International Journal of Approximate Reasoning 80 (2017) 412–427

Contents lists available at ScienceDirect

International Journal of Approximate Reasoning

www.elsevier.com/locate/ijar

Efficient learning of Bayesian networks with bounded tree-width

Siqi Nie [a], Cassio P. de Campos [b], Qiang Ji [a,*]

[a] Rensselaer Polytechnic Institute, Troy, NY, USA
[b] Queen's University Belfast, Belfast, UK

ARTICLE INFO

Article history:
Received 16 November 2015
Received in revised form 5 July 2016
Accepted 6 July 2016
Available online 11 July 2016

Keywords:
Bayesian network
Structure learning
Bounded tree-width
Hill climbing

ABSTRACT

Learning Bayesian networks with bounded tree-width has attracted much attention recently, because low tree-width allows exact inference to be performed efficiently. Some existing methods [24,29] tackle the problem by using k-trees to learn the optimal Bayesian network with tree-width up to k. Finding the best k-tree, however, is computationally intractable. In this paper, we propose a sampling method to efficiently find representative k-trees by introducing an informative score function to characterize the quality of a k-tree. To further improve the quality of the k-trees, we propose a probabilistic hill climbing approach that locally refines the sampled k-trees. The proposed algorithm can efficiently learn a quality Bayesian network with tree-width at most k. Experimental results demonstrate that our approach is more computationally efficient than the exact methods with comparable accuracy, and outperforms most existing approximate methods.

1. Introduction

Bayesian networks (BNs) use a directed acyclic graph (DAG) to compactly represent the joint probability distribution for multiple variables. The DAG encodes conditional independencies which reduces the number of parameters. Learning BNs from data has been widely studied for decades. In this paper we present our approach of score-based BN structure learning with some special constraint.

The inference problems in BNs, such as querying the probability of some value of a variable conditioned on a configuration of some other variables (belief updating), or finding a configuration of the variables that maximizes the joint probability (MAP inference), are NP-hard to compute exactly [13] or even approximately [15,30]. Existing exact algorithms have worst-case time complexity exponential in the tree-width of the graph [16,20,23,26]. Therefore, for any application that requires fast inferences, it is important to learn networks with small tree-width. Learning a BN with bounded tree-width has received growing attention recently. Besides guaranteed inference complexity, by imposing a hard constraint on the tree-width of the structure, selecting an over-complicated structure is prevented, thus the chance of over-fitting is reduced. Some empirical results [18] demonstrate that bounding the tree-width of a BN achieves better generalization performance.

Several algorithms have been proposed to learn BNs with bounded tree-width. Elidan and Gould [19] designed an approximate algorithm by combining several heuristics to compute the tree-width and to learn the structure of BNs. Korhonen and Parviainen [24] proposed a dynamic programming based algorithm for learning n-node BNs of tree-width at most k

* Corresponding author.
E-mail addresses: nies@rpi.edu (S. Nie), c.decampos@qub.ac.uk (C.P. de Campos), qji@ecse.rpi.edu (Q. Ji).

http://dx.doi.org/10.1016/j.ijar.2016.07.002
0888-613X/

(which we denote as K&P algorithm in this paper). Their algorithm guarantees to find the optimal structure over n nodes maximizing a given score function subject to the tree-width constraint with complexity $O(3^n n^{k+O(1)})$. In practice, it is quite slow for networks with more than 15 nodes, or tree-width more than 3. Parviainen et al. [31] developed an integer programming approach to solve the problem. It iteratively creates a cutting plane on the current solution to avoid exponentially many constraints. Berg et al. [6] transferred the problem into a weighted maximum satisfiability problem and solved it by weighted MAX-SAT solvers. However, all the exact algorithms work only with small networks and small tree-widths. We introduced an exact algorithm based on mixed integer linear programming (MILP) [29] and approximate methods based on k-tree sampling [29,27] to address this problem. In our latest work [28], we further improved the sampling method using the A* search algorithm. Methods have also been proposed to tackle the problem of learning undirected models with bounded tree-width [2,10,35].

In this work, we present a novel method of score-based BN structure learning with bounded tree-width. We design an approximate approach based on sampling k-trees, which are the maximal graphs of tree-width k. The sampling method is based on a fast bijection between k-trees and Dandelion codes [9]. We design a sampling scheme, called *distance preferable sampling* (DPS), in order to effectively cover the space of k-trees using limited samples, in which we give a larger probability for a sample in the unexplored area of the space, based on the existing samples. Smart rules to explore the sample space are essential, because we can only compute a few best structures respecting sampled k-trees in a reasonable amount of time. To evaluate the sampled k-trees, we design an *informative score* (I-score) function to measure the quality of k-trees based on independence tests and BDeu scores. Different from the method proposed in [29], this work focuses on identifying high quality k-trees, instead of uniformly sampling. For each sampled k-tree (represented by a Dandelion code), we first refine it by employing a hill climbing algorithm (HC) to locally identify a code with the largest I-score. One shortcoming of the HC method is that it ends up with a local optimum. To alleviate this issue, we introduce a probabilistic version of the hill climbing method (PHC) to obtain a k-tree of high quality. Once a k-tree is found, both exact [24] and approximate [29] methods are implemented to find the BN as a subgraph of the k-tree.

This paper is structured as follows. We first introduce some definitions and notations for BNs and tree-width in Section 2. Then we discuss the proposed sampling method for learning BNs with bounded tree-width in Section 3. Experimental results are given in Section 4. Finally we conclude the paper in Section 5.

2. Preliminaries

2.1. Learning Bayesian networks

A Bayesian network uses a directed acyclic graph (DAG) to represent a set of random variables $X = \{X_i\}_{i=1}^n$ and their conditional (in)dependencies. Arcs of the DAG encode parent–child relations. Denote X_{pa_i} as the parent set of variable X_i. Conditional probability tables $p(x_i|x_{pa_i})$ are given accordingly, where x_i and x_{pa_i} are instantiations of X_i and X_{pa_i}. We consider categorical variables in this work.

Given a fixed structure G and a complete data set $\mathcal{D} = \{x^{(j)}\}_{j=1}^M$ of points assumed sampled independently from a distribution P on $\mathcal{X}$, the numerical parameters θ of a BN with structure G can be efficiently obtained by maximum likelihood estimation (MLE) by finding θ that maximizes the data log-likelihood according to the model:

$$\mathcal{L}(G, \theta) = \sum_{j=1}^{M} \log P_{G,\theta}(x^{(j)}) . \tag{1}$$

The structure learning task of BNs is to identify the "best" DAG from data. In this paper we consider the score-based BN structure learning problem, in which a score $s(G)$ is assigned to each DAG G. The commonly used score functions (such as BIC [33] and BDeu [8,14,22]) are decomposable, i.e., the overall score can be written as the sum of local score functions,

$$s(G) = \sum_{i=1}^{n} s_i(X_{pa_i}) . \tag{2}$$

For each variable, its score is only related to its parent set. We assume that local scores have been computed in advance and can be retrieved in constant time.

Most score functions penalize model complexity, because increasing the number of parents of a variable never decreases data likelihood, which leads to overfitting and poor generalization. A typical example is the BIC score,

$$s_i(x_{pa_i}) = \mathcal{L}_{i,pa_i} - t_i(x_{pa_i}) \cdot w , \tag{3}$$

where the first term is log-likelihood and the second term is a penalty term to avoid overfitting. $t_i(pa_i)$ is the number of free parameters with respect to the configuration of $\{x_i, x_{pa_i}\}$, and $w = \frac{1}{2}\log M$, where M is the number of data samples. Such penalty term generally leads to structures of small in-degree, but even small in-degree graphs can have large tree-width, which is a problem for subsequent probabilistic inferences with the model. An example is the directed square grid, which has tree-width linear on the number of nodes in the diagonal of the grid, but maximum in-degree fixed in two. A more effective way to constrain the sparsity of the structure is to use a bound for the tree-width, a commonly used measure of the complexity of a graph.

2.2. Learning BN with tree-width bound

In graph theory, the *tree-width* of an undirected graph is related to the *tree decomposition* of the graph.

Definition 1 *(Tree decomposition).* Let $G = (V, E)$ denote an undirected graph, where V is the set of vertices and E is the set of edges. The tree decomposition of G is a tree T, in which each node $t \in T$ represents a subset $V_t \subseteq V$. T must satisfy three properties:

1. Each vertex $v \in V$ belongs to at least one node $t \in T$.
2. For each edge $e \in E$ connecting vertices u and v, where $u, v \in V$, there is a node $t \in T$ with $u, v \in V_t$.
3. For node $t_1, t_2 \in T$ and each node t_3 on the path between t_1 and t_2, if a vertex $v \in V$ belongs to both t_1 and t_2, it also belongs to t_3.

The tree decomposition is a mapping from an undirected graph to a tree. The *width* of a tree decomposition is the size of its largest set V_t minus one. The *tree-width* of an undirected graph is the minimum width among all possible tree decompositions of the graph. We define *tree-width* $tw(G)$ of a DAG G as the tree-width of its *moral graph*.

