Variable elimination by factor indexing

Sander Evers, Peter J.F. Lucas
Institute for Computer and Information Sciences
Radboud University Nijmegen
s.evers@cs.ru.nl, peterl@cs.ru.nl

Abstract

It is known that solving an exact inference problem on a Bayesian network with many deterministic nodes can be far cheaper than what would be expected based on its treewidth. In this article, we introduce a novel technique for this, which stores a deterministic node as an array of function values rather than one of probabilities. We propose a variable elimination algorithm, including a new elimination heuristic, that maximally exploits this encoding using *factor indexing*. A preliminary empirical evaluation gives promising results.

1 Introduction

In general, exact inference on a Bayesian network is known to take $O(d^w)$ time, where d is the number of states per variable (assumed that these are the same for each variable) and w is the treewidth of the network's moral graph (Dechter, 1999). In the canonical technique for exact inference, *variable elimination* (Zhang and Poole, 1996), this constraint manifests itself as the minimal size of the largest factor that is created during the execution of the algorithm; implemented as a multidimensional array, it has w dimensions and d entries for each dimension.

When a network contains deterministic nodes, inference can be much faster. One example where this can be seen is the approach of Chavira and Darwiche (2008), in which a Bayesian network is transformed into a logical theory, and inference is performed by counting the models of this theory. These models should be consistent with the constraints imposed by the deterministic nodes. A good model counting algorithm can use these constraints effectively to prune the model search space.

A different approach (Larkin and Dechter, 2003) stays closer to variable elimination. Here, a factor is implemented not as an array (with an entry for each possible variable assignment), but as a list of variable assignments that are nonzero (sometimes called a *sparse array*). The length of this list can be much smaller than the size of the array, but the overhead for multiplying and marginalizing factors is larger, because the list has to be searched for values (possibly using a hash table). With this alternative implementation of factors, ordinary variable elimination can be performed.

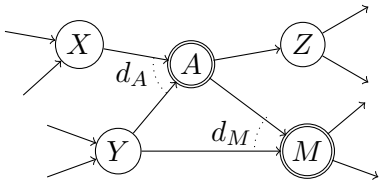

Figure 1: Fragment of a Bayesian network. Variables A and M are deterministic (indicated by a double border): their values follow directly from their parents' values. This is reflected in A's conditional probability distribution: $\mathsf{P}(a|x,y)$ equals 1 if $a = d_A(x,y)$, and 0 otherwise.

The approach we present in this article is also based on a cheaper implementation, but only of the deterministic factors. For example, consider deterministic variable A in Fig. 1. Conventionally, the corresponding factor is stored as a three-dimensional array of conditional probabilities $\mathsf{P}(a|x,y)$, which can take two values: 1 if $a = d_A(x,y)$, 0 if $a \neq d_A(x,y)$. Here, d_A is the function that determines A given X and Y. In our approach, we store this function directly, as a two-dimensional array. If A has n possible states, this array contains n times as few elements as the original array of probabilities.

The true merit of our approach, however, lies not in the efficient space use of the arrays that make up the definition of the Bayesian network; it lies in the ability to propagate this efficiency to the intermediate arrays used in variable elimination. For example, consider the elimination of variable A from Fig. 1. In conventional variable elimination, this requires the calculation of $\sum_a \mathsf{P}(a|x,y)\mathsf{P}(z|a)\mathsf{P}(m|a,y)$ for each combination x, y, z, m; a four-dimensional array of which each element is the result of summing n probabilities. On the other hand, our approach exploits the fact that

$$\begin{aligned}&\textstyle\sum_a \mathsf{P}(a|x,y)\mathsf{P}(z|a)\mathsf{P}(m|a,y)\\ &= \mathsf{P}(z|A{=}d_A(x,y))\mathsf{P}(m|A{=}d_A(x,y),y), \quad (1)\end{aligned}$$

i.e. we construct two low-dimensional arrays (over variables XYZ and MXY, resp.) and perform no summation at all. Moreover, we do not even need to multiply these arrays at this point yet; we could perhaps first eliminate Z from the former.

We integrate this operation into variable elimination by expressing it in *factor algebra*, a convenient language to describe the 'bulk operations' on multidimensional arrays that occur in most inference procedures. To the multiplication and summation operations usually encountered, we add an indexing operation written $cpd_Z[A{=}d_A]$ (for the first of the two arrays above). This *partially* indexes the array cpd_Z, which we will define as to contain the probabilities $\mathsf{P}(z|a)$, using *another array*, namely d_A. Its implementation does not require the overhead of sparse arrays, as no additional data structures are used.

