Exploiting sparsity and sharing in probabilistic sensor data models

Sander Evers

Abstract

Probabilistic sensor models defined as dynamic Bayesian networks can possess an inherent sparsity that is not reflected in the structure of the network. Classical inference algorithms like variable elimination and junction tree propagation cannot exploit this sparsity. Also, they do not exploit the opportunities for sharing calculations among different time slices of the model. We show that, using a relational representation, inference expressions for these sensor models can be rewritten to make efficient use of sparsity and sharing.

1 Introduction

In sensor data, *uncertainty* arises due to many causes: measurement noise, missing data because of sensor or network failure, the inherent 'semantic gap' between the *data* that is measured and the *information* one is interested in, and the integration of data from different sensors. *Probabilistic models* deal with these uncertainties in the well-understood, comprehensive and modular framework of probability theory, and are therefore often used in processing sensor data.

There exist a lot of probabilistic sensor models which are specialized for a certain task and sensor setup. These specialized models are accompanied by specialized *inference algorithms* which derive the probability distribution over a target variable given the observed sensor data. In contrast, there also exist generic probabilistic models, for which the de facto standard is the *Bayesian network*, in which probabilistic variables and their relations can be defined in a modular and intuitive way. The standard inference algorithms for Bayesian networks (of which *variable elimination*[11] and *junction tree propagation*[9] are the most widely known) can, in this context, be seen as meta-procedures which derive an inference algorithm from the structure of the model. In a database analogy, these meta-procedures correspond to query optimizers, while the algorithms they derive correspond to query plans. However, the derived algorithms are suboptimal for sensor data for two reasons:

1. In sensor data processing, the same calculations are made over and over. It is better to structure the calculations so that a large part of intermediate results can be *shared*.
2. The conventional 'meta-procedures' optimize under the implicit assumption that probability distributions are dense, i.e. nonzero for a large part of their domain. In the sensor data models we use, this is not the case: they are *sparse*.

As our demonstration case, we consider the following setup, in which a group of Bluetooth transceivers ('scanners') is used for localization. At a number (K) of fixed locations in a building, a scanner is installed, and performs regular scans in order to track the position of a mobile device, which we define as a discrete variable X that can take the values $1..L$. The scanning range is such that the mobile device can be seen by 2 or 3 different scanners at most places. After a certain timespan $1..T$, we want to calculate $\mathsf{P}(X_t|s_{1..T}^{1..K})$: the probability distribution over the location at time $t \in 1..T$ based on the received scan results during the timespan. This *inference* computation forms the base for different online and offline processing tasks like forward filtering (using sensor data from the past to enhance the present probability distribution) and smoothing (using sensor data from before and after the target time).

We investigate how the inference task scales up when we enlarge the area covered by the scanners, while keeping the granularity of the discrete location variable fixed, as well as the density of the scanners. In other words, we jointly increase L and K.

Using conventional inference methods, the inference time will scale quadratically: for each time t, it will take all the probabilities $\mathsf{P}(s_t^c|x_t)$ into account, where c ranges over $1..K$ and x_t ranges over $1..L$. Most of these probabilities are irrelevant, because most locations x_t are out of the question anyway (due to estimates of the location at nearby times, combined with the knowledge that the mobile device can only move with a certain speed). The number of locations that *do* need to be considered does not depend on L, so if we restrict ourselves to these, the complexity becomes linear. However, there is still a lot of redundant work: the result s_t^c of each sensor is taken into account and contributes to the processing time, although it is known beforehand that only results of nearby scanners can be positive. This means that in the joint sensor model, the number of combinations $(s_t^1, s_t^2, \ldots, s_t^K, x_t)$ with a nonzero probability grows linearly when we jointly scale up L and K. This number is small enough to simply store all these probabilities and do a lookup in logarithmic time.

We show that both optimizations can be achieved in a straightforward way when probability distributions are represented as relations (in the relational algebra sense) between domain indices and probabilities. This has two advantages:

- The base probability distributions of the model, as well as intermediate results during inference, can be stored using a *sparse* representation by omitting all tuples with zero probability. Multiplication and addition of probabilities can be performed consistently with respect to this representation using conventional relational operators.
- Optimizations like the ones we propose can be discovered, formulated and validated using the rewrite rules of relational algebra; among other things, this makes it easy to spot opportunities for sharing subcalculations.

Moreover, the relational representation frames inference optimization as a form of query optimization, which can be performed without any regard to probabilistic semantics. This allows the database community to attack the problem without requiring any insight into the semantics of probabilistic models.

At the same time, we also show how these semantics can inform query optimization under sparse representations: in the above example, it has suggested a join order which the triangulation heuristics used in variable

elimination and junction tree propagation would never have selected.

In section 2, we describe how a (dynamic) Bayesian network in general, and our Bluetooth localization model in particular, is represented as a set of relations. We also introduce variants of relational algebra operators to perform probabilistic processing on these relations.

In section 3, we present probabilistic inference using the relational representation, apply this to our localization model. We show how to structure the inference calculation such that it makes use of sparsity and sharing. We show how knowledge about the probabilistic model informs this optimization.

2 Representation of probabilistic models

A probabilistic model defines a set of variables, each with a fixed domain (which we assume to be discrete), and the relation between them. This relation is probabilistic instead of deterministic: it does not answer the question *what are the possible values of C, given that A = a and B = b?* but rather *what is the probability distribution over C, given that A = a and B = b?* In sensor data processing, the observed values a and b correspond to sensor readings, and C is a property of the sensed phenomenon; in our case, the location at a certain time. A popular way of *defining* a probabilistic model is as a Bayesian network; we will review what this means in section 2.1, and define one for our demonstration case in section 2.2. In section 2.3, we show how a Bayesian network can be represented using the relational data model.

