Bayesian Network Structure learning using Cooperative Coevolution

Olivier Barrière, Evelyne Lutton, Pierre-Henri Wuillemin

Bayesian Network Structure Learning using Cooperative Coevolution

Olivier Barrière
INRIA Saclay - Ile-de-France
Parc Orsay Université, 4, rue Jacques Monod
91893 Orsay Cedex, France
olivier.barriere@inria.fr

Evelyne Lutton
INRIA Saclay - Ile-de-France
Parc Orsay Université, 4, rue Jacques Monod
91893 Orsay Cedex, France
evelyne.lutton@inria.fr

Pierre-Henri Wuillemin
LIP6 - Pôle IA
8, rue du capitaine Scott
75015 Paris, France
pierre-henri.wuillemin@lip6.fr

ABSTRACT

We propose a cooperative-coevolution – Parisian trend – algorithm, IMPEA (Independence Model based Parisian EA), to the problem of Bayesian networks structure estimation. It is based on an intermediate stage which consists of evaluating an independence model of the data to be modelled. The Parisian cooperative coevolution is particularly well suited to the structure of this intermediate problem, and allows to represent an independence model with help of a whole population, each individual being an independence statement, i.e. a component of the independence model. Once an independence model is estimated, a Bayesian network can be built. This two level resolution of the complex problem of Bayesian network structure estimation has the major advantage to avoid the difficult problem of direct acyclic graph representation within an evolutionary algorithm, which causes many troubles related to constraints handling and slows down algorithms. Comparative results with a deterministic algorithm, PC, on two test cases (including the Insurance BN benchmark), prove the efficiency of IMPEA, which provides better results than PC in a comparable computation time, and which is able to tackle more complex issues than PC.

Categories and Subject Descriptors

G.1.6 [**Mathematics of Computing**]: Numerical analysis—*Optimization*; I.2.6 [**Computing Methodologies**]: Artificial intelligence—*Learning*

General Terms

Algorithms

Keywords

Bayesian Network Structure learning, Cooperative Coevolution, Independence Model, Parisian Evolution

1. INTRODUCTION

Bayesian networks structure learning is a NP-Hard problem [5], which has applications in many domains, as soon as we try to analyse a large set of samples in terms of statistical dependence or causal relationship. In agrifood industries for example, the analysis of experimental data using Bayesian networks helps to gather technical expert knowledge and know-how on complex processes, like cheese ripening [1].

Evolutionary techniques have been used to solve the Bayesian network structure learning problem, and were facing crucial problems like: Bayesian network representation (an individual being a whole structure like in [14], or a sub-structures like in [18]), and fitness function choice [18]. Various strategies have been used, based on evolutionary programming [24], immune algorithms [13], multi-objective strategies [22], lamarkian evolution [25] or hybrid evolution [26].

In this work, we propose using an alternate representation, independence models, in order to solve the Bayesian network structure learning in two steps. Independence model learning is still a combinatorial problem, but it is easier to embed within an evolutionary algorithm. Furthermore, it is suited to a cooperative coevolution scheme, which allows to obtain computationally efficient algorithms.

The paper is organised as follows, some notions related to Bayesian networks and independence models are recalled in section 1.2, and a brief overview of Parisian cooperative coevolution is given in section 1.3. Section 2 sketches the components of the evolutionary algorithm that is used to solve the first step of IMPEA. The second step of the algorithm is detailed in section 3. Experiments are described and analysed in section 4, before concluding in section 5.

1.1 Background on probability concepts

The joint distribution of X and Y is the distribution of the intersection of the events X and Y, that is, of both events X and Y occurring together. The *joint probability* of X and Y is written $P(X, Y)$. The *conditional probability* is the probability of some event X, given the occurrence of some other event Y and is written $P(X|Y)$.

To say that two events are *statistically independent* intuitively means that the occurrence of one event makes it neither more nor less probable that the other occurs. If two events X and Y are independent, then the conditional probability of X given Y is the same as the unconditional probability of X, that is $P(X) = P(X|Y)$.

Two events X and Y are said to be *conditionally independent* given a third event Z if knowing Z gives no more information about X once one knows Y. Specifically, $P(X|Z) = P(X|Y, Z)$. In such a case we say that X and Y are conditionally independent given Z and write it $X \perp\!\!\!\perp Y \mid Z$.

1.2 Bayesian networks

A Bayesian Network (BN) is a "graph-based model of a joint multivariate probability distribution that captures properties of conditional independence between variables" [11]. On the one hand, it is a graphical representation of the joint probability distribution and on the other hand, it encodes independences between variables. For example, a Bayesian network could represent the probabilistic relationships between diseases and symptoms. Given symptoms, the network can be used to compute the probabilities of the presence of various diseases.

Formally, a Bayesian network is a directed acyclic graph (DAG) whose nodes represent variables, and whose missing edges encode conditional independences between the variables. This graph, represented in figure 1, is called the structure of the network and the nodes containing probabilistic information are called the parameters of the network.