The remainder of the article has the following outline. Sect. 2 summarizes the formal preliminaries for inference on Bayesian networks. In Sect. 3, we review variable elimination, with an emphasis on the use of factor algebra. Our main contribution, *factor indexing*, is presented in Sect. 4, followed by an empirical evaluation in Sect. 5. In Sect. 6, we conclude and propose future work.

2 Formal preliminaries

A *Bayesian network* is a triple $(\mathbf{V}, \mathsf{par}, \mathbf{cpd})$. The set $\mathbf{V} = \{V_1, \ldots, V_n\}$ consists of n discrete variables; each V_i has a finite domain $\mathrm{dom}(V_i)$. The function par maps each variable V_j to a set of *parents* $\mathbf{V}_{\mathsf{par}(j)} \subset \mathbf{V}$ in such a way that there are no cycles. The set $\mathbf{cpd} = \{cpd_1, \ldots, cpd_n\}$ contains, for each variable V_j, the family of conditional probability distributions $\mathsf{P}(v_j|\mathbf{v}_{\mathsf{par}(j)})$; in Sect. 3, we will define this family as a *factor* that we designate cpd_j. A Bayesian network defines a joint probability distribution over $\mathbf{V}$: $\mathsf{P}(\mathbf{v}) = \prod_{1\le j\le n} \mathsf{P}(v_j|\mathbf{v}_{\mathsf{par}(j)})$.

An *inference query* is a usually defined as the conditional probability distribution $\mathsf{P}(\mathbf{q}|\mathbf{e})$ over some query variables $\mathbf{Q} \subseteq \mathbf{V}$ given an instantiation $\mathbf{e}$ of evidence variables $\mathbf{E} \subseteq \mathbf{V}$. However, for simplicity we define an inference query as the distribution $\mathsf{P}(\mathbf{q}, \mathbf{e})$ in this article. This distribution can be derived from the joint distribution by summing out the remaining variables $\mathbf{R} = \mathbf{V} \setminus (\mathbf{Q} \cup \mathbf{E})$:

$$\mathsf{P}(\mathbf{q},\mathbf{e}) = \sum_{\mathbf{r}\in\mathbf{R}} \mathsf{P}(\mathbf{q},\mathbf{e},\mathbf{r}) = \sum_{\mathbf{r}\in\mathbf{R}} \prod_{1\le j\le n} \mathsf{P}(v_j|\mathbf{v}_{\mathsf{par}(j)})$$

From this, the conditional distribution can easily be derived: $\mathsf{P}(\mathbf{q}|\mathbf{e}) = \mathsf{P}(\mathbf{q},\mathbf{e}) / \sum_{\mathbf{q}\in\mathbf{Q}} \mathsf{P}(\mathbf{q},\mathbf{e})$.

3 Factor algebra for variable elimination

In this section, we review the theory of factor algebra and variable elimination (Zhang and Poole, 1996), into which we will integrate our new factor indexing operation in Sect. 4. Many inference algorithms, including variable elimination and junction tree propagation (Lauritzen and Spiegelhalter, 1988), are implemented using multidimensional arrays; a *factor* is a mathematical description of such an array, and *factor algebra* is a convenient language to describe array operations.

Formally, a *factor* f over variables $\mathbf{V}$ is a function that maps every instantiation $\mathbf{v}$ of $\mathbf{V}$ to a number $f(\mathbf{v})$; an *instantiation* $\mathbf{v} = \{V_1{=}v_1, \ldots, V_n{=}v_n\}$ is a function mapping each V_j to a value $v_j \in \mathrm{dom}(V_j)$. We refer to the set $\dim(f) = \mathbf{V}$ as f's *dimensionality*. Thus,

each cpd_j in a Bayesian network is a factor with $\dim(cpd_j) = V_j \cup \mathbf{V}_{\mathsf{par}(j)}$ and values

$$cpd_j(V_j{=}v_j, \mathbf{v}_{\mathsf{par}(j)}) = \mathsf{P}(v_j|\mathbf{v}_{\mathsf{par}(j)})$$

The *weight* of a factor f is defined as the number of different instantiations it can be applied to, and equals the size of the array needed to store all f's values:

$$\mathsf{weight}(f) \stackrel{\text{def}}{=} \prod_{V_j \in \dim(f)} |\mathrm{dom}(V_j)|$$

It is also possible to apply a factor to an instantiation $\mathbf{e}$ of a subset of its dimensions ($\mathbf{E} \subset \mathbf{V}$): then the result $f(\mathbf{e})$ is not real number, but a factor over $\mathbf{V} \setminus \mathbf{E}$. We also find it convenient to use instantiations containing *more* variables than are in the factor's domain; in this case, the superfluous variables are just ignored. So, for example, $f(\mathbf{v}, V_{n+1}{=}y) = f(\mathbf{v})$.