Definition 2 *(Moral graph).* The moral graph $M(G)$ of a directed graph G is the undirected graph that contains an undirected edge between vertices u and v if:

1. There is a directed edge connecting u and v (in either direction);
2. u and v are both parents of the same vertex.

Existing exact inference algorithms and approximate ones with theoretical guarantees have worst-case complexity exponential in the tree-width of the graph (e.g., the variable elimination algorithm and the junction tree algorithm). In fact, under the assumption that the exponential time hypothesis (ETH) holds, it is shown that there is no algorithm that can solve arbitrary instances of the inference problem in polynomial time [25]. Moreover, if there exists an algorithm with complexity sub-exponential in the tree-width ($c^{O(k/\log k)}$) for an arbitrary inference problem, the ETH fails. The value c is defined as the number of bits needed to describe a BN. Thus, $c^{O(k/\log k)}$ can be regarded as a lower bound of the complexity of exact inference. Therefore, it is necessary to learn a BN with bounded tree-width in order to improve its inference efficiency. By constraining on the tree-width of a graph, the model is simplified, which means there is a trade-off between the representation capability and inference efficiency. We will discuss this issue (i.e., under-fitting issue) in the experiment.

The objective of this work is to find a graph G^*,

$$G^* = \arg\max_G \sum_{i=1}^{n} s_i(X_{pa_i}), \quad \text{s.t.} \quad tw(G) \le k. \tag{4}$$

Directly computing the tree-width of a graph is intractable [1]. One way of imposing the tree-width constraint is to use the family of k-trees.

Definition 3 *(k-trees).* The family of k-trees is defined inductively as follows:

1. A $(k+1)$-clique is a k-tree.
2. Let $G = (V, E)$ denote a k-tree and $C \subseteq V$ denote a set of k vertices. If the induced subgraph $G(C)$ is k-clique, the graph obtained by adding a new vertex v and an edge u–v for each $u \in C$ is a k-tree.

The k-trees are the maximal graphs with tree-width k, and no more edges can be added to them without increasing the tree-width (see [32] for details). Therefore, every graph with tree-width at most k is a subgraph of a k-tree. Learning a BN from a k-tree automatically satisfies the tree-width constraint if we ensure that the moral graph of the learned BN is a subgraph of the k-tree. A k-tree is denoted by T_k and the set of all k-trees over n nodes is denoted by $\mathcal{T}_{n,k}$. It is shown [5] that the total number of k-trees over n variables is,

$$|\mathcal{T}_{n,k}| = \binom{n}{k}(k(n-k)+1)^{n-k-2}. \tag{5}$$

3. Sampling k-trees using Dandelion codes

The basic idea is to efficiently search for k-trees with "high quality" and then use K&P algorithm to learn the optimal BN from the selected k-trees. This is accomplished in three steps. First, we propose a sampling method that can effectively cover the space of k-trees to obtain representative k-trees. Second, we establish an *informative score* (I-score) function to evaluate the quality of each k-tree. Last, we locally refine each sampled k-tree using the hill climbing or probabilistic hill climbing method. Therefore each explored area is represented by a locally optimal k-tree for learning a BN.

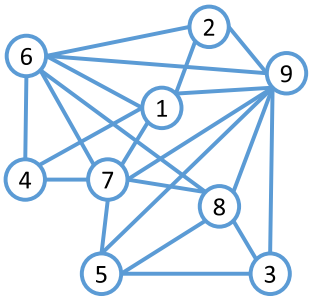

Fig. 1. An example of a k-tree.

3.1. Effective k-tree sampling

Directly sampling a k-tree is not trivial. Caminiti et al. [9] proposed to establish a one-to-one correspondence between a k-tree and what is called *Dandelion* codes.

Definition 4 *(Dandelion codes).* A Dandelion code $A_{n,k}$ is defined as a pair (Q, S), where $Q \subseteq \{1, 2, ..., n\}$ is a set of k integers, and S is a $(n-k-2) \times 2$ matrix of integers whose rows are either (i, j) such that $1 \leq i \leq n-k$ and $1 \leq j \leq k$, or $(0, \epsilon)$ where ϵ is an arbitrary number not in $\{1, 2, ..., n\}$.

Let $\mathcal{A}_{n,k}$ denote the space of Dandelion codes. According to Definition 4,

$$\mathcal{A}_{n,k} = \binom{[1,n]}{k} \times (\{(0,\epsilon)\} \cup ([1, n-k] \times [1,k]))^{n-k-2}, \tag{6}$$

and,

$$|\mathcal{A}_{n,k}| = \binom{n}{k}(k(n-k)+1)^{n-k-2}. \tag{7}$$

Therefore $|\mathcal{A}_{n,k}| = |\mathcal{T}_{n,k}|$.

An example of a Dandelion code of a 3-tree over 9 nodes (that is, $n = 9$, $k = 3$) is $Q = [7, 1, 6]$ and,

$$S = \begin{bmatrix} 1 & 3 \\ 5 & 3 \\ 6 & 2 \\ 1 & 1 \end{bmatrix}. \tag{8}$$

The corresponding k-tree is given in Fig. 1. Due to the complex mapping procedure between Dandelion codes and k-trees, it is difficult to visually establish the correspondence between the code and the structure.

Dandelion codes can be sampled uniformly at random by a trivial linear-time algorithm that uniformly chooses k elements out of N to build Q. To sample the S matrix, each row is set to $(0, \epsilon)$ with a probability of $1/(k(n-k)+1)$. Otherwise, the first and second entry are uniformly sampled from $[1, n-k]$ and $[1, k]$, respectively. Following Definition 4, each sampled code is valid. The sampling procedure for Dandelion codes naturally makes a uniform prior for k-trees, which is a quite good prior in the absence of other prior knowledge [17]. However, uniform sampling generates each sample independently, and totally ignores previous samples, which makes it possible to generate the very same sample twice, or at least samples that are too close to each other. Considering the large size of the space of all Dandelion codes ($\binom{n}{k}(k(n-k)+1)^{n-k-2}$) and the relatively small amount of samples that we can process, we would prefer the samples to be as evenly distributed as possible, which means we want a small number of samples to be more spread out than uniform sampling. This is accomplished by generating the next sample from some currently unexplored area of the sampling space. Driven by this idea, we define the *distance preferable sampling* (DPS). Given the samples of Dandelion codes $A^{(1)}, A^{(2)}, \cdots, A^{(j-1)}$ obtained so far, we want to decide how to sample the next $A^{(j)}$. A kernel density function for a new sample can be defined as

$$q(A^{(j)}) = \frac{1}{j-1}\sum_{i=1}^{j-1} K(\|A^{(j)} - A^{(i)}\|), \tag{9}$$

where $A^{(j)} \in \mathcal{A}_{n,k}$ is the jth Dandelion code sample. $q(A^{(j)})$ depends on all the previous samples, with its value decreasing as $A^{(j)}$ moves away from existing samples. $K(\cdot)$ is a Gaussian kernel function. The distance between two Dandelion codes is defined as

$$\|A^{(j)} - A^{(i)}\| = \|Q^{(j)} - Q^{(i)}\|_2 + \|S^{(j)} - S^{(i)}\|_{2,1}\,, \tag{10}$$

where $\|\cdot\|_2$ is the L_2 norm. $S^{(j)}$ is processed as a $2 \times (n-k-2)$ matrix, and $\|\cdot\|_{2,1}$ is the $L_{2,1}$ norm. For an arbitrary matrix $B \in \mathbb{R}^{r\times p}$, its $L_{2,1}$ norm is

$$\|B\|_{2,1} = \sum_{i=1}^{r} \sqrt{\sum_{j=1}^{p} B_{ij}^2}\,. \tag{11}$$

When computing the distance between two Dandelion codes, we assume the distance between 0 and n is 1 to avoid overflow issue.

Since we intend to explore the regions which have not yet been sampled, we design a proposal distribution as follows:

$$p(A^{(j)}) = 1 - \frac{q(A^{(j)})}{K(0)}\,. \tag{12}$$

$p(A^{(j)})$ increases as sample $A^{(j)}$ moves away from all the existing samples. Following the proposal distribution, we use the rejection sampling algorithm (Algorithm 1) to generate a sample of Dandelion codes, and then employ the implementation of [9] to decode it into a k-tree. One difference between DPS and uniform sampling is that DPS considers the existing samples to decide the next sample while uniform sampling does not. With an unlimited amount of samples, the proposed DPS algorithm is equivalent to uniform sampling.

Algorithm 1 Sampling a Dandelion code using DPS.

Input Previous samples of Dandelion codes $A^{(1)}, \ldots, A^{(j-1)}$.
Output a new sample of Dandelion code $A^{(j)}$.
1: Uniformly sample a Dandelion code $A^{(j)}$ in the feasible region;
2: If $j = 1$, the sample is accepted. If not, the sample is accepted with probability $p(A^{(j)})$;
3: If $A^{(j)}$ is rejected, return to step 1 for another sample, until a sample is accepted.