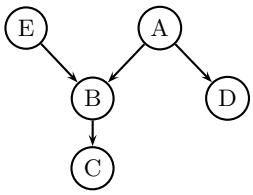

Figure 1: Directed Acyclic Graph

The set of parent nodes of a node X_i is denoted by $pa(X_i)$. In a Bayesian network, the joint probability distribution of the node values can be written as the product of the local probability distribution of each node and its parents:

$$P(X_1, X_2, \ldots, X_n) \quad = \quad \prod_{i=1}^{n} P(X_i | pa(X_i))$$

1.2.1 Parameter and structure learning

The Bayesian network learning problem has two branches: The *parameter* learning problem (i.e., to find the probability tables of each node) and the *structure* learning problem (i.e., to find the graph of the network), following the decomposition of the two constitutive parts of a Bayesian network: its structure and its parameters.

There already exists algorithms specially suited to the parameter learning problem, like expectation-maximization (EM) that is used for finding maximum likelihood estimates of parameters.

Learning the structure is a more challenging problem because the number of possible Bayesian network structures (NS) grows superexponentially with the number of nodes [21]. For example, $NS(5) = 29281$ and $NS(10) = 4.2 \times 10^{18}$. A direct approach is intractable for more than 7 or 8 nodes, it is thus necessary to use heuristics in the search space.

In a comparative study by O. Francois and P. Leray [9], authors identified some currently used structure learning algorithms, namely *PC* [23] or *IC/IC** [20] (causality search using statistical tests to evaluate conditional independence), *BN Power Constructor (BNPC)* [3] (also uses conditional independence tests) and other methods based on scoring criterion, such as *Minimal weight spanning tree (MWST)* [6] (intelligent weighting of the edges and application of the well-known algorithms for the problem of the minimal weight tree), *K2* [7] (maximisation of $P(G|D)$ using Bayes and a topological order on the nodes), *Greedy search* [4] (finding the best neighbour and iterate) or *SEM* [10] (extension of the EM meta-algorithm to the structure learning problem). However that may be, the problem of learning an optimal Bayesian network from a given dataset is NP-hard [5].

1.2.2 The PC algorithm

PC, the reference causal discovery algorithm, was introduced by Sprites, Glymour and Scheines in 1993 [23]. A similar algorithm, IC, was proposed simultaneously by Peal and Verma [20]. It is based on chi-square tests to evaluate the conditional independence between two nodes. It is then possible to rebuild the structure of the network from the set of discovered conditional independences. PC algorithm actually starts from a fully connected network and every time a conditional independence is detected, the corresponding edge is removed. Here are the first detailed steps of this algorithm:

- Step 0: Start with a complete undirected graph G
- Step 1-0: Test all conditional independences of order 0 (i.e $x \perp\!\!\!\perp y \mid \emptyset$ where x and y are two distinct nodes of G). If $x \perp\!\!\!\perp y$ then remove the edge $x - y$.
- Step 1-1: Test all conditional independences of order 1 (i.e $x \perp\!\!\!\perp y \mid z$ where x, y, and z are three distinct nodes of G). If $x \perp\!\!\!\perp y \mid z$ then remove the edge $x - y$.
- ...
- Step 1-k: Test all conditional independences of order k (i.e $x \perp\!\!\!\perp y \mid \{z_1, z_2, \ldots, z_k\}$ where $x, y, z_1, z_2, \ldots, z_k$ are $k+2$ distinct nodes of G). If $x \perp\!\!\!\perp y \mid \{z_1, z_2, \ldots, z_k\}$ then remove the edge between $x - y$.
- Next steps take particular care to detect *V-structures* and recursively detect orientation of the remaining edges.

The complexity of this algorithm depends on N, the size of the network and k, the upper bound on the fan-in, and is equal to $O(N^k)$. In practice, this implies that the value of k must remain very small when dealing with big networks.

1.2.3 Independence models

As we have seen, a Bayesian network represents a factorization of a joint probability distribution, but there can be many possible factorizations representing the same joint probability distribution. Two structures are said to be *Markov equivalent* if they represent the same joint probability distribution.

In this paper, we do not work directly on Bayesian networks but on a more general model called *Independence Model* (IM), which can be seen as the underlying model of Bayesian networks and defined as follows:

- Let M be a non-empty set of variables, then $T(M)$ denotes the collection of all triplets $\langle X, Y | Z \rangle$ of disjoint subsets of M, $X \neq \emptyset$ and $Y \neq \emptyset$. The class of elementary triplets $E(M)$ consists of $\langle x, y | Z \rangle \in T(M)$, where $x, y \in M$ are distinct and $Z \subset M \setminus \{x, y\}$.
- Let P be a joint probability distribution over M and $\langle X, Y | Z \rangle \in T(M)$. $\langle X, Y | Z \rangle$ is called an *independence statement* (IS) if X is conditionally independent of Y given Z with respect to P (i.e $X \perp\!\!\!\perp Y \mid Z$)
- An independence model (IM) is a subset of $T(M)$: Each probability distribution P defines an IM, namely, the model $\{\langle X, Y | Z \rangle \in T(M) \ ; \ X \perp\!\!\!\perp Y \mid Z\}$, called the *independence model* induced by P.

To summarize, an independence model is the set of all the independence statements, that is the set of all $\langle X, Y|Z\rangle$ satisfied by P, and different Markov-equivalent Bayesian networks induce the same independence model. By following the paths in a Bayesian network, it is possible (even though it can be combinatorial) to find a part of its independence model using algorithms based on directional separation (d-separation) or moralization criteria. Reciprocally, an independence model is a guide to produce the structure of a Bayesian network.

Consequently, as the problem of finding an independence model can be turned to an optimisation problem, we investigate here the use of an evolutionary algorithm. More precisely, we build an algorithm that let a population of triplets $\langle X, Y|Z\rangle$ evolve until the whole population comes near to the independence model, which corresponds to a cooperative coevolution scheme.

