Inference Optimization using Relational Algebra

Sander Evers, Maarten M. Fokkinga, and Peter M.G. Apers

University of Twente, The Netherlands

Abstract. Exact inference procedures in Bayesian networks can be expressed using relational algebra; this provides a common ground for optimizations from the AI and database communities. Specifically, the ability to accomodate *sparse representations* of probability distributions opens up the way to optimize for their *cardinality* instead of the *dimensionality*; we apply this in a sensor data model.

1 Introduction

Since their conception in the 1980s, Bayesian networks have rapidly become a de facto standard in the AI community for concisely and intuitively representing a probabilistic model. Recently, (dynamic) Bayesian networks have also received much interest from the database community for processing sensor data; e.g. see [1]. However, although it has been known for over a decade that exact inference in Bayesian networks can be formulated as a relational database query [2], the area of inference optimization has not yet seen a lot of interdisciplinary work.

In this article, we advocate the use of *relational algebra* in inference procedures to bridge the gap between the two communities. In database management systems, relational algebra is used to represent queries at a level between the input query language (usually SQL) and the language in which they are executed, and plays an essential role in optimization of these queries. This is possible because a relational algebra expression has a *denotational semantics* specifying *what* is calculated, and a *operational semantics* specifying *how* it is calculated; a query is optimized by substituting (sub)expressions with equivalent denotational semantics but more efficient operational semantics.

Our contributions are twofold: after reviewing Bayesian network inference in section 2, we (a) show an intimate link between numeric probability expressions and relational algebra expressions which makes it possible to write and manipulate inference procedures using relational algebra (section 3), and (b) apply this theory to a sensor data model, improving its scalability (section 4): when the number of variables K and the number of values L for one particular variable are jointly increased, inference time scales sublinearly, where using conventional methods it scales quadratically. This optimization is possible because relational algebra accommodates a *sparse representation* of probability distributions, which can exploit sparsity that is not visible in the structure of the Bayesian network.

2 Bayesian Network Inference

A Bayesian network [3] represents a probabilistic model over a set of n discrete stochastic variables $\mathbf{V} = \{V_1, \ldots, V_n\}$, and consists of:

1. A directed acyclic graph $(\mathbf{V}, \mathbf{P})$ with the variables as nodes. Variable V_h is called a *parent* of V_i if $(V_h \mapsto V_i) \in \mathbf{P}$. This induces a function *par* on the indices:
$$par(i) \stackrel{\text{def}}{=} \{\, h \mid 1 \leq h \leq n, (V_h \mapsto V_i) \in \mathbf{P} \,\}$$
2. For each V_i, the conditional probability distribution (cpd) $\mathsf{P}(v_i|\mathbf{v}_{par(i)})$.

The joint probability distribution defined by this Bayesian network is the product of these cpds:
$$\mathsf{P}(\mathbf{v}) = \prod_{1 \leq i \leq n} \mathsf{P}(v_i|\mathbf{v}_{par(i)})$$

An *inference query* $\mathsf{P}(\mathbf{v}_Q|\mathbf{v}_E)$ partitions the variables $\mathbf{V}$ into query variables $\mathbf{V}_Q$, evidence variables $\mathbf{V}_E$ and the remaining variables $\mathbf{V}_R$. The goal is to calculate $\mathsf{P}(\mathbf{v}_Q|\mathbf{v}_E)$ for all values $\mathbf{v}_Q$, given certain fixed values $\mathbf{v}_E$. The probabilities $\mathsf{P}(\mathbf{v}_Q, \mathbf{v}_E)$ also suffice; using these, the former can be calculated as $\mathsf{P}(\mathbf{v}_Q|\mathbf{v}_E) = \mathsf{P}(\mathbf{v}_Q, \mathbf{v}_E)/\mathsf{P}(\mathbf{v}_E) = \mathsf{P}(\mathbf{v}_Q, \mathbf{v}_E)/\sum_{\mathbf{v}_Q} \mathsf{P}(\mathbf{v}_Q, \mathbf{v}_E)$. Hence, to simplify expositions, we will hereafter equate inference with the calculation of $\mathsf{P}(\mathbf{v}_Q, \mathbf{v}_E)$ for all $\mathbf{v}_Q$. Substituting the definition of the joint probability for the Bayesian network gives:
$$\mathsf{P}(\mathbf{v}_Q, \mathbf{v}_E) = \sum_{\mathbf{v}_R} \mathsf{P}(\mathbf{v}) = \sum_{\mathbf{v}_R} \prod_{1 \leq i \leq n} \mathsf{P}(v_i|\mathbf{v}_{par(i)}) \tag{1}$$

The right hand side of this equation suggests a naive approach for performing the calculation: determine the value of the product (using the fixed $\mathbf{v}_E$) for all $\mathbf{v}_R$ values, and sum these products; repeat this for each $\mathbf{v}_Q$. The time taken by this approach is exponential in $|\mathbf{Q} \cup \mathbf{R}|$, the number of unobserved variables.