3.2. Informative score for k-trees

Given a k-tree, the computational complexity of the method in [24] to learn the optimal BN is super-exponential in k ($O(k \cdot 3^k \cdot (k+1)! \cdot n)$). Hence, one cannot hope to use it with too many k-trees, given current computational resources. Instead of learning from every k-tree without distinction, we define the I-score function to evaluate how well a k-tree "fits the data". The I-score of a k-tree T_k is defined as

$$IS(T_k) = \frac{S_{mi}(T_k)}{|S_l(T_k)|}\,. \tag{13}$$

The numerator, $S_{mi}(T_k)$, measures how much information is lost by representing data using the k-tree. Let e_{ij} denote the edge connecting node i and j, and let I_{ij} denote the mutual information of node i and j. Then,

$$S_{mi}(T_k) = \sum_{i,j} I_{ij} - \sum_{e_{ij}\notin T_k} I_{ij}\,. \tag{14}$$

If an edge e_{ij} is not included in the k-tree, we subtract the mutual information corresponding to that edge from the optimal score. S_{mi} is a measurement of the consistency of the k-tree and the data, and can be interpreted either as the sum of the mutual information covered by the k-tree or as constant minus the sum of the mutual information lost by the k-tree. In this work, mutual information is used to indicate how well the k-tree fits the data.

On the other hand, the denominator $S_l(T_k)$ is defined as the score[1] of the best pseudo subgraph of the k-tree by dropping the acyclicity constraint.

$$S_l(T_k) = \max_{m(G)\subseteq T_k} \sum_{i\in N} s_i(x_{pa_i})\,, \tag{15}$$

where $m(G)$ is the moral graph of DAG G. $m(G) \subseteq T_k$ means the moral graph of DAG G is a subgraph of k-tree T_k, and $s_i(x_{pa_i})$ is the local score function for x_i given parent set x_{pa_i}.

The best pseudo subgraph of a k-tree is constructed by choosing the best parent set for each node, compatible with the k-tree, in a greedy way. Combining all the parent sets will result in a directed, possibly cyclic, graph.

A simple example of discovering the best pseudo subgraph of a 2-tree is depicted in Fig. 2. Given a 2-tree with the shape of a triangle, we greedily find the best parent set for each node. In this example, suppose the best parent sets for

[1] Pre-computed score for each node, e.g., BIC, BDeu scores.

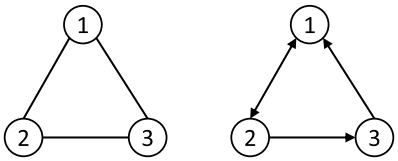

Fig. 2. An example of finding the best pseudo subgraph of a k-tree. The best pseudo subgraph of the 2-tree with 3 nodes (left) is a directed graph with a cycle (right).

nodes $1, 2, 3$ are $\{2, 3\}, \{1\}, \{2\}$, respectively. The best parent set maximizes the local score (BIC, BDeu, et al.) for each node. Combining all the parent sets will result in a directed, possibly cyclic, graph which is the best pseudo subgraph of the 2-tree. The score for this subgraph is calculated as

$$S_l(T_2) = s_1(\{2, 3\}) + s_2(\{1\}) + s_3(\{2\}) . \tag{16}$$

Given the pre-computed scores for each variable, score S_l can be computed in linear time. Typical score functions S_l are negative, so for practical reasons we use the term $1/|S_l(T_k)|$ in the I-score formulation.

The I-score can be very efficiently evaluated for any given k-tree, as computing S_{mi} requires only mutual information of pairs of nodes (which can all be pre-computed, so time complexity is at most $O(n^2)$).

3.3. Hill climbing

The sampling method in Section 3.1 only considers the distances between Dandelion codes to better spread the samples in whole Dandelion code space. It, however, does not ensure the quality of the sampled k-trees. The I-score proposed in Section 3.2 can measure a given k-tree. To combine these two ideas, we propose to refine a sampled Dandelion code using a local search algorithm based on hill climbing. Therefore, the search for the best k-tree (or Dandelion code) can be formulated as a local optimization problem.

Algorithm 2 Hill climbing to refine a Dandelion code.

Input Dandelion code A, mutual information I_{ij}, $\forall i, j \in N$;
Output Dandelion code A^*.
1: Obtain the initial code $A^0 = A$, and $A^* = A$;
2: **while** local optimum is not reached **do**
3: Search in the neighbors of the current code. Find the one with the largest I-score;

$$A^t = \arg \max_{A \in \mathcal{N}(A^{t-1})} IS(A) , \tag{17}$$

4: **if** $IS(A^*) < IS(A^t)$ **then**
5: Update the current best state $A^* = A^t$;
6: **end if**
7: **end while**
8: Return the last code A^*.

Hill climbing algorithm has been used for BN structure learning [36]. In this work, we apply the hill climbing in the space of the Dandelion codes.

Denote $\mathcal{N}(A)$ as all the neighbors of code A,

$$\mathcal{N}(A) = \{A^i : \|A^i - A\| = 1\} . \tag{18}$$

According to the definition of the distance (Eq. (10)), two Dandelion codes are neighbors to each other if they differ by exactly one entry, and all other entries are the same.

The main idea of hill climbing is that given an initial Dandelion code (initial state) resulted from the proposed sampling method, consider all the possible neighbors of the code, and make a move to the next code with the largest I-score improvement. The procedure is repeated until no neighbor has larger score than the current code. The last code is a local optimum with respect to the initial code. The hill climbing algorithm to refine a Dandelion code is summarized in Algorithm 2. Since the Dandelion code A and the k-tree T_k has a bijective relationship, we do not differentiate $IS(A)$ and $IS(T_k)$ in the following text.

Ideally, Dandelion codes close to each other should result in k-trees close to each other. If we use the Hamming distance to measure the closeness of two k-trees, and Eq. (10) to measure the closeness of Dandelion codes, it turns out that the distance is not preserved in the mapping. However, we believe it does not affect the proposed search algorithm. First of all, uniformly sampling a Dandelion code is equivalent to uniformly sampling a k-tree, since there is a one-to-one correspondence between the code and the k-tree. Second, local search in the Dandelion code space only identifies a neighboring code and the quality of the code is determined by the I-score of the corresponding k-tree. Hence, local search of the Dandelion

code improves the I-score of the k-tree, even though this k-tree is not necessarily a neighbor of the current k-tree. The goal of obtaining a k-tree of higher quality is achieved.

One shortcoming with the conventional hill climbing algorithm is that once it reaches a local optimum, it will get stuck there, unless a new starting code is considered. In order to alleviate this issue and have a better chance to reach a better code, we employ a probabilistic version of the hill climbing approach (PHC). When considering all the possible neighbors of the current code, the next move is chosen probabilistically based on their potential to improve the scores. Therefore, instead of finding the path deterministically, PHC probabilistically moves to next code and hence reduces the chance of getting trapped in a local optimum.

Algorithm 3 Probabilistic hill climbing to refine a Dandelion code.

Input Dandelion code A, mutual information I_{ij}, $\forall i, j \in N$;
Output Dandelion code A^*.
1: Initialize the state $A^0 = A$, and $A^* = A$;
2: **while** convergence is not reached **do**
3: **for** $A^i \in \mathcal{N}(A^{t-1})$ **do**
4: Compute the probability P_i of moving to state A^i using Eq. (20);
5: **end for**
6: Sample the next move A^t based on the distribution $\{P_i\}$;
7: **if** $IS(A^*) < IS(A^t)$ **then**
8: Update the current best state $A^* = A^t$;
9: **end if**
10: **end while**
11: Return A^*.

If code A is the current code with I-score s, assume its neighbor A^i has the I-score of s_i. That is,

$$IS(A^i) = s_i\,, \quad \forall A^i \in \mathcal{N}(A)\,. \tag{19}$$

To calculate the probability of moving to a certain state, we employ the Boltzmann distribution, in which the improvement of a potential move over the current code $s_i - s$ is treated as the negative energy function,

$$P_i = \frac{1}{Z} \exp\left(\frac{s_i - s}{T}\right), \tag{20}$$

where Z is the normalization term to ensure a valid probability distribution. T is a small positive number called Boltzmann constant, or temperature, which controls the shape of the distribution. A large value for T makes the distribution close to uniform (maximum entropy), which gives relatively equal chance to move to any neighbor. When using a small value of T, the distribution P_i has the shape of a delta function, strongly favoring the neighbor with the largest I-score, and probabilities for other moves are small. In the extreme case, $T \to 0$, the probability of moving to the neighbor with the highest I-score is 1, and the PHC method degenerates to HC method.

The probabilistic nature of PHC is the key feature to increase the chance of escaping from local optima. It randomly chooses a move based on the potential improvement of each neighbor. The PHC method is terminated if the scores in several consecutive moves do not improve, and the best state along the process is returned as the solution. Algorithm 3 summarizes the PHC algorithm to refine a Dandelion code.

3.4. BN learning from sampled k-trees

Combining the ideas in Sections 3.1, 3.2 and 3.3, we present Algorithm 4 as an approximate algorithm for learning BNs of bounded tree-width.

