Learning Bayesian Networks with Algebraic Differential Evolution: 15th International Conference, Coimbra, Portugal, September 8–12, 2018, Proceedings, Part II

Chapter · January 2018

DOI: 10.1007/978-3-319-99259-4_35

Marco Baioletti
Università degli Studi di Perugia

Alfredo Milani
Hong Kong Baptist University

Valentino Santucci
Università degli Studi di Perugia

Learning Bayesian Networks with Algebraic Differential Evolution

Marco Baioletti[1], Alfredo Milani[1,3], Valentino Santucci[2]

[1] Dept. of Mathematics and Computer Science, University of Perugia, Italy
[2] Dept. of Social and Human Sciences, University for Foreigners of Perugia, Italy
[3] Dept. of Computer Science, Hong Kong Baptist University, Hong Kong SAR China
{marco.baioletti,alfredo.milani}@unipg.it
valentino.santucci@unistrapg.it

Abstract. In this paper we introduce DEBN, a novel evolutionary algorithm for learning the structure of a Bayesian Network. DEBN is an instantiation of the Algebraic Differential Evolution which is designed and applied to a particular (product) group whose elements encode all the Bayesian Networks of a given set of random variables. DEBN has been experimentally investigated on a set of standard benchmarks and its effectiveness is compared with BFO-B, a recent and effective bacterial foraging algorithm for Bayesian Network learning. The experimental results show that DEBN largely outperforms BFO-B, thus validating our algebraic approach as a viable solution for learning Bayesian Networks.

Keywords: Bayesian Networks Learning, Algebraic Differential Evolution

1 Introduction and Related Work

A Bayesian Network (BN) [14] is used to represent in a compact and effective way a probability distribution of a set of discrete random variables $X_1, \ldots, X_n$. A BN is composed by two different parts. The qualitative component is a directed acyclic graph (DAG) G in which the nodes are the variables X_i and the edges denote influences among variables. Given a variable X_i, $pa(X_i)$ is the set of parents of X_i and it contains all the variables X_j connected with an incoming edge to X_i. For each variable X_i, the quantitative component contains a conditional probability distribution of X_i with respect to $pa(X_i)$, i.e., the conditional probability $p(X_i = x_i | pa(X_i) = c_j)$ for each value x_i of X_i and for each combination c_j of values for the variables in $pa(X_i)$.

The problem of learning BNs from empirical data has been extensively studied during last years [14]. In particular, the problem of learning the structure (qualitative part) of the network is well investigated, being the problem of learning the quantitative part much simpler, once the structure is given.

There are three main methodologies to learn the structure of a Bayesian Network. The first approach is to find conditional independence relations through statistical tests and to use them to infer the structure (for instance the presence of arcs), as done in [24]. Another possibility is the constraint-based approach, for instance dynamic programming [14]. Finally, a third approach is to perform a search process into a suitable space in order to find the optimal structure according to a given score metric.

Many score functions have been proposed for this purpose, e.g., K2, BDe, AIC, BIC and MDL scores [14]. In particular, we focus on the K2 and BDe scores that, given a BN with DAG structure G and a dataset D, are respectively defined as follows:

$$K2(G;D) = \prod_{i=1}^{n} \prod_{j=1}^{q_i} (\frac{(r_i - 1)!}{(N_{ij} + r_i - 1)!} \cdot \prod_{k=1}^{r_i} N_{ijk}!$$

$$BDe(G;D) = \prod_{i=1}^{n} \prod_{j=1}^{q_i} \frac{\Gamma(N'_{ij})}{\Gamma(N'_{ij} + N_{ij})} \cdot \prod_{k=1}^{r_i} \frac{\Gamma(N'_{ijk} + N_{ijk})}{\Gamma(N'_{ijk})}$$

where, for each variable X_i, r_i is the cardinality of the domain of X_i, q_i is the number of possible value combinations of $pa(X_i)$, and N_{ijk} is the number of records in D in which X_i takes the k–th value and $pa(X_i)$ take the j–th combination of values. Besides, the BDe parameters N'_{ijk}, for every triple i, j, k, are usually set to $\frac{N'}{q_i r_i}$, where N' is a constant called *equivalent sample size* which, in this paper, as in several other works [14], we set to $N' = 1$. Furthermore, for the sake of computation, the logarithm of the score functions is usually employed.

There have been many attempts to solve the BN learning problem as a combinatorial problem and one of the most studied approach is through evolutionary techniques. Starting from Larrañaga paper [18], which used genetic algorithms, the most used approach is Ant Colony Optimization, since it uses an incremental way of building solutions, making the enforcement of acyclicity constraint easy to manage [10]. Another approach is to employ evolutionary algorithms to produce good orderings among variables, which are then used as input to other DAG construction algorithms, like K2 [11, 13, 26]. Other evolutionary approaches are based on discrete variants of Particle Swarm Optimization [27, 15]. An alternative search space is the Partial DAG space [7], which represent in a compact way an equivalence class of DAGs.