However, it is possible to rewrite the expression; some factors can be pulled out of the summations due to the *distributive laws*
$$\begin{array}{ll} \sum_x(\epsilon * \eta) = (\sum_x \epsilon) * \eta & \text{if } x \text{ does not occur free in } \eta \\ \sum_x(\epsilon * \eta) = \epsilon * \sum_x \eta & \text{if } x \text{ does not occur free in } \epsilon \end{array} \tag{2}$$

in which $(\epsilon * \eta)$ is a numeric expression containing free variable x. It is sometimes suggested that applying these laws (as rewrite rules, from left to right) makes the expression more efficient to evaluate, and therefore forms the basis of efficient inference algorithms. This statement alone is somewhat misleading. For example, if the product is $\mathsf{P}(a)\mathsf{P}(b|a)\mathsf{P}(c|b)$, we can rewrite:
$$\sum_{a,b,c} \mathsf{P}(a)\mathsf{P}(b|a)\mathsf{P}(c|b) = \sum_a \mathsf{P}(a) \sum_b \mathsf{P}(b|a) \sum_c \mathsf{P}(c|b)$$

If we expand each $\sum$-expression into a list of additions in the above equation, the left hand side contains $2 * |\mathrm{dom}(A)| * |\mathrm{dom}(B)| * |\mathrm{dom}(C)|$ multiplications

$$\mu_1 \leftarrow \{\, b \mapsto \textstyle\sum_c \mathsf{P}(c|b) \mid b \in \mathrm{dom}(B) \,\}$$
$$\mu_2 \leftarrow \{\, a \mapsto \textstyle\sum_b \mathsf{P}(b|a)\mu_1(b) \mid a \in \mathrm{dom}(A) \,\} \qquad \mu_3 \leftarrow \{\, b \mapsto \textstyle\sum_a \mathsf{P}(a)\mathsf{P}(b|a) \mid b \in \mathrm{dom}(B) \,\}$$
$$\text{return } \textstyle\sum_a \mathsf{P}(a)\mu_2(a) \qquad \text{return } \textstyle\sum_{b,c} \mu_3(b)\mathsf{P}(c|b)$$

Fig. 1: Two *programs* to efficiently calculate $\sum_{a,b,c} \mathsf{P}(a)\mathsf{P}(b|a)\mathsf{P}(c|b)$, using assignments to array variables μ_i. Such an array is represented as a set-theoretic function: a set containing key-value pairs $(k \mapsto v)$. Function application $\mu_i(k)$ corresponds to array lookup.

$$\overset{+}{\pi}_{-A} \left(p[A] \bowtie \overset{+}{\pi}_{-B} \left(p[B|A] \bowtie \overset{+}{\pi}_{-C} p[C|B] \right) \right) = \overset{+}{\pi}_{-B,C} \left(\left(\overset{+}{\pi}_{-A} \left(p[A] \bowtie p[B|A] \right) \right) \bowtie p[C|B] \right)$$

Fig. 2: Relational expressions corresponding to these programs, and their equality.

while the right hand side contains $|\mathrm{dom}(A)| * (1 + |\mathrm{dom}(B)|)$, so the latter can indeed be considered more efficient; however, it still contains the same number of additions: $|\mathrm{dom}(A)| * |\mathrm{dom}(B)| * |\mathrm{dom}(C)| - 1$. A lot of them are redundant; each summation that is expanded from $\sum_c \mathsf{P}(c|b)$ is copied $|\mathrm{dom}(A)|$ times, although it does not depend on a. To eliminate this redundancy, a notion of *sharing* or *storage* has to be introduced. Therefore, a conventional inference procedure calculates the expression using a *program*; see Fig. 1.

The program on the left has the same structure as the above expression and is efficient to evaluate. However, it is more cumbersome to read, and harder to reason about. For example, it is not easy to see that it is equivalent (equal in value, not in processing time) to program on the right; one has to transform them back into single expressions, and then compare these.

In this article, we present an alternative: a *relational representation*, in which the the basic building blocks of an expression are sets of values like μ_1, instead of single values like $\mu_1(b)$. Using this representation, we are able to express both the above programs, as well as their equivalence, by the equation in Fig. 2: yhe *relational expressions* on the two sides of the equals sign can be assigned an operational semantics similar to the two programs, while their denotational semantics are equal. This equivalence can be established by using rewrite rules (see Fig. 5) similar to those used in database theory; moreover, new equivalent expressions can be obtained using these rules.

3 Relational Expressions for Inference

3.1 Relational Algebra

In relational algebra, every expression represents a *relation*, a structured collection of data. In composite expressions like $\pi_{A,B} r$ and $r \bowtie s$, unary and binary

$$\pi_{\mathbf{A}} r \stackrel{\text{def}}{=} \{\, \mathbf{A} \lhd t \mid t \in r \,\} \qquad \mathbf{A} \lhd t \stackrel{\text{def}}{=} \{\, A \mapsto v \mid (A \mapsto v) \in t, A \in \mathbf{A} \,\}$$

$$\pi_{-\mathbf{A}} r \stackrel{\text{def}}{=} \{\, \mathbf{A} \not\lhd t \mid t \in r \,\} \qquad \mathbf{A} \not\lhd t \stackrel{\text{def}}{=} \{\, A \mapsto v \mid (A \mapsto v) \in t, A \notin \mathbf{A} \,\}$$

$$r \bowtie s \stackrel{\text{def}}{=} \{\, t_r \cup t_s \mid t_r \in r, t_s \in s, \mathbf{C} \lhd t_r = \mathbf{C} \lhd t_s \,\} \quad \text{where } \mathbf{C} = \mathit{schema}(r) \cap \mathit{schema}(s)$$

$$\rho_{A \mapsto B} r \stackrel{\text{def}}{=} \{\, (\{A\} \not\lhd t) \cup \{B \mapsto t(A)\} \mid t \in r \,\}$$

$$[\![A]\!] \stackrel{\text{def}}{=} \{\, \{A \mapsto a\} \mid a \in \mathrm{dom}(A) \,\} \qquad \sigma_\theta r \stackrel{\text{def}}{=} \{\, t \mid t \in r, \theta(t) \,\}$$

$$[\![A_1, \ldots, A_n]\!] \stackrel{\text{def}}{=} [\![A_1]\!] \bowtie \ldots \bowtie [\![A_n]\!] \qquad [\![\theta]\!] \stackrel{\text{def}}{=} \sigma_\theta\, [\![\mathit{schema}(\theta)]\!]$$

