Factorization of Sparse Bayesian Networks

Julio Michael Stern and Ernesto Coutinho Colla

Abstract. This paper shows how an efficient and parallel algorithm for inference in Bayesian Networks (BNs) can be built and implemented combining sparse matrix factorization methods with variable elimination algorithms for BNs. This entails a complete separation between a first symbolic phase, and a second numerical phase.

Keywords: Bayesian networks, Probabilistic Reasoning, Variable elimination, Elimination trees, Sparse Cholesky factorization, Sparse matrix factorizations.

1 Introduction

Bayesian Networks (BNs) are probabilistic graphical models used to represent and encode uncertain expert knowledge. BNs stand out for dealing with uncertainty in decision making and statistical inference, and many algorithms were described for inference in BNs, see Dechter (1996), Heckerman (1995), Jensen (1996), Lauritzen (1988), Pearl (1988), and Zang (1996). The parallel algorithm described in this paper is based on the sequential variable elimination algorithm of Cozman(2000), using algebraic operations on potentials. These algebraic schemata for inference in BNs are not only relatively simple to understand and to implement, but also allow us to use the techniques, heuristics and abstract combinatorial structures from the sparse matrix factorizations literature, see appendix, George (1993) and Stern (1994, 2008a,b).

The main goal of this paper is to show how variations of the variable elimination algorithm can be combined with sparse matrix factorization methods to implement a fast and efficient parallel algorithm for inference in BNs. This goal is achieved with the complete separation between a first symbolic phase, and a second numerical phase. In the symbolic phase the proposed algorithm explores the graphical

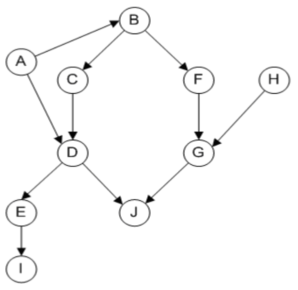
Julio Michael Stern and Ernesto Coutinho Colla
IME-USP, University of São Paulo, Brazil
e-mail: jmstern@hotmail.com

structure of the model, without computing or even accessing probabilistic information. The second numerical phase can be fully vectorized and parallelized using static data structures previously defined in the first phase. This is done examining the decoupling or separation operators of sparse matrix factorization algorithms and BNs inference procedures from a unified combinatorial framework. This unified framework is the key for implementing efficiently this parallel algorithm.

2 Inference with Bayesian Network

A BN, see Jensen (1996), is a graphical model that efficiently encodes the joint probability distribution for a set (or list) of random variables, $X = \{X_1, X_2, ..., X_n\}$, each of them having a finite number of possible states. A BN consists of two components: (i) A Directed Acyclic Graph (DAG) defining the network structure and encoding the conditional dependence relations between the variables in X; (ii) A set of local probability densities associated with each variable. Each node, i, of the DAG represents a random variable, X_i. In order to make the notation lighter, we may write a node index, i, instead of its random variable, X_i, and vice versa. We also use the vectorized notation, X_S, for the subset $\{X_i\}, i \in S$.

The DAG representing the BN structure has an arc from node i to node j, that is, i is a *parent* of j, $i \in \text{pa}(j)$, if the probability distribution of variable X_j is directly dependent on variable X_i, and the strength of this influence is expressed by conditional probability distributions. In many specific statistical models an arc can be interpreted as a direct influence or causal effect of X_i on X_j, see Pearl (1988).



Probability Densities:
$P(A)$, $P(B \mid A)$, $P(C \mid B)$, $P(D \mid A,C)$, $P(E \mid D)$
$P(F \mid B)$, $P(G \mid F,H)$, $P(H)$, $P(I \mid E)$, $P(J \mid D,G)$

B and D are children of A
C is child of B
A and C are parents of D so A and C are spouses
D and G are parents of J so D and G are spouses
G and J are descendants of F
D, E, I and J are descendants of C
H is nondescendant of B

Fig. 1 Bayesian network example

The semantics of BNs implies a correspondence between the topology of a DAG and the network's probabilistic dependence relations, determined by the *Markov condition*: Every variable is independent of its nondescendants nonparents given its parents. Therefore, every X_i is associated with a local probability density, $P(X_i \mid X_{\text{pa}(i)})$, as showed in Fig 1. Based on this condition, a BN encodes a unique probability distribution: $P(X) = \prod_i P(X_i \mid X_{\text{pa}(i)})$.