How to answer to the above question for a Bayesian network is discussed in section 3.

2.1 Defining a probabilistic model using a Bayesian network

A Bayesian network over a set $\bar{V}$ of probabilistic variables is defined by:

1. A directed acyclic graph with $\bar{V}$ as nodes. Its directed edges define a function that maps a variable to its parents: $V_i \in \mathit{Parents}(V_j)$ iff there is an edge from V_i to V_j.
2. For each variable V_i its conditional probability distribution (*cpd*) given $\mathit{Parents}(V_i)$.

Technically, this cpd is a function; for example, if $\mathit{Parents}(V_1) = \{V_2, V_3, V_4\}$, the required cpd for V_1 would be a function that produces the probability $\mathsf{P}(V_1{=}v_1 | V_2{=}v_2, V_3{=}v_3, V_4{=}v_4)$ given the arguments v_1, v_2, v_3 and v_4. However, we often simply talk about "the cpd $\mathsf{P}(v_1|v_2, v_3, v_4)$", in which we (a) conflate the function itself with its function value on a set of abstract argument values, and (b) use the syntactic shorthand a for $A{=}a$. In order to abstract away from which variables actually constitute V_1's parents, we use a further syntactic shorthand: $\mathsf{P}(v_1|\mathit{parents}(V_1))$. Notice the lowercase p in *parents* here, indicating that not the actual variables are meant, but rather a set of abstract values for these variables. We also use the lowercase shorthand for other sets of variables: if we have defined a set $\bar{X} = \{X_1, X_3\}$, then $\mathsf{P}(\bar{x})$ means $\mathsf{P}(X_1{=}x_1, X_3{=}x_3)$.

A principal property characterizing the probabilistic semantics of a Bayesian network is that, for a set of variables $\bar{X}$ closed under *Parents*,

its joint probability $\mathsf{P}(\bar{x})$ *factorizes* into the product of the cpds:

$$\mathsf{P}(\bar{x}) = \prod_{X_i \in \bar{X}} \mathsf{P}(x_i | parents(X_i)) \qquad \text{for } \bar{X} \text{ such that } X_i \in \bar{X} \Rightarrow Parents(X_i) \subseteq \bar{X}$$

(FACT-BN)

In particular, this holds for the set $\bar{V}$ of all the variables in the network. So, for an arbitrary assignment $\bar{v}$ of values to all variables, $\mathsf{P}(\bar{v})$ is obtained by multiplying all the conditional probabilities with those values as arguments.

This joint probability defines a complete and unambiguous probabilistic semantics for the model; all other probabilities over subsets of $\bar{V}$ can be derived from it (by *marginalization*, i.e. summation over the other variables).

Bayesian networks can model a variable X whose value (or probability distribution) *changes* over time by defining an instance X_t of this variable for each time t in a discrete time domain $0..T$. These kind of networks are referred to as *dynamic* Bayesian networks[4, 7, 10]. Usually, the term implies some further restrictions:

- for each point in time t, the relations between variables at t are the same
- relations between variables at different times are restricted to parent-child arrows pointing from variables at $t-1$ to variables at t (in other words, a Markov condition); these relations, too, are are same for each t

Hence, the model consists of identical 'slices'. An example of this can be seen in the next section, where we define the MSHMM model, an instance of a dynamic Bayesian network.

2.2 Bluetooth localization with the MSHMM model

We now proceed with defining a Bayesian network for the Bluetooth localization setup described above, taking into account the parameters L (the number of locations), K (the number of sensors) and T (the number of timesteps). We call this network the multi-sensor Hidden Markov Model (MSHMM). An instance with $K = 2$ and $T = 4$ is shown in Fig. 1.

There are two kinds of probabilistic variables in the model. The variable X_t for $t \in \{0, \ldots, T\}$ represents the location of the mobile device at time t and has the domain $\{1, \ldots, L\}$. The variable S_t^c for $c \in \{1, .., K\}, t \in \{1, \ldots, T\}$ represents the scan result of sensor c at time t. Its domain consists of the values 0 (device not detected) and 1 (device detected).

The cpd $\mathsf{P}(x_t | x_{t-1})$ is called the *transition model* and consists of the probabilities to go from one location to another in one time step. We assume it to be equal for each value of t. Although the Bayesian network framework does not forbid this model to contain L^2 nonzero probabilities, it is in fact always *sparse* in our localization setup, because it is only possible to move to a bounded number of locations. For example, assume a partial floor plan of the localization area looks like Fig. 2, where the numbered squares are 15 discrete values (locations) that the X variable can take. For simplicity, we assume that in one time step the mobile device can only move to an adjacent square, and only if there is no wall in between. It is also possible that it stays in the same square. Then, as is shown in Fig. 3, there are only two x_t values for which $\mathsf{P}(X_t = x_t | X_{t-1} = 7) > 0$, and only five x_t values for which $\mathsf{P}(X_t = x_t | X_{t-1} = 8) > 0$. On average, there are 3 locations x_t for which $\mathsf{P}(x_t | x_{t-1}) > 0$. We could represent this transition model using an

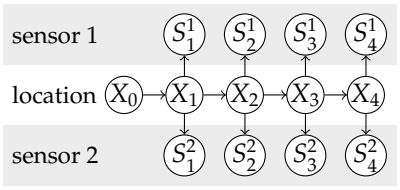

Figure 1: MSHMM with two sensors ($K = 2$) and four timesteps ($T = 4$)