Fig. 3: Relational algebra, consisting of operators for projection π, natural join $\bowtie$, renaming ρ, embodiment $[\![\ldots]\!]$ and selection σ. The restriction operators $\lhd$, $\not\lhd$ on tuples (i.e. functions) are defined for auxiliary purposes.

operators transform the operand relations (r and s) into new relations. Relational algebra comes in different variants; some used in commercial database systems support relations with duplicates (multisets), null values and aggregation operators. For expositional reasons, we define a simple variant.

We presuppose a set of attributes $\mathcal{Attr}$ and a set of values $\mathcal{Val}$; each attribute $A \in \mathcal{Attr}$ has a domain $\mathrm{dom}(A) \subseteq \mathcal{Val}$. A relation r consists of

1. A schema, a set of attributes: $\mathit{schema}(r) \subseteq \mathcal{Attr}$.
2. A set of tuples, also simply denoted as r. Each tuple $t \in r$ contains a value for each attribute in the schema. Formally, t is a function of type $\mathit{schema}(r) \to \mathcal{Val}$, where $t(A) \in \mathrm{dom}(A)$ for each $A \in \mathit{schema}(r)$. The relation's *cardinality* $|r|$ is the number of tuples in the set.

The algebra's operators are defined in Fig. 3. The $\pi_{\mathbf{A}}$, $\bowtie$ and ρ operators can be found in any database textbook; however, note that we define an additional $\pi_{-\mathbf{A}}$ variant that mentions the discarded attributes instead of those remaining. The definition of the selection operator σ_θ, which only retains tuples that satisfy the predicate (boolean expression) θ, is not so conventional. Instead of only simple comparison predicates like $A_1 < A_2$, we support an arbitrary predicate where (some of) the relation's attributes take the place of values, e.g. $A_1 + A_2 = A_3$. Therefore, we model θ as a function of type $(\mathcal{Attr} \to \mathcal{Val}) \to \mathbb{B}$: given a certain *binding* of type $\mathcal{Attr} \to \mathcal{Val}$, it yields a boolean value.

We also define a less conventional operator $[\![A]\!]$, the *embodiment* of attribute A, producing a relation with schema $\{A\}$, of which the tuples are all the values $a \in \mathrm{dom}(A)$. Likewise, we define $[\![\mathbf{A}]\!]$ for a set of attributes $\mathbf{A} = \{A_1, \ldots, A_n\}$; its tuples consist of all the possible combinations of values for these attributes. Finally, we define $[\![\theta]\!]$, whose schema consists of all the attributes appearing in θ and whose contents are all the bindings that satisfy θ.

3.2 Role of Relational Algebra in Query Optimization

As a language between the query language and machine instructions, relational algebra plays an essential role for query optimization in database systems. Essentially, an expression in a query language like SQL is a logical predicate θ; the answer to such a query consists of $[\![\theta]\!]$, all tuples that satisfy the predicate. Compound predicates using $\wedge$ and $\exists$ can be translated into compound relational expressions, because these are *represented* by $\bowtie$ and π in the following way:

$$[\![\theta]\!] \bowtie [\![\kappa]\!] = [\![\theta \wedge \kappa]\!] \tag{3}$$

$$\pi_{-A}\,[\![\theta]\!] = [\![\exists a \in \mathrm{dom}(A).\ \theta[a/A]]\!] \tag{4}$$

Here, $\theta[a/A]$ means the substitution of value a for attribute A in predicate θ. After the query is translated to relational algebra in this way, the relational algebra expression can be *optimized*, i.e. rewritten into an equivalent expression with a minimal *cost*. Indeed, two equivalent expressions can have a different cost; the reason for this is that, next to their *denotational semantics* in terms of sets defined in Fig. 3, relational algebra expressions also have an *operational semantics*: a mapping to machine instructions.

The cost (e.g. processing time, memory, number of I/O operations) of performing these instructions is estimated by a cost function. In this article, we use a very simplistic cost function, namely the *summed cardinality of the intermediate relations*. For a given query expression, this number can be reduced by considering general equivalences in the denotational semantics, for example $(r \bowtie s) \bowtie t = r \bowtie (s \bowtie t)$. Although the result on both sides is the same relation (containing, say, 100 tuples), it is possible that $|r \bowtie s| = 5000$ while $|s \bowtie t| = 50$, so the total cost for $(r \bowtie s) \bowtie t$ equals 5100 and that for $r \bowtie (s \bowtie t)$ equals 150.

Thus, relational algebra plays a double role: its denotational semantics specifies *what* is to be calculated, but its expression structure also specifies *how* to calculate it, and how much that costs. For probabilistic inference queries, relational algebra can play this double role as well.