Inference in BNs is based on queries, where the posterior marginal distribution for a set of *query variables*, X_Q, has to be computed given a set of observed variables, X_E. This set of observed variables is the *evidence* in the network and establishes the values of the variables in X_E. For example: $e = \{X_i = x_i, X_j = x_j\}$ establishes the values of X_i and X_j, so $E = \{i, j\}$.

The posterior probability of X_Q given e is:

$$P(X_Q \,|\, e) = \frac{P(X_Q, e)}{P(e)} = \frac{\sum_{X \backslash \{X_Q, X_E\}} P(X)}{\sum_{X \backslash X_E} P(X)} \,. \qquad (1)$$

The expression $X \backslash Y$ indicates the set of all variables which belong to X but do not belong to Y, and the expression $\sum_X f(X,Y)$ indicates that all variables of X were *eliminated* or *marginalized out*, that is, were summed out from the function $f(X,Y)$.

Efficient computational algorithms rely on two important technical points:

(I) Given a BN over variables X, an evidence e and a query X_Q, not all variables of X may be required to compute $P(X_Q \,|\, e)$. If the local probability density $P(X_i \,|\, X_{\mathrm{pa}(i)})$ is required to compute $P(X_Q \,|\, e)$, then X_i is a *requisite variable*, $i \in R$. Fortunately there are simple polynomial algorithms able to identify the set R. We have used Bayes-Ball algorithm, see Shachter (1998). It is important to realize that the requisite variables, X_R, can be identified exploring only the DAG topology, without any numerical information concerning probability distributions. Hence, in order to reduce the problem dimension, the identification of R should be done at the very first stage of inference calculation.

(II) At intermediate computations, it is not necessary to compute the normalization constants, that is, the denominator $P(e)$ of (1). We only need the numerator in (1),

$$P(X_Q \,|\, e) \propto P(X_Q, e) = \sum_{X_R \backslash \{X_Q, X_E\}} \left(\prod_{X_i \in X_R} P(X_i \,|\, X_{\mathrm{pa}(i)}) \right) . \qquad (2)$$

Hence, a basic rule for operation in BNs is: Compute the numerator $P(X_Q, e)$ and obtain normalization constant $P(e)$ only in the last stage. This rule means that we can perform the intermediate computations with un-normalized distributions, which are real-valued tables over a finite set of variables. These tables, ϕ, are called *potentials*, see Jensen (1996). A potential's domain, $\mathrm{dom}(\phi)$, is its correspondent set of variables. In the following, we give some important properties of the algebra of potentials:

(1) A variable X_i can be *marginalized out* of a potential ϕ resulting in a new potential $\phi'_{X_i} = \sum_{X_i} \phi$ over the domain $\mathrm{dom}(\phi'_{X_i}) = \mathrm{dom}(\phi) \backslash \{X_i\}$. Marginalization follows:
(1a) the commutative law: $\sum_{X_i} \sum_{X_j} \phi = \sum_{X_j} \sum_{X_i} \phi$; and
(1b) the distributive law: if $X_i \notin \mathrm{dom}(\phi_1)$, then $\sum_{X_i} \phi_1 . \phi_2 = \phi_1 . \sum_{X_i} \phi_2$.

(2) Two potentials can be *multiplied*, resulting in a new potential with $\mathrm{dom}(\phi_1 . \phi_2) = \mathrm{dom}(\phi_1) \cup \mathrm{dom}(\phi_2)$. Multiplication follows:
(2a) the commutative law: $\phi_1 . \phi_2 = \phi_2 . \phi_1$; and
(2b) the associative law: $(\phi_1 . \phi_2) . \phi_3 = \phi_1 . (\phi_1 . \phi_3)$.

