sensor data management with probabilistic models

sander evers

Sensor Data Management with Probabilistic Models

Samenstelling van de promotiecommissie:

prof. dr. P.M.G. Apers	promotor
prof. dr. L. Feng	promotor
dr. M.M. Fokkinga	assistent-promotor
prof. dr. ir. A.P. de Vries	Centrum voor Wiskunde en Informatica
prof. dr. ir. B.R.H.M. Haverkort	Universiteit Twente
prof. dr. ir. A. Nijholt	Universiteit Twente
prof. dr. R.J. Wieringa	voorzitter, secretaris

CTIT Ph.D. thesis series no. 09-50
ISSN: 1381-3617
Centre for Telematics and Information Technology
P.O. Box 217, 7500 AE Enschede, The Netherlands

SIKS Dissertation Series No. 2009-28
The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems.

Cover artwork: Sander Evers
Original photograph by Will Montague, licensed under the Creative Commons BY-NC 2.0 license. See also: http://www.flickr.com/photos/willmontague

This thesis is typeset using LaTeX. All diagrams are drawn using the Ti*k*Z package.

Printed by: Ipskamp Drukkers, Enschede, The Netherlands

ISBN: 978-90-365-2867-2

SENSOR DATA MANAGEMENT WITH PROBABILISTIC MODELS

PROEFSCHRIFT

ter verkrijging van
de graad van doctor aan de Universiteit Twente,
op gezag van de rector magnificus,
prof. dr. H. Brinksma,
volgens besluit van het College voor Promoties
in het openbaar te verdedigen
op vrijdag 25 september 2009 om 16.45 uur

door

Sander Evers

geboren op 1 april 1979
te Enschede

Dit proefschrift is goedgekeurd door:

prof. dr. P.M.G. Apers (promotor)
prof. dr. L. Feng (promotor)
dr. M.M. Fokkinga (assistent-promotor)

If you ask the investigator how he or she can be sure that the numbers will eventually come right [. . .], your question will be rephrased and answered in the following terms: "I am ninety-five per cent sure," or "I am ninety-eight per cent sure." What does it mean to be ninety-five per cent sure? you may ask. "It means I will be right in at least nineteen cases out of twenty; or, if not in nineteen out of twenty, then in nineteen thousand out of twenty thousand," the investigator will reply. And which case is the present one, you may ask: the nineteenth or the twentieth, the nineteen-thousandth or the twenty-thousandth?

J.M. Coetzee, *Diary of a Bad Year*

Preface

As the chairman of my dissertation committee Roel Wieringa once mentioned in a Ph.D. student career seminar I attended, every Ph.D. has learned something about dealing with uncertainty. For me this is true on two levels, as it is also the topic of this thesis.

Of course, I didn't know that this would be the case when I started in the Databases group on the NWO project *Context Aware Data Management for Ambient Intelligence*. The citation at the start of this thesis represents the mindset I had about the subject: in short, a mucky business. Not for nothing, computer science has shielded itself from the uncertainties of the world by turning a 4.82256 V voltage into a rigorous logical true.

Four and a half years later, I am convinced that uncertainty and rigor do not preclude each other, and that indeed their combination can be very fruitful. Hopefully, my work can contribute a little bit in communicating this conviction to the data management community, for which the subject is reasonably unfamiliar.

When talking about the fruits of research, you sometimes stop and wonder who will be eating them. Although I never truly came into ethical problems, the localization setup in figure 1.1 did actually exist, and I had to put up signs with the slogan 'Bluetooth Brother is watching you' on our floor. In my defense, I can only say that sensor data research can better be performed in the public domain than in a commercial, military or governmental one.

To get back to Roel Wieringa's remark, 'doing' a Ph.D. not only produces tangible results such as the one you are holding in your hands, but also some intangible ones. About the most conspicuous of these, the status of doctor, my feelings have also changed somewhat during the course. Perhaps, it is no more an award for past achievements than a driver's license is a prize for being a great driver... and it rather means: go forth, and do research!

Acknowledgements

As many Ph.D. students have grudgingly remarked before me, the section you are reading now is probably by far the most read in this thesis. This is because you are looking for either a nice statement about yourself—a narcissism of which I am guilty as well[1]—or some other personal statements from me, which are of course much more interesting than the same technical presentations you have heard me practice on you a thousand times before.

Before getting to this, I would like to thank some people I do not know personally: the (mathematical) contributors to Wikipedia, who have changed the dissemination of knowledge forever, and have helped me a lot in finding useful mathematical tools. In a similar spirit, I am indebted to the free software community for providing tools without which at least this thesis would have looked very different. I mention two members in particular: Till Tantau for Ti*k*Z, the wonderfully comprehensive package I could construct all the diagrams I needed with, and my former roommate Stefan Klinger for giving me the final nudge to try Linux.

I would also like to say thank you to the other guys I have shared room ZI3090 with: Nicolas and (especially) Harold. Spending most of your day in each other's presence, simply the way you get along makes all the difference, and I must say I have absolutely nothing to complain about in this department. Secondly, learning to do research includes seeing others do this, with roommates as the first source. I hope I contracted some of their mentality of getting things done and keeping the big picture in mind.

After my roommates, the 'next of kin' in the group are the other Ph.D. students, for much the same reasons. Although the motivation was not always as prominent, I am glad that we had weekly meetings of our own, first coordinated by Ander and later by his worthy successor Riham; I hope this nice and useful tradition will continue. One person also deserves some attention here, and not only because he provided the espresso machine. I was glad to find in Robin someone with a similar attitude towards research (and coffee).

Next in line are all the other people who made the Databases group such a closely knit one. I will definitely remember spending lunch breaks on Go,

[1] read: if you have a chance to return the favor...

competing in the Zeskamp, racing each other in karts, and the many sketches we contrived and performed. And of course, none of these activities (nor the more serious ones) would have run so smoothly without the organisation of Sandra, Ida and Suse.

Another community in Enschede that I was happy to belong to is the close harmony choir Musilon. They have probably given me a hobby for life, and I hope I have given something back by being on the board for a year. Honorable mention goes to Rob and Kris, who also accompanied me to Broodje Cultuur on many Mondays during these four years.

Seeing others do research is one thing; seeing yourself do research is harder. I have three supervisors to thank for helping me achieve a necessary balance between self-doubt and self-confidence; as most people who know me will understand, this consisted mainly of instilling the latter. Peter, Maarten and Ling all did this in their own way. (And if only it didn't sound so much like an American self-management course,[2] these could have been labelled strategical confidence, technical confidence and maybe even spiritual confidence.)

As we are nearing the end of this section, I want to make sure to include the earlier 'teachers' that shared and nurtured my interest in the mathematical side of computer science. Besides the already mentioned MMF, these include Rick van Rein, Jan Kuper, Peter Achten and Rinus Plasmeijer.

As expected, this collection of praise reaches its culmination with the people who have invested so much of their lives in mine that it is almost too obvious to mention them: my parents Anton en Marjan, and my girlfriend Pauline.

—Sander

[2] hoera voor Twentse nuchterheid

Contents

Chapter 1

Introduction

The first decade of the new millennium has fostered a vision on information technology which has found its way into corporate, national and international research agendas by the name of ubiquitous computing[60], pervasive computing[52], and ambient intelligence[22]. In this vision, computing will be freed from the desktop and move into people's pockets, clothes, furniture and buildings. Moreover, the hassle of controlling attention-demanding devices will give way to a focus on the task at hand, supported by adaptive technology that defaults to appropriate behavior in each situation (*context awareness*).

This transition is motivated by the availability of enabling technologies like small batteries, flat displays, wireless communication infrastructure and cheap sensors. It cannot be denied that it is really taking place; one has only to call to mind the fast rise of car navigation systems, touch-pad PDAs, interactive whiteboards, wearable mp3 players and the Nintendo Wii (a game computer controlled by a motion sensing remote) in the last ten years.

However, the dream is far from realized. While the hardware is there, the software infrastructure is still largely missing. It is still the case that 'almost nothing talks to anything else, as evidenced by the number of devices in a typical house or office with differing opinions as to the time of day.'[29] If a shared clock already proves difficult, how could we ever assemble constellations of devices that share sensor information?

Historically, *database management systems* (DBMS) have come to play a key part in the cooperation among a heterogeneous collection of (corporate) applications. The DBMS functions as a stable hub, ensuring that all applications have the same consistent view of the world, both on model/schema level and on data level. To update or query this view, it provides a declarative language of basic operations, and takes the responsibility that these are carried out in an efficient, reliable and consistent way.

In a ubiquitous computing architecture, a similar role for data management can be imagined; however, the requirements are quite different from the traditional

office scenarios. This thesis focuses on one of the differences: the need to deal with *uncertainty* that arises naturally from all *sensor data* and the integration thereof. Although reasoning with uncertainty has proven very fruitful in the artificial intelligence and machine learning communities, it has not been very popular in the data management community because it tends to be at odds with scalability.

1.1 New requirements for data management

It is neither desirable nor possible to design a ubiquitous computing environment as a monolithic whole. Hence, there is a need for a certain infrastructure to connect the different parts to each other in a standardized way. Chen, Li and Kotz[14] express this as follows:

> Given the overwhelming complexity of a heterogeneous and volatile ubicomp environment, it is not acceptable for individual applications to maintain connections to sensors and to process the raw data from scratch. On the other hand, it is not feasible to deploy a common context service that could meet every application's need either. Instead, we envision an infrastructure that allows applications to reuse the context-fusion services already deployed, and to inject additional aggregation functions for context customization and user personalization where necessary.

Data management will be a part of this infrastructure. To get a theoretical grip on the requirements for data management, we make the same division of processes into *sensors* (data suppliers) and *applications* (data consumers). These two sides have the following properties:

- Sensors may come and go: new sensors are installed, sensors permanently break down or become obsolete and are removed.
- On a smaller timescale, the flow of *data* from a sensor may come and go: batteries run out and are replaced, network connections are not always operational.
- Sensors are heterogeneous: even sensors that provide the same information use different data formats, data rates and accuracy specifications.
- Applications may come and go.
- The connectivity of an application may come and go; it may be turned off by a user, lose its network connection, or temporarily move out of the environment altogether.
- Applications have heterogeneous information needs: they can *monitor* something in the environment and receive continuous updates, they may define

certain events of interest and set a *trigger* on them, they may ask for a *summary* of what has happened during the time that they were disconnected.