3.3 Relational Representation of Numeric Expressions

In analogy to a boolean expression θ containing $\wedge$ and $\exists$ operators, a numeric expression ϵ (over variables $\mathbf{V}$) containing multiplication ($*$) and summation ($\sum$) operators can be represented by a relational expression as well. We will write this expression as $[\![\epsilon]\!]_{\mathsf{val}}$. The schema of this relation is $\mathbf{V} \cup \{\mathsf{val}\}$; its tuples consist of each possible combination $\mathbf{v}$ of values for $\mathbf{V}$, combined with (under val) the value of the whole expression with these values filled in. For example, the relation $[\![(A-B)*(A-C)]\!]_{\mathsf{val}}$ contains a tuple t with $t(A) = 1$, $t(B) = 2$, $t(C) = 4$ and $t(\mathsf{val}) = 3$, because $(1-2)*(1-4) = 3$. The embodiment operator $[\![\ldots]\!]_{\mathsf{val}}$ is defined in Fig. 4, together with the operators $\overset{+}{\pi}$ and $\overset{*}{\bowtie}$ (the counterparts of π and $\bowtie$ for numeric expressions) that use the dedicated attribute val.

$$
\begin{aligned}
&[\![\epsilon]\!]_{\mathsf{val}} \stackrel{\text{def}}{=} [\![\epsilon = \mathsf{val}]\!] \\
&\stackrel{+}{\pi}_{\mathbf{A}} r \stackrel{\text{def}}{=} \left\{ t \cup \{\mathsf{val} \mapsto \textstyle\sum_{t' \in r, t = (\mathbf{A} \lhd t')} t'(\mathsf{val})\} \;\middle|\; t \in \pi_{\mathbf{A}} r \right\} \\
&\stackrel{+}{\pi}_{-\mathbf{A}} r \stackrel{\text{def}}{=} \stackrel{+}{\pi}_{(schema(r) \setminus \mathsf{val}) \setminus \mathbf{A}} r \\
&r \bowtie s \stackrel{\text{def}}{=} \left\{ (\{\mathsf{val}\} \not\lhd (t_r \cup t_s)) \cup \{\mathsf{val} \mapsto t_r(\mathsf{val}) * t_s(\mathsf{val})\} \;\middle|\; t_r \in r, t_s \in s, \mathbf{C} \lhd t_r = \mathbf{C} \lhd t_s \right\} \\
&\qquad \text{where } \mathbf{C} = (schema(r) \cap schema(s)) \setminus \{\mathsf{val}\}
\end{aligned}
$$

Fig. 4: Relational representation of numeric expressions.

As announced, $\bowtie$ and $\stackrel{+}{\pi}$ satisfy the equivalences

$$[\![\epsilon]\!]_{\mathsf{val}} \bowtie [\![\eta]\!]_{\mathsf{val}} = [\![\epsilon * \eta]\!]_{\mathsf{val}} \tag{5}$$

$$\stackrel{+}{\pi}_{-A} [\![\epsilon]\!]_{\mathsf{val}} = \left[\!\!\left[\textstyle\sum_{a \in \mathrm{dom}(A)} \epsilon[a/A] \right]\!\!\right]_{\mathsf{val}} \tag{6}$$

and can therefore be used to translate a compound numeric expression into a compound relational expression. An important effect of this is that a numeric statement $\epsilon = \eta$ that holds for all values of the variables also holds as a relational statement $[\![\epsilon]\!]_{\mathsf{val}} = [\![\eta]\!]_{\mathsf{val}}$. E.g., the commutativity of $*$ carries over to $\bowtie$:

$$
\begin{aligned}
& A * B = B * A \quad \text{for all bindings of } A \text{ and } B \\
\equiv \quad & \text{definition of } [\![\ldots]\!]_{\mathsf{val}} \text{ and } [\![\ldots]\!] \\
& [\![A * B]\!]_{\mathsf{val}} = [\![B * A]\!]_{\mathsf{val}} \\
\equiv \quad & \text{by (5)} \\
& [\![A]\!]_{\mathsf{val}} \bowtie [\![B]\!]_{\mathsf{val}} = [\![B]\!]_{\mathsf{val}} \bowtie [\![A]\!]_{\mathsf{val}}
\end{aligned}
$$

By the same reasoning, associativity of $*$ carries over, so we can unambiguously write $r_1 \bowtie r_2 \bowtie r_3$, or even $\bowtie^{*}_{1 \le i \le 3} r_i$. See Fig. 5 for more rewrite rules pertaining to $\bowtie$ and $\stackrel{+}{\pi}$. The next step is representing *probability expressions* as relations. Of course, these are just numeric expressions, but as they are central to this article, we use special shorthands:

$$p[A|B,C] \stackrel{\text{def}}{=} [\![\mathsf{P}(A|B,C)]\!]_{\mathsf{val}} \qquad\qquad cpd[V_i] \stackrel{\text{def}}{=} p[V_i|\mathbf{V}_{par(i)}]$$

Using this notation, we can relationally represent specific probabilistic statements. For example, $\mathsf{P}(A,B) = \mathsf{P}(A)\mathsf{P}(B)$ (the independence of A and B) can be expressed as $p[A,B] = p[A] \bowtie p[B]$. Next, we will apply this to the inference expression for Bayesian networks.