As an example, consider the BN in Figure 1. The BN joint probability distribution can be rewritten as: $P(X) \propto \phi_A . \phi_B . \phi_C . \phi_D . \phi_E . \phi_F . \phi_G . \phi_H . \phi_I . \phi_J$, and the potentials specified for the network are: $P(A) \propto \phi_A(A), P(B|A) \propto \phi_B(B,A)$,

$P(C\,|\,B) \propto \phi_C(C,B), P(D\,|\,A,C) \propto \phi_D(D,A,C)$ and so on. Computing $P(I)$ can be accomplished by marginalizing out of $P(X)$ all the variables, except I.

$$P(I) = \sum\nolimits_{A,B,C,D,E,F,G,H,J} P(X) \ . \tag{3}$$

BNs are particularly useful for calculating new probabilities when we acquire new information. However, in the preceding calculations no evidence was entered into the network. Now, assume information e has been acquired, stating that "$A = a_t$", where A is a variable and a_t is the t-th state of A. Let A have s states with probability distribution $P(A) = (x_1, ..., x_t, ..., x_s)$. This observed evidence e means that all states except tth one are impossible. So the new (un-normalized) probability distribution is $P(A,e) = (0, ..., 0, x_t, 0, ..., 0)$ which is the result of multiplying $P(A)$ with $\underline{e}_A = (0, ..., 0, 1, 0, ..., 0)$ in which only tth value is 1. The s-dimensional 0-1 potential $\underline{e}_A$ is called *finding*.

In the current example, assume that we have the evidence $A = a$, $H = h$ and $J = j$. This evidence e would be represented using three findings $\underline{e}_A$, $\underline{e}_H$ and $\underline{e}_J$. The posterior marginal $P(I\,|\,e)$ can be obtained normalizing $P(I,E)$:

$$P(I,e) = \sum\nolimits_{A,B,C,D,E,F,G,H,J} P(X).\underline{e}_A.\underline{e}_H.\underline{e}_J \ . \tag{4}$$

To avoid calculating the product of all potentials, we use the distributive law:

$$P(I\,|\,e) = \sum_D \sum_E \phi_I(I,E).\phi_E(E,D) \sum_B \sum_C \phi_C(C,B).$$

$$\sum_A \phi_A(A).\phi_B(B,A).\phi_D(D,A,C).\underline{e}_A.$$

$$\sum_J \sum_G \phi_J(J,D,G).\underline{e}_J \sum_F \phi_F(F,B). \sum_H \phi_G(G,F,H).\phi_H(H).\underline{e}_H \ .$$

First, calculate $\phi_H' = \sum_H \phi_G(G,F,H).\phi_H(H).\underline{e}_H$, then multiply $\phi_H'(F,G)$ on $\phi_F(F,B)$ and calculate $\phi_F' = \sum_F \phi_F(F,B).\phi_H'(F,G)$. The later result is multiply on $\phi_J(J,D,G).\underline{e}_J$, to calculate $\phi_G' = \sum_G \phi_J(J,D,G).\phi_F'(B,G).\underline{e}_J$, and so forth. All the operations involved are represented in Figure 2.

Because marginalization is commutative it can be done in any order. In the preceding calculation the marginalization, also called variable elimination, was done in a particular order, namely $q = [H,F,G,E,A,C,J,B,D]$.

The diagram in Figure 2 also portrays the dependencies among potential operations to calculate $P(I\,|\,e)$. Notice that some operations could be done simultaneously. For example, at very first stage, we could perform the required operations on A, H, and I, calculating $\phi_A' = \sum_A \phi_A(A).\phi_B(B,A).\phi_D(D,A,C).\underline{e}_A$, etc.

If a parallel computer is available, we can simultaneously execute all the marginalizations using already computed potentials. Hence, it is desirable to find:
(i) An efficient way to specify all dependencies among marginalization operations.
(ii) A way to specify an elimination order entailing a "simple" dependence structure, so that many operations can be done simultaneously.

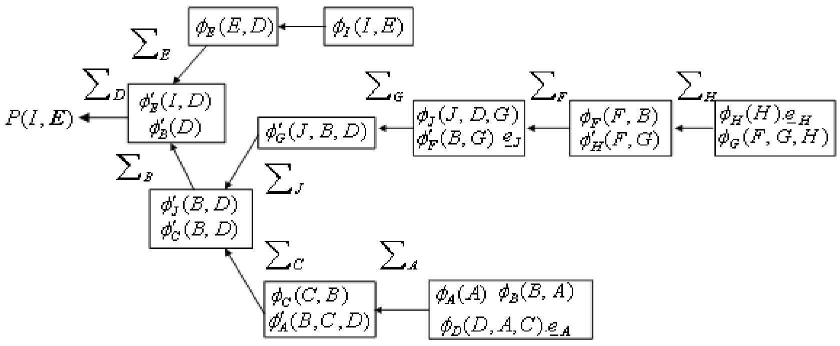


Fig. 2 The process of marginalizing down to *I*

The dependence structure of these operations is exactly the same as the dependence structure for “pivoting” operations appearing in numerical linear algebra, namely, in the Cholesky factorization of sparse matrices, see appendix, George (1993), Pissanetzky (1984), and Stern (1994, 2008b). We describe only the aspects pertinent to this paper.