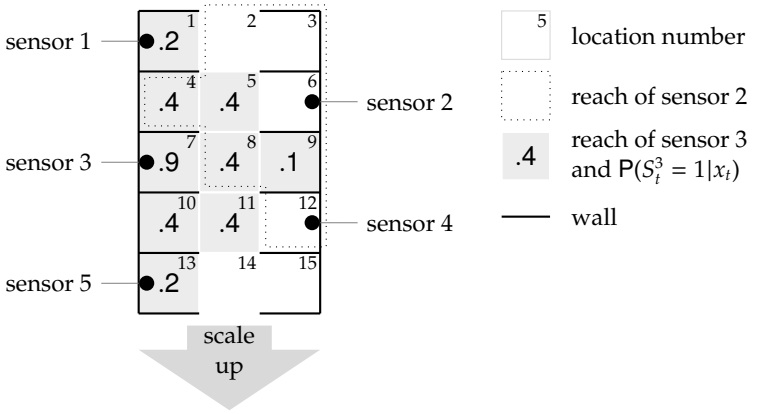

Figure 2: Example (partial) floor plan for the localization model. The numbered squares are the $L = 15$ discrete values that a location variable X_t can take. At $K = 5$ positions, a sensor is installed. In one time step, it is possible to move to an adjacent location, but not through a wall; this is encoded in the transition model $\mathsf{P}(x_t|x_{t-1})$. For sensor 3, the detection probabilities for the locations in its reach are also given; they determine the sensor model $\mathsf{P}(s_t^3|x_t)$. 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.

$\mathsf{P}(x_t\|x_{t-1})$		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

Figure 3: Partial transition model corresponding to the floor plan in Fig. 2. Rows sum to 1. This model encodes the fact that it is only possible to move to an adjacent room (and not through walls).

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

Figure 4: Sensor model for sensor 3 in Fig. 2. Columns sum to 1. This model encodes the limited reach of each sensor.

array with a *density* (fraction of nonzeros) of $3/L$ (Fig. 3), or as a relation with $3L$ tuples (which we will show in section 2.3).

The cpd $\mathsf{P}(s_t^c|x_t)$ is called the *sensor model* for sensor c and is also assumed to be equal for each t. It is different for each c, because each sensor is fixed at a different position and thus will get positive readings for different locations of the mobile device. The sensor has a bounded reach: in Fig. 2, the reach of sensor 3 is shaded in gray. Hence, there is a bounded number of x_t values for which $\mathsf{P}(S_t^c = 1|x_t) > 0$; for sensor 3, these 9 probabilities are shown in the gray squares. However, $\mathsf{P}(S_t^c = 0|x_t)$ is positive for each x_t. Therefore, the array representing $\mathsf{P}(s_t^c|x_t)$ would have a density of $(9+L)/2L$ (see Fig. 4).

We could also look at the sparsity of the sensor models in another way: given a fixed *location* x_t, there is a bounded *number of sensors* that can detect the device, i.e. a bounded number of c values with $\mathsf{P}(S_t^c = 1|x_t) > 0$. This bound depends on the sensor density and detection reach, and is 3 in our example.

2.3 Relational representation of a Bayesian network

In this section, we show how to represent the cpds of a Bayesian network, as well as the intermediate results that occur during inference, as relations. In the first place, a relation is a mathematical data model that is convenient for expressing bulk multiplication and summation operations; we use this to rewrite inference expressions into a more efficient form. In the second place, it closely maps to an *implementation* for storing a sparse multidimensional array; instead of storing all the values (where a value's position in memory is determined by its array indices), only the nonzero values are stored, in combination with their indices.

We use the following set-theoretic definition of a relation:

- A relation's schema $\mathit{schema}(r)$ consists of a set of attributes.
- Each attribute A has a domain $\mathrm{dom}(A)$.
- A relation r consists of a set of tuples.
- Each tuple $t \in r$ is a function with domain $\mathit{schema}(r)$, where $t(A) \in \mathrm{dom}(A)$ for each $A \in \mathit{schema}(r)$.

For each variable V in the Bayesian network, we define a relation $\mathit{cpd}[V]$ that contains the nonzero probabilities $\mathsf{P}(v|\mathit{parents}(V))$. The contents of $\mathit{cpd}[V]$ are as follows. Say that $\mathit{Parents}(V) = \{V_1, \ldots, V_i\}$, then

$$\{V \mapsto v, V_1 \mapsto v_1, \ldots, V_i \mapsto v_i, \mathsf{val} \mapsto \mathsf{P}(v|v_1, \ldots, v_i)\} \in \mathit{cpd}[V] \quad \text{iff} \quad \mathsf{P}(v|v_1, \ldots, v_i) > 0$$

Thus, the schema of $\mathit{cpd}[V]$ consists of the attributes $\{V\} \cup \mathit{Parents}(V) \cup \{\mathsf{val}\}$, where val is the attribute containing the probability. Every relation that we use in this article will contain such a val attribute; we will refer to the other attributes of a relation R as its *regular* attributes, and write the set as $\mathit{regattr}(R)$.

For the manipulation of probabilities necessary during inference, we define two variants of relational operators in which val plays a distinct role. We define them in terms of their conventional counterparts from an extended relational algebra, which we assume to include a generalized projection operator π[6] providing functionality similar to grouping, aggregation and renaming constructs in SQL.