1.3 Cooperative coevolution

Cooperative coevolution strategies rely on a formulation of the problem to be solved as a cooperative task, where individuals collaborate or compete (in subpopulations) in order to build a solution. They mimic the ability of natural populations to build solutions via a collective process.

Instead of dealing with a coevolution process that happens between different separated populations, we use a different implementation of cooperative coevolution principles, called *Parisian approach* [19], described in figure 2, that uses cooperation mechanisms within a single population. It is based on a two-level representation of an optimization problem, in the sense that an individual of a Parisian population represents only a part of the solution to the problem. An aggregation of multiple individuals must be built in order to obtain a solution to the problem. In this way, the co-evolution of the whole population (or a major part of it) is favoured instead of the emergence of a single best individual, as in classical evolutionary schemes.

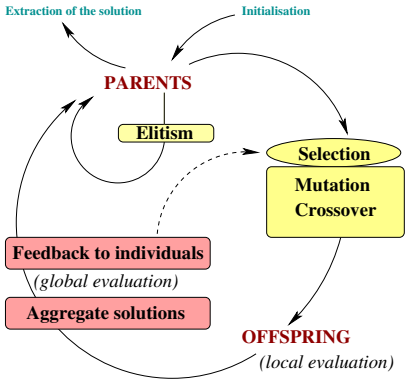

Figure 2: A Parisian EA: A monopopulation cooperative-coevolution

In a pure Parisian scheme, the evaluation of whole population through the computation of the global fitness is done at each generation and redistributed as a bonus to the individuals who participated in this aggregation. In this paper, we rely on local fitness only and just compute the global evaluation at the end, and thus don't use any feedback to the population. This approach has already been used with success for example in real-time evolutionary algorithms, such as the *flies* algorithm [15].

2. EVOLUTION OF AN INDEPENDENCE MODEL

IMPEA is a Parisian Evolutionary Algorithm that consists in two steps. First, it generates a subset of the independence model of a Bayesian network from data by evolving elementary triplets $\langle x, y|Z\rangle$, where x and y are two distinct nodes and Z is a subset of the other ones, possibly empty. Then, it uses the independence statements that it found at the first step to construct the structure of the network.

2.1 Search space and local fitness

Individuals are elementary triplets $\langle x, y|Z\rangle$. Each individual is evaluated through a chi-square test of independence which tests the null hypothesis H_0: "The nodes x and y are independent given Z". The chi-square statistic χ^2 is calculated by finding the difference between each observed O_i and theoretical E_i frequencies for each of the n possible outcomes, squaring them, dividing each by the theoretical frequency, and taking the sum of the results: $\chi^2 = \sum_{i=1}^{n} \frac{(O_i - E_i)^2}{E_i}$. The chi-square statistic can then be used to calculate a *p-value* p by comparing the value of the statistic χ^2 to a chi-square distribution with $n-1$ degrees of freedom, as represented in figure 3.

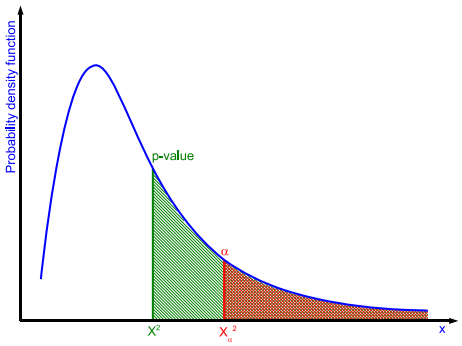

Figure 3: Chi-square test of independence

p represents the probability to make a mistake if the null hypothesis is not accepted. It is then compared to a significance level α (0.05 is often chosen as a cut-off for significance) and finally the independence is rejected if $p < \alpha$. The reader has to keep in mind that rejecting H_0 allows one to conclude that the two variable are dependent, but not rejecting H_0 means that one cannot conclude that these two variables are dependent (which is not exactly the same as claiming that they are independent). Given that the higher the p-value, the stronger the independence, p seems to be a good candidate to represent the local fitness (which measure the quality of individuals). Nevertheless, this fitness suffers from two drawbacks:

- When dealing with small datasets, individuals with long constraining set Z tends to have good p-values only because dataset is too small to get enough samples to test efficiently the statement $x \perp\!\!\!\perp y \mid Z$.
- Due to the exponential behaviour of the chi-square distribution, its tails vanishes so quickly that individuals with poor p-values are often rounded to 0, making then indistinguishable.

First, p has to be adjusted in order to promote independence statements with small Z. This is achieved by setting up a parsimony term as a positive multiplicative malus $parcim(\#Z)$ which decrease with $\#Z$, the number of nodes

in Z. Then, when $p < \alpha$ we replace the exponential tail with something that tends to zero slower. This modification of the fitness landscape allows to avoid *plateaus* which would prevent the genetic algorithm to travel all over the search space. Here is the adjusted local fitness:

$$AdjLocalFitness = \begin{cases} p \times parcim(\#Z) & if\ p \geq \alpha \\ \alpha \times parcim(\#Z) \times \frac{X_\alpha^2}{X^2} & if\ p < \alpha \end{cases}$$

2.2 Genetic operators

The genome of an individual, being $\langle x, y|Z\rangle$ where x and y are simple nodes and Z is a set of nodes is straightforward: It consists in an array of three cells (see figure 4), the first one containing the index of the node x, the second cell containing the index of y and the last one is the array of the indexes of the nodes in Z.