An *Undirected Graph* (UG), $\mathscr{G} = (\mathscr{V}, \mathscr{E})$, has undirected edges, $\{i, j\} \in \mathscr{E}$, standing for pairs of opposite directed arcs, (i, j) and (j, i). The *Moral Graph* of a DAG, $\mathscr{G}$, is the UG with the same nodes as $\mathscr{G}$, and edges joining nodes i and j if they are immediate relatives in $\mathscr{G}$. The immediate relatives of a node in $\mathscr{G}$ include its parents, children and spouses (but not brothers or sisters). i is a spouse of j is they have a child in common, that is, $i \in \mathrm{sp}(j) \Leftrightarrow \exists k \,|\, i, j \in \mathrm{pa}(k)$.

The *Markov Blanket* of X_i, $X_{\mathrm{mb}(i)}$ is defined as the minimal set of variables that makes a variable X_i independent from all other variables in the BN. This means that the Markov Blanket of a variable “decouples” this variable from the rest of network: $P(X_i \,|\, X_{\mathrm{mb}(i)}, X_j) = P(X_i \,|\, X_{\mathrm{mb}(i)})$. It can be shown that the set of immediate relatives of node i is the Markov Blanket of node i. Figure 3a shows the Moral Graph of the BN in Figure 1. It is important to realize that if X_i and X_j are both in the same domain, of a variable X_k of the BN, then the edge $\{i, j\}$ is in the Moral Graph.

Given an UG, $\mathscr{G} = (\mathscr{V}, \mathscr{E})$, $\mathscr{V} = \{1, \ldots n\}$, and $q = [q(1), \ldots q(n)]$, an elimination order, we define the elimination process of its nodes as the sequence of *elimination graphs* $\mathscr{G}_k = (\mathscr{V}_k, \mathscr{E}_k)$, for $k = 1 \ldots n$, as follows: When eliminating node $q(k)$, we make its neighbors a *clique*, adding all missing edges between them.

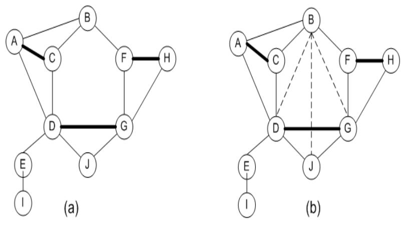


Fig. 3 (a) Moral Graph (b) Filled Graph

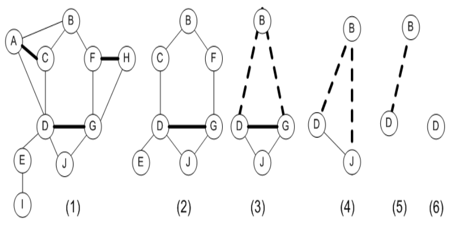


Fig. 4 Elimination graphs sequence

$$\mathscr{V}_k = \{q(k), q(k+1), \ldots q(n)\}, \quad \mathscr{E}_1 = \mathscr{E}, \text{ and, for } k > 1 ,$$

$$\{i,j\} \in \mathscr{E}_k \Leftrightarrow \begin{cases} \{i,j\} \in \mathscr{E}_{k-1} \text{ , or} \\ \{q(k-1), i\} \in \mathscr{E}_{k-1} \text{ and } \{q(k-1), j\} \in \mathscr{E}_{k-1} \text{ .} \end{cases}$$

The *Filled Graph* is the graph $(\mathscr{V}, \mathscr{F})$, where $\mathscr{F} = \cup_{k=1}^{n} \mathscr{E}_k$. The *original* edges and the *filled* edges in $\mathscr{F}$ are, respectively, the edges in $\mathscr{E}$ and in $\mathscr{F} \backslash \mathscr{E}$. There is a computationally more efficient form of obtaining the Filled Graph, known as *simplified elimination*: In the simplified version of the elimination graphs, $\mathscr{G}_k^*$, when eliminating vertex $q(k)$, we add only the clique edges incident to its neighbor, $q(l)$, that is next in the elimination order.