An inference query can then be defined as the factor $inf_{\mathbf{Q},\mathbf{E}}$:

$$inf_{\mathbf{Q},\mathbf{E}}(\mathbf{q},\mathbf{e}) \stackrel{\text{def}}{=} \mathsf{P}(\mathbf{q},\mathbf{e}) = \sum_{\mathbf{r}\in\mathbf{R}} \prod_{1\le j\le n} cpd_j(\mathbf{q},\mathbf{e},\mathbf{r})$$

where we apply each cpd_j to a lot of superfluous variables. In fact, $inf_{\mathbf{Q},\mathbf{E}}$ is a factor over $\mathbf{Q}\cup\mathbf{E}$ and contains results for all instantiations of $\mathbf{E}$; however, one is usually interested in the result for specific evidence $\mathbf{e}$. Then, the inference goal is to calculate the partial instantiation $inf_{\mathbf{Q},\mathbf{E}}(\mathbf{e})$, a factor over $\mathbf{Q}$.

The basic factor algebra we use contains an operator $\odot$ for multiplying two factors, a unit element $\mathbb{1}$ for $\odot$, and a summation operator $\Sigma_{\mathbf{W}}$ that sums out the $\mathbf{W}$ dimensions of a factor:

$$(f \odot g)(\mathbf{v}) \stackrel{\text{def}}{=} f(\mathbf{v}) \cdot g(\mathbf{v})$$
$$\mathbb{1}() \stackrel{\text{def}}{=} 1$$
$$(\Sigma_{\mathbf{W}} f)(\mathbf{u}) \stackrel{\text{def}}{=} \sum_{\mathbf{w}\in\mathbf{W}} f(\mathbf{u},\mathbf{w})$$

where we define $\dim(f \odot g) = \dim(f) \cup \dim(g)$, $\dim(\mathbb{1}) = \emptyset$, and $\dim(\Sigma_{\mathbf{W}} f) = \dim(f) \setminus \mathbf{W}$. Note that we use $\mathbf{w}{\in}\mathbf{W}$ to let variable $\mathbf{w}$ range over all possible instantiations of $\mathbf{W}$. In Sect. 4, we extend this basic factor algebra.

Algorithm 1: Variable elimination. Initialize the set of factors f_j to the conditional probability distributions. In each iteration, heuristically choose a variable V_i. From the current set of factors, replace all that have V_i in their domain by their product, with V_i summed out. For the definition of the cost heuristic, see the running text.

Input:

- Bayesian network $(\mathbf{V}, \mathsf{par}, \{cpd_1, \ldots, cpd_n\})$
- evidence $\mathbf{e}$ (instantiation of $\mathbf{E} \subset \mathbf{V}$)
- query variables $\mathbf{Q} \subset \mathbf{V}$

Output: $inf_{\mathbf{Q},\mathbf{E}}(\mathbf{e})$ (a factor over $\mathbf{Q}$)

$\mathbf{W} := \mathbf{V} \setminus (\mathbf{Q} \cup \mathbf{E})$
foreach cpd_j **do** $f_j := cpd_j(\mathbf{e})$

while $\mathbf{W}$ is not empty **do**
  choose $V_i \in \mathbf{W}$ for which the cost of *eliminate*(V_i) is smallest
  eliminate(V_i)
  $\mathbf{W} := \mathbf{W} \setminus \{V_i\}$

$inf_{\mathbf{Q},\mathbf{E}}(\mathbf{e}) := \bigodot\{\text{all remaining } f_j\}$

procedure *eliminate*(V_i)
  $p := \mathbb{1}$
  foreach f_j s.t. $V_i \in \dim(f_j)$ **do**
    $p := p \odot f_j$
    delete f_j
  $f_i := \Sigma_{V_i} p$

Variable elimination is now formulated as a procedure (Alg. 1) that stepwise constructs the factor $inf_{\mathbf{Q},\mathbf{E}}(\mathbf{e})$ out of the set of factors $cpd_1, \ldots, cpd_n$, using above operators. In each step, all the factors in the current set that contain a certain variable V_i are joined together using $\odot$, after which Σ_{V_i} is applied to the result.

