td>
</tr>
<tr>
<td>5 – 1423</td>
</tr>
<tr>
<td>6 – 1432</td>
</tr>
</tbody>
</table>
2.3. The estimation of the probability distribution

In this work, we will use two examples of EDAs: univariate marginal distribution algorithm (UMDA) and the mutual information maximization for input clustering (MIMIC). Both of them will be used in discrete and continuous domains, and take into account only low-order dependencies. The UMDA algorithm, introduced by Mühlenbein [28], assumes marginal independence between the variables (the components of the individuals), i.e., the model used to estimate $p(z)$ in the case of the discrete domain is the simplest, using the marginal frequencies to get the probability distribution:

$$p_i(z_i) = \frac{\sum_{D_{i-1}} \delta_j(z_i | \pi_i D_{i-1})}{N}$$  \hspace{1cm} (5)

where

$$\delta_j(z_i | \pi_i D_{i-1}) = \begin{cases} 1 & \text{if in the } j - \text{th case of } D_{i-1}, \quad Z_i = z_i, \\ 0 & \text{opposite case}. \end{cases}$$  \hspace{1cm} (6)

In the case of the continuous domain, the factorization of the joint density function is composed of normal univariate distributions:

$$f_\theta(x, \mu, \Sigma) = \prod_{i=1}^{n} f_\theta(x_i; \mu_i, \sigma^2_i) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma_i}} e^{-\frac{(x_i-\mu_i)^2}{2\sigma^2_i}}$$  \hspace{1cm} (7)

and the estimation of the parameters is performed using their maximum likelihood estimates:

$$\hat{\theta}_i = (\hat{\mu}_i, \hat{\sigma}_i) = \left( \frac{1}{N} \sum_{i=1}^{N} x_{ir}, \sqrt{\frac{1}{N} \sum_{i=1}^{N} \left(x_{ir} - \frac{1}{N} \sum_{i=1}^{N} x_{ir}\right)^2} \right)$$

MIMIC or Mutual Information Maximization for Input Clustering, proposed by De Bonet et al. [12], takes dependencies into account, but only between pairs of variables. In the continuous domain, GNs (Gaussian Networks) are used. The main idea is to search for the best permutation among the variables, in every generation, in order to obtain the closest probability distribution $p^\pi(z)$ to the empirical distribution of $D_i$, using the Kullback–Leibler divergence:

$$H^\pi_\theta(x) = H_i(X_{i_i}) + \sum_{j=1}^{n-1} H_i(X_{j_i} | X_{i_{j-1}})$$  \hspace{1cm} (8)

where $H(X)$ is the Shannon entropy of $X$ variable and $H(X|Y)$ is the conditional entropy of $X$ given $Y$. We try to search for the permutation $\pi^*$ that minimizes $H^\pi_\theta(x)$. The test of all the $n!$ permutations can be impossible. So, MIMIC searches for the best permutation selecting as $X_0$ variable the one that has the minor estimated entropy (the estimation is calculated using $D_i$) and, in each step, selecting the variable whose conditional entropy related to the selected variable in the previous step is the minor one. For an explanation of the continuous case, see [25].

2.4. Sampling the new generation

In both the discrete and continuous domain, we will use the probabilistic logic sampling (PLS) method of Henrion [17] to sample the new generation. In this method, the instance of a variable is generated after all its parents have already been sampled, using the distribution $p(z_i | \pi_i)$. The variables must be ordered in ancestral ordering before the simulation is performed. In the case of GNs, normal density simulation is carried out [25].

3. Recursive estimation of distribution algorithms (REDAs)

The main idea of the REDAs is to divide, in some part of the execution of the EDA algorithm, the set of nodes of the network (for which we want to obtain an optimal ordering) in two subsets. Then the new algorithm calls a REDA recursively for each subset, trying to obtain in each call an optimal ordering for it. So, in the first call, we will fix one subset of nodes and evolve recursively over the other, and in the second call we will do the opposite.

As we can see in [5], the most expensive steps of an EDA are the learning of the structure, the simulation of the new population, and, sometimes, the evaluation of the individuals if our evaluation function is very complex. All these problems grow with the number of components of the individual. If we focalize the execution of the EDA in a subset of nodes instead of the whole one, we could obtain good general results in a shorter time, because the associated individual will be smaller too. So we can increase the amount of generations without increasing the execution time.