A hybrid approach which combines conditional independence learning with searching for an optimal structure is [25].

One of the best evolutionary approach to this problem is BFO-B [28, 12], an application of Bacterial Foraging Optimization technique. BFO-B has been compared to other swarm intelligence and other techniques showing that BFO-B outperforms all its evolutionary and non-evolutionary competitors.

In this paper we present an Algebraic Differential Evolution algorithm [21, 22] to solve this problem. Differential Evolution (DE) [19] is widely adopted in optimization problems due to its capacity of self-adapting the search to the fitness landscape at hand. Although DE has been originally proposed for continuous problems, in a previous series of papers [21, 2, 20, 1, 4, 5], we have introduced an algebraic framework that allows to apply DE to combinatorial problems in which the search space is a finitely generated group.

In particular, in this paper we propose a novel representation for DAGs which allows to see the search space of all DAGs of a fixed vertex sets as a product group. In this way, it is possible to apply Algebraic DE to the BN learning problem in terms of finding the DAG with the maximum score.

Our algorithm, called DEBN, has been tested on some standard benchmarks and compared with BFO-B which, to the best of our knowledge, is the state-of-the-art evolutionary technique for BN learning. The experimental results show that DEBN largely outperforms BFO-B.

2 Differential Evolution

Differential Evolution (DE) [19] is a simple and powerful evolutionary algorithm for optimizing non-linear and even non-differentiable real functions $f : \mathbb{R}^n \to \mathbb{R}$. Hence, DE evolves a population of N real-valued vectors $x_1, \ldots, x_N \in \mathbb{R}^n$ by iteratively applying the three genetic operators: differential mutation, crossover, and selection.

The differential mutation generates a *mutant* y_i for each *target* population individual x_i. Though several mutation schemes have been proposed [19], the original one is denoted by rand/1 and it is computed as

$$y_i = x_{r_1} + F \cdot (x_{r_2} - x_{r_3}) \tag{1}$$

where r_1, r_2, r_3 are three random integers in $\{1, \ldots, N\}$ mutually different among them and with respect to i, while $F > 0$ is the scale factor parameter of DE.

For each pair formed by the target individual x_i and the mutant y_i, the crossover generates a *trial* solution z_i by recombining x_i and y_i. The most common variant is the *binomial crossover* [19] which generates z_i according to

$$z_i^{(j)} = \begin{cases} y_i^{(j)} \text{ if } r_{1,j} \leq CR \text{ or } r_2 = j \\ x_i^{(j)} \text{ otherwise} \end{cases} \tag{2}$$

where: CR is the crossover probability (another parameter of DE), $r_{1,j} \in [0, 1]$ is a random number generated for each dimension j, and $r_2 \in \{1, \ldots, n\}$ is randomly generated in order to guarantee that at least one component is inherited from the mutant y_i. Note also, that other crossover schemes are available [19, 8, 9].

Finally, the most used *selection* operator compares each target individual x_i with the corresponding trials z_i and selects the better between them to enter in the population of the next generation.

3 Algebraic Framework

Here we provide a concise description of the algebraic framework for evolutionary computation previously proposed in [21] and [3]. In particular, our attention has been mainly focused on ADE, an algebraic version of Differential Evolution, which obtained state-of-the-art performances on the permutation flow-shop scheduling problem [21, 22]. Note anyway that the framework is rather general and can be adapted to other evolutionary algorithms [1] and other search spaces.

In principle, our algebraic methodology can be applied to all the combinatorial problems whose search space X forms a finitely generated group with respect to an internal composition $\star$ and a set of generators $H \subseteq X$ [16].

Recall that a group $(X, \star)$ is *finitely generated* if there exists a finite subset $H \subseteq X$, called *generating set*, such that any $x \in X$ can be decomposed as $x = h_1 \star h_2 \star \ldots \star h_l$ for some $h_1, h_2, \ldots, h_l \in H$. We also denote by $|x|$ the length of a minimal decomposition of x in terms of H.

The *Cayley graph* of a finitely generated group is the labeled digraph whose vertexes are the solutions in X and there exists an arc from x to y labeled by $h \in H$ if and only if $y = x \star h$. Moreover, for all $x \in X$, every (shortest) path from the neutral element e to x corresponds to a (minimal) decomposition of x, i.e., if the arc labels in the path are $(h_1, h_2, \ldots, h_l)$, then $x = h_1 \star h_2 \star \ldots \star h_l$.