An often occurring task is multiplying two probability distributions, e.g. (the resulting value $f(a,b,c,d)$ might or might not be interpretable as a probability again):

$$f(a,b,c,d) = \mathsf{P}(a|b,c)\mathsf{P}(c|d)$$

We define the operator $\bowtie$ to perform this multiplication in a 'bulk' fashion. With $cpd[A]$ representing $\mathsf{P}(a|b,c)$ and $cpd[C]$ representing $\mathsf{P}(c|d)$, we want $cpd[A] \bowtie cpd[C]$ to represent f, i.e. contain $f(a,b,c,d)$ for all values a,b,c,d. Therefore, we define $r \bowtie s$ as a 'natural' join on all *regular* attributes that r and s have in common, combined with a generalized projection that performs a multiplication of their respective val attributes and renames the result to val as well:

$$r \bowtie^{*} s \stackrel{\text{def}}{=} \pi_{regattr(r) \cup regattr(s),\ r.\mathsf{val} * s.\mathsf{val} \to \mathsf{val}} \left(r \bowtie_{regattr(r) \cap regattr(s)} s \right)$$

Or, in SQL:

$$r \bowtie^{*} s \stackrel{\text{def}}{=} \textbf{select } regattr(r) \cup regattr(s),\ r.\mathsf{val} * s.\mathsf{val} \textbf{ as } \mathsf{val}$$
$$\textbf{from } r \textbf{ join } s \textbf{ using}(regattr(r) \cap regattr(s))$$

The second operator we define sums up all the probabilities in relation r for tuples that have values in common for a subset of regular attributes $\bar{V} \subseteq regattr(r)$:

$$\stackrel{+}{\pi}_{\bar{V}} r \stackrel{\text{def}}{=} \pi_{\bar{V},\ \mathsf{SUM(val)} \to \mathsf{val}} r$$

Or, in SQL:

$$\stackrel{+}{\pi}_{\bar{V}} r \stackrel{\text{def}}{=} \textbf{select } \bar{V}, \mathsf{SUM(val)} \textbf{ as } \mathsf{val} \textbf{ from } r \textbf{ group by } \bar{V}$$

The $\stackrel{+}{\pi}$ operator can be used to 'marginalize' probabilities: if we want to calculate $\mathsf{P}(a,b)$ from a relation r that contains probability $\mathsf{P}(a,b,c)$ we can use $\stackrel{+}{\pi}_{\{A,B\}} r$. This usage appeals to the notion of projecting a three-dimensional probability distribution onto two of its dimensions.

However, when we want to emphasize the correspondence to the algebraic expression $\sum_c \mathsf{P}(a,b,c)$ it is useful not to name the variables (dimensions) that remain, but those that are projected away. For this case we define the notation $\stackrel{+}{\pi}_{-C} r$:

$$\stackrel{+}{\pi}_{-\bar{V}} r \stackrel{\text{def}}{=} \pi_{regattr(r) \setminus \bar{V},\ \mathsf{SUM(val)} \to \mathsf{val}} r$$

For $\bowtie$ and $\stackrel{+}{\pi}$, similar rewrite rules as for $*$ and $\sum$ apply. The operator $\bowtie^{*}$ is associative and commutative, which means that we can unambiguously write $r \bowtie^{*} s \bowtie^{*} r$ and even $\bigbowtie^{*}_{x \in \{r,s,t\}} x$. In a multi-dimensional sum $\sum_a \sum_b f(a,b)$ order is of no importance (we might also write it $\sum_b \sum_a f(a,b,c)$ or $\sum_{a,b} f(a,b,c)$), and this also holds for $\stackrel{+}{\pi}$ if we use the negative notation for variables: $\stackrel{+}{\pi}_{-A} \stackrel{+}{\pi}_{-B} r = \stackrel{+}{\pi}_{-B} \stackrel{+}{\pi}_{-A} r = \stackrel{+}{\pi}_{-\{A,B\}} r$.

Somewhat less trivially, the distributivity property $\sum_a (\phi * \psi) = \phi * \sum_a \psi$ (if ϕ does not contain free variable a) also transfers to the relational operators:

$$\stackrel{+}{\pi}_{-A} (r \bowtie^{*} s) = r \bowtie^{*} \stackrel{+}{\pi}_{-A} s \qquad \text{if } A \notin regattr(r)$$
$$\stackrel{+}{\pi}_{-A} (r \bowtie^{*} s) = \stackrel{+}{\pi}_{-A} r \bowtie^{*} s \qquad \text{if } A \notin regattr(s)$$

We need one additional operator that does not correspond to an operator in the $\sum$,$*$-expressions. For an *observed* value (say, 7) of a probabilistic variable V, one simply substitutes the value 7 at the place of variable V in a cpd (turning it into a function with one argument less). In the relational representation, this is done by selecting the tuples that have $V = 7$.

$$\sigma_v r \stackrel{\text{def}}{=} \sigma_{V=v} r$$

Rather than a definition of a new operator σ, this is actually a syntactic shorthand again, which allows us to omit writing variable name V. We also use this with sets of values: if $\bar{E}$ is defined as $\{V_1, V_3\}$, then $\sigma_{\bar{e}} r$ expands to $\sigma_{V_1=v_1 \wedge V_3=v_3} r$.

The non-representation of zero-probability tuples plays well with the multiplication and addition semantics:

- A product $a * b$ is nonzero iff both operands are nonzero; likewise, a tuple in $r \bowtie s$ exists iff there exists a tuple in r with corresponding indices $regattr(r)$ as well as a tuple in s with corresponding indices $regattr(s)$.
- Since entries for which $\mathsf{val} = 0$ would not contribute anything to a sum, it does not matter that they are not present in the operand table. Conversely, because we are working with nonnegative numbers, a result tuple for which the sum is 0 can never come into existence.

3 Inference as a relational query

Given a probabilistic model over a set of variables $\bar{V}$, *inference* is the task of deriving the probability distribution over a subset of *query variables* $\bar{Q} \in \bar{V}$, given the observed values $\bar{e}$ of another subset $\bar{E} \in \bar{V}$ called the *evidence variables* (in sensor data processing, these correspond to sensor readings). Thus, the goal of inference is to calculate $\mathsf{P}(\bar{q}|\bar{e})$ for all values $\bar{q}$, given fixed values $\bar{e}$.