Due to the fact that k-trees with large I-scores are more likely to have better subgraphs, we give them high priority to learn the corresponding BN. This is reflected in Step 4 of Algorithm 4. Given a k-tree as the super structure, the implementation of K&P is employed as an exact method to learn the optimal BN. The goal of Algorithm 4 is to restrict the calls to K&P (which is a time consuming method with complexity $O(k3^k(k+1)!n)$, even if linear in n) only to k-trees that are promising. Alternatively, the implementation of [29] is used as an approximate method to learn a BN from a k-tree, which is more suitable for larger data sets.

Once a BN is learned, the variable elimination algorithm is used for exact inference, which requires a variable elimination order. When eliminating a variable, cliques may be generated. The *induced width* is defined as the size of the largest clique minus 1. The tree-width of a BN is the minimum induced width among all possible elimination orders. In order to perform exact inference with minimum complexity, an optimal variable elimination order is required. In general, identifying the optimal elimination order is NP-hard. However, learning from super structure (k-tree) automatically encodes an optimal elimination ordering. By converting the k-tree into a clique tree, we just need to eliminate the nodes from the leaves until the root. A node is eliminated when it does not appear in the parent (parent meaning the next clique in the direction of the root clique). Since every clique has the size of k, it provides an elimination ordering with induced width at most k. In fact, because a subgraph is learned, there can be even elimination orderings with induced width less than k, but that

Table 1
Dimensions of data sets used in the experiments.

Data set	Vars	Samples
nursery	9	12960
breast	10	699
housing	14	506
adult	15	32561
zoo	17	101
letter	17	20000
mushroom	22	8124
wdbc	31	569
audio	62	200

might be hard to find. Therefore, the complexity of exact inference is actually achieved without going through all possible orderings.

Algorithm 4 Learning a Bayesian network structure of bounded tree-width by sampling Dandelion codes.

Input score function s_i, $\forall i \in N$, mutual information I_{ij}, $\forall i, j \in N$
Output a DAG G^{best}.
1: Initialize G_i^* as an empty set for all $i \in N$;
2: **while** time limit is not reached **do**
3: Sample a Dandelion code $(Q, S) \in \mathcal{A}_{n,k}$ according to Algorithm 1;
4: Local refinement for the sampled Dandelion code using the HC method (Algorithm 2) or PHC method (Algorithm 3);
5: Transfer the code into a k-tree T_k.
6: Find a DAG G that maximizes the score function and is consistent with T_k;
7: **if** $\sum_{i \in N} s_i(G_i) > \sum_{i \in N} s_i(G_i^*)$ **then**
8: Update $G_i^*, \forall i \in N$.
9: **end if**
10: **end while**

4. Experiments

To empirically evaluate our method, we use a collection of data sets from the UCI repository [3] of varying dimensionality, as well as synthetic and real networks. The experiments consist of four parts. Firstly, we demonstrate the effectiveness of the I-scores of the k-trees by establishing its relationship with the BDeu scores of the learned BNs. Secondly, we compare the BDeu scores of the learned BNs using the proposed methods with the results from the state-of-the-art methods. Thirdly, we investigate the trade-off between inference accuracy and time. Finally, we study the under-fitting issues.

4.1. Informative score

In this section, we evaluate the I-score as a measurement of how good a k-tree would be to "produce" a BN structure as its subgraph. Nine data sets (*nursery, breast, housing, adult, zoo, letter, mushroom,* and *wdbc*), whose dimensions are summarized in Table 1, were used. Non-binary variables were binarized over the median value. Instances with missing values were discarded. In all experiments, we maximized the Bayesian Dirichlet equivalent uniform (BDeu) score with equivalent sample size equal to one [22].

For each k-tree, we compared its I-score and the BDeu score of the BN as the subgraph of the corresponding k-tree. We sampled 1,000 k-trees and discovered the optimal BN subject to the corresponding k-tree. For the *nursery, breast, housing* and *adult* data sets, we used the K&P algorithm to exactly learn the BN subject to the k-tree. For the other data sets, due to the larger number of variables, the exact algorithm fails due to memory limitation issue. We used the approximate algorithm in [29] to learn the BN from k-trees. To empirically show the correlation of the two scores, we estimated the correlation coefficients of the I-scores and the BDeu scores, as reported in Table 2. The scores for two data sets (*breast* and *housing*) with tree-width bound equal to two are plotted in Fig. 3.

As shown in Table 2 and Fig. 3, there is a strong linear positive correlation between the BDeu score of the BN and the I-score of its super-structure (k-tree). The correlation coefficients are over 0.7 in all cases. Generally, better k-trees in terms of I-scores will lead to better BNs in terms of BDeu scores. If the tree-width bound is increased, the linear correlation improves since the learned model under tree-width constraint is more similar to the model learned without tree-width constraint.

4.2. Bayesian network learning

In this section we compare the BDeu scores of the structures learned by our proposed method against the scores from the state-of-the-art approaches.

Table 2
Correlation coefficients of the I-scores and the BDeu scores on different data sets.

Data set	CorrCoef
nursery	0.976
breast	0.932
housing	0.892
adult	0.925
zoo	0.911
letter	0.782
mushroom	0.711
wdbc	0.701
audio	0.741

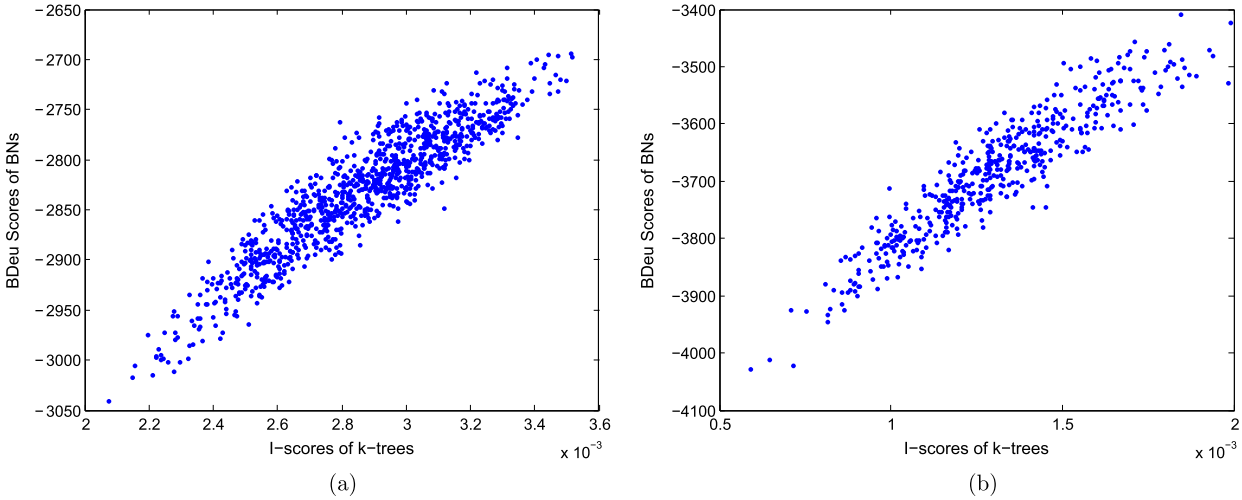

Fig. 3. Relationship of the BDeu scores of Bayesian networks and the I-scores for k-trees. (a) *breast* and (b) *housing* data sets.

We first compare with a baseline method proposed in [24], denoted as the K&P algorithm.[2] K&P method is an exact algorithm based on dynamic programming. Due to the complexity of this method, it is only applicable to some relatively small data sets, hence our comparisons are restricted to those cases. The detailed computational time that K&P uses is given in Table 3. The algorithm is run using a desktop computer with 64 GB of memory. Maximum number of parents is set to three. Due to the huge amount of memory cost, for *housing* and *adult* data sets with tree-width more than 2, as well as *breast* with tree-width bound 5, the algorithm failed to give a solution. Correspondingly, we sampled 100 k-trees using the DPS scheme with PHC and recorded the running time for the proposed algorithm to give a solution, given the same data set and the same choice of maximum tree-width. The BDeu scores of the best BNs found with both algorithms are also presented. By examining only a small portion of k-trees, the proposed algorithm finds solutions with a BDeu score difference less than 1% for most cases. The complexity does not increase with the tree-width. The PHC algorithm is much more efficient than K&P algorithm, except for the smallest two data sets and tree-widths, for which the 100 k-trees are typically identical to each other. Generally speaking, the proposed algorithm achieves comparable results to those of the exact method in terms of BDeu score difference. Yet when considering the time and memory costs of the exact solution, the proposed algorithm is more efficient against the competing method by several orders of magnitude.

It should be noted that in some cases the scores do not always increase with the tree-width bound k, as indicated in Table 3. The optimal BN given the data already has tree-width less than k, setting a larger tree-width bound will not increase the score.

The BDeu scores of the BNs learned using the PHC+S2 algorithm with varying tree-width bounds on larger data sets are reported in Table 4. Generally speaking, increasing tree-width will lead to larger BDeu scores until the actually tree-width does not increase (actual tree-width is smaller than the bound).