The Cayley graph has an important geometric interpretation. Indeed, any solution $x \in X$ can be seen both as a *point*, i.e., a vertex in the graph, but also as a *vector* because its decomposition is a sequence of generators, i.e., arcs of a path in the Cayley graph. This dichotomous interpretation allows to define the operations $\oplus, \ominus, \odot$ on X in such a way that they simulate the analogous operations of the Euclidean space.

3.1 Vector-like operations

The *addition* $x \oplus y$ is defined as the application of the vector $y \in X$, decomposed as $(h_1, h_2, \ldots, h_l)$, to the point $x \in X$. It can be proved [21] that

$$x \oplus y = x \star y. \tag{3}$$

Given $x, y \in X$ considered as points, their *difference* $y \ominus x$ is the vector $(h_1, h_2, \ldots, h_l)$ which are the labels of a path from x to y. In [21] we proved that

$$y \ominus x = x^{-1} \star y. \tag{4}$$

Given $a \in [0, 1]$ and $x \in X$, the result of the *scalar multiplication of x by the scalar a*, denoted by $a \odot x$, is defined as

$$a \odot x = h_1 \star h_2 \star \ldots \star h_k \tag{5}$$

where $(h_1, h_2, \ldots, h_l)$ is a minimal decomposition of x and $k = \lceil a \cdot |x| \rceil$. The operation $\odot$, contrarily to $\oplus$ and $\ominus$, depends on the particular minimal decomposition chosen for x. In general there can be multiple minimal decompositions, thus $\odot$ is not uniquely defined. However, since we are designing an evolutionary algorithm, we consider a random minimal decomposition of x when computing $a \odot x$.

In the following we describe the groups of the permutations and bit-strings, which will be used later in the paper.

3.2 Permutation group

The set $\mathcal{S}_n$ of the permutations of $\{1, 2, \dots, n\}$ forms a group, called *symmetric group*, with respect to the permutation composition $\circ$. Given $\pi, \rho \in \mathcal{S}_n$, their composition $\pi \circ \rho$ is defined as the permutation $(\pi \circ \rho)(j) = \pi(\rho(j))$ for all the indexes $j = 1, \dots, n$. $\mathcal{S}_n$ is not Abelian (for $n \geq 3$) and its neutral element is the identity permutation ι such that $\iota(j) = j$ for all $j = 1, \dots, n$.

Among the many generating sets of $\mathcal{S}_n$, the simplest one is the set of simple transpositions

$$ST = \{\sigma_i \in \mathcal{S}_n : 1 \leq i < n\},$$

where σ_i corresponds to an adjacent swap between positions i and $i{+}1$. Formally: $\sigma_i(i) = i + 1$, $\sigma_i(i + 1) = i$, and $\sigma_i(j) = j$ for $j \in \{1, \dots, n\} \setminus \{i, i + 1\}$.

A random decomposition algorithm for this generating set is the RandBS procedure, introduced in [21], which produces a random minimal decomposition of a given permutation by requiring $O(n^2)$ computational time. It is worth to notice that the length of a minimal decomposition of $\pi \in \mathcal{S}_n$ is the number of inversions of π.

We will denote by $\oplus_p, \ominus_p, \odot_p$, respectively, the operations $\oplus, \ominus, \odot$ defined for $\mathcal{S}_n$.

3.3 Bit-string group

The set $\mathbb{B}^m$ of all the m-length bit-strings forms an Abelian group with respect to the bitwise XOR operator $\veebar$. Its neutral element is the all-zeros string 0. Since $x \veebar x = 0$ for all $x \in \mathbb{B}^m$, the inverse of any $x \in \mathbb{B}^m$ is itself.

The most obvious generating set for $\mathbb{B}^m$ is the set

$$U = \{u_i \in \mathbb{B}^m : u_i(i) = 1 \text{ and } u_i(j) = 0 \text{ for } j \neq i\},$$

where $u_i(k)$ indicates the k-th bit of the string u_i.

A random decomposition algorithm for a bit-string b can be easily found by selecting all the indices $i \in \{1, \dots, m\}$ with $b(i) = 1$ and disposing them into a sequence with a random order. Note that the length of a minimal decomposition of b is just its Hamming weight.

We will denote by $\oplus_b, \ominus_b, \odot_b$, respectively, the operations $\oplus, \ominus, \odot$ defined for $\mathbb{B}^m$. It is important to notice that $\oplus_b$ and $\ominus_b$ coincide and both are commutative.

4 Dual representation of Bayesian Networks

In this section we introduce the representation of BN structures, i.e., DAGs, and their associated finitely generated group.