The sets $\bar{Q}$ and $\bar{E}$ do not overlap, and we define $\bar{R} = \bar{V} - (\bar{Q} \cup \bar{E})$ as the remaining variables; so, $\bar{V}$ is partitioned into three sets, and we can write the model's joint probability as $\mathsf{P}(\bar{v}) = \mathsf{P}(\bar{q}, \bar{e}, \bar{r})$. Using the axioms of probability theory, the goal probability $\mathsf{P}(\bar{q}|\bar{e})$ can be written in terms of this joint probability:

$$\mathsf{P}(\bar{q}|\bar{e}) = \frac{\mathsf{P}(\bar{q}, \bar{e})}{\mathsf{P}(\bar{e})} = \frac{\sum_{\bar{r}} \mathsf{P}(\bar{q}, \bar{e}, \bar{r})}{\sum_{\bar{q}} \sum_{\bar{r}} \mathsf{P}(\bar{q}, \bar{e}, \bar{r})}$$

It is only necessary to calculate the outcome of the numerator $\mathsf{P}(\bar{q}, \bar{e}) = \sum_{\bar{r}} \mathsf{P}(\bar{q}, \bar{e}, \bar{r})$ for all $\bar{q}$ values; the denominator can be obtained by adding all these outcomes. Therefore, to simplify the expositions, we will hereafter equate inference with the calculation of $\mathsf{P}(\bar{q}, \bar{e})$ for all $\bar{q}$.

3.1 Inference in a Bayesian network

For a Bayesian network on variables $\bar{V}$, the joint probability in $\sum_{\bar{r}} \mathsf{P}(\bar{q}, \bar{e}, \bar{r})$ can be substituted by its factorization (Fact-BN):

$$\mathsf{P}(\bar{q}, \bar{e}) = \sum_{\bar{r}} \prod_{V_i \in \bar{V}} \mathsf{P}(v_i | parents(V_i))$$

Written using the relational representation, this expression reads

$$\dot{\pi}_{-\bar{R}} \sigma_{\bar{e}} (cpd[V_1] \bowtie cpd[V_2] \bowtie \ldots \bowtie cpd[V_n])$$

In this relational representation, it is not necessary to say 'for all $\bar{q}$': this is included in the expression by default, because it contains no selection on a single value $\bar{q}$. On the other hand, we do have perform a selection on $\bar{e}$. Note that the $\bar{E}$ attributes are preserved the above expression although this is not really necessary; all tuples will contain the same values $\bar{e}$ for these

attributes. We might as well project them out. That way, the only attributes that remain in the result expression are the query variables $\bar{Q}$. Therefore, in the rest of the article, we will use

$$\overset{+}{\pi}_{\bar{Q}}\sigma_{\bar{e}}(cpd[V_1] \bowtie cpd[V_2] \bowtie \ldots \bowtie cpd[V_n])$$

as the inference expression for a Bayesian network.

To calculate the value of this expression efficiently, the multi-way $\bowtie$-join can be written as a tree of binary $\bowtie$-joins, after which $\overset{+}{\pi}$ and σ operators can be added in this tree. The σ operators can be distributed over all joins; it seems most efficient to add them directly above the cpds, and indeed, this is common practice in the exisiting inference algorithms—we show in section 3.2 that it is not the most efficient in our MSHMM model. Next, assuming we want to decrease the number of attributes as early in the tree as possible, there are two equivalent methods to place these $\overset{+}{\pi}$ operators, depending on whether the $\overset{+}{\pi}_V$ or the $\overset{+}{\pi}_{-V}$ view is used:

- Given a $\bowtie$-tree, insert a $\overset{+}{\pi}_{\bar{V}}$ node above each join, working from the leaves to the root, where $\bar{V}$ contains those variables from the join result for which holds that:
  - it is contained in $\bar{Q}$, or
  - it is contained in $\bar{R}$ and also occurs in a base relation in another part of the tree.

  If the set $\bar{V}$ turns out to contain all the variables from the join result, the $\overset{+}{\pi}_{\bar{V}}$ node can of course be omitted.
- Given a $\bowtie$-tree, start with a set of $\overset{+}{\pi}_{-R}$ operators, one for each $R \in \bar{R}$, at the top of the tree. Then, for each $\overset{+}{\pi}_{-R}$, repeatedly move it down the single branch of the tree of which the join result contains R in its regular attributes, until there are two or more of such branches. Next, add $\overset{+}{\pi}_{-E}$ operators, for each $E \in \bar{E}$, right above the selection operators σ_e.

The challenging part is to find the tree in which the variables can be projected out as early as possible.

In the AI community, the problem of efficiently (and exactly) calculating $\mathsf{P}(\bar{q}, \bar{e})$ has been the subject of extensive research, under the name of *exact inference in a discrete Bayesian network*. Two well-known algorithms that have resulted from this are *variable elimination*[11] and *junction tree propagation*[9]. In the light of the relational representation, both algorithms can be regarded as procedures to produce a $\bowtie$-tree, and both try to minimize the largest dimensionality of an intermediate relation (after the above addition of $\overset{+}{\pi}$ operators is taken into account). As this minimization problem is known to be NP-hard[1], both algorithms use heuristics.

In a setting where the intermediate relations are represented as dense arrays, it makes a lot of sense to minimize their largest dimensionality, because the size of an array (and hence, the time needed to calculate it) relates exponentially to its dimensionality. However, when using a sparse representation, it might be the case that relations with a large dimensionality actually have a small number of tuples. We will show this for our MSHMM model.