Next, we compare the proposed methods against the existing approximate methods. Based on how to obtain a k-tree, these methods include distance preferable sampling (DPS), A* search (A*) [28], hill climbing (HC) and probabilistic hill climbing (PHC). Based on the algorithm to learn a BN from the k-tree, methods are further categorized into exact method (K&P) [24] or approximate method (S2) [29]. Other exact methods are also included, such as the MILP algorithm, the

[2] http://www.cs.helsinki.fi/u/jazkorho/aistats-2013/.

Table 3
Computational time of the K&P method to find the optimal Bayesian network structure, and the proposed method to sample 100 k-trees, as well as the resulting BDeu scores of the networks found by both methods. Empty cells indicate that the method failed to solve the problem because of excessive memory consumption. s, m mean seconds and minutes, respectively.

Method	tw	Time				Score			
		Nursery $n=9$	Breast $n=10$	Housing $n=14$	Adult $n=15$	Nursery $n=9$	Breast $n=10$	Housing $n=14$	Adult $n=15$
K&P	2	7 s	26 s	128 m	137 m	−72160	−2688.4	−3295.4	−201532
	3	72 s	5 m	–	–	−72159	−2685.8	–	–
	4	12 m	103 m	–	–	−72159	−2685.3	–	–
	5	131 m	–	–	–	−72159	–	–	–
PHC	2	4.3 m	4.6 m	9.1 m	10.4 m	−72187	−2690.5	−3341.7	−209123
	3	4.4 m	4.6 m	9.2 m	10.5 m	−72163	−2685.8	−3312.1	−204116
	4	4.4 m	4.7 m	9.2 m	10.5 m	−72159	−2685.3	−3202.9	−200627
	5	4.4 m	4.8 m	9.3 m	10.5 m	−72159	−2685.3	−3202.9	−200613

Table 4
BDeu scores of the BNs learned using PHC+S2 with different tree-width bounds.

Data set	$k=2$	$k=3$	$k=4$	$k=5$
nursery ($\cdot 10^4$)	−7.219	−7.216	−7.216	−7.216
breast ($\cdot 10^3$)	−2.691	−2.686	−2.685	−2.685
housing ($\cdot 10^3$)	−3.341	−3.312	−3.203	−3.203
adult ($\cdot 10^5$)	−2.091	−2.041	−2.006	−2.006
zoo ($\cdot 10^2$)	−7.150	−5.840	−5.770	−5.770
letter ($\cdot 10^5$)	−1.983	−1.879	−1.856	−1.856
mushroom ($\cdot 10^4$)	−7.906	−6.145	−5.974	−5.910
wdbc ($\cdot 10^3$)	−9.132	−7.289	−7.128	−7.125
audio ($\cdot 10^3$)	−2.242	−2.130	−2.120	−2.118

TWILP algorithm[3] [31] and the GOBNILP algorithm[4] [4]. Note that GOBNILP is an exact structure learning algorithm without tree-width constraint. It is included to show how much information is lost by constraining the tree-width of the graph. The running time for the approximate method is set to 10 minutes. The MILP method is given 10 minutes or 3 hours running time. If the exact solution is not found, the current best structure is returned, which is the case in the *mushroom*, *wdbc* and *audio* data sets for the MILP method. The Boltzmann constant T in PHC method is set to 0.05.

The results are reported in Table 5 and Table 6. It can be seen that for the smallest networks (*nursery* and *breast*), most approximate algorithms were able to find a structure that is very close to the global optimal one. The HC and PHC methods exactly found the optimal structures. The PHC method always outperformed HC method, indicating that the probabilistic move in the PHC could help in escaping from local optima and finding better k-trees. In most cases, the PHC method achieved the best performance among all approximate methods, demonstrating its effectiveness. Specifically, compared to the second best approximate method (A* search), the PHC method can improve the accuracy by an average of 0.2%. The relative difference between PHC+S2 and unconstrained structure learning algorithm (GOBNILP) in terms of BDeu scores is less than 8% in all cases. In terms of the ILP-based algorithms, time limit of 10 minutes and 3 hours produce the same structure score for the two ILP-based methods on 5 data sets. In 10 minutes, HPC+S2 has at most 2.2% relative score difference compared to the best ILP-based algorithm. For the largest data set (*audio*), the proposed PHC method even outperforms the exact approaches.

4.3. Inference accuracy and complexity

A BN with low tree-width has guaranteed inference complexity. We constructed two heavily connected BNs with 50 and 60 nodes, *Rand50* and *Rand60* (see Fig. 4 (a) for a illustration), respectively. For simplicity, all the nodes were set binary. The (conditional) probability distribution for each node is randomly generated. Another four real networks (*Barley, Hailfinder, Hepar2*, and *Pathfinder*) were also included in the experiment, one of which (*Pathfinder* network) is shown in Fig. 4 (b). The important details of the networks are shown in Table 7. Besides the six networks, we also evaluated our algorithm on the data sets in Section 4.2, as well as two data sets with 100 random variables, *hill* and *community*. For the data sets without ground truth networks, the GOBNILP algorithm was employed to learn the structure without tree-width bound. PHC+S2 algorithm was used for structure learning with tree-width bound 3, with 10,000 samples generated from these networks.

[3] https://bitbucket.org/twilp/twilp/.
[4] https://www.cs.york.ac.uk/aig/sw/gobnilp/.

Table 5
Comparison of the performance of different approximate methods with tree-width limit 4 to the exact algorithm without tree-width bound. The mean value in 10 runs with different seeds is reported. The number in the parenthesis is the standard deviation of the scores. Gap is the relative score difference between PHC + S2 and GOBNILP. Empty cells indicate no solution found within the time limit.

Data set	DPS+K&P	A*+S2	HC+S2	PHC+S2	GOBNILP	Gap
nursery ($\cdot 10^4$)	**−7.216** (0)	**−7.216** (0)	**−7.216** (0)	**−7.216** (0)	−7.216	0
breast ($\cdot 10^3$)	−2.691 (0.005)	−2.686 (0.003)	**−2.685** (0)	**−2.685** (0)	−2.685	0
housing ($\cdot 10^3$)	−3.285 (0.039)	−3.211 (0.030)	−3.213 (0.035)	**−3.203** (0.032)	−3.159	1.4%
adult ($\cdot 10^5$)	−2.024 (0.026)	−2.007 (0.016)	−2.010 (0.014)	**−2.006** (0.010)	−2.004	0.1%
zoo ($\cdot 10^2$)	−6.091 (0.195)	−5.794 (0.156)	−5.801 (0.168)	**−5.770** (0.133)	−5.615	2.8%
letter ($\cdot 10^5$)	−1.924 (0.050)	−1.860 (0.035)	−1.867 (0.039)	**−1.856** (0.026)	−1.840	0.9%
mushroom ($\cdot 10^4$)	−6.852 (0.219)	−5.999 (0.126)	−6.149 (0.166)	**−5.974** (0.143)	−5.557	7.5%
wdbc ($\cdot 10^3$)	−8.352 (0.342)	−7.143 (0.264)	−7.268 (0.291)	**−7.128** (0.278)	−6.863	3.9%
audio ($\cdot 10^3$)	–	−2.126 (0.111)	−2.203 (0.119)	**−2.120** (0.117)	−2.013	5.3%

Table 6
BDeu scores of the structures learned using PHC+S2 and different exact methods. Tree-width bound is set to 4, except for the Chow–Liu algorithm, which learns a tree (tree-width 1). The mean in 10 runs with different seeds is reported. (*) indicates probably optimal scores.

Data set	PHC+S2	Chow–Liu	MILP$^{10\,m}$	MILP$^{3\,h}$	TWILP$^{10\,m}$	TWILP^{3h}
nursery ($\cdot 10^4$)	−7.216	−7.657	−7.216*	−7.216*	−7.216*	−7.216*
breast ($\cdot 10^3$)	−2.685	−3.877	−2.685*	−2.685*	−2.685*	−2.685*
housing ($\cdot 10^3$)	−3.203	−4.581	−3.159*	−3.159*	−3.269	−3.205
adult ($\cdot 10^5$)	−2.006	−2.153	−2.004*	−2.004*	−2.009	−2.007
zoo ($\cdot 10^2$)	−5.770	−9.690	−5.645	−5.624*	−5.782	−5.782
letter ($\cdot 10^5$)	−1.856	−2.251	−1.863	−1.843*	−1.907	−1.884
mushroom ($\cdot 10^4$)	−5.974	−10.217	−5.898	−5.783	−6.628	−6.381
wdbc ($\cdot 10^3$)	−7.128	−12.163	−7.700	−6.995	−7.831	−7.788
audio ($\cdot 10^3$)	−2.120	−2.355	−2.541	−2.541	−2.254	−2.161