Next, we can imagine a *sensor data management system* that mediates between the data supply and demand sides. This system would have the following high-level goals:

- Modularity/flexibility: to enforce a separation of concerns between sensors and applications, applications should not subscribe to specific sensor data but rather to variables in a more abstract model of the sensed world. Like in the quote above, the system should allow definition of new views that aggregate low-level into higher-level information.
- Efficiency/timeliness: due to the asymmetry between producers (low-level, high-volume data) and consumers (lower-rate, high-level), the data management system in a ubiquitous computing environment will do a lot of processing. In this respect, it resembles an OLAP (On-Line Analytic Processing) system[13] rather than an OLTP (On-Line Transaction Processing) system. It is the responsibility of the system to answer queries efficiently and on time; this calls for preprocessing and caching.
- Reliability: Failing sensors should not break the system by keeping processes waiting or not answering queries. Also, data overflow from the sensor side and query overflow from the application side should be handled gracefully.

We illustrate these requirements using a localization example that will function as a running example throughout the thesis. Figure 1.1 shows the (partial) floor plan of an office corridor, in which a group of Bluetooth transceivers ('scanners') is used for localization. At several fixed positions (in the offices), a scanner is installed which performs four scans per minute. Such a scan returns a list of mobile devices that have been discovered within the reach of the scanner during the scanning period (about 10 seconds).

For modularity reasons, applications should not be interested in these raw scan results; what matters to them is the *location* of a mobile device. This location is modeled here as a number that represents an office or part of the corridor. Assuming that mobile devices are linked to people, applications could pose such queries as:

- In which location is person P now?
- Has somebody been in locations 10–15 within the last hour?
- Have person P and person Q met yesterday?

Hence, to satisfy the modularity requirement, the data management system could provide a *view* on the data with the schema *(person,location,time)*; ideally, this view

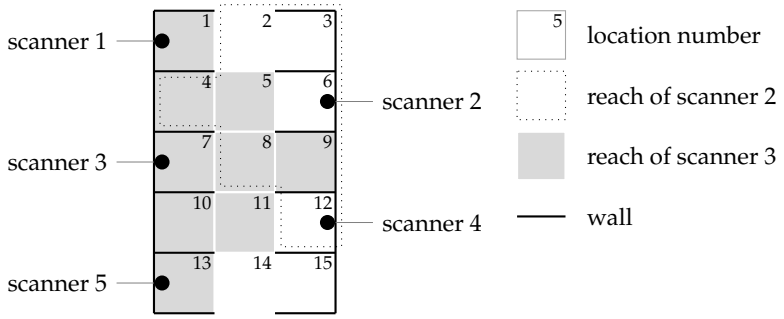


Figure 1.1: Floor plan for the Bluetooth localization example. The numbered squares are the locations that applications are interested in. At five positions, a scanner is installed that can detect a mobile Bluetooth device in a limited number of locations, as is shown for scanners 2 and 3.

would obey rules like *no person can be in two places at the same time* (a consistency constraint) and *at each time, each person is in some location* (a completeness constraint; this may require the introduction of an extra location away for the area outside of all sensor reach). There are several factors that complicate the formulation of such a view in terms of the scan results:

- The range of a scanner does not coincide with a single location.
- A device is not always detected when it is in the range of a scanner.
- Scanners are not scanning all the time.

Given these problems, it is hard to imagine an SQL query that could build such a view from the raw data (although such approaches exist, e.g. [35]); a further complication is that this query would have to be adapted when the system detects that a scanner is not working, or when additional scanners are introduced to the system. Also, consider the case where *cameras* are added to the system for extra accuracy; the query would have to fuse the information from scanner 3 that detects person P in the gray area, and from the camera that detects a person exactly in location 11, but cannot identify him or her as P.

In this thesis, we argue that data models should include uncertainty in order to define views like this; we give some supporting arguments in the next section. Above, we have highlighted the modularity requirement; as for the rest of the requirements, it is not to hard to imagine that:

- in order to process localization queries that span a considerable amount of time or space, the required data has to be *summarized* and *indexed* in some way, in order to satisfy the efficiency requirement.

- the system should behave reliably, and deal with an overload of queries in a graceful way (e.g. refuse queries, delay answers, or give answers with less accuracy).

The scope of this thesis does not further include the reliability requirement; the modularity and efficiency requirements can be found back in our research questions (section 1.3).

1.2 The case for probabilities

The vast majority of information systems only deals with certain data: to the system, a fact is either true or not. However, it can be argued that most human knowledge is uncertain. The Lowell database research self-assessment [1] acknowledges this:

> When one leaves business data processing, essentially all data is uncertain or imprecise. Scientific measurements have standard errors. Location data for moving objects involves uncertainty in current position. Sequence, image, and text similarity are approximate metrics. [...] Query processing must move from a deterministic model, where there is an exact answer for every query, to a stochastic one, where the query processor performs evidence accumulation to get a better answer to a query.

This raises the question how business systems have managed to escaped this uncertainty until now. Agre[3] provides an answer: 'The main tradition of computer system design, however, has a solution to this problem: restructure the activity itself in such a way that the computer can capture the relevant aspects of it.' For example, the activity of borrowing a book from a library is structured into a procedure where the borrower has to deal with a clerk who registers the transaction. Social practices, and even the architecture of a library with its check-out desk, are as much part of this structured activity as the information system.

As we have mentioned, *ambient* intelligence has the goal of minimizing such explicit interactions with the system. In the library system, the borrower should not even have to present his/her ID card and borrowed books to a scanner; the system should *infer*, for example using RFID tags, which action has taken place. In the words of Agre, the design choice is to 'reject the capture model, and instead register aspects of the environment that can serve as rough, heuristic (and therefore fallible) proxies for the institutional variables that are the real objects of interest.' Enter uncertainty.

Franklin[27] agrees that this is one of the 'challenges of ubiquitous data management':

> Whether the system obtains its context information from sensors, user input, PIM (personal information management) applications, or some

> combination of these, it must perform a good deal of processing over the data in order to be able to accurately assess the state of the environment and the intentions of the user. Thus, context-aware applications impose demanding requirements for inferencing and machine learning techniques. These processes will have to cope with incomplete and conflicting data, and will have to do so extremely efficiently in order to be able to interact with the user in a useful and unobtrusive manner.

This also makes clear why the uncertainty has to be dealt with *within the data management system*: the inference process has to take into account the input from multiple sources. As we stated in the previous section, it is the job of the data management system to maintain a separation of concerns between these sources.

We now focus on sensor data processing. Looking beyond the realm of traditional 'information systems', for example in the fields of artificial intelligence and scientific data processing, we see that the dominating method for dealing with the uncertainty in sensor data is probability theory, which provides a framework that is well-understood, theoretically sound and practically proven useful.[1] For example, in probabilistic localization models, uncertainty from different sources can 'cancel out' against each other, providing more accurate results[33]. Balazinska et al.[8] predict that probabilistic techniques will spread out of these domains into that of data management:

> Statistical analysis and modeling are perhaps the most ubiquitous processing tasks performed on sensor data. This has always been true of scientific data management, where sensor data collection usually aims to study, understand, and build models of real-world phenomena. Increasingly, however, the need to use statistical-modeling tools arises in nonscientific application domains as well. Many of the most common sensor-data processing tasks can be viewed as applications of statistical models. Examples include
>
> - forming a stochastic description or representation of the data,
> - identifying temporal or spatial trends and patterns in the data,
> - online filtering and smoothing (for example, Kalman filters),
> - predictive modeling and extrapolation,
> - detecting failures and anomalies, and
> - probabilistically modeling higher-level events from low-level sensor readings.

[1] For theoretical arguments in favor of probability theory over other approaches dealing with uncertainty, we refer to a section in the first chapter of Pearl's seminal work[48] by the same title as this one.

However, to make this transition, the mathematical techniques and programmatic tools would have to be 'targeted at the declarative management and processing of large-scale data sets', which is not yet the case[59]. It is here that we position the objective of our research.

1.3 Research questions

Broadly speaking, our research objective is to investigate the use of probabilistic models in a tentative sensor data management system as described above. Our first research question stems directly from the modularity goal (section 1.1) of such a system. Following well-known computer science principles, sensor data models should not be defined as monolithic wholes, but in a modular fashion: it should be possible to design, alter or remove the part of the model concerning one particular sensor with as little knowledge about (or impact on) the rest of the model as possible. This leads to the question:

Q1 How can probabilistic models be defined in a modular way?

We interpret this as an investigation into useful *structures* of a model in terms of probabilistic variables and relations between them. A similar question can be posed about the parameters *within* a probabilistic relation. In particular, we examine a transition model $\mathsf{P}(X_t|X_{t-1})$. For a variable X_t with a large and heterogeneous (discrete) state space, i.e. the set of values $\mathrm{dom}(X_t)$ that it can take, we are interested in composing the transition model out of small local transition models:

Q2 How can a transition model be constructed in a modular way?

The third question deals with probabilistic *inference*, i.e. the calculation of a probability distribution over a query variable, which makes up most of the work that a sensor data management system will perform. With standard approaches to inference, the processing time and space scale badly when the number of sensors in a model is increased; the same holds for when the discrete state space of a variable is enlarged. We ask ourselves:

Q3 How can probabilistic inference be performed efficiently in a situation where the number of sensors and the domains of variables are scaled up?

1.4 Research approach and thesis structure

We investigate the research questions using the localization example from section 1.1 as a test case; from this starting point, we try to generalize the results as much as possible. We strive for theoretical frameworks that support the construction and execution of probabilistic models for sensor data.

For the first research question, this means that we revisit the theory of (dynamic) Bayesian networks in the light of sensor data models and modularity. This is done in chapter 2 and 3. Technically, the models presented in these chapters are not very novel; the contributions of this thesis here mainly consist in summarizing relevant material, pointing out why it is relevant, and making it accessible to a scientific audience that has no background in probabilities.

The second question is answered in chapter 4. This part of the research consists of rephrasing the question in a formal way, which turns out to take the form of a system of linear equations; next, we make use of the special structure of this system to obtain faster and more insightful solutions. For this solution method, we venture a little into matrix and graph theory. The formulation of the problem and its solution are novel to our knowledge.

The answers to the third research question are to be found in chapters 5 and 6, and make up the main theoretical and technical contributions of this thesis. To obtain optimizations of the inference in the localization example (chapter 6), we first introduce a relational algebra framework to reason about the optimization of inference queries in general (chapter 5).

1.5 Related work

The need to merge information from different sensors has arisen long before the ubiquitous computing hype, namely in the military domain, where it is known as data fusion[30]. A reason why this has not led to generic data management techniques could be that these military systems (for example, the array of radars and sonars on a battleship) have fairly static configurations. The sheer cost of these sensors could also have played a role; Hellerstein, Wong and Madden[32] analyze a similar question, namely why databases have not been used for satellite data, as follows:

> The answer lies with the "market" for remote sensing. NASA is one of the only customers for remote sensing data management software. They best understand their own needs, and their software budgets are fairly large—software is cheaper than launching a set of satellites. Hence the traditional DBMS focus on general-purpose applicability and flexibility is not of primary importance to this community. Instead, they seem to prefer to write or contract out custom code.

