Structure Learning of High-Order Dynamic Bayesian Networks via Particle Swarm Optimization with Order Invariant Encoding

David Quesada(✉), Concha Bielza, and Pedro Larrañaga

Artificial Intelligence Department, Universidad Politécnica de Madrid, Madrid, Spain
dquesada@fi.upm.es

Abstract. Dynamic Bayesian networks usually make the assumption that the underlying process they model is first-order Markovian, that is, that the future state is independent of the past given the present. However, there are situations in which this assumption has to be relaxed. When this order increases, the size of the search space grows greatly, not all structure learning algorithms may be suited to learn higher-order networks, and a new appropriate order has to be found. To address the computational issues of huge networks, we propose a structure learning method that uses particle swarm optimization to search in the space of possible structures. To avoid the additional costs of increasing the Markovian order, we provide an order-invariant encoding that represents the networks as vectors of natural numbers whose length remains constant. Due to this encoding, we only need to set a maximum desired order rather than the exact one. Our experimental results show that this method is efficient in high orders and performs better than similar algorithms in both execution time and quality of the obtained networks.

Keywords: Dynamic Bayesian networks · Structure learning · Particle swarm optimization

1 Introduction

In recent years, the use of dynamic Bayesian networks (DBNs) [3] has received more attention in different areas. Their applications range from bioinformatics, where DBNs have been used to model gene regulatory networks [7,19,22], to more industrial fields like damage assessment of steel decks and estimating the remaining useful life of structures [2,12,21].

Traditionally, DBNs have been modelled with the assumption that they are first-order Markovian models, that is, their future state is independent of the past given the present [9]. However, in real world applications we can find processes

This work was partially supported by the Spanish Ministry of Science, Innovation and Universities through the PID2019-109247GB-I00 project.

where this assumption does not hold and the future state is determined not only by the last instant, as in the works of Lo et al. [11] and Vinh et al. [18].

If we increase this order, the model will require several time slices to represent the state of the system instead of only two, making it increasingly complex by adding new nodes and arcs. Given that the number of possible different Bayesian network (BN) structures is super exponential in the number of nodes [14], increasing the Markovian order also increases the search space greatly. For this reason, simply applying a structure learning algorithm for first-order Markovian DBNs in a higher-order space may not be an optimal solution.

Meta-heuristic optimization algorithms can be applied to structure learning by searching over the space of possible network structures and assessing the fitness of each solution with some score. In the case of particle swarm optimization (PSO) algorithms [8], they offer a powerful solution for big search spaces but require them to be continuous and real numbered, and the space of possible network structures does not fulfill these requirements. To fix this issue, some authors have translated the space of possible BN graphs into a continuous one over which PSO can be applied [10]. Many other authors [4,6,15,20] have opted for translating the operations of the PSO algorithm to perform discrete movements and then be able to apply the framework of PSO to BN structure learning. In particular, the work of Du et al. [4] proposes to encode particles as binary adjacency matrices that are modified as the particle moves to represent additions and deletions of arcs. Santos and Maciel [15] extend this concept by defining particles as lists with the parents of each node, and so a particle moving means adding or deleting parent nodes. However, these approaches become less efficient as the number of nodes and the Markovian order increase, and the correct order is assumed to be known beforehand. We will expand on both of those methods by defining an encoding for particles that is unaffected by the Markovian order and allows for more efficient searches in larger spaces.

The rest of the paper is organized as follows. In Sect. 2 we introduce the concepts of high-order DBNs and PSO. Section 3 contains the details of our order invariant DBN structure encoding into PSO particles and its operators. Section 4 shows the empirical results. Finally, Sect. 5 concludes the paper and gives some final remarks.

2 Background

In the field of BN structure learning, one family of methods is dedicated to applying PSO to move through the space of possible structures to find an optimal solution. Depending on the type of BN that we want to model, encoding the individuals and moving through the solution space can be done in different ways. In our case, we will center our attention in DBN models and translating the PSO operations to the discrete space defined by our encoding of a particle.