A DAG G of n vertices can be represented by a pair (π, b), where $\pi \in \mathcal{S}_n$ and $b \in \mathbb{B}^m$ with $m = \binom{n}{2}$.

The bits of b represent the skeleton of G. Let $C = \{(j, k) : 1 \leq j < k \leq n\}$ be the ordered set of vertex pairs, if the i–th pair of C is (j, k), then there exists in G an arc from X_j to X_k, or vice versa, if and only if $b_i = 1$.

The permutation π determines the direction of the arcs: if $b_i = 1$, then the arc goes from X_j to X_k if j appears before k in π, i.e., $\pi^{-1}(j) < \pi^{-1}(k)$, otherwise the arc goes in the opposite direction. Said in other words, π is a topological order of the variables $X_1, \ldots, X_n$.

One of the most important properties of this representation is that any pair (π, b) represents a DAG. This fact is an apparent advantage of this representation with respect to other forms (for instance the adjacency matrix) where constraint must be used to select which combinations correspond to directed acyclic graph. However, since a DAG can have more than one topological order, our representation is, in general, a many-to-one representation, i.e., there can be multiple pairs (π, b) that represent the same DAG.

The set of all the pairs (π, b), such that $\pi \in \mathcal{S}_n$ and $b \in \mathbb{B}^m$, is the Cartesian product $\mathcal{B} = \mathcal{S}_n \times \mathbb{B}^m$. Importantly, $\mathcal{B}$ can be endowed with the binary operation $*$ defined as

$$(\pi, b) * (\pi', b') = (\pi \circ \pi', b \veebar b') \tag{6}$$

where $\circ$ and $\veebar$ are the group operations for $\mathcal{S}_n$ and $\mathbb{B}^m$, respectively. Therefore $\mathcal{B}$ is a group with respect to $*$, namely the *product group* of $\mathcal{S}_n$ and $\mathbb{B}^m$. Its neutral element is $(\iota, 0)$, while the inverse of (π, b) is (π^{-1}, b).

Addition and subtraction on $\mathcal{B}$ can now be defined as in equations (3) and (4), by using the operation $*$ and its related inverse operator.

In order to define the multiplication of a pair (π, b) by a scalar $a \in [0, 1]$, we have to choose a generating set for $\mathcal{B}$. We describe two ways of defining a generating set for $\mathcal{B}$ starting from the generating sets for $\mathcal{S}_n$ and for $\mathbb{B}^m$.

The *additive* generating set A is defined as

$$A = ST' \cup U'$$

where $ST' = \{(\sigma_i, 0) : i = 1, \ldots, n-1\}$ and $U' = \{(\iota, u_j) : j = 1, \ldots, m\}$. Its cardinality is $|A| = n - 1 + m$. Note that the generators of ST' only influence the permutation part (since the second component of the element of ST' is 0). Conversely, the generators of U' have effect only on the binary part. Using A as generating set, it is easy to prove that $|(\pi, b)| = |\pi| + |b|$.

A randomized decomposition algorithm for A which produces a random minimal decomposition of $(\pi, b) \in \mathcal{B}$ is the following. Given a random minimal decomposition $(\sigma_{h_1}, \ldots, \sigma_{h_L})$ of π and a random minimal decomposition $(u_{k_1}, \ldots, u_{k_M})$ of b, then

– create an empty sequence r of size $L + M$,
– choose L random different indices $1 \leq j_1 < \ldots < j_L \leq L + M$ of r,
– assign $r_{j_v} \leftarrow (\sigma_{h_v}, 0)$ for $v = 1, \ldots, L$,
– fill the M unassigned positions of r with the pairs (ι, u_{k_v}) for $v = 1, \ldots, M$.

The *multiplicative* generating set P is defined as

$$P = (ST \times U) \cup A$$

whose cardinality is $n(m + 1)$. A minimal decomposition of a pair $(\pi, b) \in \mathcal{B}$ in terms of P is much shorter than a minimal decomposition in terms of A, because each generator belonging to $ST \times U$ affects both the permutation and binary part.

A minimal decomposition of $(\pi, b) \in \mathcal{B}$ can be obtained by pairing the minimal decompositions of π and b. In the general case, a certain number of copies of the neutral element have to be added to the shorter of the two minimal decompositions in order to match the length of the longest one. The generators of A (also present in P) are useful for this purpose. Using P as generating set, it can be easily proved that $|(\pi, b)| = \max\{|\pi|, |b|\}$.