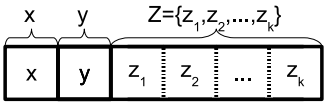

Figure 4: Representation of $\langle x, y|Z\rangle$

This coding implies specific genetic operators because of the constraints resting upon a chromosome: There must not be doubles appearing when doing mutations or crossovers. A quick-and-dirty solution would have been to first apply classical genetic operators and then apply a *repair operator* a posteriori. Instead, we propose wise operators (which do not create doubles), namely two types of mutations and an robust crossover.

- Genome content mutation

  This mutation operator involves a probability p_{mG} that an arbitrary node will be changed from its original state. In order to avoid the creation of doubles, this node can be muted into any nodes in N except the other nodes of the individual, but including itself (see figure 5).

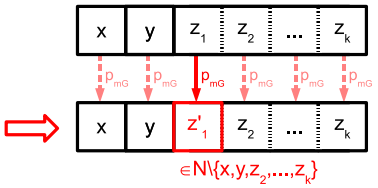

Figure 5: Genome content mutation

- Add/remove mutation

  The previous mutation randomly modifies the content of the individuals, but does not modify the length of the constraining set Z. We introduce a new mutation operator called *add/remove mutation*, represented in figure 6, that allows to randomly add or remove nodes in Z. If this type of mutation is selected, with probability P_{mAR}, then new random nodes are either added with a probability P_{mAdd} or removed with $1 - P_{mAdd}$. These probabilities can vary along generations. Moreover, the minimal and the maximal number of nodes allowed in Z can evolve as well along generations, allowing to tune the growth of Z

- Crossover

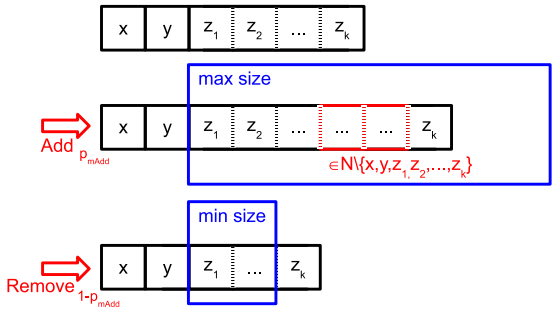

Figure 6: Add/remove mutation

The crossover consist in a simple swapping mechanism between x, y and Z. Two individuals $\langle x, y|Z\rangle$ and $\langle x', y'|Z'\rangle$ can exchange x or y with probability p_{cXY} and Z with probability p_{cZ} (see figure 7). When a crossover occurs, only one swapping among $x \leftrightarrow x'$, $y \leftrightarrow y'$, $x \leftrightarrow y'$, $y \leftrightarrow x'$ and $Z \leftrightarrow Z'$ is selected via a wheel mechanism which implies that $4p_{cXY} + p_{cZ} = 1$. If the exchange is impossible, then the infeasible individual is repaired: problematic nodes are automatically muted in order to keep clear of doubles.

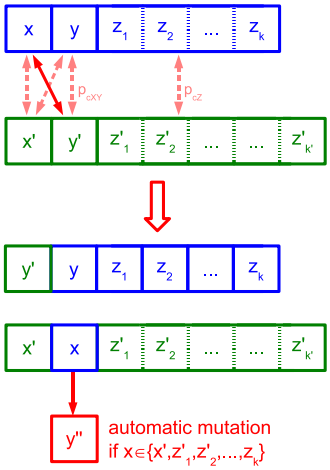

Figure 7: Robust crossover

2.3 Sharing

So as not to converge to a single optimum, but enable the genetic algorithm to identify multiple optima, we use a sharing mechanism that maintains diversity within the population by creating *ecological niches*. The complete scheme is described in [8] and is based on the fact that fitness is considered as a shared resource, i.e that individuals having too many neighbours are penalized. Thus we need a way to compute the distance between individuals so that we can count the number of neighbours of a given individual. A simple Hamming distance was chosen: Two elementary triplets $\langle x, y|Z\rangle$ and $\langle x', y'|Z'\rangle$ are said to be neighbours if they test the same two nodes (i.e $\{x, y\} = \{x', y'\}$), whatever Z. Finally, dividing the fitness of each individual by the number of its neighbours would result in sharing the population into subpopulations whose size is proportional to the height of the peak they are colonising [12]. Instead, we take into account the relative importance of an individual with respect to its neighbourhood, and the fitness of each individual is divided by the sum of the fitnesses of its neighbours [16]. This scheme allows to equilibrate the subpopulations within peaks, whatever their height.

2.4 Immortal archive and embossing points

Recall that the aim of IMPEA is to construct a subset of the independence model, and thus the more independence statements we get, the better. Using a classical Parisian Evolutionary Algorithm scheme would allow to evolve a number of independence statements equal to the population size. In order to be able to evolve larger independence statements sets, IMPEA implements an *immortal archive* that gather the best individuals found so far. An individual $\langle x, y|Z\rangle$ can become immortal if any of the following rules applies:

- Its p-value is equal to 1 (or numerically greater than $1-\epsilon$, where ϵ is the precision of the computer)
- Its p-value is greater than the significance level and $Z = \emptyset$
- Its p-value is greater than the significance level and $\langle x, y|\emptyset\rangle$ is already immortal

This archive serves two purposes: The most obvious one is that at the end of the generations, not only we get all the individuals of the current population but also all the immortal individuals, which can make a huge difference. But this archive also plays a very important role as *embossing points*: When computing the sharing coefficient, immortal individuals that are not in the current population are added to the neighbours counting. Therefore a region of the search space that has already been explored but that has disappeared from the current population is *marked as explored* since immortals individuals count as neighbours and thus penalize this region, encouraging the exploration of other zones.