2.1 High-Order Dynamic Bayesian Networks

Dynamic Bayesian networks [13] are a type of probabilistic graphical model that extend the BN framework to the case of time series. As in the static scenario, a DBN is comprised of a directed acyclic graph that defines its structure and a set of parameters that define the probabilistic relationships of the variables. In the case of DBNs, time is discretized into time slices that represent consecutive instants. Let $\mathbf{X}^t = \{X_0^t, X_1^t, \ldots, X_n^t\}$ be the set of all variables in a single time slice t. For some horizon t = T, we can define the joint probability distribution of the network as:

$$p(\mathbf{X}^T, \mathbf{X}^{T-1}, \ldots, \mathbf{X}^0) = p(\mathbf{X}^{T:0}) = p(\mathbf{X}^T) \prod_{t=T-1}^{0} p(\mathbf{X}^t | \mathbf{X}^{T:t+1}), \qquad (1)$$

where $p(\mathbf{X}) = \prod_{i=1}^{n} p(x_i | \mathbf{Pa}(x_i))$ represents the probability distribution of a set of nodes $\mathbf{X}$ and $\mathbf{Pa}(x_i)$ represents the set of parent nodes of X_i in the graph. Usually, in the literature the oldest instant is the one defined as $\mathbf{X}^0$, but we will reverse this notation and define $\mathbf{X}^0$ as the most recent instant to differentiate it from the others. In a dynamic scenario, nodes can have parents in previous time slices and so all variables in all instants $\mathbf{X}^{T:0}$ have to be taken into account to calculate the joint probability distribution as in Eq. 1. In this situation, a very common assumption is to suppose the DBN to be first-order Markovian [9]. This assumes that the future state of the system is independent of the past given the present, that is, $p(\mathbf{X}^t | \mathbf{X}^{T:t+1}) = p(\mathbf{X}^t | X^{t+1})$. Although the model is greatly simplified, it cannot represent systems where the future state is not determined only by the state of the variables in the last instant. We call a high-order dynamic Bayesian network (HO-DBN) a DBN model where the first-order Markovian assumption is relaxed and the network is represented with more than two time slices.

To leverage the increase in complexity of this kind of model, we can restrict the arcs in the network so that they can only be directed to nodes in the most recent time slice, in our case $\mathbf{X}^0$. This kind of DBN structures are called transition networks [15] and they avoid by definition any kind of cycles, which simplifies the search. The space of possible structures that transition networks allow is also much smaller than that of regular DBN models. To prove this, let G be a DBN network structure and G' be a transition network, both with the same number of nodes n_0 per time slice, total number of nodes n and Markovian order m. Let $\mathbf{D}$ be the set of all possible inter-slice arcs in G and $\mathbf{T}$ be the set of all possible arcs in G'. We can calculate the number of possible DBN structures g^{DBN} as the different combinations of the elements in $\mathbf{D}$, that is, the different combinations of all the possible arcs in the network:

$$g^{DBN} = \sum_{i=0}^{|\mathbf{D}|} \binom{|\mathbf{D}|}{i} = \sum_{i=0}^{|\mathbf{D}|} \frac{|\mathbf{D}|!}{i!(|\mathbf{D}| - i)!}. \qquad (2)$$

By definition, $|\mathbf{T}| \leq |\mathbf{D}|$ because even though both have the same number of nodes, the arcs in $\mathbf{T}$ are restricted in a way that satisfies $\mathbf{T} \subseteq \mathbf{D}$. If we apply

Eq. (2) to calculate the number of possible transition networks g^{TN}, we can see that:

$$|\mathbf{T}| < |\mathbf{D}| \implies g^{TN} << g^{DBN}. \tag{3}$$

If we would also take into account the possible intra-slice arcs in $\mathbf{D}$, the inequality in Eq. (3) would be super-exponentially bigger. Moreover, increasing m by one translates into adding only n_0^2 arcs in the case of the transition network, as we only allow arcs from the new n_0 nodes to the nodes in $\mathbf{X}^0$. This means that increasing m by one increases $|\mathbf{T}|$ by the constant n_0^2. On the other hand, this operation means adding $n_0 * n$ new inter-slice arcs in the case of a regular DBN, which increases $|\mathbf{D}|$ exponentially with each increase in m. This shows that not only $|\mathbf{T}|$ is always smaller than $|\mathbf{D}|$, but it also increases drastically slower when we increase the Markovian order of the network. For both the lack of cycles in transition networks and their reduced, although still vast, space of possible structures we will be using this kind of network in the rest of the paper. An example of a HO-DBN with the restrictions of a transition network can be seen in Fig. 1.