A randomized minimal decomposition for $(\pi, b) \in \mathcal{B}$ in terms of P is computed as follows. Given a random minimal decomposition $(\sigma_{h_1}, \ldots, \sigma_{h_L})$ of π and a random minimal decomposition $(u_{k_1}, \ldots, u_{k_M})$ of b, if $L < M$, then

- create an empty sequence r of size M,
- choose L random different indices $1 \leq j_1 < \ldots < j_L < M$ in r,
- assign $r_{j_v} \leftarrow (\sigma_{h_v}, u_{k_{j_v}})$ for $v = 1, \ldots, L$,
- fill the $M - L$ unassigned positions of r with the pairs (ι, u_{k_v}) for $v = 1, \ldots, M$.

The method works in a similar way when $L \geq M$.

5 The algorithm DEBN

In this section we describe DEBN, the algorithm based on the Algebraic Differential Evolution for learning Bayesian Networks. It has the same structure of a classical DE algorithm: its pseudo-code is depicted in Alg. 1.

Algorithm 1 DEBN Pseudo-Code

```
function DEBN
    Initialize and Evaluate the Population
    while termination criterion is not met do
        for $i \leftarrow 1$ to $N$ do
            $y_i \leftarrow$ DifferentialMutation($x_i, F, pm$)
            $z_i \leftarrow$ Crossover($x_i, y_i, CR$)
            Evaluate($z_i$)
        for $i \leftarrow 1$ to $N$ do
            $x_i \leftarrow$ selection($x_i, z_i$)
    return the best BN structure found
```

Any population individual x_i is represented by means of the dual representation introduced in Section 4, i.e., $x_i = (\pi_i, b_i)$, where $\pi_i \in \mathcal{S}_n$, $b_i \in \mathbb{B}^m$, and $m = \binom{n}{2}$. The individuals are evaluated by means of a BN score function selected by the user. In this work, K2 and BDe have been considered (see Section 1).

Each individual $x_i = (\pi_i, b_i)$ is randomly initialized by selecting a permutation π_i uniformly at random on $\mathcal{S}_n$, while each bit of b_i is set to 1 with probability $\frac{2}{n-1}$, thus that the average number of edges in the BN represented by x_i is n.

The discrete differential mutation uses the algebraic operations $\oplus$, $\ominus$, $\odot$ of $\mathcal{B}$. Moreover, in order to mitigate the diversity loss phenomenon, typical in combinatorial spaces, the rand/1 scheme of classical DE has been extended by introducing a random term as follows:

$$y_i = (x_{r_1} \oplus t) \oplus F \odot (x_{r_2} \ominus x_{r_3}), \tag{7}$$

where, as in equation (1), x_{r_1}, x_{r_2}, and x_{r_3} are three random population individual different to each other and with respect to x_i, while $F \in [0, 1]$ is the scale factor parameter.

Furthermore, $t \in \mathcal{B}$ is randomly generated by means of the *pre-mutation probability* $pm \in (0, 1)$ such that $|t| = k$ with probability pm^k. Operatively, t is initialized to the neutral element $(\iota, 0)$, then, during a loop, a random number $r \in [0, 1]$ is generated and, if $r < pm$, a suitable randomly selected generator (from A or P) is applied to t. As soon as $r \geq pm$, the loop is stopped and t is returned.

Two crossover operators are separately applied to the permutation and binary parts of $x_i = (\pi_i, b_i)$ and $y_i = (\pi'_i, b'_i)$, thus obtaining the trial individual $z_i = (\pi''_i, b''_i)$. We have implemented different crossover schemes for the permutation and binary parts. After preliminary tests we decide to use this combination of crossover:

$$\pi'' = CYC(\pi_i, \pi'_i)$$

$$b''_i = BIN(b_i, b'_i, CR)$$

where CYC is the (parameterless) cycle crossover described in [17], and BIN is the usual binomial crossover of DE, as defined in equation (2).

The generation is then concluded by applying the 1-to-1 selection scheme of classical DE.

DEBN has also been equipped with a self-adaptive procedure, inspired by the well known jDE method [6], that allows to self-regulate the three parameters pm, F, and CR. Each population individual maintains its own parameter values. Then, independently for each parameter, when mutant and trial are generated, with probability 0.9, they inherit the value of the target population individual, otherwise they randomly sample a new value in the allowed range for the parameter at hand, i.e., $[0.1, 1]$ for F, $[0, 1]$ for CR, and $[0.1, 0.3]$ for pm.

Finally, two implementations of DEBN can be devised, namely DEBN$_+$, where the operation $\odot$ is defined with respect to the generating set A, and DEBN$_\times$, which is based on P.

6 Experimental results

In this section we describe the experimental results obtained with the implementation of DEBN algorithm. Experiments have been conducted using 8 popular BN benchmarks. For each one, we have generated two datasets of different sizes by using the sampling procedure of the *bnlearn* R package [23].