Regarding uncertainty in databases, a community effort of the past decade centers around *probabilistic databases*[10, 5, 16], which mostly take the approach of modeling the *existence* of a tuple in a certain table as a stochastic variable, with independence assumptions among the tuples within a table as well as among tables. This approach is not applicable to sensor data, where these dependencies do exist[38]—moreover, they are exploited to improve accuracy.

Perhaps the work that comes closest to ours is that of Kanagal and Deshpande[38]; they offer a system in which a dynamic Bayesian network is used to provide a *view* that consists of probabilistic variables that are continuously updated by incoming sensor data. Where they have taken an experimental systems approach, we take a more theoretical one. Also very relevant are [59] and [53], which acknowledge the optimization opportunities for sensor data that stem from the repetition of conditional probability distributions throughout a probabilistic model or query.

Chapter 2

Modeling sensor data using a Bayesian Network

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.

Apart from externally imposed factors, uncertainty can also stem from the inability or unwillingness either to model all the world's intricacies or to reason with such a complicated model[51]. Choosing a simpler probabilistic model provides a way to trade off accuracy for reasoning power.

A probabilistic model defines a set of variables and the relation between them. This relation is probabilistic instead of deterministic: it does not answer the question *what is the value of C, given that $A = a$ and $B = b$?* but rather *what is the probability distribution over C, given that $A = a$ and $B = b$?* or *what is the probability distribution over C, given certain probability distributions over A and B?*. In sensor data processing, the values a and b are readings provided by sensors, and C is a property of the sensed phenomenon.

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. However, in the context of our data management requirements, we focus on the *Bayesian network*, a generic model in which probabilistic variables and their relations can be defined in a modular and intuitive way. Bayesian networks are the most popular member of a family called *graphical models*:

> Graphical models are a marriage between probability theory and

> graph theory. They provide a natural tool for dealing with two problems that occur throughout applied mathematics and engineering—uncertainty and complexity—and in particular they are playing an increasingly important role in the design and analysis of machine learning algorithms. Fundamental to the idea of a graphical model is the notion of modularity—a complex system is built by combining simpler parts. Probability theory provides the glue whereby the parts are combined, ensuring that the system as a whole is consistent, and providing ways to interface models to data. The graph theoretic side of graphical models provides both an intuitively appealing interface by which humans can model highly-interacting sets of variables as well as a data structure that lends itself naturally to the design of efficient general-purpose algorithms.
>
> Michael Jordan, *Learning in graphical models* [36]

It is perhaps more accurate to view graphical models not as a class of models themselves, but rather as meta-models: a Bayesian network is the language in which a probabilistic model can be defined. In this aspect, Bayesian networks play about the same role for probabilistic data modeling as entity-relationship diagrams do for relational data modeling.

The goal of this chapter is to provide a solid theoretical framework for the application of Bayesian networks in sensor data processing, which is needed for subsequent chapters. It assumes no knowledge of probabilistic modeling, and starts with a review of the relevant concepts from probability theory (section 2.1), followed by a review of the semantics of a Bayesian network (section 2.2). In section 2.3, we discuss the general use of Bayesian networks for sensor data in particular; in section 2.4, we present some concrete models. The chapter's focus is on *modeling*, but it concludes with a section on querying these models (2.5) and one on evaluating and learning them (2.6). Our approach to the theory is perhaps somewhat more formal than usual—we consider the ability to formally manipulate probabilistic models and queries as important for optimizing inference (see chapters 5 and 6).

2.1 Theoretical foundations of probability

Probability is conventionally defined either as a *degree of belief* in the occurrence of an event or as a long-run *frequency* of this occurrence. These interpretations have been subject to much academic debate. However, regardless of what semantics are given to the probabilities, a probabilistic model has a rigorous formal definition in terms of set theory, which we present here. In its most generic form, this definition is quite intricate, as it accommodates continuous variables and uncountable probability spaces. However, we restrict ourselves to discrete (even

finite) variables, and can use a simpler definition. A probabilistic model is defined in terms of a *probability space* and *random variables*.

2.1.1 Probability spaces and random variables

A *probability space* is a pair (P, Ω) of a *probability measure* P and a *sample space* Ω. This Ω is a countable set that represents the universe of discourse of our model. An *event* is a subset of Ω, and a probability measure is a function from events to $\mathbb{R}_0^1$ (the real numbers between 0 and 1, inclusive), satisfying the following criteria:

$$\begin{aligned} \mathsf{P}(\emptyset) &= 0 \\ \mathsf{P}(\Omega) &= 1 \\ \mathsf{P}(a \cup b) &= \mathsf{P}(a) + \mathsf{P}(b) \quad \text{for all disjoint } a, b \subseteq \Omega \end{aligned} \qquad \textbf{(Prob-Space)}$$

An example probability space is $\Omega = \{\omega_1, \omega_2, \omega_3, \omega_4\}$, with $\mathsf{P}(\{\omega_1\}) = 0.1$, $\mathsf{P}(\{\omega_2\}) = 0.2$, $\mathsf{P}(\{\omega_3\}) = 0.3$ and $\mathsf{P}(\{\omega_4\}) = 0.4$. The probabilities for all other subsets of Ω follow from (Prob-Space), e.g. $\mathsf{P}(\{\omega_2, \omega_3\}) = \mathsf{P}(\{\omega_2\}) + \mathsf{P}(\{\omega_3\}) = 0.5$.

A *random* (or *stochastic*) *variable* is a function on the sample space. For example, we could define the variables X and Y on the above defined sample space:

$$\begin{aligned} X(\omega_i) &= i \\ Y(\omega_i) &= i \bmod 2 \end{aligned}$$

These functions have *range* $\{1, 2, 3, 4\}$ and $\{0, 1\}$, respectively; however, we refer to this as the variable's *domain*, and write $\mathrm{dom}(Y) = \{0, 1\}$. Random variables are used in *predicates*, for example $X > 2$. This predicate is used as a shorthand for the event

$$\{\, \omega_i \mid X(\omega_i) > 2 \,\},$$

for example in $\mathsf{P}(X{>}2) = \mathsf{P}(\{\omega_3, \omega_4\}) = 0.7$. Predicates can involve multiple random variables: $\mathsf{P}(X{=}Y) = \mathsf{P}(\{\omega_1\}) = 0.1$. The *product* XY of variable X and variable Y is a variable whose values consist of tuples of X and Y values:

$$XY(\omega) = (X(\omega), Y(\omega))$$

Note that, by this definition, the event $XY = (x, y)$, i.e. $\{\, \omega \mid XY(\omega){=}(x, y) \,\}$, equals $\{\, \omega \mid X(\omega){=}x \wedge Y(\omega){=}y \,\}$: the event $X{=}x \wedge Y{=}y$. In the remainder, we will abbreviate this conjunction to $X{=}x, Y{=}y$.

The *probability distribution* of a random variable X is a function that maps each value x in the variable's domain to the probability $\mathsf{P}(X{=}x)$. The probability distributions f_X, f_Y and f_{XY} on the above defined variables are:

$f_X(1) = 0.1$		$f_{XY}(1,0) = 0$	$f_{XY}(1,1) = 0.1$
$f_X(2) = 0.2$	$f_Y(0) = 0.6$	$f_{XY}(2,0) = 0.2$	$f_{XY}(2,1) = 0$
$f_X(3) = 0.3$	$f_Y(1) = 0.4$	$f_{XY}(3,0) = 0$	$f_{XY}(3,1) = 0.3$
$f_X(4) = 0.4$		$f_{XY}(4,0) = 0.4$	$f_{XY}(4,1) = 0$

Note that some probabilities of f_{XY} are 0 because they correspond to empty events. For example, $\mathsf{P}(XY{=}(1,0)) = \mathsf{P}(\{\,\omega_i \mid X(\omega_i){=}1, Y(\omega_i){=}0\,\}) = \mathsf{P}(\emptyset) = 0$. Also, note that the values of a probability distribution always add to 1. This holds because a variable X *partitions* the sample space into the events $X{=}1$, $X{=}2$, et cetera. Because of (Prob-Space), the P values on these events add up to one.

2.1.2 Implicit probability spaces

The above examples are given only to clarify the theory; in the practice of probabilistic modeling, the sample space is never explicitly defined. One starts by postulating some random variables and their domains; the sample space is then implicitly defined as all possible combinations of values for the variables. For example, let us model a world where a red and a blue die are thrown. We represent the number that comes up on the red die with the random variable R, and that on the blue die with B; $\mathrm{dom}(R) = \mathrm{dom}(B) = \{1,2,3,4,5,6\}$. The implicitly defined sample space is then $\Omega = \mathrm{dom}(R) \times \mathrm{dom}(B) = \{(1,1),(1,2),\ldots,(6,6)\}$, and the random variables are the following functions on Ω:

$$R(x,y) = x$$
$$B(x,y) = y$$

The predicate $R = 2$ thus denotes the event

$$\{\,(x,y) \mid R(x,y){=}2\,\} = \{(2,1),(2,2),(2,3),(2,4),(2,5),(2,6)\}$$

The probability measure is usually defined by means of a given joint probability distribution over all variables, in this case f_{BR}. We have intentionally reversed the order of R and B here to highlight the difference between the sample space and the domain of the joint probability distribution. There is also a correspondence: each event $BR = (b,r)$ is a distinct singleton set in Ω, and vice versa. The given value $f_{BR}(b,r)$ tells us the value of P on this event; as we will show, this defines the P function completely. For example, the event $BR = (3,2)$ is defined as

$$\{\,\omega \mid B(\omega){=}3, R(\omega){=}2\,\} = \{(2,3)\}$$

If $f_{BR}(3,2) = 0.03$ is given, this tells us that $\mathsf{P}(BR{=}(3,2)) = \mathsf{P}(\{(2,3)\}) = 0.03$. Thus, P is defined on all singleton sets $\{\,\{\omega\} \mid \omega \in \Omega\,\}$, and the value of P on other subsets can be derived using (Prob-Space). The only criterion that the joint probability distribution has to satisfy, in order not to violate the second axiom, is that the individual probabilities add up to 1.