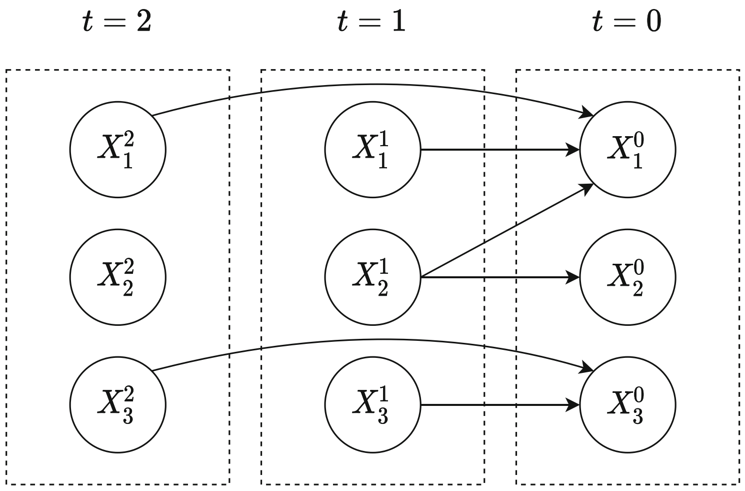

Fig. 1. An example of an order 2 Markovian transition network with tree nodes per time slice. Only arcs directed to t_0 from earlier time slices are allowed.

2.2 Particle Swarm Structure Learning

The PSO algorithm [8] is a meta-heuristic technique that simulates a swarm consisting of n particles moving in a k-dimensional space to find the optimal solution. Particles can be defined as $\mathbf{Pr}_i = \{P_i, V_i, P_l, P_g\}$, where P_i is its current position, V_i is its current velocity, P_l represents the best position found by the particle so far and P_g is the best position found by the whole swarm. A position represents one specific solution of the optimization problem, and the velocities modify these positions, allowing the particles to move in the solution space. To calculate in each iteration t the next position P_i^{t+1} and the next velocity V_i^{t+1} of each particle, the following updating rules are applied:

$$V_i^{t+1} = wV_i^t + c_1 r_1 (P_l - P_i^t) + c_2 r_2 (P_g - P_i^t), \tag{4}$$

$$P_i^{t+1} = P_i^t + V_i^{t+1}, \tag{5}$$

where $w, c_1, c_2 \in \mathbb{R}$, w is the inertia factor of the last velocity, c_1 is the factor that weighs the importance of the local best position, c_2 weighs the global best position and r_1 and r_2 are two real numbers sampled uniformly from the interval $[0, 1]$. One of the key factors in PSO is the pondered effects that the global and local best positions have on the current velocity of a particle, which can change in each iteration if a particle finds a position that has a better score than P_l or P_g. The inertia factor defines how much importance is given to the random search of each particle, increasing or decreasing the exploratory capabilities of the swarm. A higher inertia factor will mean that the $t+1$ velocity of the particle will be very similar to the one it had in the previous instant t.

Although PSO was originally designed for continuous and real valued spaces, there are adaptations to discrete scenarios. In particular, the approach established by Du et al. [4] defines positions and velocities as the binary adjacency matrix of a BN. In this case, velocities matrices can take any value from the set $\{-1, 0, 1\}$, representing deletions, non modifications or additions of arcs respectively. This same approach is taken by Santos and Maciel [15], but instead of adjacency matrices they define a structure called *causality list* that establishes positions and velocities as sets of parent nodes.

To be able to apply PSO to the problem of learning DBN structures, we need a score that measures how likely it is that a network structure fits some data. Let $\mathcal{D}$ be our training data and let G be the network structure represented by a position. Our objective can now be defined as:

$$\arg\max_{G \in g^{DBN}} score(G, \mathcal{D}). \tag{6}$$

This score of fitness is necessary to assess which particles in the solution space fit the training data better and guide the exploration towards them. There are many examples of scores in the literature, such as the Bayesian information criterion (BIC) [16] and the Akaike information criterion (AIC) [1] scores for discrete networks, or the Bayesian Gaussian equivalent (BGe) [5] score and the adapted BIC and AIC scores for Gaussian Bayesian networks. Depending on the type of score used, the same algorithms can be used to learn both discrete and Gaussian Bayesian networks.

3 Encoding and Operators

One of the crucial elements of meta-heuristic optimization algorithms is the encoding used. A suboptimal encoding could generate losses in efficiency by having redundant solutions, having to fix invalid individuals each iteration or by not allowing an even exploration of the solution space.

Our proposed encoding maps each possible transition network structure to a vector of natural numbers. This mapping is bijective in both sets: a transition network structure can only be represented with one specific vector, and a vector only represents one specific transition network structure.

3.1 Natural Vector Encoding

In a particle swarm scenario, each particle has a position and a velocity. In our case, positions represent specific transition network structures and velocities represent additions and deletions of arcs.

