Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

A parallel algorithm for Bayesian network structure learning from large data sets

Anders L. Madsen[a,b,*], Frank Jensen[a], Antonio Salmerón[d], Helge Langseth[c], Thomas D. Nielsen[b]

[a] HUGIN EXPERT A/S, DK-9000 Aalborg, Denmark
[b] Aalborg University, DK-9220 Aalborg, Denmark
[c] Norwegian University of Science and Technology, NO-7491 Trondheim, Norway
[d] University of Almería, ES-04120 Almería, Spain

ARTICLE INFO

Article history:
Received 28 February 2016
Revised 8 July 2016
Accepted 23 July 2016
Available online 25 July 2016

Keywords:
Bayesian network
PC algorithm
Parallelization

ABSTRACT

This paper considers a parallel algorithm for Bayesian network structure learning from large data sets. The parallel algorithm is a variant of the well known PC algorithm. The PC algorithm is a constraint-based algorithm consisting of five steps where the first step is to perform a set of (conditional) independence tests while the remaining four steps relate to identifying the structure of the Bayesian network using the results of the (conditional) independence tests. In this paper, we describe a new approach to parallelization of the (conditional) independence testing as experiments illustrate that this is by far the most time consuming step. The proposed parallel PC algorithm is evaluated on data sets generated at random from five different real-world Bayesian networks. The algorithm is also compared empirically with a process-based approach where each process manages a subset of the data over all the variables on the Bayesian network. The results demonstrate that significant time performance improvements are possible using both approaches.

1. Introduction

A *Bayesian network* (BN) [1–5] is a powerful model for probabilistic inference. It consists of two main parts: a graphical structure specifying a set of dependence and independence relations between its variables and a set of conditional probability distributions quantifying the strengths of the dependence relations. The graphical nature of a Bayesian network makes it well-suited for representing complex problems, where the interactions between entities, represented as variables, are described using *conditional probability distributions* (CPDs). Both parts can be elicited from experts or learnt from data, or a combination. Here we focus on learning the graphical structure from data using a variant of the PC algorithm [6] exploiting parallel computations.

Large data sets both in terms of the number of variables and cases may challenge the efficiency of pure sequential algorithms for learning the structure of a Bayesian network from data. Since the computational power of computers is ever increasing and access to computers supporting parallel processing is improving, it is natural to consider exploiting parallel computations to improve the performance of learning algorithms. A number of different approaches to parallel structure learning have been considered in the literature. In [7] the authors describe a MapReduce-based method for learning Bayesian networks from massive data using a search & score algorithm while [8] describes a MapReduce-based method for machine learning on multi-core computers. Also, [9] presents the R package **bnlearn** which provides implementations of some structure learning algorithms including support for parallel computing. [10] introduces a method for accelerating Bayesian network parameter learning using Hadoop and MapReduce. Other relevant work on parallelization of learning Bayesian networks from data include [11–15].

In this paper, we consider two different approaches to parallelization of the PC algorithm. First, we describe a new parallel version of the PC algorithm for learning the structure of a Bayesian network from large data sets on a shared memory computer using threads. The proposed parallel PC algorithm is inspired by the work in [16] on vertical parallelization of TAN learning using Balanced Incomplete Block (BIB) designs [17]. Second, we consider

* Corresponding author.
E-mail addresses: alm@hugin.com (A.L. Madsen), fj@hugin.com (F. Jensen), antonio.salmeron@ual.es (A. Salmerón), helgel@idi.ntnu.no (H. Langseth), tdn@cs.aau.dk (T.D. Nielsen).

http://dx.doi.org/10.1016/j.knosys.2016.07.031

an embarrassingly parallel version of the PC algorithm. This approach uses processes where each process manages a subset of the data over all variables. In order to distinguish between the two approaches, the latter approach is referred to as the *horizontal PC algorithm*. The horizontal PC algorithm is developed for distributed memory concurrent computers using the standardized and portable message-passing system referred to as the *Message Passing Interface* (MPI) [18]. The horizontal PC algorithm also takes advantage of BIB designs to improve efficiency. The results of an empirical evaluation show a significant improvement in time performance over a purely sequential implementation for both approaches.

This paper is organized as follows. Section 2 presents preliminaries and notation, including an introduction to BIB designs and the PC algorithm. Section 3 describes the details of both methods for parallel structure learning while Section 4 presents the results of an empirical evaluation of the algorithms on both real-world Bayesian networks and examples from literature. Finally, Section 5 gives a discussion of the results and Section 6 conclusions.

2. Material and methods

Let $\mathcal{X} = \{X_1, \ldots, X_n\}$ be a set of random variables such that $\text{dom}(X)$ is the state space of X when X is discrete. The state space size is $||X|| = |\text{dom}(X)|$. A BN $\mathcal{N} = (\mathcal{X}, G, \mathcal{P})$ over the set $\mathcal{X}$ consists of an acyclic directed graph (DAG) $G = (V, E)$ with vertices V and edges E and a set of CPDs $\mathcal{P} = \{P(X \,|\, \text{pa}(X)) : X \in \mathcal{X}\}$, where $\text{pa}(X)$ denotes the parents of X in G. The BN $\mathcal{N}$ specifies a joint probability distribution over $\mathcal{X}$:

$$P(\mathcal{X}) = \prod_{i=1}^{n} P(X_i \,|\, \text{pa}(X_i)).$$

We use upper case letters, e.g., X_i and Y, to denote variables while sets of variables are denoted using calligraphy letters, e.g., $\mathcal{X}$ and $\mathcal{S}$. In this paper, we only consider discrete variables.