So, we have shown how an implicit probability space can be defined given a set of variables and a joint probability over them. The structure of this probability space is actually never needed in probability calculations. For example, the probability $\mathsf{P}(R{=}2)$ can be directly expressed in terms of the joint probability. This

is done by expressing the event $R = 2$ in terms of the events of the form $BR = (b, 2)$:

$$\begin{aligned}
& \{\, \omega \mid R(\omega){=}2 \,\} \\
= \\
& \{\, \omega \mid R(\omega){=}2 \,\} \cap \Omega \\
= \qquad & B{=}b \text{ events partition } \Omega \\
& \{\, \omega \mid R(\omega){=}2 \,\} \cap \bigcup_{b \,\in\, \mathrm{dom}(B)} \{\, \omega \mid B(\omega){=}b \,\} \\
= \\
& \bigcup_{b \,\in\, \mathrm{dom}(B)} \{\, \omega \mid R(\omega){=}2, B(\omega){=}b \,\} \\
= \\
& \bigcup_{b \,\in\, \mathrm{dom}(B)} \{\, \omega \mid BR(\omega){=}(b, 2) \,\}
\end{aligned}$$

The fact that the $B = b$ events partition Ω also causes the events $BR{=}(b, 2)$ to be disjoint, and therefore, using (Prob-Space),

$$\mathsf{P}(R{=}2) = \sum_{b \,\in\, \mathrm{dom}(B)} \mathsf{P}(BR{=}(b, 2))$$

The same reasoning can be followed in an arbitrary probabilistic model, for any predicate θ and any variable X:

$$\mathsf{P}(\theta) = \sum_{x \,\in\, \mathrm{dom}(X)} \mathsf{P}(\theta, X{=}x) \qquad \textbf{(Marginalize)}$$

Applying this rule repeatedly, the predicate on the rhs can be supplemented to include all the variables in the joint distribution; the expression then becomes a multi-dimensional summation over the variables that are not in θ.

Bottom line: it is perfectly possible to define and use a probabilistic model without ever mentioning the probability space. One reason why we do mention it is that it provides a formal means of comparing two probabilistic models with each other. The two models implicitly define two probability spaces (P, Ω) and (P', Ω'); these models are said to be *consistent* with respect to a predicate θ if $\mathsf{P}(\theta) = \mathsf{P}'(\theta)$.

2.1.3 Conditional probabilities

The *conditional probability* $\mathsf{P}(b|a)$ (where a and b are events) is defined as follows:

$$\mathsf{P}(b|a) = \frac{\mathsf{P}(b \cap a)}{\mathsf{P}(a)} \quad \text{if } \mathsf{P}(a) \neq 0$$

It is undefined if $\mathsf{P}(a) = 0$. However, in Bayesian networks conditional probabilities are used the other way around; in the specification of the model, the probabilities $\mathsf{P}(X{=}x)$ and $\mathsf{P}(Y{=}y|X{=}x)$ are given for all x and y, and together determine:

$$\mathsf{P}(XY{=}(x,y)) = \mathsf{P}(X{=}x)\mathsf{P}(Y{=}y|X{=}x)$$

Thus, one specifies a value $\mathsf{P}(Y{=}y|X{=}x) = v$, even if $\mathsf{P}(X{=}x) = 0$. The value just does not matter, because $\mathsf{P}(XY{=}(x,y)) = 0$ for any v.

Then, the reader may ask, why bother specifying this value at all? The answer is modularity: the probability $\mathsf{P}(Y{=}y|X{=}x)$ is specified without full knowledge of variable X. For example, let X model the location of a transmitting device, and Y the strength of its received signal at a certain fixed receiver. The conditional probability $\mathsf{P}(Y{=}y|X{=}x)$ models only the uncertainty in the *sensing process*, and can be very well defined without needing to know anything about the probability distribution over the location: if the transmitter would be 50m up in the air ($X = x$), it would generate signal strength y with a probability $\mathsf{P}(Y{=}y|X{=}x) = v$. Afterward, this sensing model may be used in a situation where this location is impossible (i.e. X is the location of a car), in which case $\mathsf{P}(X{=}x) = 0$, and the value v is irrelevant.

The *conditional probability distribution* (cpd) of Y given X is the function mapping any combination of values y and x (from $\mathrm{dom}(Y)$ and $\mathrm{dom}(X)$, respectively) to $\mathsf{P}(Y{=}y|X{=}x)$. For a cpd, it holds that $\sum_{y\in\mathrm{dom}(Y)} \mathsf{P}(Y{=}y|X{=}x) = 1$ for every x; again, this follows from (Prob-Space)[p. 13], provided that $\mathsf{P}(X{=}x) > 0$. For Bayesian networks, it is true by definition, also when $\mathsf{P}(X{=}x) = 0$; see section 2.2.

2.1.4 Conditional independence

Two variables A and B in a probabilistic model are called *independent* iff

$$\forall a, b.\ \mathsf{P}(AB{=}(a,b)) = \mathsf{P}(A{=}a)\mathsf{P}(B{=}b)$$

This implies that

$$\forall a, b.\ \mathsf{P}(A{=}a|B{=}b) = \mathsf{P}(A{=}a)$$
$$\forall a, b.\ \mathsf{P}(B{=}b|A{=}a) = \mathsf{P}(B{=}b)$$

or, in words, *knowledge about one variable does not change the probability of the other*. Actually, no probability ever really *changes*, of course—the probability measure P is as immutable as any other mathematical function. However, in colloquial speech, the notion of probabilities that change (or *beliefs* that are *updated*) when more evidence becomes available is often used.

Independence does not occur often in probabilistic models, because the very reason that one puts two variables in one model is that knowledge of one affects

knowledge of the other. A notion that is used much more often is *conditional independence*. The variables A and B are conditionally independent *given* C iff

$$\forall a, b, c.\ \mathsf{P}(AB{=}(a,b)|C{=}c) = \mathsf{P}(A{=}a|C{=}c)\mathsf{P}(B{=}b|C{=}c)$$

Again, this implies that

$$\forall a, b, c.\ \mathsf{P}(A{=}a|B{=}b, C{=}c) = \mathsf{P}(A{=}a, C{=}c)$$
$$\forall a, b, c.\ \mathsf{P}(B{=}b|A{=}a, C{=}c) = \mathsf{P}(B{=}b, C{=}c)$$

Here, a translation in natural language would be: *given that $C = c$, knowledge of B is irrelevant for the probabilities over A, and vice versa*. Asserting these kind of independencies is an important part of probabilistic modeling; it makes models easier to specify or learn (because it reduces the number of parameters), and it makes reasoning more efficient. In section 2.2.2, we discuss how the graphical structure of a Bayesian network corresponds to assertions of conditional independence.

2.1.5 Notational shorthands

We introduce some notational shorthands, most of which are commonly used in probabilistic modeling:

- We write $\mathsf{P}(x)$ instead of $\mathsf{P}(X{=}x)$; the implicit variable X is syntactically derived from the abstract value x.
- If we have defined the random variables V_1 through V_n, then $\mathsf{P}(v_{1..n})$ means $\mathsf{P}(V_1 V_2 \cdots V_n = (v_1, v_2, \ldots, v_n))$.
- If we have not defined an order on the set of variables $\bar{V}$, we write $\mathsf{P}(\bar{v})$; in that case, any order can be taken (but the order in the product of variables should be the same as the order in the tuple of values).
- Sometimes we index a variable by a set instead of by numbers. For example, we define the variables X_A, X_B, X_C, and $\bar{S} = \{A, B, C\}$; then $\mathsf{P}(x_{\bar{S}})$ means $\mathsf{P}(X_A X_B X_C = (x_A, x_B, x_C))$ (again, any order on $\bar{S}$ can be taken).
- Note: the above expansion of $x_{\bar{S}}$ is *syntactic*: the symbolic values x_A, x_B and x_C can be captured. For example, $\sum_{x_A} \mathsf{P}(x_{\bar{S}}) = \mathsf{P}(x_B, x_C)$.
- In summations, we implicitly sum over the whole domain of a variable: $\sum_x \mathsf{P}(\ldots)$ means $\sum_{x \in \mathrm{dom}(X)} \mathsf{P}(\ldots)$. Again, the variable over which to sum (X) is syntactically derived from the abstract value (x).

There is also a common notational shorthand that we explicitly *do not* use: $\mathsf{P}(X)$ for the probability distribution over X. In our notation, probabilities $\mathsf{P}(\ldots)$ are always real numbers between 0 and 1, and never distributions (functions/arrays). Part of

the reason for this is that we introduce a notation $p[X]$ to represent a distribution in chapter 5, and we want to make a clear distinction between the two.

Informally, we do talk about "the distribution $\mathsf{P}(x)$", or "the conditional distribution $\mathsf{P}(y|x)$", as is common language in the AI literature. In this usage, there are always abstract values x and y, and never concrete values (like 79 or *true*); formally, the distributions that are meant are $\lambda x.\ \mathsf{P}(x)$ and $\lambda y, x.\ \mathsf{P}(y|x)$.

2.2 Defining a model using a Bayesian network

In the previous section, we explained that a probabilistic model is usually defined by a set of random variables (including their domains), and a joint probability distribution over these. In this section, we show how this joint probability can be defined in a concise way by means of a *Bayesian network*.

Since their introduction by Pearl[48] in the 1980s, Bayesian networks—formerly also known as *belief networks* or *causal networks*—have evolved into a de facto standard for probabilistic modeling. In spite of these names, Bayesian networks are not restricted to representing beliefs or causal relations, nor do they imply a commitment to the "Bayesian interpretation" of probability. A Bayesian network defines a joint probability distribution in terms of (smaller) conditional probability distributions, and it is up to the modeler to attach semantics to this distribution.

2.2.1 Formal definition

A Bayesian network over a set $\bar{V}$ of random variables consists of:

1. A directed acyclic graph with $\bar{V}$ as nodes.

2. For each variable $V \in \bar{V}$, the conditional probability distribution (cpd) given all its *parents* in the graph (i.e. all variables U for which an arrow $U \to V$ exists).

This Bayesian network defines a probabilistic model—i.e. a joint probability over all variables—in the following way: $\mathsf{P}(\bar{v})$ is defined to be the product of all cpds on the values $\bar{v}$.

We expand this into a more formal definition. In this section, we make a purely technical distinction between the nodes $\bar{V}$ in the graph and the random variables in the probabilistic model: we denote the latter by X, indexed by $\bar{V}$. There is thus a one-to-one correspondence between $\bar{V}$ and $X_{\bar{V}}$; if $\bar{V} = \{A, B\}$, then node A in the graph corresponds to variable X_A, and B corresponds to X_B. A Bayesian network then consists of:

1. A set $\bar{V}$ and a domain function $\mathrm{dom} : \bar{V} \to \wp\mathcal{V}al$ (for some universal set of values $\mathcal{V}al$). The set $\bar{V}$ is ordered $\{V_1, \ldots, V_n\}$.

2. A directed acyclic graph (DAG) on $\bar{V}$ that respects the order, i.e. for each arrow $V_i \to V_j$ in the graph, $i < j$ holds. (This does not impose any restrictions; for every DAG, there is at least one such order on the variables.) This graph induces a function *Parents*, which maps every $V_i \in \bar{V}$ to its list of parents, ordered in the node order mentioned above. Thus, if V_5 has parents V_3 and V_2, then $\mathit{Parents}(V_5) = [V_2, V_3]$. To address an element of this list, we write $\mathit{Parents}(V_5)_1 = V_2$ (we use a 1-based index).