In order to present the recursive algorithm of the REDAs, we have to define some concepts:

- $z^\pi$ will be a cache of orderings of $m$ nodes. This cache is composed of the $M$ best orderings found among the execution of the REDA (where $M$ is the population size). The cache is ordered using the evaluation of the orderings. As we will see later, we will use this cache in the evaluation of the individuals and in the initialization of the population.
• \( G \) is the set of \( m \) nodes of the Bayesian network we are trying to triangulate.

• \( G_r \subseteq G \) is the subset of size \( m_r \) of \( G \) composed of the nodes that are not fixed in the recursion level number \( r \).\(^2\) The recursion level is the number of consecutive recursive calls. That is, at first, in the main code, the level is 0, and in the first recursive call the level will be 1, and so on.

• \( g_m \) is the number of components of the individual that represents an ordering of \( m_r \) nodes. Notice that, whereas in the continuous domain \( m_r = g_m \), this is not true in the discrete domain.

• \( D_l \) is a population of individuals of size \( g_m \leq g_m \) in the generation \( l \), where \( r \) is the recursive level. We could use the \( D_l \) notation to incorporate the level of the recursion not only to the population, but also to the other variables described here. But it is not necessary to show this information in the description of the pseudocodes because the variables are local ones, and we prefer to simplify the notation.

• \( c_i^m \) is the \( i \)th ordering of \( m \) nodes that belongs to \( c^m \). The \( c_i^m \) ordering will always be the best one found at each moment.

• \( c_i^b \) is an ordering of \( m_r \) nodes extracted from a \( c_i^m \), and is composed of the nodes of \( c_i^m \) that are not fixed in the recursive level \( r \), maintaining their relative ordering.

• \( i^a \) is an ordering of \( m \) nodes. \( c^a \) is an \( i^a \) that belongs to the cache.

• \( i^c_{\text{ord}} \) is the individual of \( g_m \) components associated with the ordering \( i^c \).

• \( i^c \) is an ordering of \( m_r \leq m \) nodes. \( c^c \) is an \( i^c \) extracted from a \( c^a \).

• \( i^c_{\text{ind}} \) is the individual of \( g_m \) components associated with the ordering \( i^c \).

• \( i \) is an ordering.

• \( \text{Node}_i(j) \) is the \( j \)th node of the \( i \) ordering.

• \( i^r \) is an ordering of \( m \) nodes created using an \( i^r \) ordering. We could construct it placing the fixed nodes of the recursive step \( r \) in their respective place and then placing the nodes that appear in the \( i^r \) ordering in the resulting gaps, maintaining their relative subordering.

• \( \text{Evaluate}^a(i^a) \) if a function that returns the metric of an ordering of size \( m \).

• \( \text{Evaluate}^c(i^c, i^r) \) is a procedure that constructs the \( i^c \) ordering using the \( i^c \) one, and returns \( \text{Evaluate}^c(i^c) \), and the \( i^c \) ordering itself.

\(^2\) They are the nodes that will appear in the \( i^r \) orderings and participate in the evolution of the recursive call of the EDA.
We can only calculate the weight of an ordering if it has \( m \) nodes: so, we have to transform the \( i_b \) into an \( i_c \) before calculating its metric. But, instead of placing the fixed components of the recursive step \( r \) in the same order all the time, we can use their relative ordering that appears in the best individual of the \( fa \) cache. We can see an example of this construction in Fig. 5.

Figs. 6–8 show the pseudocode used. As we can see, the size of the subset of nodes that are not fixed determines the general case and the base case of the recursive algorithm. It must be taken into account that, after and before the recursive calls, we use a standard EDA (BaseEda) to evolve over the whole subset of nodes.

If in each recursive call we use a random initialization of the population, then we will not be able to take advantage of the previous good individuals (of the other recursive levels). But we can initialize the population by extracting the relative ordering of the non fixed nodes in the actual recursive level from all the orderings of the now old \( fa \) cache (not exactly with the relative ordering but with the individual associated with it). In order to not to determine the evolution of REDA with the first random population too much, we can maintain the five best individuals of the old \( fa \) in the new cache and fill the rest of the \( M/C_0 \) individuals with the best \( i_c \) individuals proposed by the REDA in the actual recursive level.