The benchmark names are provided in Table 1, where we also report the dataset sizes, together with the number of nodes and edges of the true networks from which the datasets are generated. The aim of the experimentation is thus to try to the recover the original networks by maximizing the K2 and BDe scores, computed by only looking at the datasets.

Table 1. Datasets

Network	**Size1**	**Size2**	**#vars**	**#edges**
alarm	4000	8000	37	46
asia	1000	5000	8	8
barley	5000	10000	48	84
child	2000	5000	20	25
hailfinder	5000	10000	56	66
insurance	3000	6000	27	52
water	4000	8000	32	66
win95pts	5000	10000	76	112

Three algorithms have been compared: DEBN$_+$, DEBN$_\times$, and the recent state-of-the-art evolutionary algorithm BFO-B, which has been implemented by faithfully following the description given in [28]. As indicated by its authors, the BFO-B parameters have been set to $N_s = 4$, $N_{re} = 4$, $N_{ed} = 3$, $S = 80$, $N_c = 30$, and $P_{ed} = 0.1$.

Our DEBN$_+$ and DEBN$_\times$ only require to set the population size N. After some preliminary tests (here not reported for the lack of space), we decided to use $N = 50$ for both the variants.

In order to choose a fair termination criterion for DEBN, we have observed that BFO-B performed around 100 000 fitness evaluations in average, so we used this number of evaluations also for DEBN.

All the three algorithms have been run 20 times per dataset using both the considered score metrics. Tables 2 and 3 provide the average and best scores obtained by all the algorithm and considering, respectively, BDe and K2 as score functions. For each dataset, the best average and maximum scores are indicated in, respectively, bold and italic.

Tables 2 and 3 clearly show that our proposals largely outperform BFO-B on almost all the comparisons. The only exceptions are on *water_4000* and *water_8000* where BFO-B obtains a better average K2 score. However, in the larger BNs *hailfinder* and *win95pts*, the score differences are remarkably large in favor of the DEBN algorithms.

Regarding the comparison between the two DEBNs and considering the BDe metric, DEBN$_\times$ obtained better average scores, while DEBN$_+$ shows better peak performances.

On the other end, considering the K2 metric, DEBN$_+$ is slightly preferable with respect to DEBN$_\times$ both in terms of average and peak performances.

Table 2. Results with *BDe* score

Dataset	**BFO-B**		**DEBN$_+$**		**DEBN$_\times$**	
	avg	**max**	**avg**	**max**	**avg**	**max**
alarm_4000	-43794.98	-43437.00	**-42570.45**	*-42502.57*	-42610.05	-42519.19
alarm_8000	-87576.63	-86859.20	-85234.69	*-85086.80*	**-85209.07**	-85087.68
asia_1000	-2310.45	*-2310.37*	**-2310.37**	*-2310.37*	**-2310.37**	*-2310.37*
asia_5000	-11394.74	-11394.50	**-11394.49**	*-11394.49*	**-11394.49**	*-11394.49*
barley_5000	-268201.94	-264423.00	**-266737.42**	*-262211.07*	-267429.76	-263562.99
barley_10000	-532904.94	-527937.00	-524224.13	*-512050.47*	**-523856.84**	-517022.55
child_2000	-25041.73	-25040.10	-25040.90	*-25040.08*	**-25040.08**	*-25040.08*
child_5000	-61385.42	-61382.90	**-61382.93**	*-61382.93*	**-61382.93**	*-61382.93*
hailfinder_5000	-251569.15	-250809.00	-249938.90	*-249555.91*	**-249935.54**	-249635.38
hailfinder_10000	-503277.10	-499926.00	-497099.00	-496475.76	**-496891.03**	*-496451.81*
insurance_3000	-40884.81	-40718.00	-40472.34	*-40347.91*	**-40454.16**	-40389.54
insurance_6000	-80903.57	-80495.30	-80146.05	-79966.23	**-80109.22**	*-79960.52*
water_4000	-52234.43	-52154.70	**-52068.78**	*-51993.13*	-52071.86	-52001.85
water_8000	-103631.50	-103443.00	**-103320.64**	*-103155.90*	-103352.41	-103168.28
win95pts_5000	-60481.39	-58908.90	-51460.67	-49924.90	**-51348.43**	*-49440.49*
win95pts_10000	-121667.70	-118890.00	-105275.83	*-99497.04*	**-104804.92**	-100368.35