As for which V_i to choose next, Alg. 1 uses a greedy heuristic: it always takes the one with minimal cost. We define this cost to be the size of the largest array constructed in this step, i.e. $\mathsf{weight}(p)$ for the final value of p in *eliminate*(V_i). This heuristic is known as the minweight heuristic; other heuristics are also possible, but minweight is known in practice to

Table 1: Laws of factor algebra.

$$f \odot \mathbb{1} = f \tag{2}$$
$$f \odot g = g \odot f \tag{3}$$
$$f \odot (g \odot h) = (f \odot g) \odot h = \bigodot\{f, g, h\} \tag{4}$$
$$\Sigma_V \Sigma_W f = \Sigma_W \Sigma_V f = \Sigma_{V,W} f \tag{5}$$
$$\Sigma_V (f \odot g) = \Sigma_V f \odot g \quad \text{if } V \in \dim(f),\ V \notin \dim(g) \tag{6}$$
$$\Sigma_V (f \odot g) = f \odot \Sigma_V g \quad \text{if } V \notin \dim(f),\ V \in \dim(g) \tag{7}$$
$$\big(f \odot g\big)(\mathbf{e}) = f(\mathbf{e}) \odot g(\mathbf{e}) \tag{8}$$
$$\big(\Sigma_V f\big)(\mathbf{e}) = \Sigma_V f(\mathbf{e}) \quad \text{if } \mathbf{e} \text{ does not instantiate } V \tag{9}$$

perform best when variables have different domain sizes (Kjærulff, 1990). Note that the algorithm can calculate the cost of *eliminate*(V_i) before actually executing it, as it is not necessary to construct the array p to determine $\mathsf{weight}(p)$.

As a matter of fact, although Alg. 1 can certainly be read to perform array operations at factor assignments such as $p := p \odot f_j$ and $f_i := \Sigma_{V_i} p$, it does not have to perform any array operations *at all*. Instead, it can perform a symbolic construction of a new factor algebra expression at these points. In that case, the result of the algorithm is not an array, but a large symbolic expression which can be evaluated at a later stage to produce said array. Thus, the inference procedure is divided into a *search* phase and an *evaluation* phase.

Correctness

Factor algebra is not only a tool for expressing variable elimination concisely, but also for analyzing its correctness. Using some general laws of factor algebra (see Table 1), we can prove that the factor constructed by Alg. 1 indeed corresponds to the factor $\mathit{inf}_{\mathbf{Q},\mathbf{E}}(\mathbf{e})$ specified in the text above. Making use of the definitions of $\odot$ and $\Sigma_{\mathbf{R}}$, we rewrite the definition of $\mathit{inf}_{\mathbf{Q},\mathbf{E}}$ into:

$$\mathit{inf}_{\mathbf{Q},\mathbf{E}} = \Sigma_{\mathbf{R}} \bigodot_{1 \le j \le n} \mathit{cpd}_j$$

Using laws (8) and (9), the instantiation of evidence is pushed into the expression:

$$\mathit{inf}_{\mathbf{Q},\mathbf{E}}(\mathbf{e}) = \Sigma_{\mathbf{R}} \bigodot_{1 \le j \le n} \mathit{cpd}_j(\mathbf{e})$$

Now, we will prove that the following invariant holds at the start of each iteration:

$$\mathit{inf}_{\mathbf{Q},\mathbf{E}}(\mathbf{e}) = \Sigma_{\mathbf{W}} \bigodot \{\text{all remaining } f_j\}$$

For the first iteration, this is trivial, as $\mathbf{W}$ was set to $\mathbf{R}$ and each f_j to $\mathit{cpd}_j(\mathbf{e})$. Next, eliminating a variable V_i corresponds to the following rewriting:

$$\begin{aligned} & \Sigma_{\mathbf{W}} \bigodot \{\text{all remaining } f_j\} \\ = \quad & \Sigma_{\mathbf{W}} \left(\bigodot_{V_i \notin \dim(f_j)} f_j \ \odot \bigodot_{V_i \in \dim(f_j)} f_j \right) \\ = \ & \Sigma_{\mathbf{W} \setminus V_i} \left(\bigodot_{V_i \notin \dim(f_j)} f_j \ \odot \ \Sigma_{V_i} \bigodot_{V_i \in \dim(f_j)} f_j \right) \end{aligned}$$

in which the product is restructured using (3,4) into a group that does not contain V_i and one that does; next, distributive law (7) is used to push the summation over V_i into the expression.

The bottom expression corresponds to the invariant for the next iteration, where $\mathbf{W}$ is set to $\mathbf{W} \setminus \{V_i\}$, and the f_j factors with $V_i \in \dim(f_j)$ have been replaced with

$$f_i := \Sigma_{V_i} \bigodot_{V_i \in \dim(f_j)} f_j$$

After the last loop, $\mathbf{W}$ is empty, so

$$\mathit{inf}_{\mathbf{Q},\mathbf{E}}(\mathbf{e}) = \bigodot \{\text{all remaining } f_j\}$$

which is what we wanted to prove.