4. The experiments

As we have seen in Section 1, we will use the known method of the vertex elimination in our experiments.

4.1. Networks used in the experiments

The networks used to compare EDAs and REDAs are named sparse and dense [21]. Sparse network is computer-generated and has 50 variables and 100 arcs. Dense, also computer-generated, has 50 variables and a greater number of arcs: 359.

---

3 Remember that, in the case of the discrete domain, the individual cannot be obtained directly from the ordering, that is, in this case the relative ordering must be transformed into an individual before adding it to the population.
Strictly speaking, sparse and dense are not Bayesian networks, because the structure in both of them is undirected acyclic graphs. This simplifies the process of the triangulation, because it is not necessary to moralize the network.

In addition, we have used two more complex networks named Medianus I and Medianus II [21] in order to compare the results of REDAs with other triangulation techniques in the case of real world networks. Medianus I is a Bayesian network describing relations between disorders, pathophysiological features and measurements for the human median nerve, and originates from the development of the MUNIN4 system [1]. It has 43 nodes and 66 arcs. Medianus II is a modified version of Medianus I with 56 nodes and 161 arcs. In both networks, node values are between 3 and 21.

### 4.2. Choices about the parameters of EDAs and REDAs

- We always use a population size of $M = 100$ and $M = 500$ individuals for sparse and dense networks. We have executed the experiments for each EDA 50 times (continuous UMDA or UMDA, discrete UMDA or UMDA, continuous MIMIC or MIMIC, and discrete MIMIC or MIMIC).
- In EDAs we print a sample (the best individual) every $M$ different individuals evaluated. In REDAs, every $6 \cdot M$ different evaluations, because we can take advantage of the greater speed of the algorithm when the size of the subindividual is small.
- The EDA stops when we have printed 100 samples: there is not a convergence criterion because it does not guarantee the termination of the algorithm. The REDA, taking advantage of its execution speed, is executed five consecutive times over the successive caches. In each execution we print 50 samples. So, we stop when we have printed a total of 250 samples. Take into account that the printing of the 50 samples are divided between all recursive calls, making the execution of BaseEda faster, because the number of iterations is very small and, in addition, the size of the individual is smaller in the majority of the recursive calls.
- In EDAs, we select the first $N = M/10$ best individuals from the actual population to generate the Bayesian network of EDA and then simulate the next generation. After a previous set of experiments, we selected the values that appear in Table 2 for REDAs.
- In EDAs, we only keep the five best individuals from one generation to the next, sampling $M – 5$ individuals at each step or generation. In REDAs, we sample the $M$ individuals.

### 4.3. Results with EDAs for sparse and dense networks

In Table 3 we can see the average results of 50 executions for EDAs, in continuous (UMDA, MIMIC,) and discrete (UMDA, MIMIC,) domains, for sparse network. In Table 4 we have the same information for dense network.

<table>
<thead>
<tr>
<th></th>
<th>MIMIC$_c$</th>
<th>UMDA$_c$</th>
<th>Best MIMIC$_c$</th>
<th>Best UMDA$_c$</th>
</tr>
</thead>
<tbody>
<tr>
<td>M = 100</td>
<td>26.79 ± 2.35</td>
<td>26.80 ± 2.34</td>
<td>23.48</td>
<td>23.48</td>
</tr>
<tr>
<td>M = 500</td>
<td>27.55 ± 0.23</td>
<td>27.43 ± 0.33</td>
<td>26.65</td>
<td>26.26</td>
</tr>
<tr>
<td>M = 100</td>
<td>27.61 ± 1.61</td>
<td>26.55 ± 0.90</td>
<td>24.20</td>
<td>24.38</td>
</tr>
<tr>
<td>M = 500</td>
<td>27.72 ± 0.63</td>
<td>23.35 ± 0.09</td>
<td>25.73</td>
<td>22.66</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>MIMIC$_d$</th>
<th>UMDA$_d$</th>
<th>Best MIMIC$_d$</th>
<th>Best UMDA$_d$</th>
</tr>
</thead>
<tbody>
<tr>
<td>M = 100</td>
<td>58.31 ± 2.11</td>
<td>58.03 ± 2.08</td>
<td>55.04</td>
<td>54.01</td>
</tr>
<tr>
<td>M = 500</td>
<td>56.65 ± 2.53</td>
<td>56.50 ± 1.80</td>
<td>53.04</td>
<td>53.22</td>
</tr>
<tr>
<td>M = 100</td>
<td>55.16 ± 2.80</td>
<td>56.10 ± 2.72</td>
<td>51.87</td>
<td>52.75</td>
</tr>
<tr>
<td>M = 500</td>
<td>54.27 ± 2.79</td>
<td>53.78 ± 1.86</td>
<td>51.85</td>
<td>51.81</td>
</tr>
</tbody>
</table>