3. For each $V \in \bar{V}$, a function

$$c_V : \mathrm{dom}(V) \times \mathrm{dom}(\mathit{Parents}(V)_1) \ldots \times \mathrm{dom}(\mathit{Parents}(V)_n) \to \mathbb{R}_0^1$$

with the following restriction: for each combination $x_{\mathit{Parents}(V)}$, it should hold that $\sum_{x_V} c_V(x_V, x_{\mathit{Parents}(V)}) = 1$.

The random variables of this Bayesian network are defined to be $X_{\bar{V}}$, with $\mathrm{dom}(X_V) = \mathrm{dom}(V)$ for all $V \in \bar{V}$. Over these variables, the following joint probability is defined:

$$\mathsf{P}(x_{\bar{V}}) = \prod_{V \in \bar{V}} c_V(x_V, x_{\mathit{Parents}(V)})$$

In the literature, this definition is usually formulated:

$$\mathsf{P}(x_{\bar{V}}) = \prod_{V \in \bar{V}} \mathsf{P}(x_V | x_{\mathit{Parents}(V)}) \qquad \textbf{(Fact-BN)}$$

However, this definition is circular: the probability space is defined in terms of the probability space. Therefore, we favor the first definition, as it is theoretically clean; we will show below that (Fact-BN) follows from it as a theorem.

We will first show that the joint probability over any subset $\bar{W} \subseteq \bar{V}$ which is closed under *Parents*, i.e. for which $W \in \bar{W} \Rightarrow \mathit{Parents}(W) \subseteq \bar{W}$ holds, is also a product of the c_V functions. We derive this probability by summing out all the other variables $\bar{U} = \bar{V} \setminus \bar{W}$ from $\mathsf{P}(x_{\bar{V}})$, and then moving the summations into the expression. This is possible because of the distributive law $ab + ac = a(b + c)$, which takes the following form for $\sum$-expressions:

$$\sum_x (\phi * \psi) = \left(\sum_x \phi\right) * \psi \qquad \text{if } \psi \text{ does not contain free variable } x \qquad \textbf{(Σ-Distr-L)}$$

$$\sum_x (\phi * \psi) = \phi * \sum_x \psi \qquad \text{if } \phi \text{ does not contain free variable } x \qquad \textbf{(Σ-Distr-R)}$$

When we arrange the product of c_U functions in $\bar{V}$ order, every summation $\sum_{x_U}$ can be moved until the corresponding c_U factor using (Σ-Distr-R), because the factors to the left of it can not contain variable x_U. The $\bar{U}$ variables, ordered in $\bar{V}$ order, are denoted $U_1, \ldots, U_m$. In the product, the $\bar{U}$ factors can be arranged after all $\bar{W}$ factors: as $\bar{W}$ is closed under *Parents*, the latter do not contain any

x_U variables. After moving the summations, we can repeatedly eliminate the rightmost summation, because $\sum_{x_{U_i}} c_{U_i}(x_{U_i}, \ldots) = 1$:

$$
\begin{aligned}
& \mathsf{P}(x_{\bar{W}}) \\
= \quad & \text{by (MARGINALIZE)}^{\text{p. 15}} \\
& \sum_{x_{\bar{U}}} \mathsf{P}(x_{\bar{V}}) \\
= \quad & \text{joint probability of a Bayesian network} \\
& \sum_{x_{\bar{U}}} \left(\prod_{W \in \bar{W}} c_W(x_W, x_{Parents(W)})\right) \prod_{U \in \bar{U}} c_U(x_U, x_{Parents(U)}) \\
= \quad & \text{by (Σ-DISTR-R); } \bar{W} \text{ is closed under } Parents \\
& \left(\prod_{W \in \bar{W}} c_W(x_W, x_{Parents(W)})\right) \sum_{x_{\bar{U}}} \prod_{U \in \bar{U}} c_U(x_U, x_{Parents(U)}) \\
= \quad & \text{by (Σ-DISTR-R); the } c_U \text{ factors are arranged in } \bar{V} \text{ order} \\
& \left(\prod_{W \in \bar{W}} c_W(x_W, x_{Parents(W)})\right) \\
& \quad \sum_{x_{U_1}} c_{U_1}(x_{U_1}, x_{Parents(U_1)}) \sum_{x_{U_2}} c_{U_2}(x_{U_2}, x_{Parents(U_2)}) \cdots \sum_{x_{U_m}} c_{U_m}(x_{U_m}, x_{Parents(U_m)}) \\
= \quad & c_{U_m} \text{ adds up to 1} \\
& \left(\prod_{W \in \bar{W}} c_W(x_W, x_{Parents(W)})\right) \sum_{x_{U_1}} c_{U_1}(x_{U_1}, x_{Parents(U_1)}) \sum_{x_{U_2}} c_{U_2}(x_{U_2}, x_{Parents(U_2)}) \cdots 1 \\
= \quad & \\
& \ldots \\
= \quad & c_{U_2} \text{ adds up to 1} \\
& \left(\prod_{W \in \bar{W}} c_W(x_W, x_{Parents(W)})\right) \sum_{x_{U_1}} c_{U_1}(x_{U_1}, x_{Parents(U_1)}) \cdot 1 \\
= \quad & c_{U_1} \text{ adds up to 1} \\
& \left(\prod_{W \in \bar{W}} c_W(x_W, x_{Parents(W)})\right) \cdot 1
\end{aligned}
$$

Thus, we conclude

$$\mathsf{P}(x_{\bar{W}}) = \prod_{W \in \bar{W}} c_W(x_W, x_{Parents(W)}) \qquad \text{if } \bar{W} \text{ is closed under } Parents$$

(JOINT-SUBSET-BN)

Also, note that if we take $\bar{W} = \emptyset$ (so $\bar{U} = \bar{V}$), the above calculation (without the first equality) yields $\sum_{x_{\bar{V}}} \mathsf{P}(x_{\bar{V}}) = 1$, and therefore the function P derived via the construction in section 2.1.2 is a valid probability space.

Next, we derive the value of a cpd $\mathsf{P}(x_V | x_{Parents(V)})$ over a single variable V in a Bayesian network. We cannot use the fraction of $\mathsf{P}(x_V, x_{Parents(V)})$ and $\mathsf{P}(x_{Parents(V)})$ directly, because the sets $\{V\} \cup Parents(V)$ and $Parents(V)$ are generally not closed

under *Parents*. However, we can use the transitive closure of these sets by extending them with the *ancestors* of V:

$$Anc(V) = \{\, W \mid \text{there is a path from } W \text{ to } V \wedge W \neq V \wedge W \notin Parents(V) \,\}$$

Now, $V^* = \{V\} \cup Parents(V) \cup Anc(V)$ and $V^+ = Parents(V) \cup Anc(V)$ are both closed under *Parents*, so

$$\begin{aligned}
& \mathsf{P}(x_V, x_{Parents(V)}, x_{Anc(V)}) \\
= \quad & \text{apply (Joint-Subset-BN) to } V^* \\
& \textstyle\prod_{W \in V^*} c_W(\ldots) \\
= \quad & \\
& c_V(x_V, x_{Parents(V)}) \textstyle\prod_{W \in V^+} c_W(\ldots) \\
= \quad & \text{apply (Joint-Subset-BN) to } V^+ \\
& c_V(x_V, x_{Parents(V)}) \mathsf{P}(x_{Parents(V)}, x_{Anc(V)})
\end{aligned}$$

Summing over all $x_{Anc(V)}$ values on both sides of the above equation:

$$\begin{aligned}
& \sum_{x_{Anc(V)}} \mathsf{P}(x_V, x_{Parents(V)}, x_{Anc(V)}) = c_V(x_V, x_{Parents(V)}) \sum_{x_{Anc(V)}} \mathsf{P}(x_{Parents(V)}, x_{Anc(V)}) \\
\equiv \quad & \text{by (Marginalize)}^{\text{p. 15}} \\
& \mathsf{P}(x_V, x_{Parents(V)}) = c_V(x_V, x_{Parents(V)}) \mathsf{P}(x_{Parents(V)}) \\
\equiv \quad & \text{divide both sides, assume nonzero} \\
& \frac{\mathsf{P}(x_V, x_{Parents(V)})}{\mathsf{P}(x_{Parents(V)})} = c_V(x_V, x_{Parents(V)}) \\
\equiv \quad & \text{definition of conditional probability} \\
& \mathsf{P}(x_V | x_{Parents(V)}) = c_V(x_V, x_{Parents(V)})
\end{aligned}$$

Hence, the cpds (whenever they are formally defined) are the same as the c_V functions from our definition of a Bayesian network, and (Fact-BN) follows as a theorem.

2.2.2 Graphical properties

In general, the factorization of the joint distribution of a Bayesian network leads to a lot of conditional independencies. The independencies we mean here are those purely derived from the *form* of the joint probability distribution, and not from the actual values of the cpds. For example, in a graph $A \to B \to C$, applying (Joint-Subset-BN) to $\{A, B, C\}$ and $\{A, B\}$ yields, respectively,

$$\begin{aligned}
\mathsf{P}(x_A, x_B, x_C) &= \mathsf{P}(x_A)\mathsf{P}(x_B|x_A)\mathsf{P}(x_C|x_B) \\
\mathsf{P}(x_A, x_B) &= \mathsf{P}(x_A)\mathsf{P}(x_B|x_A)
\end{aligned}$$

and hence

$$\begin{aligned} & \mathsf{P}(x_C|x_A, x_B) \\ = & \quad \text{definition of conditional probability} \\ & \frac{\mathsf{P}(x_A, x_B, x_C)}{\mathsf{P}(x_A, x_B)} \\ = & \quad \text{just derived} \\ & \mathsf{P}(x_C|x_B) \end{aligned}$$

so X_C is conditionally independent of X_A given X_B. As it turns out, these necessary independencies can be described by a property of the graph called *d-separation*[48]: a conditional independence of X_A and X_B given a set of variables $X_{\bar{E}}$ follows from the form of the factorization iff the nodes A and B are d-separated by the set of nodes $\bar{E}$. This d-separation is defined as follows: A and B are d-separated by $\bar{E}$ if there is no d-connecting path between A and B given $\bar{E}$. An undirected path between A and B, with intermediate nodes $N_1, N_2, \ldots, N_m$ is *d-connecting* given $\bar{E}$ if for every N_i holds:

- if, from the two arrows connecting N_i to the rest of the path, at least one points away from N_i, then $N_i \notin \bar{E}$;
- if both arrows point towards N_i, then N_i or a descendant of N_i is in $\bar{E}$.