The marginalization of variable X_i out of $P(X)$ corresponds to the elimination of the correspondent node in the elimination sequence. In order to marginalize on X_i, we have first to multiply all the potentials having X_i in its domain, and than sum out X_i. The domain of the resulting potential includes all the neighbors of X_i. In the Elimination Graphs, the corresponding elimination of X_i forms a clique with all of X_i's neighbors. Figure 3b and 4 show the Filled Graph and a synthetic version of the eliminations graphs for the order $q = [H, F, G, E, A, C, J, B, D]$.

The *Elimination Tree*, see appendix, George (1993), Pissanetzky (1984), and Stern (1994, 2008b), portrays the dependencies among numeric operations on potentials, corresponding to dependencies in the node elimination process in the elimination graph. Hence, building the Elimination Tree for the corresponding Moral Graph makes it easy to see which variables can be eliminated simultaneously. Figure 5 shows the Elimination Tree for the order $q = [H, F, G, E, A, C, J, B, D]$.

According to the Figure 5, six steps would be enough to eliminate all nodes: Variables I, H and A could be eliminated at first step and variables E, F and C at second one. Note that: (i) The Elimination Tree has the same structure of the tree of operations portrayed in Figure 2; (ii) A serial elimination would require 10 steps.

Clearly the Elimination Tree depends on the chosen elimination order, and the sparse matrix literature has many heuristics designed for finding good elimination orders. In this paper we adopted an heuristic based on a nested dissections of the breadth-first tree rooted at a pseudo-peripheral vertex which, in turn, was found using the Gibbs heuristic, see Figures 6 and 7. These procedures are described in the appendix, see also George (1993), Pissanetzky (1984) and Stern (1994, 2008b).

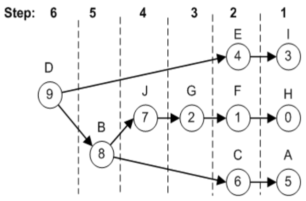


Fig. 5 Elimination Tree for order $q = [H, F, G, E, A, C, J, B, D]$

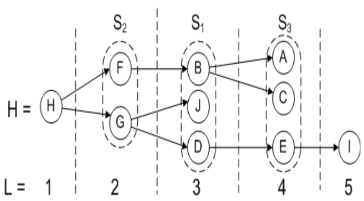


Fig. 6 Nested Dissection

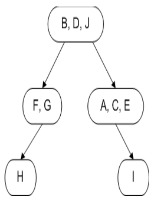


Fig. 7 Nested Dissection

3 Parallel Variable Elimination Algorithm

The sequence of operations described in the previous section for inference in BNs can be summarized in the *parallel variable elimination algorithm*:

1. Symbolic phase:

1.1* Define the requisite variables X_R (ex. using Bayes-Ball algorithm);

1.2 Build the Moral Graph (including only variables in X_R);

1.3 Choose a good elimination order using the Gibbs heuristics to find a pseudo-peripheral vertex used as a root for the Nested Dissection heuristic;

1.4 Symbolic Factorization: Execute the simplified elimination on the Moral Graph, and build the Elimination Tree;

1.5 Allocate the computation resources and prepare the data structures to execute the numeric operations.

2. Numeric phase: Using static data structures defined in the first phase:

2.1 While the root of the Elimination Tree was not executed: Based on the Elimination Tree hierarchy, trigger all threads executeing variable eliminations ready to be done, including its numeric operations of multiplication and marginalization;

2.2 Normalize the remaining potential at the root.

4 Results and Conclusions

The proposed parallel algorithm was implemented and its performance was compared with a serial implementation. Both implementations were done in C and use the same functions to execute the basic operations for: Load the network; Multiply and marginalize potentials; and define the elimination order. The only difference between the two implementations is that the parallel version builds the Elimination Tree and, if possible, eliminate two or more variables simultaneously. Following this strategy we hope to isolate the effect of parallelization.