In a transition network, there are several nodes representing the same variable in different instants of time. We define the concept of a temporal family of nodes $\mathbf{X}_i^f = \{X_i^0, X_i^1, \ldots, X_i^T\}$ as the set of nodes representing a single variable X_i in all existing time slices of the network. Inside a temporal family, we call a *receiving node* the node X_i^0 in the present time slice. This node is the only one in a temporal family that can have arcs pointing to him from any other node in earlier time slices. In our encoding, we will divide a vector in as many sections as receiving nodes X_i^0 there are in the network. Each of these sections is further subdivided into a subsection consisting of a single natural number for each existing temporal family $\mathbf{X}_i^f$ in the network. This number defines with its binary representation the existing arcs from a certain temporal family to a specific receiving node. Each 1-bit encodes an arc from a specific member of the temporal family to the receiving node. By definition, this encoding does not allow invalid individuals because only receiving nodes can have arcs pointing to them and no cycles can appear. Furthermore, the length of the encoded vectors only depends on the number of existing receiving nodes, and not on the Markovian order. Higher orders will only mean bigger natural numbers in the vector. To clarify this explanation, an example of a position and the network it encodes can be seen in Fig. 2.

The velocities follow the same encoding, but represent arc additions or deletions instead of the presence or absence of an arc. Each velocity is composed of two vectors, $\mathbf{V}_p$ and $\mathbf{V}_n$, defining additions and deletions of arcs respectively.

3.2 Position and Velocity Operators

To perform the operations of the PSO, we will adapt them to the discrete space defined by our encoding. In essence, we will need to be able to add positions and velocities, subtract positions and multiply velocities by real numbers. All operators shown are supposed to be bitwise logical operations. Both positions and velocities have the same vector length, so when operated together this bitwise

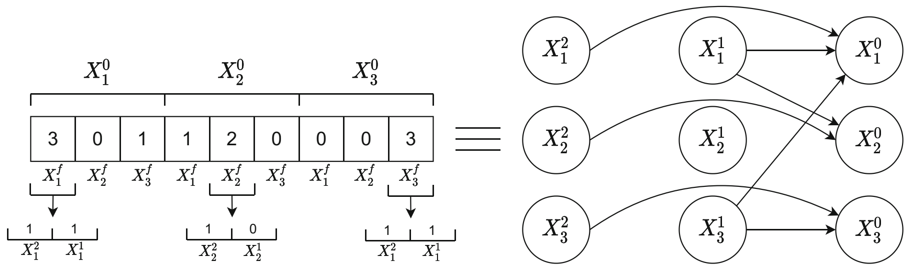

Fig. 2. Representation of a position natural vector on the left and its equivalent transition network on the right. The transition network has three nodes per time slice and a Markovian order 2 for simplicity and clarity. Notice how increasing the order, thus adding many possible nodes and arcs, only implies bigger natural numbers in the vector, but it does not increase its length. For example, increasing the order up to 3 in the figure would only mean that natural numbers up to 7 can now appear in the vector.

operations will be performed throughout both vectors to each pair of natural numbers.

Position Plus Velocity
To add a velocity to a position, first we add all the arcs in $\mathbf{V}_p$ by performing a logical 'or' in the form of $\mathbf{P'} = \mathbf{P} \vee \mathbf{V}_p$. As for the negative part, we define the $\ominus$ operator as:

$$x_1 \ominus x_2 = x_1 \wedge \neg x_2. \tag{7}$$

This operator is equivalent to a 1-bit subtractor without borrow. By performing $\mathbf{P'} \ominus \mathbf{V}_n$, we remove the 1-bits that are present in both the position and negative velocity temporal families and maintain the rest unaffected. The consecutive positive and negative operations will add and remove the marked arcs of the velocity in the original position. An example of this operation can be seen in Fig. 3.

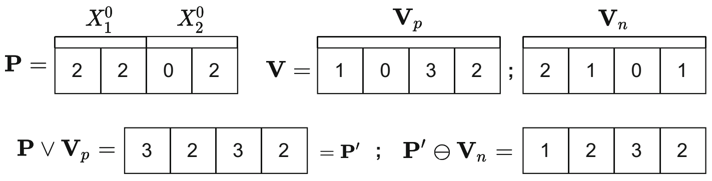
X_1^0 X_2^0
$\mathbf{P} =$ 2 2 0 2
$\mathbf{V}_p$
$\mathbf{V}_n$
$\mathbf{V} =$ 1 0 3 2 ; 2 1 0 1
$\mathbf{P} \vee \mathbf{V}_p =$ 3 2 3 2 $= \mathbf{P'}$;
$\mathbf{P'} \ominus \mathbf{V}_n =$ 1 2 3 2

Fig. 3. An example of adding a position and a velocity encoding a network with two receiving variables X_0^0 and X_1^0 and maximum Markovian order 2 for simplicity. All the operations are performed bitwise on the natural numbers of all vectors.