4 Factor indexing

This section presents the main contribution of this article: *factor indexing*, and its integration in variable elimination. We propose a new factor algebra operator and laws, and extend Alg. 1 into Alg. 2, which uses them to eliminate deterministic variables.

A variable $Y \in \mathbf{V}$ is called *deterministic* if its value is functionally determined by the value of its parents (here $\mathbf{X} \subset \mathbf{V}$). This means that its conditional probability distribution has the following form:

$$cpd_Y(Y{=}y, \mathbf{x}) = \begin{cases} 1 & \text{if } y = d_Y(\mathbf{x}) \\ 0 & \text{if } y \neq d_Y(\mathbf{x}) \end{cases}$$

where d_Y is a factor over $\mathbf{X}$ with values in $\mathrm{dom}(Y)$, which we call Y's *deterministic factor*.

To account for deterministic variables, we extend the definition of a Bayesian network as follows: next to the set **cpd** of conditional probability distributions for non-deterministic variables $V_1, \ldots, V_m$, we include a set $\mathbf{d}$ of factors for deterministic variables $V_{m+1}, \ldots, V_n$. Factor d_j is then a factor over $\mathsf{par}(V_j)$ with values in $\mathrm{dom}(V_j)$. So, unlike a cpd_j factor, $V_j \notin \dim(d_j)$.

Like d_Y and cpd_Y above, every deterministic factor d has a 'probabilistic representation'. Although we want to keep a factor deterministic whenever possible during variable elimination, sometimes it is unavoidable to translate it to its probabilistic representation. For this, we define the factor algebra operator $\mathbb{1}_{V=d}$:

$$\mathbb{1}_{V=d}(V{=}v, \mathbf{u}) \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } v = d(\mathbf{u}) \\ 0 & \text{if } v \neq d(\mathbf{u}) \end{cases}$$

where $\dim(d) = \mathbf{U}$, and $\dim(\mathbb{1}_{V=d}) = \{V\} \cup \mathbf{U}$.[1]

So, translating a network with deterministic variables into a conventional one can now be defined as $cpd_j = \mathbb{1}_{V_j=d_j}$ for all $m < j \leq n$.

Consider the nodes A and M in Fig. 1 with the following deterministic factors:

$$d_A(X{=}x, Y{=}y) = x + y$$
$$d_M(A{=}a, Y{=}y) = a \cdot y$$

Conventionally, the probabilistic representations of d_A and d_M are used for variable elimination. For example, when eliminating variable A, the following expression would be constructed:

$$\Sigma_A(\mathbb{1}_{A=d_A} \odot \mathbb{1}_{M=d_M} \odot cpd_Z)$$

[1] From these dimensionalities, it immediately follows that $\mathsf{weight}(\mathbb{1}_{V=d}) = |\mathrm{dom}(V)| \cdot \mathsf{weight}(d)$, so an array storing the probabilistic representation has $|\mathrm{dom}(V)|$ as many elements as one storing the deterministic factor.

Let us focus on the values of sub-expression $\mathbb{1}_{A=d_A} \odot \mathbb{1}_{M=d_M}$, a factor over $XYAM$:

$$(\mathbb{1}_{A=d_A} \odot \mathbb{1}_{M=d_M})(X{=}x, Y{=}y, A{=}a, M{=}m) = \begin{cases} 1 & \text{if } a{=}x{+}y \wedge m{=}a{\cdot}y \\ 0 & \text{otherwise} \end{cases}$$

Note that we can rewrite the condition into $a{=}x{+}y \wedge m{=}(x{+}y){\cdot}y$, so m depends on x instead of a. This suggests that we can rewrite the factor into $\mathbb{1}_{A=d_A} \odot \mathbb{1}_{M=d'_M}$, with

$$d'_M(X{=}x, Y{=}y) = (x{+}y){\cdot}y$$

Similarly, cpd_Z can be transformed into

$$cpd'_Z(X{=}x, Y{=}y, Z{=}z) = \mathsf{P}(Z{=}z \,|\, A{=}x{+}y)$$

Neither d'_M nor cpd'_Z contain variable A anymore. Therefore, after rewriting the variable elimination expression using these new factors, the summation over A can be pushed inwards:

$$\begin{aligned} &\Sigma_A(\mathbb{1}_{A=d_A} \odot \mathbb{1}_{M=d_M} \odot cpd_Z) \\ &= \Sigma_A(\mathbb{1}_{A=d_A} \odot \mathbb{1}_{M=d'_M} \odot cpd'_Z) \\ &= \Sigma_A \mathbb{1}_{A=d_A} \odot \mathbb{1}_{M=d'_M} \odot cpd'_Z \end{aligned}$$