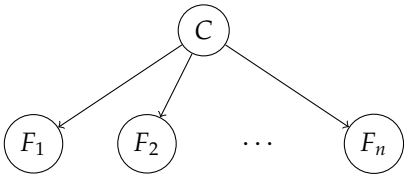
In practice, it often suffices to know that:

- a node is conditionally independent from its non-descendants given its parents, and
- a node is conditionally independent from the nodes outside of its *Markov blanket* given the nodes in its Markov blanket. A node's Markov blanket consists of its parents, its children, and the parents of its children.

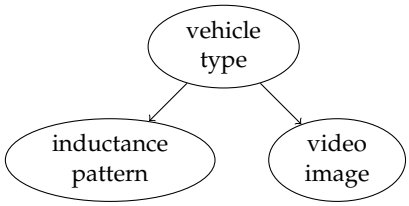
These conditional independencies are what makes a Bayesian network meaningful as a probabilistic model *even if only the graph is defined*.

2.3 Probabilistic models for sensor data

In this section, we discuss two desirable properties of probabilistic models for sensor data: the conditional independence of different sensor variables, and the ability to represent changes. From now on, we drop the distinction between the nodes in the graph and the probabilistic variables, because the readability of probabilistic expressions suffers if all variables start with X. We will use the distinction again in chapter 5, where we make a formal connection between probabilistic variables and attributes in relation schemas.



(a) Generic naive Bayes classifier.



(b) Naive Bayes classifier for road vehicles, with two sensors: an induction loop detector and a video camera.

Figure 2.1: A simple type of Bayesian network: the naive Bayes classifier.

2.3.1 Naive Bayes classifiers for sensor data

A particularly simple type of model is the so-called *naive Bayes classifier*. Its Bayesian network consists of one unobservable 'class variable' C, several observable 'feature variables' F_i, and arrows $C \rightarrow F_i$ from the target variable to every feature variable. The term *classifier* refers to the inference task associated with this model: given some observed features $\bar{f}$, determine the most likely class c, i.e. $\arg\max_c \mathsf{P}(C{=}c|\bar{F}{=}\bar{f})$.

The graph structure implies that, given a value of the class variable, all feature variables are conditionally independent of each other. Even in cases when this independence assumption is clearly invalid, the model often has surprisingly good accuracy[21], and it is popular due to its inference speed and ease of learning (see also sections 2.5 and 2.6 on inference and learning).

The model can be applied to a sensor environment where multiple sensors are used to observe the same phenomenon (but possibly different properties of it). The class variable corresponds to that phenomenon, and feature variable F_i to the input from sensor i. For example, one may be interested in the type of vehicles (car, truck, bus, motorcycle) passing at a certain point of a highway[37]. This is sensed using two different sensors: an induction loop and a video camera. The probabilistic model contains a *sensor model* $\mathsf{P}(f_i|c)$ for both sensors, specifying the probability for vehicle type c to cause sensor observation f_i. (Additionally, the model contains the so-called *class prior* $\mathsf{P}(c)$ specifying the relative frequency of vehicle type c.)

The sensors models contain uncertainty: a bus may confuse the video camera, and (perhaps under certain lighting conditions) produce an image i that is more likely to result from a truck than from a bus (i.e. $\mathsf{P}(F_2{=}i|C{=}\text{truck}) > \mathsf{P}(F_2{=}i|C{=}\text{bus})$. The conditional independence assumption now states that $\mathsf{P}(F_1{=}p|C{=}\text{bus}, F_2{=}i) = \mathsf{P}(F_1{=}p|C{=}\text{bus})$: the fact that a bus produces a confusing image i does not influence the probability that it produces a confusing inductance pattern p. This seems a fair assumption, because the source of uncertainty for the camera (different lighting conditions) seems unrelated to the source of uncertainty for the induction loop

(say, different materials).

A major practical advantage of using the naive Bayes classifier for sensor data is that adding, removing or changing a sensor (or the associated software) is done without touching the models for the other sensors; the naive Bayes classifier meets our *modularity requirement* (section 1.1). For example, the camera feature extraction software is changed so the variable F_2 gets a new meaning (let us model this by a different variable F_2') and a new domain with different values. Then a new model $\mathsf{P}(f_2'|c)$ for the camera is needed, but the induction loop model $\mathsf{P}(f_1|c)$ does not have to be altered. If it were dependent on F_2 (the graph has an arrow $F_2 \to F_1$), the model $\mathsf{P}(f_1|c, f_2)$ would have to be modified to $\mathsf{P}(f_1|c, f_2')$ as well.

2.3.2 Formal modularity of Bayesian networks

We now formalize the following statement: *Adding a sensor does not change the probabilistic model over the existing variables.* When we add a node to a Bayesian network, and only add edges from the existing network to the new node, the model over the existing variables does not change. First, note that adding an edge the other way around, i.e. from the new node to an existing node, actually means that the cpd on the existing node has to be changed; it gets an extra dimension, because it now depends on one more variable. When we only add edges from existing variables, this does not happen.

Say that the network consists of $\{V_1, \ldots, V_n\}$ and defines a joint probability

$$\mathsf{P}(v_{1..n}) = c_{V_1}(\ldots)c_{V_2}(\ldots)\cdots c_{V_n}(\ldots)$$

Then, a new node V_{n+1} with cpd $c_{V_{n+1}}$ is added; the new model defines a joint probability $\mathsf{P}'(v_{1..n+1})$. In this new model, the set $\{V_1, \ldots, V_n\}$ is still closed under *Parents* (because none of them has V_{n+1} as parent), so by (Joint-Subset-BN)[p. 20],

$$\mathsf{P}'(v_{1..n}) = c_{V_1}(\ldots)c_{V_2}(\ldots)\cdots c_{V_n}(\ldots)$$

So, the old and new model are consistent with respect to the probability distribution over the old variables, and hence with all predicates over these variables. By a similar argument, removal or modification of a variable V_i and its cpd has no effect on the joint probability over the other variables, *as long as V_i has no children*. (In the naive Bayes classifier, all sensor variables satisfy this requirement.)

2.3.3 Dynamic Bayesian networks

The naive Bayes classifier can be applied in a streaming setting on a snapshot base; in that case, we do not define any temporal relations between the features or class at time t and those at $t + 1$. However, such temporal relations are often present; e.g. the location of an object at $t + 1$ is probabilistically dependent on its location at t (and vice versa).

Bayesian networks can model a variable X whose value *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[17, 41, 47]. Usually, the term implies some further restrictions:

- for each t, the same variables exist; let us refer to them as $\{V_t^1, \ldots, V_t^n\}$
- the parents of each V_t^j are among the variables at t and $t-1$, and are the same for each t
- the cpd $c_{V_t^j}$ is the same for each t

The variables at $t = 0$ form an exception, as they cannot have any parents in $t-1$. Therefore, for $t = 0$ different variables and cpds are allowed. The rest of the model consists of identical 'slices': it is said to be (time-)*homogeneous*.

Examples are given in the next section: the HMM and MSHMM models are both dynamic Bayesian networks that satisfy the above restrictions. More models are defined in chapter 3, where we widen our definition of a dynamic Bayesian network somewhat.

In anticipation of chapter 6, we derive a property of the product of the cpds in slice t, namely:

$$\prod_{j=1..n} c_{V_t^j}(\ldots) = \mathsf{P}(v_t^{1..n} | \bar{\imath}_{t-1}) \qquad \textbf{(Cpd-Slice)}$$

It equals the conditional probability of the variables V_t^j given their parents in slice $t-1$. We call this set of parents the *interface* between $t-1$ and t, which we write $\bar{I}_{t-1}$. Formally, we define, for each t:

$$\bar{I}_t \stackrel{\text{def}}{=} \bar{V}_t \cap \bigcup_j Parents(V_{t+1}^j)$$

In this definition, we follow Murphy[47], except that he calls this the *forward interface* of slice t. Note that $\bar{I}_{t-1}$ d-separates $\{V_t^1, \ldots, V_t^n\}$ from the other variables in slices $t-1$ and earlier. For the derivation of (Cpd-Slice), we start with an auxiliary calculation:

$$\begin{aligned}
& \mathsf{P}(v_{0..t}^{1..n}) \\
= & \quad \text{by (Joint-Subset-BN)}^{\text{p. 20}} \\
& \prod_{\substack{j=1..n \\ k=0..t}} c_{V_t^j}(\ldots) \\
= & \\
& \left(\prod_{\substack{j=1..n \\ k=0..t-1}} c_{V_t^j}(\ldots) \right) \prod_{j=1..n} c_{V_t^j}(\ldots) \\
= & \quad \text{by (Joint-Subset-BN)} \\
& \mathsf{P}(v_{0..t-1}^{1..n}) \prod_{j=1..n} c_{V_t^j}(\ldots)
\end{aligned}$$

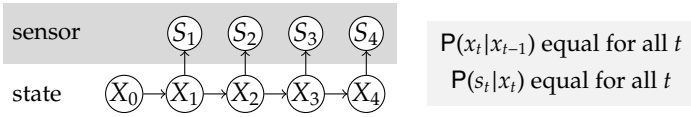


Figure 2.2: Hidden Markov Model (HMM) with $T = 4$

The property is now derived as follows.

$$
\begin{aligned}
& \prod_{j=1..n} c_{V_t^j}(\ldots) \\
= & \quad \text{just derived} \\
& \frac{\mathsf{P}(v_{0..t}^{1..n})}{\mathsf{P}(v_{0..t-1}^{1..n})} \\
= & \\
& \frac{\mathsf{P}(v_t^{1..n}, v_{0..t-1}^{1..n})}{\mathsf{P}(v_{0..t-1}^{1..n})} \\
= & \quad \text{definition of conditional probability} \\
& \mathsf{P}(v_t^{1..n} | v_{0..t-1}^{1..n}) \\
= & \quad \text{by d-separation} \\
& \mathsf{P}(v_t^{1..n} | \bar{\imath}_{t-1})
\end{aligned}
$$

A similar, more general result for a group of variables in a Bayesian network is derived in section 3.3.

2.4 Concrete models

2.4.1 Hidden Markov Model

One of the most simple dynamic Bayesian networks that can be applied to sensor data is the Hidden Markov Model (HMM), shown in figure 2.2. For each discrete point in time $t \in \{0, \ldots, T\}$, it contains a state variable X_t and a sensor variable S_t representing an (imperfect/partial) observation of the state. The state variables are linked together in a *Markov chain*, which means that X_{t+1} is conditionally independent of $X_0, \ldots, X_{t-1}$ given X_t (as can be verified from the d-separation in the graph). As for the sensor variables, S_t is conditionally independent of all other variables given X_t.