Table 1 displays some illustrative results. These experiments were done in a bi-processed machine running Linux and consists of 100 inferences for 7 distinct queries using the Hailfinder25 network (55 variables). The set of experiments suggests that the parallel implementation is much faster than the serial one for larger experiments. Queries requiring more variables or with a branched structure in the Elimination Tree allow the simultaneous elimination of several variables, for example experiments 1 to 6. Models requiring less variables, or with a more linear structure in the elimination tree allow less parallelization of elimination operations. Consequently, in these examples, the serial implementation performed better due to the computational overheads imposed by the parallel version, namely, building of the Elimination Tree and the heavy context switch during execution. This was the case of experiment 7 in which the relations of dependence between the operations reduce the possibilities of parallelization.

Practitioners always want to solve larger models, most large models used in practice are sparse, and parallel or distributed computer are increasingly available. Hence, we see great potential for the parallel algorithm presented in this article.

Table 1 Query example, Numb. or requisite vars., Parallel time, Serial Time, Parallel context switches. Serial context switches

Q.E.	N.R.	P.T.	S.T.	P.C.S.	S.C.S.
1	44	174	812	6498	901
2	44	95	125	6514	142
3	45	71	155	6553	183
4	46	74	155	6681	164
5	46	104	126	6817	138
6	48	106	125	7067	152
7	22	98	66	2948	78

Appendix: Sparse Cholesky Factorization

In orther to highlight the analogy between sparse network and sparse matrix factorizations, this appendix presents a few (very summarized and condensed) examples of sparse Cholesky factorization. For a complete explanation, see Stern (1994, 2008b).

Figure 8 shows the positions filled in the Cholesky factorization of a matrix A, $A = LL'$, and in the Cholesky factorization of two symmetric permutation of the same matrix, $A(q,q)$. Initial Non Zero Elements, NZEs, are represented by x, initial zeros filled during the factorization are represented by 0, and initial zeros left unfilled are represented by blank spaces.

Fig. 8 Filled Positions in Cholesky Factorization

$$\begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{bmatrix} 1 & x & x & & & x \\ x & 2 & x & & & 0 \\ x & x & 3 & x & & 0 \\ & & x & 4 & & 0 \\ & & & & 5 & x \\ x & 0 & 0 & 0 & x & 6 \end{bmatrix} \begin{bmatrix} 1 & x & x & x & & \\ x & 3 & 0 & x & x & \\ x & 0 & 6 & 0 & 0 & x \\ x & x & 0 & 2 & 0 & 0 \\ & x & 0 & 0 & 4 & 0 \\ & & x & 0 & 0 & 5 \end{bmatrix} \begin{bmatrix} 5 & & & x & & \\ & 4 & & & x & \\ & & 2 & & x & x \\ x & & & 6 & & x \\ & x & x & & 3 & x \\ & & x & x & x & 1 \end{bmatrix}$$

The Elimination Lemma, see Stern (1994, 2008b) states that, when eliminating the j-th column in the Cholesky factorization of matrix $A(q,q) = LL'$, we fill the positions in L corresponding to the filled edges in $\mathscr{F}$ at the elimination of vertex $q(j)$.

The *elimination tree*, $\mathscr{H}$, is defined by

$$h(j) = \begin{cases} j, \ \text{ if } \operatorname{nze}(L^j) = \{j\}, \ \text{ or} \\ \min\{i > j \mid i \in \operatorname{nze}(L^j)\} \ , \ \text{ otherwise ;} \end{cases}$$

where $h(j)$, the parent of j in $\mathscr{H}$, is the first (non diagonal) NZE in column j of L.

Figure 9 shows the elimination trees corresponding to the example A.1.

The elimination tree portrays the dependencies among the columns for the numeric factorization process. More exactly, we can eliminate column j of A, that is, compute all the multipliers in column j and update all the elements affected by these

$$\begin{tikzcd}[row sep=small, column sep=small] 6 \arrow[r] & 5 \arrow[dr] & & & & \\ & & 4 \arrow[r] & 3 \arrow[r] & 2 \arrow[r] & 1 \end{tikzcd} , \quad \begin{tikzcd}[row sep=small, column sep=small] 6 \arrow[r] & 5 \arrow[r] & 4 \arrow[d] \\ 1 & 2 \arrow[l] & 3 \arrow[l] \end{tikzcd} , \quad \begin{tikzcd}[row sep=small, column sep=small] & & 2 \\ 6 \arrow[r] & 5 \arrow[r] \arrow[ur] \arrow[dr] & 3 \\ & & 4 \arrow[r] & 1 \end{tikzcd} .$$