Knowledge based system for diagnosing diseases and malfunctions in the human neuromuscular system.
case of dense network, 55.16 is the best average result (without taking into account the case of UMDA$_d$), that is far from the worst result for the simulated annealing in [22] (54.25) or the 428 possible parameter combinations with an average weight lower than 52.88 for the genetic algorithms in [24]. However, the results of UMDA$_d$ are competitive with simulated annealing and genetic algorithms.

We can see a comparison between the different EDAs in Figs. 9 and 10 for $M = 500$ using a box-and-whisker plot. With sparse network and $M = 500$, UMDA$_d$ outperforms all the other discrete algorithms with a $p$-value near zero in the Wilcoxon rank-sum test. There are no statistical significant differences between the rest of the algorithms, except in the case of MIMIC$_d$ and UMDA$_c$ ($p$-value of 0.04). With dense, the result of comparisons is the same: UMDA$_d$ outperforms the rest of the algorithms with $p$-values $< 10^{-9}$, and MIMIC$_d$ outperforms MIMIC$_c$ and UMDA$_c$ with a $p$-value near zero.

4.4. Results with REDAs for sparse and dense networks

The behaviour of REDAs is more competitive than those of EDAs. In Tables 5 and 6 we can see the average results of 50 executions for REDAs, in continuous and discrete domains respectively, for sparse and dense networks.

We can see a comparison between the different REDAs in Figs. 11 and 12 for $M = 500$ using a box-and-whisker plot. REDAs undoubtedly outperforms EDAs. In the case of UMDA$_d$, REDA outperforms the EDA’s one with a $p$-value of 0.0003 in the case

Table 5
Average results of 50 executions for sparse and REDAs

<table>
<thead>
<tr>
<th></th>
<th>MIMIC$_c$</th>
<th>UMDA$_c$</th>
<th>Best MIMIC$_c$</th>
<th>Best UMDA$_c$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M = 100$</td>
<td>23.28 ± 0.43</td>
<td>23.42 ± 0.35</td>
<td>22.67</td>
<td>22.63</td>
</tr>
<tr>
<td>$M = 500$</td>
<td>23.01 ± 0.09</td>
<td>23.20 ± 0.24</td>
<td>22.66</td>
<td>22.66</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>MIMIC$_d$</th>
<th>UMDA$_d$</th>
<th>Best MIMIC$_d$</th>
<th>Best UMDA$_d$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M = 100$</td>
<td>23.28 ± 0.17</td>
<td>23.23 ± 0.20</td>
<td>22.63</td>
<td>22.66</td>
</tr>
<tr>
<td>$M = 500$</td>
<td>23.30 ± 0.25</td>
<td>23.15 ± 0.22</td>
<td>22.61</td>
<td>22.64</td>
</tr>
</tbody>
</table>