3.2 Inference in the MSHMM model

When constructing a join tree for an inference query on a dynamic Bayesian network, it is possible to take advantage of its repetitive structure: build the same join tree for each timeslice and connect these to each other.

We do this for the MSHMM model, where the inference query variable is X_u (for some u with $1 \leq u \leq T$) and the evidence consists of all the sensor readings of the form $S_t^c{=}s_t^c$. We write the collection of all these readings $s_{1..T}^{1..K}$, and use $s_t^{1..K}$ for the collection of all the readings at a certain time t.

The inference query $\mathsf{P}(x_u, s_{1..T}^{1..K})$ is written as follows:

$$\overset{+}{\pi}_{X_u} \sigma_{s_{1..T}^{1..K}} \left(cpd[X_0] \bowtie \overset{*}{\bowtie}_{t=1..T} cpd[X_t] \bowtie cpd[S_t^1] \bowtie \ldots \bowtie cpd[S_t^K] \right)$$

Following common practice, this is split into $F \bowtie B$, consisting of a 'forward' factor F from 0 to u and a 'backward' factor B from T to $u+1$. The evidence is split over the factors:

$$F = \overset{+}{\pi}_{X_u} \sigma_{s_{1..u}^{1..K}} \left(cpd[X_0] \bowtie \overset{*}{\bowtie}_{t=1..u} cpd[X_t] \bowtie cpd[S_t^1] \bowtie \ldots \bowtie cpd[S_t^K] \right)$$

$$B = \overset{+}{\pi}_{X_u} \sigma_{s_{u+1..T}^{1..K}} \left(\overset{*}{\bowtie}_{t=u+1..T} cpd[X_t] \bowtie cpd[S_t^1] \bowtie \ldots \bowtie cpd[S_t^K] \right)$$

We will discuss only factor F from now; the reasoning for B is very similar. Apart from $cpd[X_0]$, the expression F consists of similar subexpressions for each time t. We can write F as a chain $f_u(f_{u-1}(\ldots f_1(cpd[X_0])\ldots))$, where we have defined:

$$f_t(r) = \overset{+}{\pi}_{X_t} \sigma_{s_t^{1..K}} \left(cpd[X_t] \bowtie cpd[S_t^1] \bowtie \ldots \bowtie cpd[S_t^K] \bowtie r \right)$$

This partially determines the join tree for the F expression: the different f_t parts are connected to each other as a right-deep tree, but we have not yet specified how the f_t parts are structured inside. However, what we do already know is that we can push down the projections removing the S_t^c variables and the X_{t-1} variable into the f_t part, as these variables do not occur higher up in the tree. Therefore, the only variable (regular attribute) that remains is X_t. Likewise, we push down every selection s_t^c regarding the evidence for a sensor c at time t into the f_t part.

The reason that we can not project out X_t is that it occurs in a cpd relation higher in the tree, in f_{t+1}: $regattr(cpd[X_{t+1}]) = \{X_t, X_{t+1}\}$. We say that $\{X_t\}$ forms the interface between f_t and f_{t+1}. In general dynamic Bayesian networks, the *interface* between slices t and $t+1$ consists of those variables in slice t that have a child in slice $t+1$. (In this definition, we follow Murphy[10], except that he calls this the *forward interface* of slice t.)

Each f_t consists of the same kind of relations; although their contents differ, their schema is the same (modulo the variable subscript t). This is also true for r; it always contains the interface variables between $t-1$ and t. Therefore, one can construct a similar join tree for each f_t, and then connect them to each other as the chain we defined above. In construction of this join tree, r plays the same role as the cpd relations, the evidence consists of $s_t^{1..K}$, and the query variables are those that form the interface to f_{t+1}.

In comparison to constructing a global join tree, this approach of chaining together local join trees can save a lot on optimization time. Moreover, it can be done in a streaming way; the join tree can be grown every time a batch of sensor readings arrives.

For the MSHMM, two possible join trees that the conventional algorithms could construct for $f_t(r)$ are the following:

$$f_t(r) = \overset{+}{\pi}_{-S_t^1} \sigma_{s_t^1} cpd[S_t^1] \bowtie (\ldots \bowtie (\overset{+}{\pi}_{-S_t^K} \sigma_{s_t^K} cpd[S_t^K] \bowtie \overset{+}{\pi}_{-X_{t-1}} (cpd[X_t] \bowtie r))\ldots)$$

$$f_t(r) = (\overset{+}{\pi}_{-S_t^1} \sigma_{s_t^1} cpd[S_t^1] \bowtie (\ldots \bowtie \overset{+}{\pi}_{-S_t^K} \sigma_{s_t^K} cpd[S_t^K]\ldots)) \bowtie \overset{+}{\pi}_{-X_{t-1}} (cpd[X_t] \bowtie r)$$

Judging by the number of attributes of the intermediate relations, these trees are optimal: all attributes are projected out immediately (except for X_t, which can not be projected out anyway, because it is the query variable). If we use a dense (array) representation of the relations, the processing times will not be very different. When we jointly scale up L and K, these times will scale quadratically for two reasons:

1. The relation r, which contains L tuples, is joined with $cpd[X_t]$, which contains L^2 tuples.
2. There are K joins of $\sigma_{s^c} cpd[S^c]$ relations, which all contain L tuples.