Table 3. Results with *K*2 score

Dataset	**BFO-B**		**DEBN$_+$**		**DEBN$_\times$**	
	avg	**max**	**avg**	**max**	**avg**	**max**
alarm_4000	-43802.95	-43429.90	**-42760.22**	*-42694.80*	-42790.03	-42695.92
alarm_8000	-87576.81	-86298.90	**-85451.64**	*-85301.86*	-85573.49	-85318.76
asia_1000	-2289.97	*-2289.94*	**-2289.94**	*-2289.94*	**-2289.94**	*-2289.94*
asia_5000	-11373.64	-11373.50	**-11373.47**	*-11373.47*	**-11373.47**	*-11373.47*
barley_5000	-282797.80	-280376.00	**-278009.54**	*-274887.52*	-279218.10	-275726.47
barley_10000	-558264.08	-551764.00	-538313.54	*-531544.39*	**-536969.49**	-531683.39
child_2000	-25022.86	-25019.60	**-25019.56**	*-25019.56*	-25020.45	*-25019.56*
child_5000	-61366.06	*-61363.50*	**-61363.76**	-61363.55	**-61363.76**	-61363.55
hailfinder_5000	-252976.70	-251587.00	**-250506.92**	*-250204.53*	-250543.04	-250266.94
hailfinder_10000	-506897.80	-504137.00	**-497874.99**	-497546.61	-497917.03	*-497194.28*
insurance_3000	-41273.06	-41150.30	**-40905.16**	-40857.28	-40921.03	*-40823.76*
insurance_6000	-81655.57	-81088.70	**-80868.01**	*-80642.36*	-80905.42	-80707.54
water_4000	**-52451.31**	*-52429.50*	-52487.60	-52433.88	-52490.21	-52449.57
water_8000	**-103793.35**	-103760.00	-103854.43	-103771.33	-103842.66	*-103755.88*
win95pts_5000	-54866.60	-52726.90	**-50442.92**	-48932.97	-50852.83	*-48509.02*
win95pts_10000	-112196.95	-110883.00	**-103897.32**	*-97930.23*	-104792.63	-101290.97

7 Conclusions and Future Work

In this paper we have described DEBN, an Algebraic Differential Evolution algorithm [21] for learning the structure of a Bayesian Network (BN). DEBN is based on a novel BNs representation based on the algebraic concept of product group, where a DAG is represented by a permutation and a bit-string. Both permutations and bit-strings are finitely generated groups, hence it is possible to apply the principles of Algebraic Differential Evolution.

Two variants of DEBN have been proposed and experimentally compared with BFO-B, one of the best evolutionary algorithms for BN learning. The experimental results clearly show that DEBN largely outperforms BFO-B.

As future lines of research, we will investigate: the use of other generating sets for the permutation part (see [3]), and the application of the DEBN scheme to other problems whose solutions are DAGs.

References

1. M. Baioletti, A. Milani, and V. Santucci. Algebraic particle swarm optimization for the permutations search space. In *Proc. of 2017 IEEE Congress on Evolutionary Computation (CEC 2017)*, pages 1587–1594, 2017. DOI: 10.1109/CEC.2017.7969492.
2. Marco Baioletti, Alfredo Milani, and Valentino Santucci. Linear ordering optimization with a combinatorial differential evolution. In *Proc. of 2015 IEEE International Conference on Systems, Man, and Cybernetics (IEEE SMC 2015)*, pages 2135–2140, 2015. DOI: 10.1109/SMC.2015.373.
3. Marco Baioletti, Alfredo Milani, and Valentino Santucci. An extension of algebraic differential evolution for the linear ordering problem with cumulative costs. In *Proc. of 14th International Conference on Parallel Problem Solving from Nature (PPSN 2016)*, pages 123–133, 2016. DOI: 10.1007/978-3-319-45823-6_12.
4. Marco Baioletti, Alfredo Milani, and Valentino Santucci. Automatic algebraic evolutionary algorithms. In *Artificial Life and Evolutionary Computation*, pages 271–283, Cham, 2018. Springer. DOI: 10.1007/978-3-319-78658-2_20.
5. Marco Baioletti, Alfredo Milani, and Valentino Santucci. MOEA/DEP: An algebraic decomposition-based evolutionary algorithm for the multiobjective permutation flowshop scheduling problem. In *Evolutionary Computation in Combinatorial Optimization*, pages 132–145, Cham, 2018. Springer. DOI: 10.1007/978-3-319-77449-7_9.
6. J. Brest, S. Greiner, B. Boskovic, M. Mernik, and V. Zumer. Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems. *IEEE Transactions on Evolutionary Computation*, 10(6):646–657, 2006.
7. Rónán Daly and Qiang Shen. Learning bayesian networks equivalence with ant colony optimization. *Journ. of Artif. Intell. Res.*, 35:391–447, 2009.
8. S. Das and P. N. Suganthan. Differential evolution: A survey of the state-of-the-art. *IEEE Transactions on Evolutionary Computation*, 15(1):4–31, Feb 2011.
9. Swagatam Das, Sankha Subhra Mullick, and P.N. Suganthan. Recent advances in differential evolution an updated survey. *Swarm and Evol. Comp.*, 27:1 – 30, 2016.
10. Luis M. de Campos, Juan M. Fernández-Luna, José A. Gámez, and Jose Miguel Puerta. Ant colony optimization for learning bayesian networks. *Int. J. Approx. Reasoning*, 31(3):291–311, 2002.