3.4 Relationally Rewriting the Inference Expression

Following the method above, the inference expression for a Bayesian network (1) is translated into a relational expression:

$$p[\mathbf{V}_Q, \mathbf{V}_E] = \stackrel{+}{\pi}_{-\mathbf{V}_R} \bowtie^{*}_{1 \le i \le n} cpd[V_i]$$

$$r \bowtie s = s \bowtie r \tag{7}$$

$$r \bowtie (s \bowtie t) = (r \bowtie s) \bowtie t \tag{8}$$

$$\overset{+}{\pi}_{-A} \overset{+}{\pi}_{-B} r = \overset{+}{\pi}_{-B} \overset{+}{\pi}_{-A} r \tag{9}$$

$$\overset{+}{\pi}_{-A} (r \bowtie s) = \begin{cases} \overset{+}{\pi}_{-A} r \bowtie s & \text{if } A \notin \mathit{schema}(s) \\ r \bowtie \overset{+}{\pi}_{-A} s & \text{if } A \notin \mathit{schema}(r) \end{cases} \tag{10}$$

$$\overset{+}{\pi}_{-E} \sigma_{E=e} (r \bowtie s) = \begin{cases} \overset{+}{\pi}_{-E} \sigma_{E=e} r \bowtie s & \text{if } E \in \mathit{schema}(r),\ E \notin \mathit{schema}(s) \\ r \bowtie \overset{+}{\pi}_{-E} \sigma_{E=e} s & \text{if } E \notin \mathit{schema}(r),\ E \in \mathit{schema}(s) \\ \overset{+}{\pi}_{-E} \sigma_{E=e} r \bowtie \overset{+}{\pi}_{-E} \sigma_{E=e} s & \text{if } E \in \mathit{schema}(r),\ E \in \mathit{schema}(s) \end{cases} \tag{11}$$

$$\overset{+}{\pi}_{-A} \sigma_{B=b} r = \sigma_{B=b} \overset{+}{\pi}_{-A} r \qquad \text{if } A \neq B \tag{12}$$

Fig. 5: Rewrite rules for $\bowtie$ and $\overset{+}{\pi}$. Eq. (10) represents the distributive law (2).

However, when performing an inference query, we are not interested in the answer for *all* bindings of $\mathbf{V}_E$; we are interested in one particular $\mathbf{v}_E$ value. In our original discussion of the inference query, this value was bound by the *context* in which we used the expression; in the relational representation, it has to be specified in the expression itself. We do this by adding a selection $\sigma_{\mathbf{V}_E = \mathbf{v}_E}$ on both sides of the above equation. After this selection, we might as well discard the $\mathbf{V}_E$ attributes from the tuples using a $\overset{+}{\pi}_{-\mathbf{V}_E}$ operator (which is in this case equivalent to a $\pi_{-\mathbf{V}_E}$ operator). This leaves us with:

$$\overset{+}{\pi}_{-\mathbf{V}_E} \sigma_{\mathbf{V}_E = \mathbf{v}_E} p[\mathbf{V}_Q, \mathbf{V}_E] = \overset{+}{\pi}_{-\mathbf{V}_E} \sigma_{\mathbf{V}_E = \mathbf{v}_E} \overset{+}{\pi}_{-\mathbf{V}_R} \overset{*}{\bowtie}_{1 \leq i \leq n} \mathit{cpd}[V_i] \tag{13}$$

Now, we can formulate the central thought of this article:

> *Efficient inference in a Bayesian network is performed by rewriting the right hand side of Eq.* (13) *into an equivalent expression with low cost.*

This will involve rewriting the multi-way $\overset{*}{\bowtie}$ into a parenthesized expression of $n-1$ binary $\bowtie$ operators (*join ordering*) and pushing the $\overset{+}{\pi}_{-\mathbf{V}_E}$, $\sigma_{\mathbf{V}_E = \mathbf{v}_E}$ and $\overset{+}{\pi}_{-\mathbf{V}_R}$ operators into the expression (observing the rules in Fig. 5).

Indeed, the conventional inference procedures can be translated into procedures that rewrite a relational expression in this way. We demonstrate this for *variable elimination* [4], also known as *bucket elimination* [5];[1] see Algorithm 1. Proof that it obeys the rewrite rules is omitted due to space constraints.

Algorithm 1: Variable elimination.

Input:
- unoptimized inference expression $\overset{+}{\pi}_{-\mathbf{V}_E} \sigma_{\mathbf{V}_E=\mathbf{v}_E} \overset{+}{\pi}_{-\mathbf{V}_R} \overset{*}{\bowtie}_{1\le i\le n} cpd[V_i]$
- variable elimination order α, ordering the m variables $\mathbf{V}_R$ as $V_{\alpha(1)}, \ldots, V_{\alpha(m)}$

Output: an expression e equivalent to the input expression

$\mathbf{s} \leftarrow \left\{ \overset{+}{\pi}_{-\mathbf{V}_L} \sigma_{\mathbf{V}_L=\mathbf{v}_L} cpd[V_i] \;\middle|\; 1 \le i \le n \right\}$
where $L \overset{\text{def}}{=} E \cap \{\, j \mid V_j \in schema(cpd[V_i]) \,\}$

for $i = 1..m$ **do**
$\mathbf{r} \leftarrow \{\, s \mid s \in \mathbf{s},\ V_{\alpha(i)} \in schema(s) \,\}$
$\mathbf{s} \leftarrow (\mathbf{s} \setminus \mathbf{r}) \cup \{ \overset{+}{\pi}_{-V_{\alpha(i)}} \overset{*}{\bowtie}_{r\in\mathbf{r}} r \}$
end
$e \leftarrow \overset{*}{\bowtie}_{s\in\mathbf{s}} s$

Note: where the algorithm specifies a multi-way join, any order can be taken.