If we use a sparse representation, things look quite different. In the first query tree:

The relation r contains the locations for which $\mathsf{P}(x_{t-1}, \bar{s}_{0..t-1}) > 0$ (see section 3.3 for the probabilistic semantics of intermediate relations); let us designate the number of these locations by m. If there has just been a sensor reading $S^c_{t-1} = 1$, m is around 3; if the last positive sensor reading has occurred a short time ago, it is the number of locations to which transitions could have happened since then—which is bounded by a constant. Joining r with $cpd[X_t]$ will produce around $3m$ tuples; we assume this takes $O(m \log L)$ time. Joining with $\sigma_{s^c_t} cpd[S^c_t]$ will maintain the number of tuples for $s^c_t = 0$, and reduce it to around 3 for $s^c_t = 1$. All the sensor models together take $O(Km \log L)$ time. Thus, jointly scaling up L and K increases the costs with $O(n \log n)$.

In the second query tree, the $\sigma_{s^c_t} cpd[S^c_t]$ are joined independent of the contents of r. This can have a positive or negative effect, depending on the values s^c_t. If they are all zero, these relations all contain L tuples just like in the dense case; joining K tables would have a quadratic cost again. On the other hand, if a reading for sensor c deep down in the tree (i.e. for a c close to K) is positive, $\sigma_{s^c_t} cpd[S^c_t]$ will contain 9 tuples, and all the intermediate relations higher in the tree will have an equal or smaller size. This would render the cost linear in K, and independent of m, which is an advantage if m is large due to a lack of positive readings in the past. In an average case analysis, however, a positive sensor reading will occur somewhere half way between 1 and K, which results again in a quadratic cost.

A possible remedy would be to *dynamically* choose a join tree for $f_t(r)$ depending on the values of $s^{1..K}_t$: reorder the $\sigma_{s^c_t} cpd[S^c_t]$ relations such that one with $s^c_t = 1$ is at the bottom, and if there is no such reading put r at the bottom. However, we will present a better alternative with a static join tree for $f_t(r)$:

$$f_t(r) = \dot{\pi}_{-X_{t-1}} \left(\dot{\pi}_{-S^{1..K}_t} \sigma_{s^{1..K}_t} (cpd[X_t] \bowtie (cpd[S^1_t] \bowtie (\ldots \bowtie cpd[S^K_t] \ldots))) \bowtie r \right)$$

At first sight, this seems like a very bad idea: the $\sigma_{s^c_t}$ and $\dot{\pi}_{-S^c_t}$ operators have been pulled out of the join of the $cpd[S^c_t]$ relations. Using the dense representation, the result of this join would contain $2^K L$ tuples, which is unmanageable when scaling up L and K. Again, using a sparse representation leads to a totally different picture. Because a fixed location x_t can only trigger 3 sensors, there are only $2^3 = 8$ possible $s^{1..K}_t$ combinations for this x_t; the 3 sensors in question can produce either 0 or 1, and all other sensors readings are 0. So, the join of all $cpd[S^c_t]$ relations contains about $8L$ tuples. This *inherent linearity* in the MSHMM model can not be exploited by the conventional inference algorithms, because it is not apparent in the structure of the Bayesian network.

Still, why pull the selection operators out of the join? The answer is that it gives an opportunity for *sharing* among the different f_t subtrees. The

results of the join are now independent of the sensor readings at time t, and because the sensor models $cpd[S_t^c]$ are equal for each t, we can reuse these results in every f_t tree. The only operations specific to each time step are the selection $\sigma_{s_t^{1..K}}$ and joining its result with r. With a good index, this can be done in $O(\log K)$ time. So, we have traded a lot of repeated small calculations for a one-time big calculation and repeated lookup.

3.3 Inference in a generic dynamic Bayesian network

In this section, we generalize the sharing optimization from the previous section to a generic Dynamic Bayesian Network. We refer to the variables in slice t as $V_t^1, \ldots, V_t^n$; for variables $V_t^1, \ldots, V_t^m$ we observe the values $v_t^{1..m}$. The query variables $\bar{Q}_u$ are all in slice u. The forward factor is then:

$$F = \dot{\pi}_{\bar{Q}_u} \sigma_{v_{1..u}^{1..m}} \left(prior \bowtie \overset{*}{\bowtie}_{t=1..u} cpd[V_t^1] \bowtie \ldots \bowtie cpd[V_t^n] \right)$$

We define the set $\bar{I}_t$ as the interface between t and $t+1$: the variables V_t^i that occur in some $cpd[V_{t+1}^j]$. The relation *prior* contains the so-called prior probability distribution $\mathsf{P}(\bar{\imath}_0)$ derived from the model (compare to $\mathsf{P}(x_0)$ in the MSHMM model). For the above relation F, a chain structure is constructed as follows:

$$\begin{aligned} F &= \dot{\pi}_{\bar{Q}_u} f_u(f_{u-1}(\ldots f_1(prior)\ldots)) \\ f_t(r) &= \dot{\pi}_{\bar{I}_t} \left(\dot{\pi}_{\bar{I}_t \cup \bar{I}_{t-1}} \sigma_{v_t^{1..m}} f_t' \bowtie r \right) \\ f_t' &= \dot{\pi}_{\bar{I}_t \cup \bar{I}_{t-1} \cup V_t^{1..m}} \left(cpd[V_t^1] \bowtie \ldots \bowtie cpd[V_t^n] \right) \end{aligned}$$

Reading from bottom to top:

- The expression f_t' represents an unoptimized join tree that joins all relations in slice t to each other and projects onto the union of its two interfaces and its observed variables. It is supposed to be replaced by an optimized tree. This can be done using conventional inference techniques, where $\bar{I}_t \cup \bar{I}_{t-1} \cup V_t^{1..m}$ are taken as query variables. The result of this expression has to be calculated only once (f_t' is the same for each t except for the variable names).
- The result f_t' is used at every step of the chain; its values for the current sensor readings $v_t^{1..m}$ are looked up, joined with the results r from the previous step into $f_t(r)$, and propagated to the next step.
- From these steps, a chain is built, starting with *prior*. In the above formulas, it is assumed that $\bar{Q}_u \subseteq \bar{I}_u$; if this is not the case, a different join tree should be built for slice u, taking into regard the extra query variables.