2.5 Clustering and partial restart

Despite the sharing mechanism, we observed experimentally that some individuals became over-represented within the population. Therefore, we add a mechanism to reduce this undesirable effect: If an individual has too many redundant representatives then the surplus is eliminated and new random individuals are generated to replace the old ones.

2.6 Description of the main parameters

The table 1 describes the main parameters of IMPEA and their typical values or range of values, in order of appearance in the paper. Some of these parameters are scalars, like the number of individuals, and are constant along the whole evolution process. Others parameters, like the minimum or maximum number of nodes in Z, are arrays indexed by the number of generations, allowing these parameter to follow a profile of evolution (for example increasing or decreasing in value, but any kind of profile is possible). This enables to guide the algorithm in a specific direction.

3. BAYESIAN NETWORK STRUCTURE ESTIMATION

The last step of IMPEA consist in reconstructing the structure of the Bayesian network. This is achieved by aggregating all the immortal individuals and only the *good ones* of the final population. An individual $\langle x, y|Z\rangle$ is said to be *good* if its p-value allows not to reject the null hypothesis $x \perp\!\!\!\perp y \mid Z$. There are two strategies in IMPEA: A pure one, called *P-IMPEA*, which consists in strictly enforcing independence statements and an constrained one, called C-IMPEA, which adds a constraint on the number of desired edges.

Name	Description	Typical value
MaxGens	Number of generations	$50 \dots 200$
Ninds	Number of individuals	$50 \dots 500$
Alpha	Significance level of the χ^2 test	$0.01 \dots 0.25$
Parcim (#Z)	Array of parsimony coefficient (decreases with the length of Z)	$0.5 \dots 1$
PmG	Probability of genome content mutation	$0.1/(2+\#Z)$
PmAR	Probability of adding or removing nodes in Z	$0.2 \dots 0.5$
PmAdd (#Gen)	Array of robability of adding nodes in Z along generations	$0.25 \dots 0.75$
MinNodes (#Gen)	Array of minimal number of nodes in Z along generations	$0 \dots 2$
MaxNodes (#Gen)	Array of maximal number of nodes in Z along generations	$0 \dots 6$
Pc	Probability of crossover	0.7
PcXY	Probability of swapping x and y	1/6
PcZ	Probability of swapping Z	1/3
Epsilon	Numerical precision	10^{-5}
MaxRedundant	Maximal number of redundant individuals in the population	$1 \dots 5$

Table 1: Parameters of IMPEA. Values are chosen within their typical range depending on the size of the network and the desired computation time.

3.1 Pure conditional independence

Then, as in PC, P-IMPEA starts from a fully connected graph, and for each individual of the aggregated population, it applies the rule "$x \perp\!\!\!\perp y \mid Z \Rightarrow$ no edge between x and y" to remove edges whose nodes belong to an independence statement. Finally, the remaining edges (which have not been eliminated) constitute the undirected structure of the network.

3.2 Constrained edges estimation

C-IMPEA needs an additional parameter which is the desired number of edges in the final structure. It proceeds by accumulation: It starts from an empty adjacency matrix and for each $\langle x, y|Z\rangle$ individual of the aggregated population, it adds its fitness to the entry (x, y). An example of a matrix obtained this way is shown in figure 8.

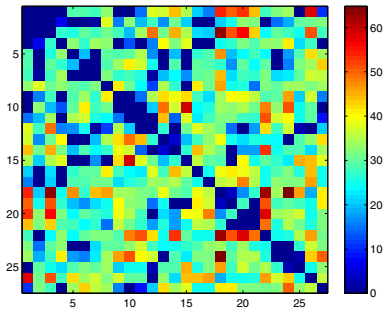

Figure 8: Accumulated adjacency matrix of a network with 27 nodes (from Insurance network).

At the end of this process, if an entry (at the intersection of a row and a column) is still equal to zero, then it means that there was no independence statement with this pair of nodes in the aggregated population. Thus, these entries exactly correspond to the strict application of the conditional independences. If an entry has a low sum, then it is an entry for which IMPEA found only a few independence statements (and/or independence statements with low fitness) and thus there is a high expectancy of having an edge between its

nodes. Therefore, to add more edges in the final structure (up to the desired number of edges), we just have to select edges with the lowest values and construct the corresponding network.

This approach seems to be more robust since it allows some "errors" in the chi-square tests, but strictly speaking, if an independence statement is discovered, there cannot be any edge between the two nodes.

4. EXPERIMENTS AND RESULTS

4.1 Test case: Comb network

To evaluate the efficiency of IMPEA, we forge a test-network which looks like a *comb*. A n-comb network has $n+2$ nodes: x, y, and $z_1, z_2, \ldots, z_n$, as one can see in figure 9. The Conditional Probability Tables (CPT) are filled in with a uniform law. It can be seen as a kind of classifier: Given the input $z_1, z_2, \ldots, z_n$, it classifies the output as x or y. For example, it could be a classifier that accepts a person's salary details, age, marital status, home address and credit history and classifies the person as acceptable/unacceptable to receive a new credit card or loan.