The so-called state transition probabilities $\mathsf{P}(x_t|x_{t-1})$ do not depend on t; together they make up the *transition model* of the HMM. Only X_0 is different; the probability distribution $\mathsf{P}(x_0)$ is called the *state prior*. The observation probabilities $\mathsf{P}(s_t|x_t)$ make up the *sensor model*, which is also the same for each t.

Hidden Markov Models are popular in speech processing[49]. Simply put, the S variables represent a sequence of observed audio frames, and the X variables

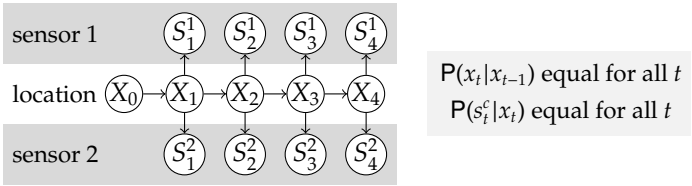


Figure 2.3: Multi-sensor Hidden Markov Model (MSHMM) with $T = 4$, $K = 2$

represent the sequence of *phones* (the sounds that form the units of speech). The task of the speech processing system consists of finding the most probable X sequence.

2.4.2 Multi-sensor HMM

In order to accommodate multiple (say K) sensors in the HMM, one can treat them as one sensor and join their observations $s_t^1, \ldots, s_t^K$ into one value $s_t = (s_t^1, \ldots, s_t^K)$. However, this causes the size of $\text{dom}(S_t)$ and hence the number of values in the distribution $\mathsf{P}(s_t|x_t)$ to grow (exponentially) large as more sensors are added, which is a problem for specifying, learning and storing the model. Furthermore, this model does not satisfy the modularity requirement in the way the naive Bayes classifier does (see section 2.3.1).

As a solution to these problems, we introduce the Multi-sensor Hidden Markov Model (MSHMM), which combines the HMM and the naive Bayes classifier. Instead of one, it contains K sensor variables S_t^c ($1 \leq c \leq K$) per time point t, all conditionally independent given X_t: see figure 2.3.

2.4.3 Localization setup

Using the MSHMM, we can model the localization scenario from section 1.1, which we repeat and formalize here (in chapter 3, we will further refine the scenario to allow for non-synchronized observations). At K fixed positions in a building, a Bluetooth transceiver ('scanner') is installed, and performs regular scans in order to track the location of a mobile device (note: we restrict ourselves to a single device for simplicity), which 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 time T, we want to calculate $\mathsf{P}(x_t|s_{1..T}^{1..K})$: the probability distribution over the location at time t (with $1 \leq t \leq T$) based on the received scan results during the time span. As we will explain in section 2.5, this probabilistic 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).

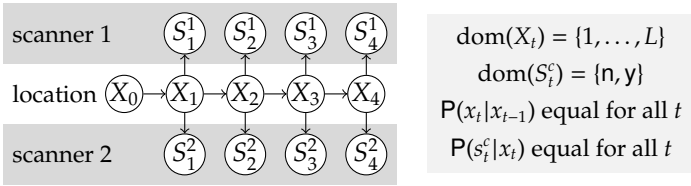


Figure 2.4: MSHMM for localization setup, with $T = 4$, $K = 2$

In the MSHMM, each sensor c corresponds to a scanner; the variable S_t^c represents the result of the scan at time t and can take the values n (device not detected) and y (device detected). The state 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 transition model $\mathsf{P}(x_t|x_{t-1})$ consists of the probabilities to go from one location to another in one time step. 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 figure 2.5, 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 figure 2.6, 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 $\mathsf{P}(x_t|x_{t-1}) > 0$.

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 figure 2.5, the reach of sensor 3 is shaded in gray. Hence, there is a bounded number of x_t values for which $\mathsf{P}(S_t^c = \mathsf{y}|x_t) > 0$; for sensor 3, these 9 probabilities are shown in the gray squares. However, $\mathsf{P}(S_t^c = \mathsf{n}|x_t)$ 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 figure 2.7).

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 $\mathsf{P}(S_t^c = \mathsf{y}|x_t) > 0$. This bound depends on the sensor density and detection reach, and is 3 in our example. In section 6.2, we discuss how this sparsity can be exploited for saving computational resources.

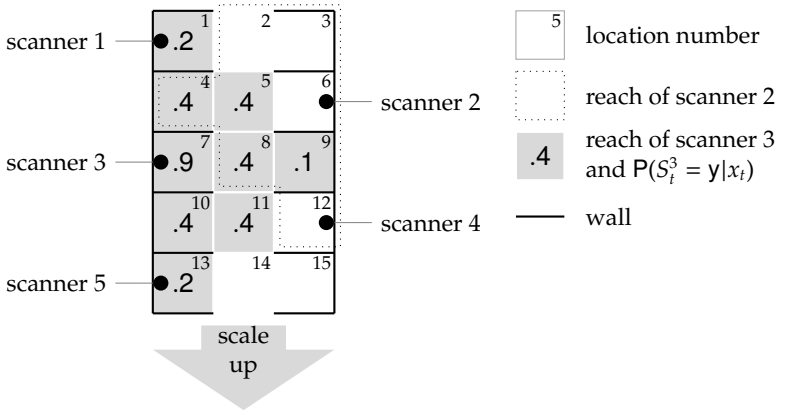


Figure 2.5: Example (partial) floor plan for the localization model. The numbered squares are the $L = 15$ discrete values that a location variable X_t can take. At $K = 5$ positions, a scanner is installed. In one time step, it is possible to move to an adjacent location, but not through a wall; this is encoded in the transition model $\mathsf{P}(x_t|x_{t-1})$. For scanner 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 scanner has a reach of 9 locations, as is shown for scanners 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 2.6: Partial transition model corresponding to the floor plan in figure 2.5. 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	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

Figure 2.7: Sensor model for sensor 3 in figure 2.5. Columns sum to 1. This model encodes the limited reach of each sensor.

2.5 Common inference queries

When a probabilistic model of a sensor setup has been established, it can be put to work: using the data obtained from the sensors, it should be able to answer queries with regard to variables in the model. Although these answers are always probabilistic, they satisfy the requirements laid down in section 1.1. Firstly, they are *consistent*: information from different sensors that would be conflicting in a deterministic model can be suitably processed into the probability distribution. Secondly, they are complete: no matter how little information is actually obtained from the sensors, a probability distribution can always be produced. Queries in a probabilistic model are formulated as *conditional probabilities*: given the observed readings $\bar{s}$ for sensor variables $\bar{S}$, what is the conditional distribution $\mathsf{P}(Q{=}q|\bar{S}{=}\bar{s})$ over a certain variable Q?

For example, in the localization MSHMM with $K = 10$ and $T = 100$, one could query X_{42} (the location at time 42), given the 1000 sensor readings $\bar{s} = \left\{ s_t^c \;\middle|\; 1 \le c \le K, 1 \le t \le T \right\}$; this query corresponds to $\mathsf{P}(x_{42}|\bar{s})$. Note that, through the transition model, this probability distribution is affected by sensor readings not only with $t = 42$, but also by readings from earlier and later times: a detection at $t = 45$ by a scanner near location 22 increases the odds that $X_{42} = 22$, because the transition model dictates that the location cannot change very much within three time steps (i.e. the probability of a large change is zero or very low).

How to calculate the answer to an inference query will be discussed extensively in chapters 5 and 6. Here, we give an overview of some common queries for dynamic Bayesian networks, or rather, common inference tasks consisting of *sets* of queries. This overview is taken from [51] and applied to the MSHMM.

The first task is called *smoothing*. Like the query above, it considers all sensor readings up to a certain point T; however, the goal is to find the probability distribution over the location for every time t instead of for one specific point. Hence, the set of queries is $\{\, \mathsf{P}(x_t|\bar{s}) \mid 1 \le t \le T \,\}$; note that the intended result is a set of distributions, and formally it is clearer to write $\{\, \lambda x_t.\, \mathsf{P}(x_t|\bar{s}) \mid 1 \le t \le T \,\}$. The results can, for example, be used to calculate the most likely location at each point in time: $\left\{ \arg\max_{x_t \in \mathrm{dom}(X_t)} \mathsf{P}(x_t|\bar{s}) \;\middle|\; 1 \le t \le T \right\}$.

A different but related task is to find the probability distribution over the whole sequence of locations, i.e. $\mathsf{P}(x_1, \ldots, x_T|\bar{s})$. It is usually infeasible to store this distribution in a multi-dimensional array, as the number of entries is exponential in T. However, it often suffices to find just the *most likely sequence*, i.e. $\arg\max_{(x_1,\ldots,x_T)} \mathsf{P}(x_1, \ldots, x_T|\bar{s})$, which can be done without calculating the whole array, using the well-known Viterbi algorithm[26]. Note: in general, this most likely sequence of locations differs from the sequence of most likely locations described above.

The above two tasks can only be performed after all sensor readings up to T have been collected; until now we have silently assumed that T is the fixed length of the model. However, this assumption does not make much sense in situations

where sensors are continuously collecting new data, and applications want to be kept up to date as new data arrives. In these situations, T can be considered as infinite or continuously growing. A common task is called *filtering*: continuously calculating $\mathsf{P}(x_t|\bar{s}_1,\ldots,\bar{s}_t)$ for the *current* time t (where $\bar{s}_u \stackrel{\text{def}}{=} s_u^1,\ldots,s_u^K$).

As we explained, new sensor readings can improve the accuracy of earlier location estimates; this means that, if the application can tolerate some latency, it is better to wait some time after t before giving out an X_t estimate. If this delay is fixed (say, δ), this is called *fixed-lag smoothing*: the continuous inference query is then $\mathsf{P}(x_t|\bar{s}_1,\ldots,\bar{s}_{t+\delta})$. The opposite is also possible; $\mathsf{P}(x_t|\bar{s}_1,\ldots,\bar{s}_{t-\delta})$ would be a fixed-lag *prediction*.

A reason for a system to group a set of queries into a specific inference task instead of considering them all by themselves is that it is often possible to re-use (intermediate) results of a query in order to calculate the answer to the following queries. In fact, this is the only way that the continuous queries, whose answer depends in principle on the whole history of sensor data, can be made tractable. For example, the filtering query $\mathsf{P}(x_t|\bar{s}_1,\ldots,\bar{s}_t)$ can be calculated using only the answer to the previous query $\mathsf{P}(x_{t-1}|\bar{s}_1,\ldots,\bar{s}_{t-1})$ and the new sensor readings $\bar{s}_t$.

Until now, we have not explicitly mentioned what to do with missing sensor readings. The answer is almost trivial: the corresponding variables are simply left out of the conditional probabilities. As we mentioned at the start of this section, probabilistic models can always give an answer, no matter how few sensor information is used. Of course, this makes the quality of the answer deteriorate: the less input, the more the probability mass 'spreads out' over the alternatives.