In order to gain more understanding about the size of the relation f_t', it is helpful to examine its probabilistic semantics. If the set $\{V_t^1, \ldots, V_t^n\}$ were closed under the *Parents* function, the semantics of the cpd product $\prod_{j=1..n} \mathsf{P}(v_t^j | parents(v_t^j))$ (which corresponds to $\overset{*}{\bowtie}_{j=1..n} cpd[V_t^j]$) would have been directly given by (Fact-BN). However, the set has parents in slice $t-1$: the interface variables $\bar{I}_{t-1}$. Therefore, we rewrite the cpd product as the fraction of two cpd products for sets that do meet the closure requirement: $\bigcup_{\substack{j=1..n \\ k=0..t}} V_k^j$ (all the variables up to t) and $\bigcup_{\substack{j=1..n \\ k=0..t-1}} V_k^j$ (the variables up to $t-1$).

$$\prod_{j=1..n} \mathsf{P}(v_t^j | parents(v_t^j)) = \frac{\prod_{j=1..n} \mathsf{P}(v_k^j | parents(v_k^j))}{\prod_{\substack{j=1..n \\ k=0..t-1}} \mathsf{P}(v_k^j | parents(v_k^j))} = \frac{\mathsf{P}(v_{1..t}^{1..n})}{\mathsf{P}(v_{1..t-1}^{1..n})}$$
$$= \mathsf{P}(v_t^{1..n} | v_{1..t-1}^{1..n}) = \mathsf{P}(v_t^{1..n} | \bar{\imath}_{t-1})$$

In the last equality, we have used a conditional independence property of dynamic Bayesian networks: the set of variables $V_t^{1..n}$ is conditionally independent of all variables in previous slices given the interface $\bar{I}_{t-1}$.

So, the tuples in relation $cpd[V_t^1] \bowtie \ldots \bowtie cpd[V_t^n]$ contain the probabilities $\mathsf{P}(v_t^{1..n} | \bar{\imath}_{t-1})$ that are nonzero. Relation f_t' consists of the operator $\dot{\pi}_{\bar{I}_t \cup \bar{I}_{t-1} \cup V_t^{1..m}}$ applied to this join, so these probabilities are marginalized to $\mathsf{P}(\bar{w}_t | \bar{\imath}_{t-1})$, where $\bar{W}_t = V_t^{1..m} \cup \bar{I}_t$.

In the MSHMM model, this corresponds to $\mathsf{P}(x_t, s_t^{1..K} | x_{t-1})$: as we have explained, the number of these probabilities which are nonzero is linearly bounded when scaling up L and K. In other models, a similar line of reasoning may be possible.

4 Related work

Sparse/relational representations for probabilistic processing have been considered in the areas of constraint propagation[8] and information retrieval models[2], where good performance is reported. The well-known variable elimination algorithm can be successfully combined with conventional database query optimizations[3]. However, none of this work considers the area of sensor data, whose properties lead to specific optimizations.

In our previous work[5], we represented inference query plans using so-called sum-factor diagrams. The connection is as follows: a sum-factor diagram represents a right-deep join tree, with the base relations ordered from left to right as they would occur in the relational algebra expression. A dot indicates that the variable occurs in the schema of the base relation; a grey area indicates that the variable occurs in the schema of the intermediate relation that represented by the subexpression that starts at that point. A vertical bar indicates that the corresponding variable is projected away at that point.

References

[1] Stefan Arnborg, Derek G. Corneil, and Andrzej Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM J. Algebraic Discrete Methods*, 8(2):277–284, 1987.

[2] Roberto Cornacchia, Sándor Héman, Marcin Zukowski, Arjen P. de Vries, and Peter A. Boncz. Flexible and efficient IR using array databases. *VLDB J.*, 17(1):151–168, 2008.

[3] Héctor Corrada Bravo and Raghu Ramakrishnan. Optimizing MPF queries: decision support and probabilistic inference. In *SIGMOD Conference*, pages 701–712, 2007.

[4] Thomas Dean and Keiji Kanazawa. A model for reasoning about persistence and causation. *Computational Intelligence*, 5(3):142–150, 1989.

[5] Sander Evers, Maarten Fokkinga, and Peter M. G. Apers. Probabilistic processing of interval-valued sensor data. In *Proceedings of the 5th Workshop on Data Management for Sensor Networks, in conjunction with VLDB*, pages 42–48, August 2008.

[6] Ashish Gupta, Venky Harinarayan, and Dallan Quass. Aggregate-query processing in data warehousing environments. In Umeshwar Dayal, Peter M. D. Gray, and Shojiro Nishio, editors, *VLDB*, pages 358–369. Morgan Kaufmann, 1995.

[7] Uffe Kjærulff. A computational scheme for reasoning in dynamic probabilistic networks. In Didier Dubois and Michael P. Wellman, editors, *UAI*, pages 121–129. Morgan Kaufmann, 1992.

[8] David Larkin and Rina Dechter. Bayesian inference in the presence of determinism. In *Proceedings of the Ninth International Workshop on Artificial Intelligence and Statistics*, January 2003.

[9] S. L. Lauritzen and D. J. Spiegelhalter. 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, 1988.

[10] Kevin P. Murphy. *Dynamic Bayesian Networks: Representation, Inference and Learning*. PhD thesis, University of California, Berkeley, 2002.

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