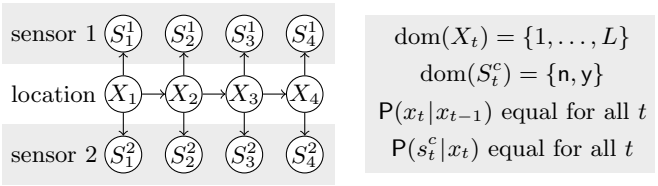


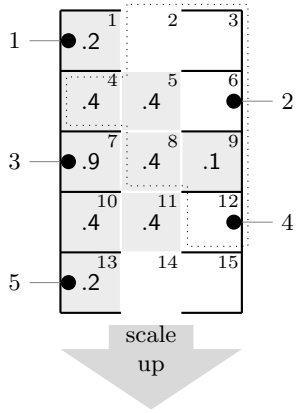
Fig. 6: MSHMM with two sensors ($K = 2$) and four timesteps ($T = 4$)

4 Sensor Data Inference

In this section, we put the above theory to use in a sensor data setup, in which a group of Bluetooth transceivers ('scanners') is used for localization. At K fixed locations in a building, a scanner is installed, performing a scan at discrete times $1 \le t \le T$ in order to track the position of a mobile device. The scanning range is such that the device can be seen by 2–3 different scanners at most places.

We model this using the Bayesian network in Fig. 6, which we call a multi-sensor Hidden Markov Model (MSHMM). The position of the mobile device at time t is modelled as a discrete variable X_t that can take the values 1–L; the different X_t variables form a Markov chain with transition model $\mathsf{P}(x_t|x_{t-1})$. The result of scanner c at time t is modelled by variable S_t^c; it can be n (device not detected) and y (device detected). An example floor plan and the resulting transition and sensor models are shown in Fig. 7.

[1] In principle, it is also possible for *junction tree propagation* [6, 7], but this is more complex as it performs multiple inference queries at once. In the relational representation, this means that some subexpressions are shared between multiple queries.



5 location number

wall

reach of sensor 2

c sensor c

.4 reach of sensor 3 and $\mathsf{P}(S_t^3 = \mathsf{y}|x_t)$

$\mathsf{P}(x_t\|x_{t-1})$		x_t 1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
x_{t-1}	7	0	0	0	0	0	0	.95	.05	0	0	0	0	0	0	0
	8	0	0	0	0	.2	0	.15	.3	.15	0	.2	0	0	0	0

$\mathsf{P}(s_t^3\|x_t)$		x_t 1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
s_t^3	n	.8	1	1	.6	.6	1	.1	.6	.9	.6	.6	1	.8	1	1
	y	.2	0	0	.4	.4	0	.9	.4	.1	.4	.4	0	.2	0	0

Fig. 7: Example (partial) floor plan for the localization model. The numbered squares are the $L = 15$ discrete values that location variable X_t can take. At $K = 5$ positions, a sensor is installed. In one time step, it is possible to move to an adjacent location, but not through a wall; this is encoded in the transition model $\mathsf{P}(x_t|x_{t-1})$ of which the table is partially shown. For sensor 3, the detection probabilities for the locations in its reach are also given; they determine the sensor model $\mathsf{P}(s_t^3|x_t)$ (also shown in a table). Simultaneously scaling up L and K can be imagined as extending this floorplan in the direction of the arrow. If the upper and lower edges of the floor plan are ignored, each sensor has a reach of 9 locations, as is shown for sensors 2 and 3.

The inference query is $\mathsf{P}(x_u|\mathbf{s})$, the probability distribution over the location at time u based on the received scan results $\mathbf{s} = \{\, s_t^c \mid 1 \le c \le K, 1 \le t \le T \,\}$.

4.1 Dynamic Bayesian Network Inference

The MSHMM model is an example of a *dynamic Bayesian network* [8, 9], which means that it has a special repetitive structure: it repeats for every t, and the parents of a variable at time t are either at time $t-1$ or at time t as well. We can rewrite the inference expression to reflect this structure:

$$\overset{+}{\pi}_{X_u} \sigma_{\mathbf{S}=\mathbf{s}} p[X_u, \mathbf{S}] = \overset{+}{\pi}_{X_u} \sigma_{\mathbf{S}=\mathbf{s}} \bigmes_{1 \le t \le T} \left(cpd[X_t] \mes \bigmes_{1 \le c \le K} cpd[S_t^c] \right) = f_u \mes b_{u+1}$$

where we define

$$f_t \overset{\text{def}}{=} \overset{+}{\pi}_{X_t} \sigma_{\mathbf{S}_t=\mathbf{s}_t} \left(f_{t-1} \mes cpd[X_t] \mes \bigmes_{1 \le c \le K} cpd[S_t^c] \right) \qquad f_0 \overset{\text{def}}{=} [\![1]\!]_{\mathsf{val}}$$

$$b_t \overset{\text{def}}{=} \overset{+}{\pi}_{X_{t-1}} \sigma_{\mathbf{S}_t=\mathbf{s}_t} \left(cpd[X_t] \mes \left(\bigmes_{1 \le c \le K} cpd[S_t^c] \right) \mes b_{t+1} \right) \qquad b_{T+1} \overset{\text{def}}{=} [\![1]\!]_{\mathsf{val}}$$

In the last rewrite step, we do two things at the same time. Firstly, we order the parentheses in the outer join: assuming r_t as a shorthand for the operands, we rewrite $\overset{*}{\bowtie}_{1\leq t\leq T} r_t$ into $((\llbracket 1 \rrbracket_{\mathsf{val}} \bowtie r_1) \ldots \bowtie r_u) \bowtie (r_{u+1} \bowtie \ldots \bowtie (r_T \bowtie \llbracket 1 \rrbracket_{\mathsf{val}}))$. Secondly, we push σ and $\overset{+}{\pi}$ operators down this expression.