Examining the first factor, we see that

$$\begin{aligned} &(\Sigma_A \mathbb{1}_{A=d_A})(X{=}x, Y{=}y) \\ &= \sum_{a \in \mathrm{dom}(A)} (1 \text{ iff } a{=}x{+}y) \quad = 1 \end{aligned}$$

because for each combination x, y there is only one a s.t. $a{=}x{+}y$. Thus, we conclude that

$$\Sigma_A(\mathbb{1}_{A=d_A} \odot \mathbb{1}_{M=d_M} \odot cpd_Z) = \mathbb{1}_{M=d'_M} \odot cpd'_Z$$

where the right hand side represents a much cheaper way to eliminate A than the left hand side. Note that this is the factor algebra equivalent of Eq. 1 (see *Introduction*).

In order to integrate this rewrite rule into variable elimination, we will now formalize the transformations $d_M \Rightarrow d'_M$ and $cpd_Z \Rightarrow cpd'_Z$ in factor algebra.

We do this by introducing a new operation of the form $f[V{=}d]$, where factor f is *indexed* by factor d in dimension V:

$$f[V{=}d](\mathbf{u}) \stackrel{\text{def}}{=} f(\mathbf{u}, V{=}d(\mathbf{u}))$$

where the dimensionality of resulting factor $f[V{=}d]$ is $(\dim(f) \setminus \{V\}) \cup \dim(d)$— so $\mathbf{u}$ is an instantiation over these variables.

With this operation, we can define the above transformations as follows:

$$d'_M = d_M[A{=}d_A]$$
$$cpd'_Z = cpd_Z[A{=}d_A]$$

Note that, contrary to conventional indexing, the dimensionality of $f[V{=}d]$ can be larger than that of f. For example, $\dim(cpd'_Z) = XYZ$, while $\dim(cpd_Z) = AZ$.

To the laws of factor algebra (Table 1), we add the following:

$$f[V{=}d] = \Sigma_V(f \odot \mathbb{1}_{V=d}) \tag{10}$$
$$(f \odot g)[V{=}d] = f[V{=}d] \odot g[V{=}d] \tag{11}$$
$$\mathbb{1}_{W=f[V=d]} = \mathbb{1}_{W=f}[V{=}d] \quad \text{if } V \neq W \tag{12}$$

Using these, we can generalize the rewrite rule above. The elimination of deterministic variable V_i from a product of factors f_j and deterministic factors d_j can be rewritten as follows:

$$\Sigma_{V_i}\left(\mathbb{1}_{V_i=d_i} \odot \bigodot_{V_j \in \dim(f_j)} f_j \odot \bigodot_{V_j \in \dim(d_j)} \mathbb{1}_{V_j=d_j}\right)$$
$$= \bigodot_{V_j \in \dim(f_j)} f_j[V_i{=}d_i] \odot \bigodot_{V_j \in \dim(d_j)} \mathbb{1}_{V_j=d_j[V_i=d_i]} \tag{13}$$

We apply this in a variable elimination algorithm with factor indexing (Alg. 2). It has the same structure as Alg. 1, but is extended as follows:

- For a deterministic variable V_i, we store d_i instead of $\mathbb{1}_{V_i=d_i}$.
- To eliminate a deterministic variable V_i, we use Eq. 13: we index all currently existing f_j and d_j factors over V_i by $V_i{=}d_i$, and delete d_i itself.
- Not all deterministic variables are eliminated like this: during the elimination of a non-deterministic variable V_i, all deterministic factors over V_i have to be expanded to their probabilistic representation. Also, for a deterministic *evidence* variable, its factor is expanded during initialization.

Algorithm 2: Variable elimination with factor indexing. Next to the set of factors f_j, maintain a set of deterministic factors d_j. When eliminating a deterministic variable V_i (for which d_i still exists), do not replace factors by their product but *index* them by d_i. Note: in the absence of deterministic nodes, the algorithm 'degenerates' to Alg. 1.