11. Luis M. de Campos, José A. Gámez, and Jose Miguel Puerta. Learning bayesian networks by ant colony optimization: Searching in two different spaces. *Mathware & Soft Computing*, 9(2–3):251–268, 2002.
12. Junzhong Ji, Cuicui Yang, Jiming Liu, Jinduo Liu, and Baocai Yin. A comparative study on swarm intelligence for structure learning of bayesian networks. *Soft Comput.*, 21(22):6713–6738, 2017.
13. R. Kabli, F. Herrmann, and J. McCall. A chain-model genetic algorithm for bayesian network structure learning. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 1264–1271. ACM, 2007.
14. Daphne Koller and Nir Friedman. *Probabilistic Graphical Models Principles and Techniques*. MIT Press, USA, 2009.
15. Shu-Ching Kuo, Hung-Jen Wang, Hsiao-Yi Wei, Chih-Chuan Chen, and Sheng-Tun Li. Applying mdl in pso for learning bayesian networks. In *Fuzzy Systems (FUZZ), 2011 IEEE International Conference on*, pages 1587–1592. IEEE, 2011.
16. Serge Lang. *Algebra*, volume 211. Springer, 2002.
17. Pedro Larrañaga, Cindy M. H. Kuijpers, Roberto H. Murga, Iñaki Inza, and S. Dizdarevic. Genetic algorithms for the travelling salesman problem: A review of representations and operators. *Artif. Intell. Rev.*, 13(2):129–170, 1999.
18. Pedro Larrañaga, Mikel Poza, Yosu Yurramendi, Roberto H. Murga, and Cindy M. H. Kuijpers. Structure learning of bayesian network by genetic algorithms: A performance analysis of control parameters. *IEEE Trans. Pattern Anal. Mach. Intell.*, 18(9):912–926, 1996.
19. K.V. Price, R.M. Storn, and J.A. Lampinen. *Differential evolution: a practical approach to global optimization*. Springer Verlag, 2005.
20. Valentino Santucci, Marco Baioletti, and Alfredo Milani. An algebraic differential evolution for the linear ordering problem. In *Proc. of the Comp. Publ. of the 2015 Annual Conf. on Genetic and Evolutionary Computation (GECCO 2015)*, pages 1479–1480, New York, 2015. ACM. DOI: 10.1145/2739482.2764693.
21. Valentino Santucci, Marco Baioletti, and Alfredo Milani. Algebraic differential evolution algorithm for the permutation flowshop scheduling problem with total flowtime criterion. *IEEE Transactions on Evolutionary Computation*, 20(5):682–694, 2016. DOI: 10.1109/TEVC.2015.2507785.
22. Valentino Santucci, Marco Baioletti, and Alfredo Milani. Solving permutation flowshop scheduling problems with a discrete differential evolution algorithm. *AI Communications*, 29(2):269–286, 2016. DOI: 10.3233/AIC-150695.
23. M Scutari. Learning bayesian networks with the bnlearn r package. *Journal of Statistical Software*, 35(3):1–22, 2010.
24. I. Tsamardinos, L.E. Brown, and C.F. Aliferis. The max-min hill-climbing bayesian network structure learning algorithm. *Machine Learning*, 65(1):31–78, 2006.
25. Steven van Dijk, Linda C. van der Gaag, and Dirk Thierens. A skeleton-based approach to learning bayesian networks from data. In *Knowledge Discovery in Databases: PKDD 2003*, pages 132–143, 2003.
26. Yanghui Wu, John A. W. McCall, and David W. Corne. Two novel ant colony optimization approaches for bayesian network structure learning. In *IEEE Congress on Evolutionary Computation*, pages 1–7, 2010.
27. Heng Xing-Chen, Qin Zheng, Tian Lei, and Shao Li-Ping. Learning bayesian networks structures with discrete particle swarm optimization algorithm. In *IEEE Symp. on Foundations of Computational Intelligence, 2007*, pages 47 – 52, 2007.
28. Cuicui Yang, Junzhong Ji, Jiming Liu, Jinduo Liu, and Baocai Yin. Structural learning of bayesian networks by bacterial foraging optimization. *Int. J. Approx. Reasoning*, 69:147–167, 2016.