Table 6
Average results of 50 executions for dense and REDAs

<table>
<thead>
<tr>
<th></th>
<th>MIMIC$_c$</th>
<th>UMDA$_c$</th>
<th>Best MIMIC$_c$</th>
<th>Best UMDA$_c$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M = 100$</td>
<td>53.09 ± 1.13</td>
<td>53.04 ± 1.05</td>
<td>51.15</td>
<td>51.15</td>
</tr>
<tr>
<td>$M = 500$</td>
<td>52.73 ± 1.58</td>
<td>52.70 ± 1.44</td>
<td>50.88</td>
<td>50.88</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>MIMIC$_d$</th>
<th>UMDA$_d$</th>
<th>Best MIMIC$_d$</th>
<th>Best UMDA$_d$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M = 100$</td>
<td>53.04 ± 0.85</td>
<td>52.88 ± 1.19</td>
<td>50.88</td>
<td>50.99</td>
</tr>
<tr>
<td>$M = 500$</td>
<td>52.75 ± 0.73</td>
<td>52.62 ± 1.31</td>
<td>50.89</td>
<td>50.88</td>
</tr>
</tbody>
</table>
of sparse network, and there are no statistical significant differences with dense. In addition, independent of population size, kind of the algorithm (UMDA or MIMIC) or characteristics of the domain (continuous or discrete), REDAs seem to be more stable in their results. For sparse network and $M = 500$, there are only statistically significant differences between MIMIC$_c$ and MIMIC$_d$ ($p$-value = 0.00077) and MIMIC$_c$ and UMDA$_d$ ($p$-value = 0.033). But in the case of dense network, there are no significant differences between any of the algorithms. If we compare the differences between population sizes ($M = 100$ and $M = 500$), the behaviour is similar: there are significant statistical differences only in the case of MIMIC$_c$ ($p$-value 0.01). In other experiments, not presented in this paper, we have stated that standard EDAs are very sensible to the size of the set of selected individuals used to generate the Bayesian network, but we have not detected the same tendency in REDAs. In Fig. 13 we can see a comparison for a population of 100 individuals, sparse network and UMDA$_c$ algorithm between REDAs and EDAs of the average results of 10 executions, with several different sizes of the set. The results shows more stability in the case of REDAs.

![Fig. 11. Comparison for the sparse network, REDAs and $M = 500$.](image1)

![Fig. 12. Comparison for the dense network, REDAs and $M = 500$.](image2)

![Fig. 13. Comparison for the sparse network and $M = 100$ of the average results of 10 executions of UMDA$_c$ for REDAs and EDAs with different sizes of the set of selected individuals. $n$ REDA means $n$ consecutive executions of the REDA over the successive caches. In the experiments of this paper, we have used $n = 5$.](image3)
4.5. Comparison between REDAs and other triangulation methods with real world networks

We have used the discrete and continuous version of the UMDA algorithm (in the BaseEda call) to compare the results of the associated REDA with other triangulation techniques. We have applied the REDAs to the Medianus I and II networks. As we can state in the Tables 7–9, the REDAs are competitive with other triangulation techniques, not only in the case of the average evaluation, but also in the case of the best one.

5. Previous work in triangulation

The method related to minimize \( w(G') \) is not the only one proposed in the early work to best triangulate a Bayesian network. Robertson and Seymour [32] and Becker and Geiger [4] do not try to minimize the weight of the triangulated graph, but want to obtain the one that has a clique number less than \( k + 1 \), given a \( k \), being that all the nodes of the graph are binary (the treewidth criterion). The clique number is the size of the bigger clique of the graph. Their algorithms assure an approximate solution to the best one by a constant.

Won et al. [37] have discovered a bijective relationship between a hypergraph and a relational database scheme. There is also a biunivocal relation between the set of triangulated graphs and the set of acyclic hypergraphs. So, the authors are able to use well known properties and methods associated with the database schemes in order to obtain the associated triangulated graph.

Bodlaender et al. [8] try to preprocess the original graph, making it smaller but without losing the possibility of extrapolating the results for the preprocessed graph to the original one. The reduction is managed by rules such as; if \( V \) is a simplicial vertex with a degree \( d \leq 0 \), then we can remove \( V \) and update the found maximum degree, where a vertex is simplicial if its neighbours conform a clique (and its degree is the size of this clique). The algorithm applies these rules until it can not go any further. But this algorithm does not take the different values of each vertex into account.

Darwiche and Hopkins [11] work on the balance of a \( d \)-tree, because it is easy to obtain a junction tree from a \( d \)-tree or an ordering for the removal of the nodes. A \( d \)-tree for a graph \( G \) is a complete binary tree where the leaves are the families of the graph and the nodes that are not leaves serve to divide the families of \( G \) into two branches.