The result consists of two expressions f_u and b_{u+1} with a repetitive structure, known in conventional inference procedures as a *forward* and *backward pass*. The repeating f_t and b_t parts can be seen as small inference expressions themselves; for example, in f_t, the query variable is X_t and the evidence variables are $\mathbf{S}_t$. The remaining variables consist of all the other attributes (except val) in the relations joined in f_t: this happens to be only one, namely X_{t-1}, which occurs in $cpd[X_t]$ and in f_{t-1} (as this relation starts with a $\overset{+}{\pi}_{X_{t-1}}$ operator, X_{t-1} is its only variable).

To efficiently rewrite f_t, we can apply an inference procedure of our choice. In this case, rewriting is almost trivial:

$$f_t = \overset{+}{\pi}_{-X_{t-1}} (f_{t-1} \bowtie cpd[X_t]) \bowtie \overset{*}{\bowtie}_{1\leq c\leq K} \overset{+}{\pi}_{-S_t^c} \sigma_{S_t^c = s_t^c} cpd[S_t^c] \tag{14}$$

in which the structure of parentheses in the $\overset{*}{\bowtie}$ factor is irrelevant. This is also the expression generated by Alg. 1 (where the $\mathbf{V}_R$ variables consist only of X_{t-1}).

As the f_t expressions are all similar—except f_1, because $cpd[X_1]$ is different—and all b_t expressions as well, we only need to apply the inference procedure over three expressions of $2 + K$ variables. This saves a lot of query optimization time compared to applying it over the whole model ($T(1 + K)$ variables). *This procedure can be applied to* any *dynamic Bayesian network.*

4.2 Exploiting Sparsity and Sharing

The cpds in the MSHMM model contain a lot of zeros. In the common usage of inference procedures, this is irrelevant: the cpds, as well as intermediate results, are represented by arrays in which zeros are treated the same as any other value. Up to a certain number of zeros, this representation is optimal, as it incurs low overhead. The size of these arrays, however, grows exponentially with the number of variables that are represented (the array's *dimensionality*): if each of the variables $V_1, \ldots, V_n$ has a domain of size d, the array contains d^n entries. This also holds for the relations that we have considered up to this point: due to the definition of $\llbracket \epsilon \rrbracket_{\mathsf{val}}$, an expression ϵ over these variables is represented by a relation with cardinality d^n. (In fact, such a relation can be directly represented by an array: the values $t(V_1), \ldots, t(V_n)$ of a tuple t together determine the index, at which the value $t(\mathsf{val})$ is stored.)

If we consider the simple cost function from Sect. 3.2, each intermediate f_t relation will contribute, apart from f_{t-1}, a term of the order $O(L^2 + KL)$ to the total cost. This will become a problem as the model is scaled up. By scaling up we mean that the detection area is expanded by installing more scanners; the granularity of the discrete location variable (i.e. the number of m^2 per x_t value) is kept fixed, as well as the density of the scanners (the number of scanners per

m^2). In other words, the K and L parameters of the model are jointly increased (see Fig. 7); this causes the inference cost to increase quadratically.

However, as we will show, the number of non-zero values in the intermediate relations only increases linearly; therefore, for a certain size of the model, it will become more efficient to represent only these non-zero values. We define this *sparse representation* by replacing the earlier $[\![\epsilon]\!]_{\mathsf{val}}$ representation with

$$[\![\epsilon]\!]_{\mathsf{val}} \stackrel{\text{def}}{=} [\![\epsilon = \mathsf{val} \wedge \mathsf{val} > 0]\!]$$

Crucially, equations (5) and (6) still hold when we use this sparse representation, so the relational inference equation (1) is still valid, as are all the rewrite rules; therefore, we can still use the rewritten expression (14) for f_t.

What are the effects on the inference cost? This depends on the size of the f_{t-1} relation. Some probabilistic reasoning will show that this relation is equal to $\stackrel{+}{\pi}_{X_{t-1}} \sigma_{\mathbf{S}_{1..t-1}=\mathbf{s}_{1..t-1}} p[X_{t-1}, \mathbf{S}_{1..t-1}]$; therefore, its size equals the number of X_{t-1} locations that have a nonzero joint probability with the sensor input up to $t-1$. If one scanner produced y at $t-1$, this number is 9: see the gray area in Fig. 7. When this f_t is joined with $cpd[X_t]$, the resulting relation will contain 13 tuples: all the locations reachable from this area in one step. Unfortunately, the other half of the f_t expression will still cost $O(KL)$: almost all scans S_t^c return a n, and $\stackrel{+}{\pi}_{-S_t^c} \sigma_{S_t^c=\mathsf{n}} cpd[S_t^c]$ contains L tuples.