Fig. 9 Elimination Trees

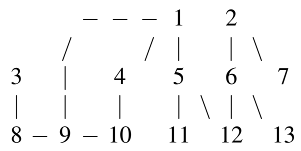
multipliers, if and only if we have already eliminated all the descendents of j in the elimination tree. If we are able to perform parallel computations, we can simultaneously eliminate all the columns at a given level of the elimination tree, beginning with the leaves, and finishing at the root. For example, let us consider an elimination with the same pattern of the last permutation in 8, 9. This elimination tree has three levels that, from the leaves to the root, are: $\{1,3,2\}$, $\{4,5\}$, e $\{6\}$. Hence, the corresponding factorization requires only 2 steps, as illustrated in the following numerical example (in order to avoid taking square roots, we present the LU instead of the Cholesky factorization):

$$\begin{bmatrix} \mathbf{1} & & & 7 & & \\ & \mathbf{2} & & & 8 & \\ & & \mathbf{3} & & 6 & 9 \\ 7 & & & 53 & & 2 \\ & 8 & 6 & & 49 & 23 \\ & & 9 & 2 & 23 & 39 \end{bmatrix} \begin{bmatrix} 1 & & & 7 & & \\ & 2 & & & 8 & \\ & & 3 & & 6 & 9 \\ \mathit{7} & & & \mathbf{4} & & 2 \\ & \mathit{4} & \mathit{2} & & \mathbf{5} & 5 \\ & & \mathit{3} & 2 & 5 & 12 \end{bmatrix} \begin{bmatrix} 1 & & & 7 & & \\ & 2 & & & 8 & \\ & & 3 & & 6 & 9 \\ \mathit{7} & & & 4 & & 2 \\ & \mathit{4} & \mathit{2} & & 5 & 5 \\ & & \mathit{3} & \mathit{\tfrac{1}{2}} & \mathit{1} & \mathbf{6} \end{bmatrix}$$

The sparse matrix literature has many heuristics designed for finding good elimination orders. The example in Figures 10 and 11 show a good elimination order for a 13×13 sparse matrix.

The elimination order in Figure 10 was found using the Gibbs heuristic, described in Stern (1994, ch.6) or Pissanetzky (1984). The intuitive idea of Gibbs heuristic, see Figure 11, is as follows: 1- Starting from a 'peripheral' vertex, in our example, vertex 3; 2- Grow a breath-first tree $\mathscr{T}$ in $\mathscr{G}$. Notice that the vertices at a given level, l, of $\mathscr{T}$ form a separator, S_l, in the graph $\mathscr{G}$. 3- Chose a separator, S_l, that is 'small',

$$\begin{array}{r|ccccccccccccc} 1 & 3 & x & & & & & & & & & & & \\ 2 & x & 8 & & & x & & & & & & & & \\ 3 & & & 1 & & x & & & & & & & x & x \\ 4 & & & & 10 & x & & & & & & & x & \\ 5 & & x & x & x & \mathbf{9} & & & & & & & 0 & \\ 6 & & & & & & 12 & & & & x & & & x \\ 7 & & & & & & & 13 & & & x & & & \\ 8 & & & & & & & & 2 & x & x & & & \\ 9 & & & & & & & & x & 7 & 0 & & & \\ 10 & & & & & & x & x & x & 0 & \mathbf{6} & & & 0 \\ 11 & & & & & & & & & & & 11 & & x \\ 12 & & & x & x & 0 & & & & & & & \mathbf{4} & 0 \\ 13 & & & x & & 0 & x & & & & 0 & x & 0 & \mathbf{5} \end{array}$$



$$\begin{tikzcd}[row sep=small, column sep=small] & & 5;4 \arrow[dll] \arrow[d] \arrow[drr] & & \\ 6 \arrow[d] \arrow[dr] \arrow[drr] & & 11 & & 9 \arrow[d] \arrow[dr] \arrow[drr] & & \\ 12 & 13 & 7;2 & & 10 & 1 & 8;3 \end{tikzcd}$$