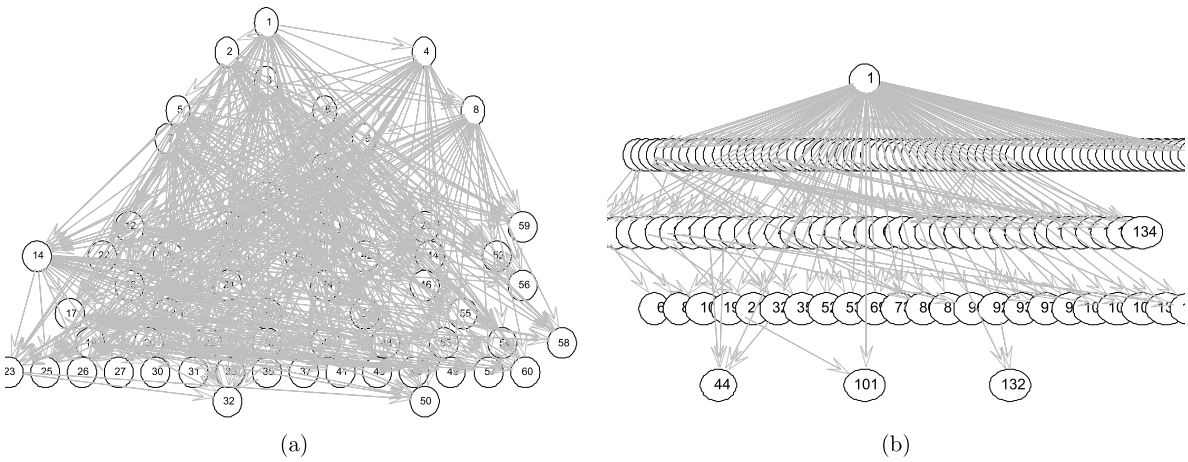

Fig. 4. Illustration of (a) *Rand60* network and (b) *Pathfinder* network.

Specifically, from a ground true graph G_0, we generated samples and learned the structures G_1 with different tree-width bounds. For low tree-width, we learned networks with tree-width 0 (a graph with no edge) and 1 (a tree). For moderate tree-width, we used the PHC+S2 method to learn structures with tree-width bounds 2, 3 and 4. For high tree-width, we used the ground truth networks with unbounded tree-width. Training time limit is set to 30 minutes. Junction tree algorithm is used for exact inference, to which we provide the corresponding optimal elimination order obtained from the k-tree (Section 3.3).

We want to study the trade-off between inference complexity and inference accuracy. We first study the marginal distribution of a single variable. As a measurement of the inference complexity, we use the total running time of 1,000 marginal distribution computation, averaged over all variables. The measurement of model accuracy includes three parts. First, we use the average KL divergence of the marginal distribution of each node, denoted as KL_1,

$$KL_1 = \frac{1}{n}\sum_i KL\left(P(x_i)\|P_0(x_i)\right), \tag{21}$$

Table 7
Six networks (2 synthetic, 4 real) used for the evaluation of the trade-off between inference complexity and accuracy. fan-in and fan-out means maximum number of parents and children.

Network	Nodes	Edges	fan-in	fan-out
Rand50	50	283	7	22
Rand60	60	517	10	41
Barley	48	84	4	5
Hailfinder	56	66	4	16
Hepar2	70	123	6	17
Pathfinder	135	200	4	130

Table 8
Inference accuracy and time of *Rand50* and *Rand60* networks in terms of marginal distribution of a single variable. Tree-width bound is set to 3.

	Rand50		Rand60	
	Learned	Original	Learned	Original
Time (s)	0.17	0.38	0.17	5.27
KL_1	9.96×10^{-4}		9.17×10^{-4}	
D_{abs}	1.29×10^{-3}		6.36×10^{-4}	
D_{rel}	0.21%		0.13%	

Table 9
The inference accuracy and time of real networks and data sets. Tree-width bound is set to 3. Ratio is the ratio of the inference time of learned network to that of the original network. t_0 and t_1 are the average inference time for the original and learned networks, respectively.

Network	KL_1	D_{abs}	D_{rel}	t_0	t_1	Ratio
Barley	0.0174	0.0056	1.08%	0.746	0.199	26.7%
Hailfinder	0.0020	0.0004	1.25%	0.238	0.232	97.7%
Hepar2	0.0010	0.0047	6.04%	0.215	0.205	95.3%
Pathfinder	0.0322	0.0346	6.76%	0.256	0.254	99.4%
nursery	0	0	0%	0.045	0.045	100%
breast	0	0	0%	0.056	0.056	100%
housing	0.0008	0.0038	0.52%	0.113	0.109	96.5%
adult	0.0025	0.0012	0.92%	0.134	0.116	86.6%
zoo	0.0009	0.0004	1.03%	0.110	0.094	85.4%
letter	0.0011	0.0033	0.68%	0.127	0.117	92.1%
mushroom	0.0048	0.0112	1.91%	0.247	0.139	56.3%
wdbc	0.0021	0.0043	0.88%	0.285	0.184	64.6%
audio	0.0024	0.0032	1.34%	0.379	0.264	69.7%
hill	0.0078	0.0069	1.21%	0.256	0.221	86.3%
community	0.0061	0.0045	1.03%	0.452	0.311	68.8%

where n is the number of variables; $p(x_i)$ and $p_0(x_i)$ are the estimated and true marginal distribution for node x_i, respectively.

To have an intuitive understanding of the KL divergence, we use the absolute and relative distance of the marginal distributions as the second and third measurements of accuracy,

$$D_{abs} = \frac{1}{n} \sum_i \| P(x_i) - P_0(x_i) \|_1 , \tag{22}$$

$$D_{rel} = \frac{1}{n} \sum_i \| \frac{P(x_i) - P_0(x_i)}{P_0(x_i)} \|_1 , \tag{23}$$

where $\| \cdot \|_1$ is the L_1 norm. For each variable, it is computed as the summation of all values. The results are then averaged over all variables.

The inference time is measured by the ratio of the running time of the learned network to that of the original network. The results are given in Tables 8 and 9. For *Rand50*, *Rand60* and *Barley* networks, there is a huge saving in the inference time (44.2%, 3.2% and 27.1%, respectively). The inference complexity is significantly reduced using the proposed algorithm, while the accuracy of marginals is well maintained. The average marginal distribution of the learned network is merely 0.21% away from the true distribution for *Rand50* network, 0.13% for *Rand60* and 1.08% for *Barley*, which means the learned network is close to the original network in terms of the marginal distribution of a single variable. For the data sets, a larger

Table 10
Absolute KL-1 values for different networks with different tree-width bounds.

Network	$k=0$	$k=1$	$k=2$	$k=3$	$k=4$	$k=\infty$
Barley	0.0185	0.0180	0.0174	0.0173	0.0173	0.0173
Hailfinder	0.0023	0.0022	0.0022	0.0022	0.0022	0.0022
Hepar2	0.0012	0.0011	0.0010	0.0010	0.0010	0.0010
Pathfinder	0.0342	0.0336	0.0322	0.0321	0.0321	0.0321
Rand50	0.0012	0.0010	0.0010	0.0010	0.0010	0.0010
Rand60	0.0012	0.0010	0.0009	0.0009	0.0009	0.0009

Table 11
Absolute KL-2 values for different networks with different tree-width bounds.

Network	$k=0$	$k=1$	$k=2$	$k=3$	$k=4$	$k=\infty$
Barley	0.0120	0.0067	0.0065	0.0065	0.0065	0.0065
Hailfinder	0.0153	0.0094	0.0092	0.0092	0.0092	0.0092
Hepar2	0.0147	0.0079	0.0077	0.0078	0.0078	0.0078
Pathfinder	0.0214	0.0105	0.0101	0.0100	0.0102	0.0102
Rand50	0.0110	0.0042	0.0041	0.0040	0.0040	0.0040
Rand60	0.0161	0.0080	0.0074	0.0074	0.0074	0.0074

Table 12
Absolute KL-3 values for different networks with different tree-width bounds.

Network	$k=0$	$k=1$	$k=2$	$k=3$	$k=4$	$k=\infty$
Barley	0.0138	0.0087	0.0075	0.0069	0.0069	0.0069
Hailfinder	0.0109	0.0076	0.0064	0.0063	0.0063	0.0062
Hepar2	0.0082	0.0047	0.0043	0.0041	0.0041	0.0041
Pathfinder	0.0241	0.0114	0.0102	0.0100	0.0095	0.0095
Rand50	0.0148	0.0053	0.0040	0.0039	0.0038	0.0038
Rand60	0.0102	0.0057	0.0048	0.0047	0.0046	0.0046

size of the network results in more significant savings in the running time. Comparing with Table 5, a large score gap in the learned structure does not translate to larger inference gap. For example, the relative score difference for the mushroom data set is 7.5%, the relative different in the marginal distribution is only 1.91%.

The first three networks are very complex. However, if the original network already has a low tree-width, there will not be much savings on the inference time. For example, in the *Hailfinder*, *Hepar2* and *Pathfinder* networks, most of the cliques in the junction tree have the size of 2. Therefore, the inference computation in the original network is already efficient, especially for the *Pathfinder* network, which has a tree-like structure, as shown in Fig. 4 (b), even though the network has over 100 nodes. In such cases, bounding the tree-width will not result in considerable reduction of the inference complexity.