We let $\mathcal{D} = (c_1, \ldots, c_N)$ denote a data set of N complete cases over variables $\mathcal{X} = \{X_1, \ldots, X_n\}$ and we let $I(X, Y; \mathcal{S})$ denote conditional independence between X and Y given $\mathcal{S}$. When learning the structure of a DAG G from $\mathcal{D}$, we use a test statistic to test the hypothesis $I(X, Y; \mathcal{S})$ based on counts in $\mathcal{D}$. That is, to test the conditional independence hypothesis $I(X, Y; \mathcal{S})$ between two discrete variables X and Y conditional on $\mathcal{S}$ based on counts in $\mathcal{D}$, we use the test statistic $G^2 = \sum_{\mathcal{S}=s} G^2_s$ where

$$G^2_s = 2 \sum_{x,y} O_{xy|s} \log \frac{O_{xy|s}}{E_{xy|s}}, \qquad (1)$$

where $O_{xy|s}$ is the observed count for x and y given s and $E_{xy|s}$ is the expected count for x and y given s under the null-hypothesis.

2.1. PC algorithm

The task of learning the structure of a Bayesian network from $\mathcal{D}$ amounts to determining the structure G. The PC algorithm of [6] consists of five steps:

1. Determine pairwise (conditional) independence $I(X, Y; \mathcal{S})$.
2. Identify the skeleton of G.
3. Identify v-structures in G.
4. Identify derived directions in G.
5. Complete orientation of G making it a DAG.

Step 1 is performed such that tests for marginal independence (i.e., $\mathcal{S} = \emptyset$) are performed first followed by conditional independence tests where the size of $\mathcal{S}$ iterates over $1, 2, 3, \ldots$ taking the adjacency of vertices into consideration. That is, in the process of determining the set of conditional independence statements $I(X, Y; \mathcal{S})$, the results produced earlier are exploited to reduce the number of tests. This means that we stop testing conditional independence of X and Y once a subset $\mathcal{S}$ has been identified such that the independence hypothesis is not rejected. When testing the conditional independence hypothesis $I(X, Y; \mathcal{S})$, the conditioning set $\mathcal{S}$ is restricted to contain only potential neighbors of either X or Y, i.e., a variable Z is excluded from $\mathcal{S}$, if the independence hypothesis between X (or Y) and Z was previously not rejected. This is referred to as the PC* algorithm by [6], but we will refer to it as the PC algorithm.

Steps 2–5 use the results of Step 1 to determine the DAG G. We will not consider Step 2–5 further in this paper as experiments demonstrate that the combined time cost of these steps is negligible compared to the time cost of Step 1. This is clearly demonstrated in the empirical evaluation. The interested reader is referred to, e.g., [6] for more details.

Hence, our proposal for scaling up the PC algorithm is based on parallelizing Step 1, which involve the calculation of the G^2 score (see Eq. (1)) between each pair of variables. An immediate approach for scaling up the algorithm could be to simply generate one computing thread for each pair of variables and then process the threads in parallel. However, with n variables this approach would require accessing the underlying database $\binom{n}{2}$ times, inducing a significant overhead in terms of disk/network access. Alternatively, one might group the variables in blocks so that each block only accesses the data a single time in order to calculate the sufficient statistics required for computing the G^2 score for all pairs of variables within the block. A key issue here is finding an appropriate block size and at the same time ensuring that the blocks, in combination, guarantee that all pairs of variables are considered exactly once.

To get an intuitive understanding of this process we can as an analogy consider the organization of the Speedway World Championship (SWC). After the initial pre-qualifying rounds for the SWC, the remaining 16 highest ranked riders should be compared to each other to obtain a final ranking of the riders. One approach to achieve this would be to pair-up the riders so that each rider will participate in 15 races, yielding a total of 120 rounds with two riders competing in each round. This setup would put a strain on the riders and not use the full capacity of the speedway track, which is designed to accommodate four riders simultaneously. Instead, the SWC employs a heat-system ensuring that each of the 16 riders will meet each of the other riders at some time during the competition. Specifically, the heat-system consists of 20 heats with four riders in a heat. Each rider participates in only five heats, and within a single heat all riders compete jointly, thereby meeting each other. After completing the 20 heats, all pairs of riders will have met exactly once. This can also be seen by labeling the riders $\{0, \ldots, 15\}$ and constructing these heats: $H_1 = \{3, 6, 12, 15\}$, $H_2 = \{4, 5, 10, 13\}$, $H_3 = \{0, 4, 6, 7\}$, $H_4 = \{0, 10, 11, 15\}$, $H_5 = \{7, 10, 12, 14\}$, $H_6 = \{0, 8, 9, 14\}$, $H_7 = \{0, 1, 3, 13\}$, $H_8 = \{1, 6, 8, 10\}$, $H_9 = \{7, 9, 13, 15\}$, $H_{10} = \{1, 5, 14, 15\}$, $H_{11} = \{8, 11, 12, 13\}$, $H_{12} = \{5, 6, 9, 11\}$, $H_{13} = \{1, 4, 9, 12\}$, $H_{14} = \{3, 5, 7, 8\}$, $H_{15} = \{3, 4, 11, 14\}$, $H_{16} = \{2, 6, 13, 14\}$, $H_{17} = \{1, 2, 7, 11\}$, $H_{18} = \{0, 2, 5, 12\}$, $H_{19} = \{2, 4, 8, 15\}$, and $H_{20} = \{2, 3, 9, 10\}$.

When it comes to computing the G^2 scores, the 16 riders correspond to variables and each heat represents a block consisting of four variables to be pairwise compared. Thus, rather than handling pairs of variables independently and having to make data access $\binom{16}{2} = 120$ times, we can instead make 20 blocks/heats of four variables each and thereby only having to access the full dataset 20 times. Note that with the particular setup above, we are guaranteed not to make redundant calculations as the G^2 score is computed exactly once for each pair X_i, X_j, $1 \leq i, j \leq n$.