Addition of Velocities
Given two velocities $\mathbf{V}^1$ and $\mathbf{V}^2$, to add them we first need to combine their positive and negative parts. For this, we operate $\mathbf{V}'_p = \mathbf{V}^1_p \vee \mathbf{V}^2_p$ and $\mathbf{V}'_n = \mathbf{V}^1_n \vee \mathbf{V}^2_n$. Afterwards, we perform a bitwise logical 'and' operation to identify redundancies in the form of $\mathbf{Rd} = \mathbf{V}'_p \wedge \mathbf{V}'_n$. Any 1-bit present in $\mathbf{Rd}$ means that an arc is being added and deleted at the same time in the resulting velocity, and it has to be set to 0 in both positive and negative vectors of $\mathbf{V}'$ with the 'xor' operator by performing $\mathbf{V}'_p \oplus \mathbf{Rd}$ and $\mathbf{V}'_n \oplus \mathbf{Rd}$ respectively. An example of this operation is shown in the following lines:

$$\begin{aligned}
&\mathbf{V}^1 = [1,0,2,0];[2,0,0,3], \mathbf{V}^2 = [0,1,2,1];[0,2,0,2],\\
&\mathbf{V}^1 \vee \mathbf{V}^2 = [1,1,2,1];[2,2,0,3] = \mathbf{V}',\\
&\mathbf{V}'_p \wedge \mathbf{V}'_n = [0,0,0,1] = \mathbf{Rd},\\
&\mathbf{V}' \oplus \mathbf{Rd} = [1,1,2,0];[2,2,0,2].
\end{aligned}$$

Subtraction of Positions
Given two positions $\mathbf{P}_1$ and $\mathbf{P}_2$, the operation $\mathbf{P}_1 - \mathbf{P}_2 = \mathbf{V}'$ returns the velocity $\mathbf{V}'$ such that $\mathbf{P}_1 + \mathbf{V}' = \mathbf{P}_2$. This effect is obtained by using the operator $\ominus$ defined in Eq. (7) to calculate both the positive part $\mathbf{V}'_p = \mathbf{P}_2 \ominus \mathbf{P}_1$ and the negative part $\mathbf{V}'_n = \mathbf{P}_1 \ominus \mathbf{P}_2$ of the velocity. Notice that the $\ominus$ operator is not commutative, and so the inverted positions in each operation give different results. The $\ominus$ operator can be used to get the bits that need to be added to transform a position into another. An example to clarify this operation is shown in the following lines:

$$\begin{aligned}
&\mathbf{P}_1 = [1,0,2,1], \mathbf{P}_2 = [1,1,0,3],\\
&\mathbf{V}'_p = \mathbf{P}_2 \ominus \mathbf{P}_1 = [0,1,0,2],\\
&\mathbf{V}'_n = \mathbf{P}_1 \ominus \mathbf{P}_2 = [0,0,2,0],\\
&\mathbf{V}' = [0,1,0,2];[0,0,2,0].
\end{aligned}$$

Multiplication of Velocities by Real Numbers
Let $|\mathbf{V}|$ be the total population count in a velocity, that is, the total number of 1-bits in both positive and negative vectors. As proposed by [15], multiplying a velocity by a real number increases or decreases $|\mathbf{V}|$ in the form of $\lfloor \alpha * |\mathbf{V}| \rfloor = |\mathbf{V}|'$. This means that we will randomly add or delete 1-bits in the velocity until the new total number of operations is obtained. We will follow a uniform distribution when sampling a temporal family in the vector and the open bits in the natural numbers. If $\alpha < 0$, $\mathbf{V}_p$ and $\mathbf{V}_n$ will be swapped with each other to invert all additions and deletions of arcs in the velocity and the absolute value of α will be used.