Next we evaluate the performance of the proposed learning algorithm in terms of computing the marginal distribution of two or more random variables. Specifically, for every pair of variables x_i and x_j in the network, the KL-2 divergence is computed as,

$$KL_2 = \frac{2}{n(n-1)} \sum_{i,j} KL(P(x_i, x_j) \| P_0(x_i, x_j)) \,. \tag{24}$$

The KL-3 divergence is computed as,

$$KL_3 = \frac{6}{n(n-1)(n-2)} \sum_{i,j,k} KL(P(x_i, x_j, x_k) \| P_0(x_i, x_j, x_k)) \,. \tag{25}$$

The evaluation criterion for inference accuracy is the KL divergence between the learned structure G_1 and the ground truth structure G_0, using the marginal distribution of singletons, pairs and triples of variables. The evaluation criterion for inference complexity is the ratio of inference time for G_1 and G_0. The KL divergence and running time are shown in Fig. 5. The absolute values of the KL divergence are given in Table 10 – Table 12.

For the inference complexity, in all experiments, the time required to perform inference increases with the tree-width bound, unless the tree-width bound reaches the tree-width of the ground truth graphs, which is the case for the *Hailfinder*, *Hepar2* and *Pathfinder* networks.

For the inference accuracy, generally speaking, increasing the tree-width from 0 to 3 can improve the inference accuracy. Tree-width 3, 4 and unbounded tree-width have similar performances, indicating that networks with moderate tree-width are nearly as good as the ones with high tree-width for the data sets we used. One thing to mention is that for two of the networks (*Barley* and *Pathfinder*), an empty network (tree-width 0) achieves the KL-divergence 7% worse than the tree structure (tree-width 1). This is because the empty network assumes variables to be independent, and there are no complex

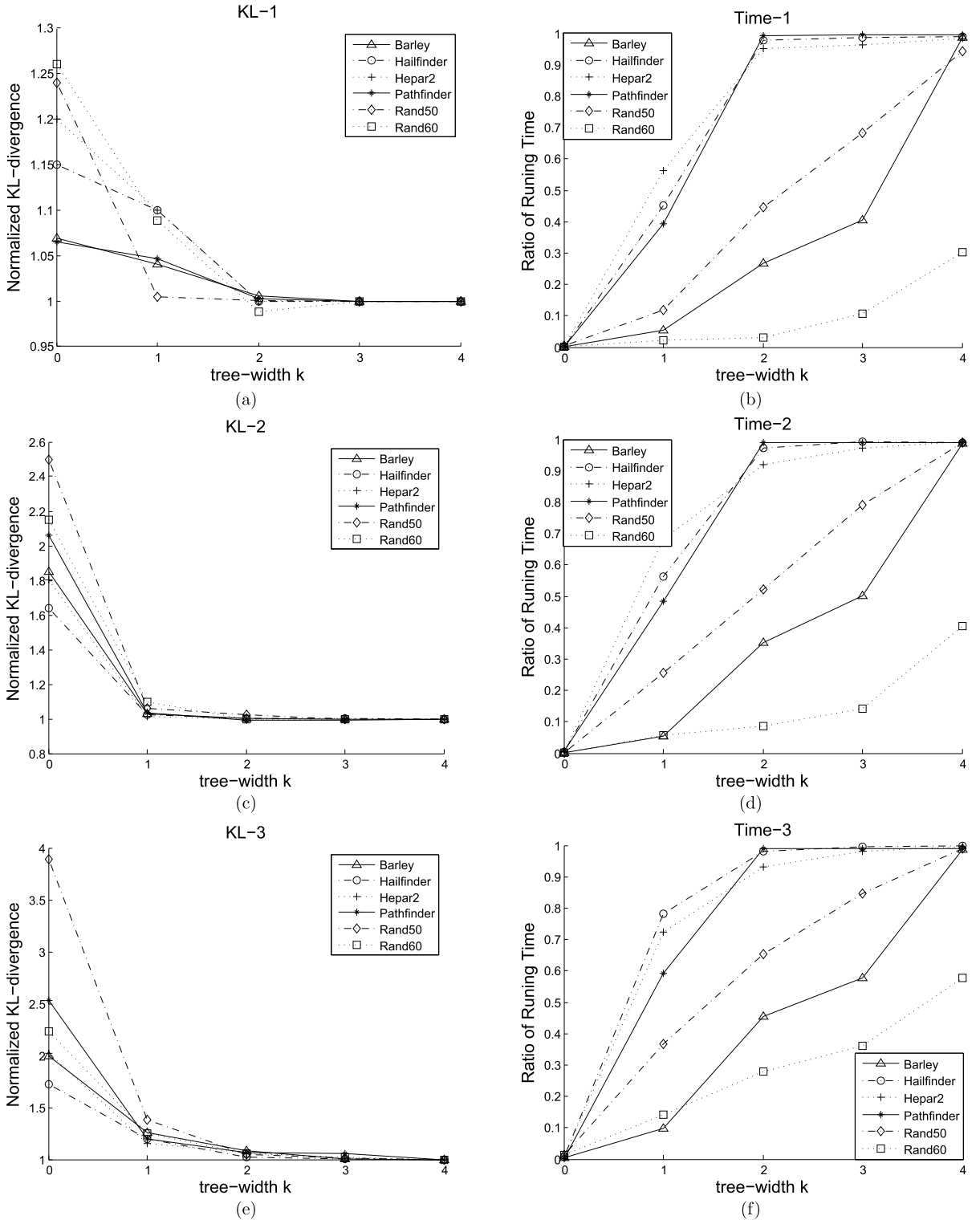

Fig. 5. The KL divergence (normalized using the network with unbounded tree-width for easy view of relative improvement) and ratio of the running times for singletons (a, b), pairs (c, d), and triples (e, f).

Table 13
Details about the data sets used to evaluate the classification performance of different classifiers.

Data set	Vars	Classes	Samples	Training	Testing
australian	14	2	690	552	138
breast	9	2	683	544	136
tic-tac-toe	9	2	958	764	191
cmc	9	2	1473	1176	294
chess	36	2	3196	2130	1066

Table 14
Mean values for 5-fold cross-validation experiments of the classification accuracy of the PHC+S2 algorithm in comparison to those of the PC, TPDA, GES, MMHC, SC, NBC and RAI algorithms. Standard deviations are given in the parenthesis except the chess data set because it uses a holdout division for training and testing sets. The results may vary from [37] due to different folds in the cross-validation.

Data set	australian	breast	tic-tac-toe	cmc	chess
PC	85.8 (0.4)	95.3 (1.9)	71.4 (1.3)	51.2 (2.2)	91.1
TDPA	84.8 (0.3)	94.2 (2.9)	72.5 (3.2)	42.4 (2.8)	89.1
GES	84.5 (2.2)	96.8 (1.1)	70.3 (2.9)	45.7 (1.4)	**95.2**
MMHC	**86.5** (1.2)	95.4 (1.5)	70.8 (3.2)	47.9 (2.5)	94.5
SC	85.3 (1.1)	96.7 (0.9)	71.1 (4.3)	48.1 (2.3)	93.4
NBC	85.7 (3.4)	**97.4** (1.1)	69.9 (3.2)	**51.8** (1.2)	86.2
RAI	85.3 (1.3)	96.4 (1.3)	73.2 (1.8)	53.3 (2.9)	93.8
PHC ($k=2$)	85.8 (3.1)	96.0 (2.8)	66.1 (3.2)	51.3 (3.0)	88.7
PHC ($k=3$)	86.2 (2.9)	95.7 (2.8)	**82.7** (2.9)	51.3 (3.0)	90.3
PHC ($k=4$)	86.2 (2.9)	95.5 (2.8)	81.9 (2.9)	51.3 (3.0)	91.5

interactions among the variables in the true networks (the *Pathfinder* network has a tree structure as shown in Fig. 4 (b)). For the other four networks, empty networks are at least 15% worse than tree structures.

4.4. Under-fitting

The tree-width bound is a constraint to ensure the sparsity of the learned structure. Therefore, there is a potential that learning with bounded tree-width could lead to under-fitting. It is shown that [7] to approximate a target distribution using a BN within a certain distance δ in terms of KL divergence, there exists the effective tree-width $k(\delta)$, which is the smallest achievable tree-width. If the tree-width bound we selected is smaller than the effective tree-width, under-fitting could happen.

To study this issue in terms of classification, we empirically evaluated the classification capability of the learned BN. We took 5 data sets from the UCI machine learning repository (i.e., *australian, breast, tic-tac-toe, cmc*, and *chess*), and implemented 7 other structure learning methods (i.e., PC [34], TDPA [11], GES [12], MMHC [36], SC [21], Naive Bayesian Classifier (NBC) and RAI [37]) in terms of classification accuracy. The details of the data sets are given in Table 13. All data sets were analyzed using a 5-fold cross-validation experiment, except *chess*, which was analyzed using the holdout methodology due to the large size. Continuous variables were binarized over the mean value, and instances with missing values were removed. The data sets contain a variable representing the label. For example, in the *australian* data set, one binary variable represents the Australian credit card approval decision. The other variables are used as features. The classification is to determine the label given all the features, by computing its posterior probability in the BN. The PHC+S2 algorithm was given 10 minute running time for the *australian, breast, tic-tac-toe* and *cmc* data sets. For the *chess* data set, due to its relatively larger size, we gave it 30 minute running time.