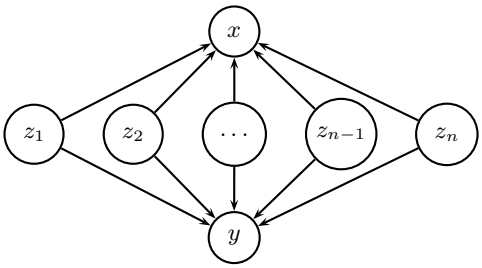

Figure 9: A n-comb network

The interest of such a network is that its independence model can be generated (using semi-graphoid rules) from the following independence statements:

$$\forall i, j \text{ such as } i \neq j, z_i \perp\!\!\!\perp z_j$$
$$x \perp\!\!\!\perp y \mid \{z_1, z_2, \ldots, z_n\}$$

Thus it has only one complex independence statement and a lot of simple (short) ones. In particular, the only way to remove the edge between x and y using statistical chi-square tests is to test the triplet $\langle x, y \mid \{z_1, z_2, \ldots, z_n\}\rangle$. This cannot be achieved by the PC algorithm as soon as $k < n$ (and in practice, k is limited to 3 due to combinatorial complexity).

4.1.1 Typical run

We choose to test P-IMPEA with a simple 6-comb network. It has been implemented using an open source toolbox, the *Bayes Net Toolbox for Matlab* [17] available at `http://bnt.sourceforge.net/`. We draw our inspiration from PC and initialise the population with individuals with an empty constraining set and let it grow along generations up to 6 nodes, in order to find the independence statement $x \perp\!\!\!\perp y \mid \{z_1, \ldots, z_6\}$. As shown in figure 10, the minimal number of nodes allowed in Z is always 0, and the maximal number is increasing on the first two third of the generations and is kept constant to 6 on the last ones. The average number of nodes in the current population is also slowly rising up but remains rather small since in this example, there are a lot of small *easy to find* independence statements and only a unique big one.

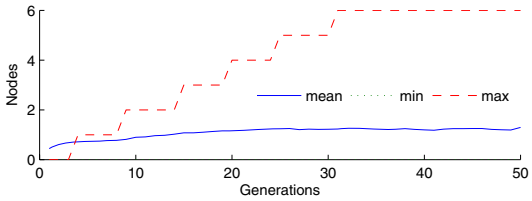

Figure 10: Evolution of Minimal, Maximal and Average number of nodes in Z along generations

The correct structure (figure 11) is found after 40 (out of 50) generations.

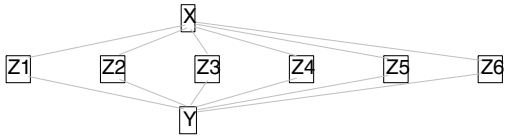

Figure 11: Final evolved structure for the comb network

Figure 12 represents the evolution of the number of errors along generations. The current evolved structure is compared with the actual structure: An *added* edge is an edge present in the evolved structure but not in the actual comb network, and a *deleted* edge is an edge that has been wrongly removed. The total number of errors is the sum of added and deleted edges. Note that even if the number of errors of the discovered edges is extracted at each generation, it is by no means used by IMPEA or reinjected in the population because this information is only relevant in that particular test-case where the Bayesian network that generated the dataset is known.

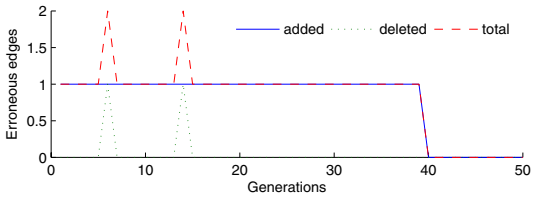

Figure 12: Evolution of the number of erroneous edges of the structure along generations

4.1.2 Statistical results

The previous example gives an idea of the behaviour of P-IMPEA, but to compare it fairly with PC we must compare them not only over multiple runs but also with respect to the size of the dataset. So we set up the following experimental protocol:

- A 4-comb network is created and we use the same Bayesian network (structure and CPT) throughout the whole experiment.
- We chose representative sizes for the dataset: $\{500, 1000, 2000, 5000, 10000\}$, and for each size, we generate the corresponding number of cases from the comb network.
- We run 100 times both PC and P-IMPEA, and extract relevant information (see tables 2 and 3):

  - How many edges were found? Among these, how many were erroneous? (added or deleted)
  - Did the algorithm remove the edge $x - y$?
- PC is tuned with a fan-in k equal to 3 and P-IMPEA is tuned with 50 generation of 50 individuals in order to take the same computational time as PC. They both share the same significance level α.

The actual network contains 8 edges and 6 nodes. Therefore, the number of possible alternative is $2^6 = 64$ and if we roughly want to have 30 samples per possibility, we would need approximatively $64 * 30 \approx 2000$ samples. That explains why performances of the chi-square test are very poor with only 500 and 1000 cases in the dataset. Indeed, when the size of the dataset is too small, PC removes the $x - y$ edge (see the last row of table 2) while it does not even test $\langle x, y \mid \{z_1, z_2, z_3, z_4\}\rangle$ because it is limited by k to 3 nodes in Z.