Input:

- Bayesian network w/ deterministic nodes $(\mathbf{V}, \mathsf{par}, \{cpd_1, \ldots, cpd_m\}, \{d_{m+1}, \ldots, d_n\})$
- evidence $\mathbf{e}$ (instantiation of $\mathbf{E} \subset \mathbf{V}$)
- query variables $\mathbf{Q} \subset \mathbf{V}$

Output: $inf_{\mathbf{Q},\mathbf{E}}(\mathbf{e})$ (a factor over $\mathbf{Q}$)

$\mathbf{W} := \mathbf{V} \setminus (\mathbf{Q} \cup \mathbf{E})$
foreach cpd_j **do** $f_j := cpd_j(\mathbf{e})$
foreach d_j **do**
  if $V_j \in \mathbf{E}$ **then**
    $f_j := \mathbb{1}_{V_j=d_j}(\mathbf{e})$
  else
    $d_j := d_j(\mathbf{e})$

while $\mathbf{W}$ is not empty **do**
  choose $V_i \in \mathbf{W}$ for which the cost of $eliminate(V_i)$ is smallest
  $eliminate(V_i)$
  $\mathbf{W} := \mathbf{W} \setminus \{V_i\}$

$inf_{\mathbf{Q},\mathbf{E}}(\mathbf{e}) := (\bigodot\{\text{all remaining } f_j\}) \odot \bigodot \{\, \mathbb{1}_{V_j=d_j} \mid \text{all remaining } d_j \,\}$

procedure $eliminate(V_i)$
  if d_i exists **then**
    foreach d_j s.t. $V_i \in \dim(d_j)$ **do**
      $d_j := d_j[V_i = d_i]$
    foreach f_j s.t. $V_i \in \dim(f_j)$ **do**
      $f_j := f_j[V_i = d_i]$
    delete d_i
  else
    $p := \mathbb{1}$
    foreach d_j s.t. $V_i \in \dim(d_j)$ **do**
      $p := p \odot \mathbb{1}_{V_j=d_j}$
      delete d_j
    foreach f_j s.t. $V_i \in \dim(f_j)$ **do**
      $p := p \odot f_j$
      delete f_j
    $f_i := \Sigma_{V_i} p$

The used elimination heuristic is still the *cost* of the next elimination step. However, the definition of this cost is also extended. If V_i has *no* deterministic factor d_j associated with it, the cost is still $\mathsf{weight}(p)$. If it *does*, the cost is

$$\sum_{V_i \in \dim(f_j)} \mathsf{weight}(f_j[V_i{=}d_i]) + \sum_{V_i \in \dim(d_j)} \mathsf{weight}(d_j[V_i{=}d_i])$$

Although space does not permit it here, a correctness proof can also be given for Alg. 2. The invariant is:

$$inf_{\mathbf{Q},\mathbf{E}}(\mathbf{e}) = \Sigma_{\mathbf{W}} \Big(\Big(\bigodot \{\text{all remaining } f_j\} \Big) \odot \bigodot \big\{ \mathbb{1}_{V_j=d_j} \,\big|\, \text{all remaining } d_j \big\} \Big)$$

5 Empirical evaluation

We have implemented the factor algebra described above in Python, using the package NumPy which provides an n-dimensional array and executes array operations using fast C loops (not unlike MATLAB). The $\odot$ operator directly translates to NumPy's array multiplication, which can handle the situation where the operands have different dimensions. Indexing an array with another array is supported in NumPy as well.

We perform inference on 4 networks with deterministic nodes known from the Bayesian network literature (the students network is from the UAI'08 evaluation track). We also investigated 6 generated networks of 100 nodes, with 30 root nodes and 70 nodes with 2 parents (randomly chosen from earlier generated nodes). Each node has randomly generated probabilities; each of the 70 non-root nodes has a chance of being deterministic, in which case we randomly generate a deterministic function. Each variable has the same domain; between networks, we vary the domain size (2 or 4). Also, we vary the fraction of deterministic nodes (30%, 60%, 90% of the non-root nodes).

For each network, we take medians over 10 runs; in each run, we instantiate 10 randomly chosen[2] evidence variables $\mathbf{e}$ and choose one random query variable Q. Then we use algorithms Alg. 1 and Alg. 2 to generate a symbolic expression (a *plan*) for $inf_{\mathbf{Q},\mathbf{E}}(\mathbf{e})$, i.e. we execute them as a *search phase* as discussed in Sect. 3. As it is completely implemented in Python (without regard for speed), we do not time this phase; its performance would severely distort the overall timing results.

We record the cost of the generated plans, i.e. the summed weight of all the intermediate factors. In the second phase, we evaluate the plans and record the (wall clock) duration. The experiments were performed on a machine with a 3GHz Intel Core2Duo processor and 2GB RAM.

Results are shown in Table 2: the factor indexing technique provides speedups ranging up to 16×. Expectations are confirmed that it works best with a high fraction of deterministic nodes and/or larger domain sizes. However, we noticed that the variance in performance between runs can be high: we suspect that the current heuristic can easily guide the algorithm in the wrong way, and will investigate more robust heuristics in the future.