The results are summarized in Table 14. On three data sets (i.e., *australian, tic-tac-toe*, and *cmc*), the proposed PHC algorithm outperformed the competing algorithms in terms of classification accuracy, and on the other data sets (i.e., *breast*, and *chess*), it is comparable to other algorithms. This empirically indicates that under-fitting did not happen on these data sets. The NBC method learns a tree structure with tree-width 1. In terms of classification, it gives a very good approximation in most cases. However it requires all variables, besides the label, for classification, while other networks employ a subset of variables consisting of the Markov Blanket of the label, which can perform feature dimension reduction and feature selection. For the other learning algorithms, the learned structures already have small tree-width on the first four data sets (3 for *tic-tac-toe*, 4 for *australian*, *breast* and *cmc*), therefore classification performance is similar. For the *chess* data set, the structure learned by other approaches has tree-width at least 6. The classification result indicates that for these data sets, the relationship of random variables can be effectively captured using networks with a small tree-width. Bounding the tree-width to 4 does not lose much accuracy in representing the underlying distribution.

5. Conclusion

In this paper we present a sampling method for learning Bayesian networks with bounded tree-width. The sampling is based on a bijection between Dandelion codes and k-trees. We design a distance preferable sampling scheme to effectively cover the space of k-trees, as well as an informative score function to evaluate each k-tree. Based on the I-score, a local search method is employed to refine the k-trees. These ideas allow to quickly find representative k-trees of high quality. To further improve the quality of the sampled k-trees, a conventional hill climbing approach and a probabilistic hill climbing are proposed to refine the sampled k-trees locally. Experiments indicate that the proposed method reaches comparable accuracy to the exact algorithms in terms of BDeu scores, but is much more efficient in terms of learning speed, and can scale up to larger networks and larger tree-widths. Moreover, experiments show that the proposed probabilistic hill climbing method outperforms most existing approximate methods on various data sets.

Acknowledgements

We thank the anonymous reviewers for their useful suggestions. This work is supported in part by the grant N00014-12-1-0868 from the Office of Navy Research, by the grant W911NF-12-1-0473 from the Army Research Office, and by the grant from French ANR, project ID: ANR-14-CE24-0026.

References

[1] S. Arnborg, D.G. Corneil, A. Proskurowski, Complexity of finding embeddings in a k-tree, SIAM J. Algebraic Discrete Methods 8 (2) (1987) 277–284.
[2] F.R. Bach, M.I. Jordan, Thin junction trees, in: Advances in Neural Information Processing Systems, 2001, pp. 569–576.
[3] K. Bache, M. Lichman, UCI machine learning repository, http://archive.ics.uci.edu/ml, 2013.
[4] M. Barlett, J. Cussens, Advances in Bayesian network learning using integer programming, in: Proceedings of the 29th Conference on Uncertainty in Artificial Intelligence, 2013, pp. 182–191.
[5] L.W. Beineke, R.E. Pippert, The number of labeled k-dimensional trees, J. Comb. Theory 6 (2) (1969) 200–205.
[6] J. Berg, M. Järvisalo, B. Malone, Learning optimal bounded treewidth Bayesian networks via maximum satisfiability, in: Proceedings of the 17th International Conference on Artificial Intelligence and Statistics, 2014, pp. 86–95.
[7] A. Beygelzimer, I. Rish, Approximability of probability distributions, in: Advances in Neural Information Processing Systems, 2003.
[8] W. Buntine, Theory refinement on Bayesian networks, in: Proceedings of the 7th Conference on Uncertainty in Artificial Intelligence, 1991, pp. 52–60.
[9] S. Caminiti, E.G. Fusco, R. Petreschi, Bijective linear time coding and decoding for k-trees, Theory Comput. Syst. 46 (2) (2010) 284–300.
[10] A. Chechetka, C. Guestrin, Efficient principled learning of thin junction trees, in: Advances in Neural Information Processing Systems, 2007, pp. 273–280.
[11] J. Cheng, D.A. Bell, W. Liu, Learning belief networks from data: an information theory based approach, in: Proceedings of the 6th International Conference on Information and Knowledge Management, ACM, 1997, pp. 325–331.
[12] D.M. Chickering, Optimal structure identification with greedy search, J. Mach. Learn. Res. 3 (2003) 507–554.
[13] G.F. Cooper, The computational complexity of probabilistic inference using Bayesian belief networks, Artif. Intell. 42 (2) (1990) 393–405.
[14] G.F. Cooper, E. Herskovits, A Bayesian method for the induction of probabilistic networks from data, Mach. Learn. 9 (4) (1992) 309–347.
[15] P. Dagum, M. Luby, Approximating probabilistic inference in Bayesian belief networks is NP-hard, Artif. Intell. 60 (1) (1993) 141–153.
[16] A. Darwiche, Modeling and Reasoning with Bayesian Networks, Cambridge University Press, 2009.
[17] D. Eaton, K. Murphy, Bayesian structure learning using dynamic programming and mcmc, in: Proceedings of the 23rd Conference on Uncertainty in Artificial Intelligence, 2007, pp. 101–108.
[18] G. Elidan, S. Gould, Learning bounded treewidth Bayesian networks, J. Mach. Learn. Res. 9 (2008) 2699–2731.
[19] G. Elidan, S. Gould, Learning bounded treewidth Bayesian networks, in: Advances in Neural Information Processing Systems, 2008, pp. 417–424.
[20] S. Ermon, C.P. Gomes, A. Sabharwal, B. Selman, Taming the curse of dimensionality: discrete integration by hashing and optimization, in: Proceedings of the 30th International Conference on Machine Learning, 2013, pp. 334–342.
[21] N. Friedman, I. Nachman, D. Peér, Learning bayesian network structure from massive datasets: the sparse candidate algorithm, in: Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence, 1999, pp. 206–215.
[22] D. Heckerman, D. Geiger, D.M. Chickering, Learning Bayesian networks: the combination of knowledge and statistical data, Mach. Learn. 20 (3) (1995) 197–243.
[23] D. Koller, N. Friedman, Probabilistic Graphical Models, MIT Press, 2009.
[24] J.H. Korhonen, P. Parviainen, Exact learning of bounded tree-width Bayesian networks, in: Proceedings of the 16th International Conference on Artificial Intelligence and Statistics, 2013, pp. 370–378.
[25] J.H.P. Kwisthout, H.L. Bodlaender, L.C. van der Gaag, The necessity of bounded treewidth for efficient inference in Bayesian networks, in: Proceedings of the 19th European Conference on Artificial Intelligence, 2010, pp. 237–242.
[26] D.D. Mauá, C.P. de Campos, Anytime marginal MAP inference, in: Proceedings of the 28th International Conference on Machine Learning, 2012, pp. 1471–1478.
[27] S. Nie, C.P. de Campos, Q. Ji, Learning bounded tree-width Bayesian networks via sampling, in: Symbolic and Quantitative Approaches to Reasoning with Uncertainty, Springer, 2015, pp. 387–396.
[28] S. Nie, C.P. de Campos, Q. Ji, Learning Bayesian networks with bounded tree-width via guided search, in: 13th AAAI Conference on Artificial Intelligence, 2016.
[29] S. Nie, D.D. Mauá, C.P. de Campos, Q. Ji, Advances in learning Bayesian networks of bounded treewidth, in: Advances in Neural Information Processing Systems, 2014, pp. 2285–2293.
[30] J.D. Park, A. Darwiche, Complexity results and approximation strategies for map explanations, J. Artif. Intell. Res. 21 (2004) 101–133.
[31] P. Parviainen, H.S. Farahani, J. Lagergren, Learning bounded tree-width Bayesian networks using integer linear programming, in: Proceedings of the 17th International Conference on Artificial Intelligence and Statistics, 2014, pp. 751–759.
[32] H.P. Patil, On the structure of k-trees, J. Comb. Inf. Syst. Sci. 11 (2–4) (1986) 57–64.
[33] G. Schwarz, Estimating the dimension of a model, Ann. Stat. 6 (2) (1978) 461–464.
[34] P. Spirtes, C.N. Glymour, R. Scheines, Causation, Prediction, and Search, Lecture Notes in Statistics, vol. 81, MIT Press, 2000.
[35] N. Srebro, Maximum likelihood bounded tree-width Markov networks, Artif. Intell. 143 (1) (2003) 123–138.
[36] I. Tsamardinos, L.E. Brown, C.F. Aliferis, The max–min hill-climbing Bayesian network structure learning algorithm, Mach. Learn. 65 (1) (2006) 31–78.
[37] R. Yehezkel, B. Lerner, Bayesian network structure learning by recursive autonomy identification, J. Mach. Learn. Res. 10 (2009) 1527–1570.