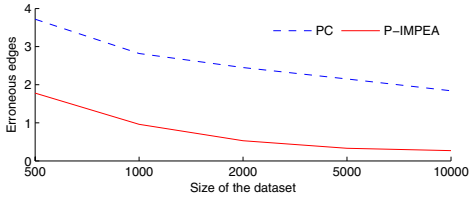

Figure 13: Number of erroneous edges (added+deleted) for PC and P-IMPEA, depending on the size of the dataset

Regarding the global performance, figure 13 puts up the average number of erroneous nodes (either *added* or *deleted*) of both algorithms. As one can expect, the number of errors decreases with the size of the dataset, and it is clear that P-IMPEA clearly outperforms PC in every case.

Cases	Edges	Added	Removed	Errors	x-y?
500	5.04±0.85	0.38±0.50	3.34±0.78	3.72±1.01	97%
1000	6.50±1.24	0.66±0.71	2.16±1.01	2.82±1.23	83%
2000	8.09±1.18	1.27±0.80	1.18±0.68	2.45±0.91	39%
5000	9.71±0.74	1.93±0.57	0.22±0.46	2.15±0.73	0%
10000	9.84±0.58	1.84±0.58	0 ± 0	1.84±0.58	0%

Table 2: Averaged results of PC after 100 runs

Cases	Edges	Added	Removed	Errors	x-y?
500	6.64±0.79	0.05±0.21	1.73±1.90	1.78±1.94	100%
1000	7.32±0.91	0.18±0.50	0.78±1.01	0.96±1.24	100%
2000	8.87±1.04	0.24±0.51	0.29±0.60	0.53±0.82	97%
5000	8.29±0.32	0.30±0.59	0.03±0.17	0.33±0.63	90%
10000	8.27±0.31	0.27±0.54	0 ± 0	0.27±0.54	89%

Table 3: Averaged results of P-IMPEA after 100 runs

Finally, if one has a look to the average number of discovered edges, it is almost equal to 8 (which is the actual number of edges in the 4-comb structure) for P-IMPEA whereas it is greater than 9 for the PC algorithm since it can't remove the $x - y$ edge.

4.2 Classical benchmark: The Insurance Bayesian network

Insurance [2] is a network for evaluating car insurance risks. The Insurance Bayesian network contains 27 variables and 52 arcs (figure 14). We use in our experiments a database containing 50000 cases generated from the network.

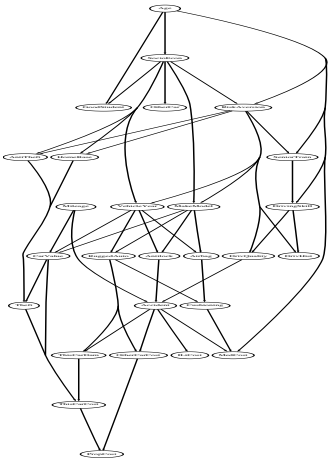

Figure 14: The Insurance Bayesian Network

Once again, we start from a population with small Z and let it increase up to 4 nodes. Figure 15 illustrates this growth: The average size of the number of nodes in Z of the current population follows the orders given by the minimum and the maximum values.

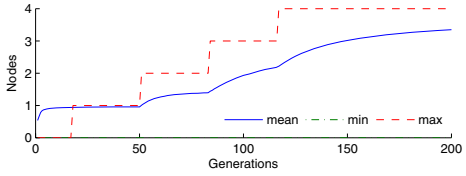

Figure 15: Evolution of Minimal, Maximal and Average number of nodes in Z along generations

Concerning the evolution of the number of erroneous edges, represented in figure 16, it quickly decreases during the first half of the generation (the completely connected graph has more than 700 edges) and then stagnates. At the end, P-IMPEA finds 39 edges out of 52 among which there is no added edge, but 13 which are wrongly removed. It is slightly better than *PC* which also wrongly removes 13 edges, but which adds one superfluous one.

The best results are obtained with C-IMPEA and a desired number of edges equal to 47. Then, only 9 errors are made (see table 4). When asking for 52 edges, the actual number of edges in the Insurance network, it makes 14 errors (7 additions and 7 deletions).

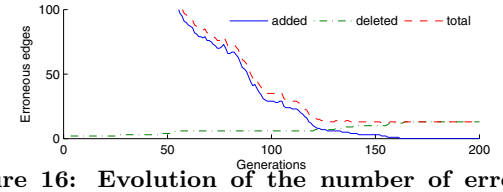

Figure 16: Evolution of the number of erroneous edges of the structure along generations

Algorithm	Edges	Added	Removed	Errors
PC	40	1	13	14
P-IMPEA	39	0	13	13
C-IMPEA	47	2	7	9
C-IMPEA	52	7	7	14

Table 4: Number of detected edges for all algorithms

5. CONCLUSION

In this work we compared performances on the basis of undirected graphs produced by both algorithms. The edge directions estimation has not been yet programmed in IMPEA, this will be done in future developments, using a low combinatorial strategy similar to PC. Comparisons between both algorithms do not actually depend on this step.

The two experiments of section 4 prove that IMPEA favourably compares to PC, actually, besides the fact that IMPEA relies on a convenient problem encoding, PC performs a deterministic and systematic search while IMPEA uses evolutionary mechanisms to prune computational efforts and to concentrate on promising parts of the search space. The limitation of PC according to problem size is obvious in the first test (Comb network): PC is unable to capture a complex dependency, even on a small network. Additionally it is to be noticed that IMPEA better resists to a current problem of real life data, that is the insufficient number of available samples.