However, we can rewrite f_t into:

$$f_t = \stackrel{+}{\pi}_{-X_{t-1}} (f_{t-1} \bowtie cpd[X_t]) \bowtie \stackrel{+}{\pi}_{-\mathbf{S}_t} \sigma_{\mathbf{S}_t=\mathbf{s}_t} \bigbowtie^{*}_{1 \le c \le K} cpd[S_t^c]$$

At first sight, this seems strange: each $cpd[S_t^c]$ contains $L+9$ tuples, so joining them will certainly cost $O(KL)$ or more. Also, it goes totally against the heuristics in conventional inference procedures, which try to minimize the largest dimensionality in the intermediate relations. The trick is that *evaluating this join can be done upfront*, as it does not depend on the evidence $\mathbf{s}_t$. Because the relations $cpd[S_t^c]$ are the same for each t, it does not even depend on t, and can thus be reused within an inference query as well as among different inference queries. The relation is equal to $p[\mathbf{S}_t | X_t]$, from which we can deduce its size: for each location x_t, 3 scanners may or may not produce y, giving $2^3 = 8$ possible $(x_t, \mathbf{s}_t)$ combinations with nonzero probability. Hence, the relation contains $8L$ tuples; so storage size scales linearly when we increase the detection area.

When the cost for this relation is not taken into account (which is reasonable if T gets large), the $\sigma_{\mathbf{S}_t=\mathbf{s}_t} \bigbowtie^{*}_{1 \le c \le K} cpd[S_t^c]$ part of f_t will only contribute a constant term (at most 9) to the cost if at least one scan is y. If all scans are n, it will contribute an $O(L)$ term; then, it is better to postpone the selection, and first join on X_t instead.

In conclusion, when the upfront calculation is not taken into account, the inference cost *remains constant* when scaling up the model using a sparse representation. Under a more realistic cost metric that also takes into account the *time* taken by selections and joins on base relations, it will scale logarithmically.

5 Related Work

The realization that probabilistic inference can be expressed as a relational query goes back to [2]. More recently, it has been shown[10] that variable elimination can be combined with a query optimization[11] that pushes down $\overset{+}{\pi}_{-A}$ operators. Although both acknowledge that an inference query can be processed and optimized by a relational database, neither shows the intimate connection between probability expressions and relational expressions such as (1) and (13). Also, neither mentions the advantages of a sparse representation.

Sparse/relational representations for probabilistic processing have been considered in the areas of constraint propagation [12] and information retrieval models [13], where good performance is reported. However, none of this work considers the area of sensor data management, whose scalability requirements make a sparse representation absolutely necessary.

6 Conclusions

We have shown how inference can be analysed and carried out using a relational representation. As its main advantage, we see the use of rewrite rules like in Fig. 5 for deriving or checking new inference optimizations. These rules can be applied by database researchers without any probabilistic knowledge, or indeed by automatic query optimizers. In this article, we have applied them manually, and shown that a (sparse) relational representation and a cost function depending on *cardinality* instead of *dimensionality* can be crucial for scalable sensor data processing. We hope this research clears a little bit of the path connecting database and AI research in inference query optimization.

References

1. Kanagal, B., Deshpande, A.: Online filtering, smoothing and probabilistic modeling of streaming data. In: Proceedings of the 24th International Conference on Data Engineering (ICDE2008). (April 2008) 1160–1169
2. Wong, S.K.M., Butz, C.J., Xiang, Y.: A method for implementing a probabilistic model as a relational database. In: Proc. 11th Conf. on Uncertainty in AI. (1995) 556–564
3. Pearl, J.: Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann Publishers Inc., San Francisco, USA (1988)
4. Zhang, N.L., Poole, D.: Exploiting causal independence in Bayesian network inference. J. Artif. Intell. Res. (JAIR) **5** (1996) 301–328
5. Dechter, R.: Bucket elimination: A unifying framework for reasoning. Artif. Intell. **113**(1-2) (1999) 41–85
6. Lauritzen, S.L., Spiegelhalter, D.J.: Local computations with probabilities on graphical structures and their application to expert systems. Journal of the Royal Statistical Society. Series B **50**(2) (1988) 157–224
7. Huang, C., Darwiche, A.: Inference in belief networks: A procedural guide. Int. J. Approx. Reasoning **15**(3) (1996) 225–263

8. Dean, T., Kanazawa, K.: A model for reasoning about persistence and causation. Computational Intelligence **5**(3) (1989) 142–150
9. Murphy, K.P.: Dynamic Bayesian Networks: Representation, Inference and Learning. PhD thesis, University of California, Berkeley (2002)
10. Corrada Bravo, H., Ramakrishnan, R.: Optimizing MPF queries: decision support and probabilistic inference. In: SIGMOD Conference. (2007) 701–712
11. Chaudhuri, S., Shim, K.: Including group-by in query optimization. In Bocca, J.B., Jarke, M., Zaniolo, C., eds.: VLDB, Morgan Kaufmann (1994) 354–366
12. Larkin, D., Dechter, R.: Bayesian inference in the presence of determinism. In: Proceedings of the Ninth International Workshop on Artificial Intelligence and Statistics. (January 2003)
13. Cornacchia, R., Héman, S., Zukowski, M., de Vries, A.P., Boncz, P.A.: Flexible and efficient IR using array databases. VLDB J. **17**(1) (2008) 151–168