4 Results

In this section we will first discuss the implementation of our PSO structure learning algorithm (natPSOHO) and compare it with two other algorithms: the PSO for HO-DBNs proposed by Santos and Maciel [15] and a variation of the dynamic max-min hill climbing algorithm [17]. This comparison will consist of recovering several synthetic randomly generated networks from sampled datasets, evaluating the execution time and how many of the original arcs are recovered from the data. The Markovian order and the number of receiving nodes will vary, to assess the efficiency and precision of the algorithms as both these factors increase. The number of iterations of the PSO algorithms is set to 50 with populations of 300 particles, and all the datasets will consist of 10.000 instances. These parameters remain constant through all the experiments.

4.1 Implementation

All the algorithms have been implemented in R and C++. The code of the natPSOHO algorithm, the experiments and the generation of the synthetic datasets have been combined into an R package that is publicly available in a *GitHub* repository[1]. All experiments were conducted on an Ubuntu 18 machine with an Intel i7-4790K processor and 16 Gb of RAM.

Due to the translation of the position and velocity operations to our specific encoding, we can use the normal pipeline of the PSO algorithm shown in Fig. 4 without any change. For the evaluation of the positions based on the dataset, we will use the BGe score.

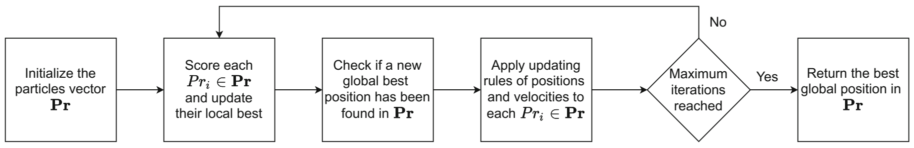
Initialize the particles vector **Pr**
Score each $Pr_i \in \mathbf{Pr}$ and update their local best
Check if a new global best position has been found in **Pr**
Apply updating rules of positions and velocities to each $Pr_i \in \mathbf{Pr}$
Maximum iterations reached
No
Yes
Return the best global position in **Pr**

Fig. 4. Pipeline of the PSO algorithm. The particles move applying the updating rules described in Sect. 2.2 and the operators described in Sect. 3.2. The position with the best fitness is translated into its DBN equivalent and returned as the best solution found.

As recommended in other PSO works [10], the inertia value w is set high at the beginning and slowly decreases as the iterations advance to favour exploration at first, and c_1 is set high and decreases over time while c_2 is set low and increases over time, so that the positions close to the global optimum are properly explored prior to finishing the execution.

[1] https://github.com/dkesada/natPSOHO.

4.2 Experimental Comparison

The results for Markovian orders 1 and 2 networks can be found in Table 1. We can see that for smaller networks with few nodes and low Markovian order, the DMMHC algorithm is much faster than the particle swarm ones, but its execution time scales rapidly with the number of variables. We will not be testing the DMMHC algorithm on higher orders, given its poor scalability and the similar performance to the other two algorithms in terms of number of real arcs recovered. On the other hand, the natPSOHO algorithm is less efficient in time than the binary PSOHO for low-order networks and many receiving nodes, but it consistently outperforms the other two algorithms in terms of real recovered arcs.

Table 1. Results for low-order networks

$[\text{Order}, n_0, \text{Arcs}]$	Algorithm	Rec. arcs	Exec. time
$[1, 10, 47]$	natPSOHO	47	1.49 m
	PSOHO	42	1.55 m
	DMMHC	36	0.22 s
$[1, 15, 111]$	natPSOHO	96	3.24 m
	PSOHO	78	2.81 m
	DMMHC	69	2.54 s
$[1, 20, 201]$	natPSOHO	152	5.81 m
	PSOHO	118	4.22 m
	DMMHC	100	42.21 s
$[2, 10, 96]$	natPSOHO	86	3.37 m
	PSOHO	68	2.88 m
	DMMHC	71	24.1 s
$[2, 15, 234]$	natPSOHO	175	7.1 m
	PSOHO	139	12.68 m
	DMMHC	132	40.73 m
$[2, 20, 398]$	natPSOHO	288	14.24 m
	PSOHO	215	23.15 m
	DMMHC	233	19.1 h

The results for recovering high-order networks from data can be seen in Table 2 and in Fig. 5. We can see how the execution time for the natPSOHO algorithm scales better than the other method as we increase the Markovian order of the networks. We can also see that the number of recovered arcs is consistently higher in the case of the natPSOHO algorithm. In order to evaluate the networks, the BGe score is used to find the fitness of the particles. We have enhanced this score by omitting the scoring of nodes outside of t_0, due to the