Fig. 10 Gibbs Heuristic's Elimination Order and Tree

$$\begin{tikzcd}
 & & & & 10 \arrow[r] & 4 & 11 & & 13 & \\
\mathscr{T} = & 3 \arrow[r] & 8 \arrow[r] & 9 \arrow[r] \arrow[ur] & 1 \arrow[r] & 5 \arrow[r] \arrow[ur] & 12 \arrow[r] & 6 \arrow[r] \arrow[ur] & 2 \arrow[r] & 7 \\
l = & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9
\end{tikzcd}$$

Fig. 11 Nested Dissection by Gibbs Heuristic

i.e. with few vertices, and 'central', i.e. dividing $\mathscr{G}$ in 'balanced' components. 4- Place in q, first the indices of each component separated by S_l, and, at last, the vertices in S_l. 5- Proceed recursively, separating each large component into smaller ones. In our example, we first use separator $S_5 = \{4,5\}$, dividing $\mathscr{G}$ in three components, $C_1 = \{3,8,1,10,9\}$ $C_2 = \{12,13,2,7,6\}$ $C_3 = \{11\}$. Next, we use separators $S_3 = \{9\}$ in C_1, and $S_7 = \{6\}$ in C_2.

References

Colla, E.C.: Aplicação de Técnicas de Fatoração de Matrizes Esparsas para Inferência em Redes Bayesianas. Ms.S. Thesis, MAC-IME-USP, Institute of Mathematics and Statistics, University of São Paulo (2007)

Cozman, F.G.: Generalizing variable elimination in Bayesian networks. In: IBERAMIA-SBIA, Workshop proceedings. São Paulo, Tec. Art, pp. 27–32 (2000)

Mandani, A., Heckerman, D., Wellman, M.P.: Real-world applications of Bayesian networks. Comm. of the ACM 38(3), 24–26 (1995)

Dechter, R.: Bucket elimination: An unifying framework for probabilistic inference. In: 12th UAI proceedings, pp. 211–219. Morgan Kaufmann Publishers, San Francisco (1996)

George, A., Gilbert, J.R., Liu, J.W.H. (eds.): Graph Theory and Sparse Matrix Computation. Springer, NY (1993)

Jensen, F.V.: An introduction to Bayesian networks. Springer, NY (1996)

Lauritzen, S.L., Spiegelhalter, D.J.: Local computations with probabilities on graphical structures and their application to expert systems. J. Royal Statistical Soc., B 50(2), 157–224 (1988)

Pearl, J.: Probabilistic reasoning in intelligent systems: Networks of plausive inference. Morgan Kaufmann, San Francisco (1988)

Pissanetzky, S.: Sparse matrix technology. Academic Press, New York (1984)

Shachter, R.: Bayes-ball: The rational pastime (for determining irrelevance and requisite information in belief networks and influence diagrams). In: 14th UAI proceedings, pp. 480–487. Morgan Kaufmann, San Francisco (1998)

Stern, J.M.: Simulated Annealing with a Temperature Dependent Penalty Function. ORSA Journal on Computing 4, 311–319 (1992)

Stern, J.M.: Esparsidade, Estrutura, Estabilidade e Escalonamento em Álgebra Linear Computacional. IX Escola de Computação. UFPE, Recife (1994)

Stern, J.M.: Decoupling, Sparsity, Randomization, and Objective Bayesian Inference. Cybernetics and Human Knowing 15(2), 49–68 (2008a)

Stern, J.M.: Cognitive Constructivism and the Epistemic Significance of Sharp Statistical Hypotheses. Tutorial book for MaxEnt, The 28th International Workshop on Bayesian Inference and Maximum Entropy Methods in Science and Engineering, July 06-11, Boracéia, São Paulo, Brazil (2008b)

Stern, J.M., Vavasis, S.A.: Nested Dissection for Sparse Nullspace Bases. SIAM Journal on Matrix Analysis and Applications 14(3), 766–775 (1993)
Stern, J.M., Vavasis, S.A.: Active Set Algorithms for Problems in Block Angular Form. Computational and Applied Mathemathics 12(3), 199–226 (1994)
van der Vorst, H.A., van Dooren, P. (eds.): Parallel Algorithms for Numerical Linear Algebra. North-Holland, Amsterdam (1990)
Zhang, N.L., Poole: Exploiting casual independence in Bayesian network inference. Journal of Artificial Intelligence Research, 301–328 (1996)