Future work on this topic will be devoted to statistical tests on large benchmarks and on a real industrial agrifood application, where we will have to consider incomplete data.

6. REFERENCES

[1] C. Baudrit, P. Wuillemin, M. Sicard, and N. Perrot. A Dynamic Bayesian Network to Represent a Ripening Process of a Soft Mould Cheese. In *KES '08: 12th international conference on Knowledge-Based Intelligent Information and Engineering Systems, Part II*, pages 265–272. Springer-Verlag, 2008.

[2] J. Binder, D. Koller, S. Russell, and K. Kanazawa. Adaptive probabilistic networks with hidden variables. *Machine Learning*, 29:213–244, 1997.

[3] J. Cheng, D. A. Bell, and W. Liu. Learning Belief Networks from Data: An Information Theory Based Approach. In *Sixth ACM International Conference on Information and Knowledge Management*, pages 325–331, 1997.

[4] D. M. Chickering and C. Boutilier. Learning equivalence classes of Bayesian-network structures. In *Journal of Machine Learning Research*, pages 150–157. Morgan Kaufmann, 1996.

[5] D. M. Chickering, D. Heckerman, and C. Meek. Large-Sample Learning of Bayesian Networks is NP-Hard. *Journal of Machine Learning Research*, 5:1287–1330, 2004.

[6] C. Chow and C. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, 14(3):462–467, 1968.

[7] G. F. Cooper and E. Herskovits. A Bayesian Method for the Induction of Probabilistic Networks from Data. *Machine Learning*, 09(4):309–347, 1992.

[8] K. Deb and D. Goldberg. An investigation of niche and species formation in genetic function optimization. In *Third international conference on Genetic algorithms*, pages 42–50. Morgan Kaufmann Publishers Inc., 1989.

[9] O. Francois and P. Leray. Etude comparative d'algorithmes d'apprentissage de structure dans les réseaux Bayésiens. In *Rencontres des Jeunes Chercheurs en IA*, 2003.

[10] N. Friedman. Learning Belief Networks in the Presence of Missing Values and Hidden Variables. In *14th International Conference on Machine Learning*, pages 125–133. Morgan Kaufmann, 1997.

[11] N. Friedman, M. Linial, I. Nachman, and D. Pe'er. Using Bayesian Network to Analyze Expression Data. *J. Computational Biology*, 7:601–620, 2000.

[12] D. E. Goldberg and J. Richardson. Genetic algorithms with sharing for multimodal function optimization. In *Second International Conference on Genetic Algorithms and their application*, pages 41–49. Lawrence Erlbaum Associates, Inc., 1987.

[13] H. Jia, D. Liu, and P. Yu. Learning dynamic Bayesian network with immune evolutionary algorithm. 2005.

[14] P. Larrañaga and M. Poza. Structure Learning of Bayesian Networks by Genetic Algorithms: A Performance Analysis of Control Parameters. *IEEE Journal on Pattern Analysis and Machine Intelligence*, 18(9):912–926, 1996.

[15] J. Louchet, M. Guyon, M. J. Lesot, and A. M. Boumaza. Dynamic flies: a new pattern recognition tool applied to stereo sequence processing. *Pattern Recognition Letters*, 23(1-3):335–345, 2002.

[16] E. Lutton and P. Martinez. A Genetic Algorithm with Sharing for the Detection of 2D Geometric Primitives in Images. In *AE '95: Selected Papers from the European conference on Artificial Evolution*, pages 287–303. Springer-Verlag, 1995.

[17] K. Murphy. The Bayes Net Toolbox for Matlab. *Computing Science and Statistics*, 33(2):1024–1034, 2001.

[18] J. W. Myers, K. B. Laskey, and K. A. DeJong. Learning Bayesian Networks from Incomplete Data using Evolutionary Algorithms. In *Genetic and Evolutionary Computation Conference*, volume 1, pages 458–465. Morgan Kaufmann, 1999.

[19] G. Ochoa, E. Lutton, and E. Burke. The Cooperative Royal Road: Avoiding Hitchhiking. In *Artificial Evolution*, pages 184–195, 2007.

[20] J. Pearl and T. Verma. A theory of inferred causation. In *Second International Conference on the Principles of Knowledge Representation and Reasoning*, 1991.

[21] R. W. Robinson. Counting unlabeled acyclic digraphs. *Combinatorial mathematics V*, 622:28–43, 1977.

[22] B. J. Ross and E. Zuviria. Evolving dynamic Bayesian networks with Multi-objective genetic algorithms. *Applied Intelligence*, 26, 2007.

[23] P. Spirtes, C. Glymour, and R. Scheines. *Causation, Prediction, and Search*. The MIT Press, second edition, 2001.

[24] A. Tucker and X. Liu. Extending Evolutionary Programming Methods to the Learning of Dynamic Bayesian Networks. In *GECCO '99*, 1999.

[25] S. C. Wang and S. P. Li. Learning Bayesian Networks by Lamarckian Genetic Algorithm and Its Application to Yeast Cell-Cycle Gene Network Reconstruction from Time-Series Microarray Data. 3141/2004:49–62.

[26] M. L. Wong and K. S. Leung. An efficient data mining method for learning Bayesian networks using an evolutionary algorithm-based hybrid approach. 8:378–404, 2004.