Table 2. Results for high-order networks

[Order, n_0, Arcs]	Algorithm	Rec. arcs	Exec. time
[3, 10, 147]	natPSOHO	125	4.93 m
	PSOHO	85	7.56 m
[3, 20, 616]	natPSOHO	398	23.8 m
	PSOHO	308	37.06 m
[4, 10, 208]	natPSOHO	157	6.87 m
	PSOHO	110	11.06 m
[4, 20, 825]	natPSOHO	533	38.02 m
	PSOHO	423	1.04 h
[5, 10, 247]	natPSOHO	181	9.23 m
	PSOHO	148	16.87 m
[5, 20, 982]	natPSOHO	622	57.45 m
	PSOHO	425	1.3 h
[6, 10, 294]	natPSOHO	208	12.43 m
	PSOHO	170	21.6 m
[6, 20, 1171]	natPSOHO	739	1.4 h
	PSOHO	517	2.04 h

fact that in transition networks these nodes never have parents and their structure remains constant. Regardless of this, the score takes longer to compute for networks with a higher number of arcs. Given that the natPSOHO algorithm consistently recovers more arcs from the real networks that generated the synthetic data, the execution time of the score is also consistently higher. This makes it impossible to achieve a constant execution time on the evaluation of the networks, but the constant execution time in the operations of the particles is shown in the overall better performance of the algorithm.

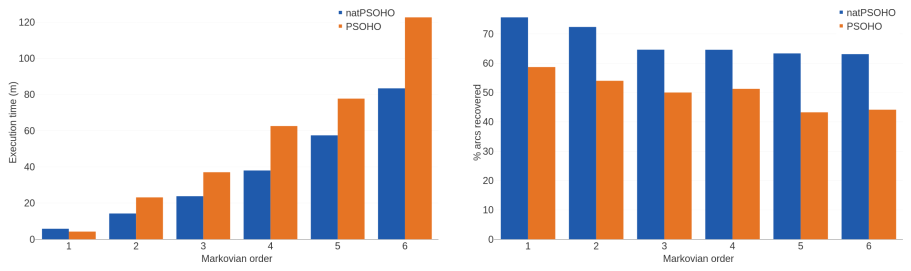

Fig. 5. Execution time in minutes and percentage of recovered arcs of both PSO algorithms with 300 particles, 50 iterations and 20 receiving nodes when learning transition networks as we increase the Markovian order.

The percentage of recovered arcs decreases in both algorithms as we increase the order due to the PSO not being close to convergence. The solutions obtained are the best ones found after 50 iterations, but it is expected that the algorithms did not yet converge to a solution due to the number of iterations being constant in all experiments. As the size of the search space increases, the number of iterations should also increase accordingly.

5 Conclusions

We have presented a new high-order dynamic Bayesian network structure learning algorithm that employs particle swarm optimization to find the network structure that best fits the training data provided. Its order invariant encoding allows a good scalability to bigger networks and high Markovian orders. It also offers the possibility to search up to a maximum desired order rather than having to specify it beforehand.

When learning high-order networks, the search space becomes huge rapidly. Algorithms that rely on independence tests, like the DMMHC algorithm, can become unfeasible in terms of execution time due to the high number of variables and the datasets with thousands of instances. On the other hand, particle swarm algorithms can scale well to finding solutions in bigger search spaces, but suffer slightly in smaller ones.

The execution time of the proposed natPSOHO algorithm scales much better in high orders due to the underlying data structures being constant in size and the operations being performed bitwise. It is shown to be an effective algorithm when dealing with processes with big search spaces generated by high-order networks. In situations where the exact Markovian order is not known beforehand, a higher number of iterations and a maximum desired order can be provided and the algorithm will search for a fitting network in that scenario.

In future work, we would like to refine our encoding and generalize it to be used with any meta-heuristic algorithm, not only with PSO. The main issue that it presents is that even though it relies on vectors of natural numbers, the operators must be bitwise. This would require a transition to a different encoding that is able to take advantage of the natural numbers, like the Gray code, or a generalization of the bitwise treatment before being able to extend it to other frameworks.