In other strategies related to the direct vertex removing, Kjaerulff proposes three heuristics in [21]:

| Table 7 | Average results of 50 executions for Medianus I, Medianus II and REDAs |
|---------|------------------|------------------|------------------|------------------|
|         | UMDA\(_c\)       | UMDA\(_d\)       | Best UMDA\(_c\)  | Best UMDA\(_d\)  |
| Medianus I and \( M = 100 \) | 23.36 ± 0.23 | 23.35 ± 0.21 | 22.63 | 22.63 |
| Medianus I and \( M = 500 \) | 23.30 ± 0.19 | 23.22 ± 0.10 | 22.63 | 22.63 |
| Medianus II and \( M = 100 \) | 21.78 ± 0.66 | 21.32 ± 0.63 | 20.54 | 20.46 |
| Medianus II and \( M = 500 \) | 21.41 ± 0.45 | 21.16 ± 0.40 | 20.45 | 20.45 |

| Table 8 | Results of several triangulation techniques for Medianus I |
|---------|--------------|--------------|
|         | Average      | Best         |
| Random elimination | 41.66 | 27.75 |
| Lexicographic search | 30.29 | 23.44 |
| Maximum cardinality | 29.00 | 23.63 |
| Minimum size heuristic | 23.27 | 22.71 |
| Minimum weight heuristic | 22.99 | 22.94 |
| Minimum fill heuristic | 23.14 | 22.71 |
| Annealing (best control scheme) | 22.71 | 22.63 |

| Table 9 | Results of several triangulation techniques for Medianus II |
|---------|--------------|--------------|
|         | Average      | Best         |
| Random elimination | 46.35 | 29.28 |
| Lexicographic search | 30.59 | 22.28 |
| Maximum cardinality | 29.17 | 22.38 |
| Minimum size heuristic | 22.19 | 20.50 |
| Minimum weight heuristic | 20.65 | 20.65 |
| Minimum fill heuristic | 21.78 | 20.57 |
| Annealing (best control scheme) | 20.54 | 20.49 |
• H1 – Removes, in each step, the variable that produces the smaller maximal clique 
  Mindgree.
• H2 – Removes the variable that minimizes the number of added arcs.
• H3 – Removes the variable that minimizes the whole weight of the triangulated graph.

None of the three algorithms can assure the three minimizations at the same time. H1 is the fastest one, but it can return a bad ordering if the graph is already triangulated [34]. Cano et al. [9] use modifications of H1 also taking into account the number of values of the variable to remove, or the increment of the greatest clique size. Gamez et al. [16] propose an ant colony optimization algorithm (ACO), a multi-agent bio-inspired algorithm. This method achieves good and faster solutions than other combinatorial optimization methods.

Kjaerulf tries in [21] to eliminate the unnecessary arcs added by a previous triangulation done by means of a heuristic, and uses an ordering of the arcs. In [22] the same author proposes a search for an ordering of the nodes using simulated annealing [20], and obtains good results but sacrifices execution speed. Wen also uses the simulated annealing in [36], but with two different methods of perturbation and more restrictive state transitions. Here, we also found a sacrifice of the speed for the optimization of the resulting ordering.

Larrañaga et al. [24] use the genetic algorithms to search in the space of orderings. They use a complete set of crossover and mutation operators. Normally, the winning combination contains the most expensive operators (in the sense of the execution speed).

Flores and Gámez [14] triangulate the network not over the complete graph but over subsets of it (Maximal Prime Subgraphs or MPS), working over a smaller space, as REDAs do. This technique is useful only if the network can be decomposed into more than one MPS. The same authors show in [15] a review related to BNs triangulation.

6. Concluding remarks

The EDA parametrization is easier than other evolutionary methods, such as Genetic Algorithms, but continues to be a difficult task that determines the results of a standard EDA. It is difficult to select a priori a specific EDA algorithm (UMDA or MIMIC in our case) for a specific problem (the triangulation of Bayesian networks in this paper). We can say the same thing about the type of domain of EDA individuals (continuous or discrete). But with REDAs we can obtain good results, improve the speed (because in the majority of the recursive calls, the size of the individual is smaller than in standard EDA) and make the execution of the EDA relative independent of the parametrization, the specific EDA algorithm and the type of the domain of the components.

Regarding future work, we plan to study the stability of REDAs with other problems, like the learning of a Bayesian structure from a database of cases or the partial abductive inference, and make them parallel in order to increase the speed of their execution.

Acknowledgements

This work has been partially supported by the following grants: TIN2005-03824 and Consolider Ingenio 2010-CSD2007-00018 projects (Spanish Ministry of Education and Science) and COMBIOMED network in computational biomedicine (Carlos III Health Institute).

References