6 Conclusions and future work

We propose a new variable elimination technique for exact inference on Bayesian networks, in which deterministic variables are eliminated not by summation but by a factor indexing operation. We emphasize the role of factor algebra, which enables (a) a concise definition of the algorithm, (b) a straightforward correctness proof, and (c) a model for defining an elimination order heuristic in terms of the *cost* of array operations. Indeed, our updated heuristic has little to do with the network's graph structure anymore; this is in line with common knowledge that treewidth is not so important for highly deterministic networks.

A preliminary empirical evaluation shows that the technique performs decently on real-world networks (small speedups) and good on randomly generated networks (speedups of 1–16). We expect much room for improvement here: first, by developing heuristics that take into account the *actual* cost of performing

[2] However, for students, we took the 9 easiest evidence files from the UAI'08 evaluation.

Table 2: Experimental results. Numbers are median values over 10 random queries.

network	# vars (det.)	plan cost Alg. 1	plan cost Alg. 2	cost impr. Alg. 1/Alg. 2	eval. time (s) Alg. 1	eval. time (s) Alg. 2	speedup Alg. 1/Alg. 2
munin-1	189 (65)	278M	260M	1.00	6.94	7.91	0.935
munin-4	1041 (411)	23.3M	19.2M	1.22	0.481	0.382	1.25
diabetes	413 (24)	13.2M	13.1M	1.00	0.148	0.151	0.994
students	376 (304)	4.32M	14.7K	293	0.205	0.053	4.13
random-2-30	100 (±21)	16.3K	3.85K	2.91	0.0120	0.0106	1.15
random-2-60	100 (±42)	19.6K	2.47K	5.82	0.0121	0.0088	1.35
random-2-90	100 (±63)	14.6K	0.711K	15.0	0.0117	0.0064	1.90
random-4-30	100 (±21)	6.28M	2.38M	9.23	0.122	0.0536	5.38
random-4-60	100 (±42)	2.27M	49.0K	55.1	0.0504	0.0098	5.39
random-4-90	100 (±63)	4.41M	14.7K	257	0.0908	0.0065	16.3

the different array operations instead of the size of the resulting array; second, by exploiting low-level machine knowledge to decrease these actual costs. For example, current CPUs and GPUs often feature vectorized processing modes, which we expect can be exploited for the bulk array operations of probabilistic inference. When used properly, this might outperform inference techniques for determinism that cannot be expressed as array operations, e.g. (Chavira and Darwiche, 2008; Larkin and Dechter, 2003).

Furthermore, we argue that our technique has much potential for combination with other inference algorithms, e.g. with junction tree propagation (Lauritzen and Spiegelhalter, 1988), recursive conditioning (Darwiche, 2001) and factor decomposition techniques (Heckerman and Breese, 1996; Vomlel, 2002; Díez and Galán, 2003).

Acknowledgements

The authors have been supported by the OCTOPUS project under the responsibility of the Embedded Systems Institute. The OCTOPUS project is partially supported by the Netherlands Ministery of Economic Affairs under the Embedded Systems Institute program.

References

Mark Chavira and Adnan Darwiche. 2008. On probabilistic inference by weighted model counting. *Artif. Intell.*, 172(6-7):772–799.

Adnan Darwiche. 2001. Recursive conditioning. *Artif. Intell.*, 126(1-2):5–41.

Rina Dechter. 1999. Bucket elimination: A unifying framework for reasoning. *Artif. Intell.*, 113(1-2):41–85.

Francisco Javier Díez and Severino F. Galán. 2003. Efficient computation for the Noisy MAX. *Int. J. Intell. Syst.*, 18(2):165–177.

D. Heckerman and J. S. Breese. 1996. Causal independence for probability assessment and inference using Bayesian networks. *IEEE Transactions on Systems, Man and Cybernetics, Part A*, 26(6):826–831.

Uffe Kjærulff. 1990. Triangulation of graphs — algorithms giving small total state space. Technical Report R-90-09, Dept. of Mathematics and Computer Science, Aalborg University.

David Larkin and Rina Dechter. 2003. Bayesian inference in the presence of determinism. In C M Bishop and B J Frey, editors, *Proceedings of Ninth International Workshop on Artificial Intelligence and Statistics*, Key West, USA.

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

Jiří Vomlel. 2002. Exploiting functional dependence in Bayesian network inference. In *UAI02*, pages 528–535.

Nevin Lianwen Zhang and David Poole. 1996. Exploiting causal independence in bayesian network inference. *J. Artif. Intell. Res. (JAIR)*, 5:301–328.