References

1. Akaike, H.: Information theory and an extension of the maximum likelihood principle. In: Parzen, E., Tanabe, K., Kitagawa, G. (eds.) Selected Papers of Hirotugu Akaike. SSS, pp. 199–213. Springer, New York (1998). https://doi.org/10.1007/978-1-4612-1694-0_15
2. Cai, B., et al.: Remaining useful life estimation of structure systems under the influence of multiple causes: subsea pipelines as a case study. IEEE Trans. Ind. Electron. **67**(7), 5737–5747 (2019)

3. Dean, T., Kanazawa, K.: A model for reasoning about persistence and causation. Comput. Intell. **5**(2), 142–150 (1989)
4. Du, T., Zhang, S.S., Wang, Z.: Efficient learning Bayesian networks using PSO. In: Hao, Y., et al. (eds.) CIS 2005. LNCS (LNAI), vol. 3801, pp. 151–156. Springer, Heidelberg (2005). https://doi.org/10.1007/11596448_22
5. Geiger, D., Heckerman, D.: Learning Gaussian networks. In: Uncertainty in Artificial Intelligence Proceedings 1994, pp. 235–243. Elsevier (1994)
6. Gheisari, S., Meybodi, M.R.: BNC-PSO: structure learning of Bayesian networks by particle swarm optimization. Inf. Sci. **348**, 272–289 (2016)
7. Godsey, B.: Improved inference of gene regulatory networks through integrated Bayesian clustering and dynamic modeling of time-course expression data. PloS One **8**(7) (2013)
8. Kennedy, J., Eberhart, R.: Particle swarm optimization. In: Proceedings of 1995 IEEE International Conference on Neural Networks, vol. 4, pp. 1942–1948. IEEE (1995)
9. Koller, D., Friedman, N.: Probabilistic Graphical Models: Principles and Techniques. The MIT Press, Cambridge (2009)
10. Liu, X., Liu, X.: Structure learning of Bayesian networks by continuous particle swarm optimization algorithms. J. Stat. Comput. Simul. **88**(8), 1528–1556 (2018)
11. Lo, L.Y., Wong, M.L., Lee, K.H., Leung, K.S.: High-order dynamic Bayesian network learning with hidden common causes for causal gene regulatory network. BMC Bioinform. **16**(1), 1–28 (2015)
12. Ma, Y., Wang, L., Zhang, J., Xiang, Y., Liu, Y.: Bridge remaining strength prediction integrated with Bayesian network and in situ load testing. J. Bridge Eng. **19**(10) (2014)
13. Murphy, K.P.: Dynamic Bayesian Networks: Representation, Inference and Learning (2002)
14. Robinson, R.W.: Counting unlabeled acyclic digraphs. In: Little, C.H.C. (ed.) Combinatorial Mathematics V. LNM, vol. 622, pp. 28–43. Springer, Heidelberg (1977). https://doi.org/10.1007/BFb0069178
15. Santos, F.P., Maciel, C.D.: A PSO approach for learning transition structures of higher-order dynamic Bayesian networks. In: 5th ISSNIP-IEEE Biosignals and Biorobotics Conference: Biosignals and Robotics for Better and Safer Living, pp. 1–6. IEEE (2014)
16. Schwarz, G.: Estimating the dimension of a model. Ann. Stat. **6**(2), 461–464 (1978)
17. Trabelsi, G., Leray, P., Ben Ayed, M., Alimi, A.M.: Dynamic MMHC: a local search algorithm for dynamic Bayesian network structure learning. In: Tucker, A., Höppner, F., Siebes, A., Swift, S. (eds.) IDA 2013. LNCS, vol. 8207, pp. 392–403. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-41398-8_34
18. Vinh, N.X., Chetty, M., Coppel, R., Wangikar, P.P.: Gene regulatory network modeling via global optimization of high-order dynamic Bayesian network. BMC Bioinform. **13**(1), 1–16 (2012)
19. Wang, Y., Berceli, S.A., Garbey, M., Wu, R.: Inference of gene regulatory network through adaptive dynamic Bayesian network modeling. In: Zhang, L., Chen, D.-G.D., Jiang, H., Li, G., Quan, H. (eds.) Contemporary Biostatistics with Biopharmaceutical Applications. IBSS, pp. 91–113. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-15310-6_5
20. Xing-Chen, H., Zheng, Q., Lei, T., Shao, L.P.: Research on structure learning of dynamic Bayesian networks by particle swarm optimization. In: 2007 IEEE Symposium on Artificial Life, pp. 85–91 (2007)

21. Zhu, J., Zhang, W., Li, X.: Fatigue damage assessment of orthotropic steel deck using dynamic Bayesian networks. Int. J. Fatigue **118**, 44–53 (2019)
22. Zou, M., Conzen, S.D.: A new dynamic Bayesian network (DBN) approach for identifying gene regulatory networks from time course microarray data. Bioinformatics **21**(1), 71–79 (2005)