2.6 Evaluation and learning of sensor data processing

This thesis focuses on performing *exact inference queries* on a Bayesian network with known parameters; i.e. the answer to a query corresponds exactly to the conditional probability distribution that is asked for. Evaluation and optimization of these systems is expressed in terms of *efficiency*: the amount of consumed resources for a query (collection). In the current section, we briefly place this into a broader context.

Firstly, *approximate* inference methods are widely used when exact inference is unacceptably inefficient. Mostly, these methods are 'probabilistic' themselves: to represent an intermediate probability distribution in the calculation, they use a collection of (weighted) random samples instead of a large table with all the exact probabilities. Sensor data processing systems that use approximate inference are subject to an additional dimension of evaluation: next to efficiency, *accuracy* has to be taken into account as well.

Accuracy, the extent to which the given answer to a query resembles the 'true' answer, can be defined in many ways; we will give some examples with the

localization query $\mathsf{P}(x_{42}|\bar{s})$. For now, we define the 'true answer' as the probability distribution obtained by exact inference, and write $\mathsf{P}'(x_{42}|\bar{s})$ for the approximate answer. Some possible accuracy measures are:

- The similarity between the two distributions $\mathsf{P}'(x_{42}|\bar{s})$ and $\mathsf{P}(x_{42}|\bar{s})$; a common measure for this is the Kullback-Leibler divergence[44].
- Whether the approximation yields the same most probable value: $\arg\max_{x_{42}} \mathsf{P}'(x_{42}|\bar{s}) = \arg\max_{x_{42}} \mathsf{P}(x_{42}|\bar{s})$
- The difference in probability of the most probable value.
- The similarity between the ranked lists of most probable values; a well-known measure for this is Kendall's tau[39].

It is also possible to look at system evaluation in a broader context, and evaluate both the inference system and the probabilistic model at the same time. In this case, we compare the given answer to the *ground truth*; in the localization example, this would be the *actual location* at $t = 42$. Of course, this kind of evaluation presumes that there is a well-defined ground truth for the query variable (this would be harder for probabilistic variables such as 'current terrorism threat level'), and that it is known. Say that the ground truth is g; some possible evaluation metrics are:

- Whether the most probable answer equals the ground truth: $g = \arg\max_{x_{42}} \mathsf{P}'(x_{42}|\bar{s})$
- In case a distance measure d on the locations is defined, the expected error $\sum_{x_{42}} \mathsf{P}'(x_{42}|\bar{s})d(g, x_{42})$.

Above methods evaluate the system for one dataset $\bar{s}$ and one query X_{42}; for real evaluations to be useful, representative collections of datasets and queries are needed, over which the above metrics can be averaged. In the case of *binary* query variables (*Is Sander currently in room 3090?*), the evaluation metrics of *precision* and *recall* can also be used, derived from the numbers of true/false positives/negatives among the answers (for a set of queries with different evidence). A positive answer is given when the probability for *true* exceeds a certain cutoff point. Similar metrics are *sensitivity/specificity* and the *ROC curve*[25], which is obtained by plotting sensitivity against specificity while varying the cutoff point.

If the parameters of a Bayesian network are not available, they can be *learned* from training data. The goal is to maximize the accuracy of the system (compared to ground truth); in general, machine learning principles dictate that this accuracy is measured on a different data set to avoid *overfitting* the model to the training data. However, this is not always done. A common practice is to search for the *maximum likelihood* parameters: those that maximize the probability of the data.

For the MSHMM, finding the maximum likelihood parameters is especially easy in the case that, next to sequences of sensor data, the ground truth (sequences of locations) is also known; i.e. we have a set of instances of the model where all

the variables are observed. This means that for each variable of the model, a conditional frequency table can be constructed (where the tables for all time-instances of a variable are aggregated into one); the maximum likelihood cpd for this variable then corresponds to that table.

Finding maximum likelihood parameters without ground truth is also possible, but much more time consuming and not always successful. The canonical approach here is called *expectation maximization*[19] or simply EM. It is an iterative process; from an arbitrary set of starting cpds, it uses inference to derive the expected values for the unobserved variables, then alters the cpds for maximum likelihood, and repeats these steps until convergence.

For more information about learning probabilistic models, we refer to machine learning textbooks[11, 46].

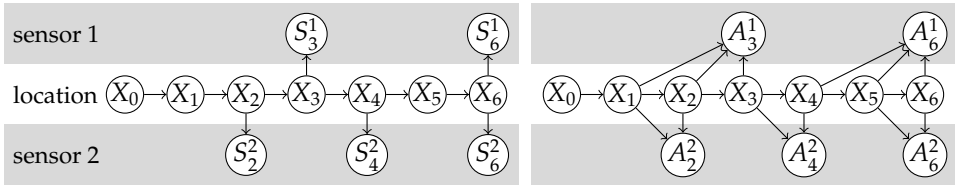
Chapter 3

Complex models

There is one large disadvantage about the Multi-sensor Hidden Markov Model (MSHMM) we introduced in section 2.4.2. Each point in time t defines, besides state variable X_t, a variable S_t^c for each sensor, and therefore it *synchronizes* the readings from the different sensors with each other (and with the state variable). However, to satisfy the modularity goal in section 1.1, it would be useful to be able to:

- integrate data from sensors that have different observation frequencies
- integrate data from sensors that have the same observation frequency but a different *phase* (i.e. observations are not synchronized)
- incorporate state variables for which transitions can happen with a higher frequency than the sensor's observation frequency (apart from enabling more accurate transition models, this is useful as an interface to other parts of a probabilistic model that define a higher frequency)

In this chapter, we introduce probabilistic sensor models that fulfill these requirements. In section 3.1 we define a very general model, which we refine (in order to keep inference tractable) in section 3.4. In order to do this, we introduce some theory about deterministic functions (section 3.2) and how Bayesian networks can be refined in general (section 3.3). Although this theory should not come as a surprise to AI researchers, it is not often explicitly formulated, and we deem it important for a working knowledge of Bayesian networks. The chapter concludes with a discussion of how complex queries can be represented in a Bayesian network (section 3.5), which also depends on these theoretical concepts.



(a) Simple integration model **(b)** MSHMM-AO

Figure 3.1: Two models for the integration of data from sensors with different frequencies. Sensor 1 reports an observation every 3 time steps, and sensor 2 reports every 2 time steps.

3.1 MSHMM-AO

For illustration purposes, we assume the localization scenario outlined in section 2.4.3, only with two scanners that have a different frequency. Scanner 1 reports an observation every 3 time steps, while scanner 2 reports an observation every 2 steps. In principle, an easy solution for the integration of the data from these scanners would be to use the model in figure 3.1a, which is essentially the MSHMM model with observation variables left out at the times when there is no observation. (Note: as argued in section 2.3.2, the probability distribution over the remaining variables would stay the same if the model *would* include these left-out variables. Hence, inference over the model in figure 3.1a is equivalent to inference over the MSHMM with no evidence on the left-out variables.)

However, this model does not really apply to the scenario with Bluetooth scans; in the model, the observation S^1_3 is only directly "caused" by X_3, the location at time 3. If $S^1_3 = \mathsf{y}$, the probability of locations around scanner 1 at $t = 3$ will increase given this evidence; it will as well for $t = 2$, but only indirectly through the transition model, and not to the same extent. This is because the model does not capture the fact that the result of the scan S^1_3 is caused by locations in the *interval* $\{1, 2, 3\}$.

The most simple and general model that does accommodate observations of a sequence of states is shown in figure 3.1b; we call it the MSHMM-AO, for *aggregate observations*, and denote these observation variables with A^c_t. However, this model has its own drawback. Note that the cpd $\mathsf{P}(a^1_3 | x_1, x_2, x_3)$ has four dimensions, and in the localization scenario, where $|\mathrm{dom}(A^c_t)| = 2$ and $|\mathrm{dom}(X_t)| = L$, consists of $2L^3$ probabilities.

Hence, the size of the cpd grows exponentially w.r.t. the number of time points in the interval. As the cost of learning and inference are directly related to this size (see chapter 6), the MSHMM-AO model is only usable for very small intervals. This problem can be solved by imposing a so-called Noisy-OR *structure* on the cpd; in order to understand this structure, we first introduce some theory about deterministic functions.

3.2 Deterministic functions in a Bayesian network

A nice property of Bayesian networks that we have not discussed yet is that, alongside probabilistic relationships between variables, it can very easily accommodate conventional *functions*. For example, we can introduce a probabilistic variable Y whose value y is always fully determined by the values of the variables X_1 and X_2: $y = f(x_1, x_2)$. Formally, in terms of the sample space (see section 2.1.1), this variable is defined:

$$Y(\omega) \stackrel{\text{def}}{=} f(X_1(\omega), X_2(\omega))$$

This definition leads to a cpd $\mathsf{P}(Y{=}y|X_1{=}x_1, X_2{=}x_2)$ which is a so-called *degenerate distribution*: its value can only be 1 or 0. In particular, it is 1 when $y = f(x_1, x_2)$ and 0 otherwise. This necessarily follows from the definition of Y:

$$\begin{aligned}
&\mathsf{P}(Y{=}y|X_1{=}x_1, X_2{=}x_2) \\
= &\quad \text{definition of conditional probability} \\
&\frac{\mathsf{P}(Y{=}y, X_1{=}x_1, X_2{=}x_2)}{\mathsf{P}(X_1{=}x_1, X_2{=}x_2)} \\
= &\quad \text{definition of } \mathsf{P}(A{=}a) \\
&\frac{\mathsf{P}(\{\,\omega \mid Y(\omega){=}y, X_1(\omega){=}x_1, X_2(\omega){=}x_2\,\})}{\mathsf{P}(\{\,\omega \mid X_1(\omega){=}x_1, X_2(\omega){=}x_2\,\})} \\
= &\quad \text{exhaustive case enumeration; see below} \\
&\begin{cases} \dfrac{\mathsf{P}(\{\,\omega \mid X_1(\omega){=}x_1, X_2(\omega){=}x_2\,\})}{\mathsf{P}(\{\,\omega \mid X_1(\omega){=}x_1, X_2(\omega){=}x_2\,\})} & \text{if } y = f(x_1, x_2) \\ \dfrac{\mathsf{P}(\{\,\omega \mid \mathit{false}\,\})}{\mathsf{P}(\{\,\omega \mid X_1(\omega){=}x_1, X_2(\omega){=}x_2\,\})} & \text{if } y \neq f(x_1, x_2) \end{cases} \\
= &\quad \text{(second case) by axiom } \mathsf{P}(\emptyset) = 0 \text{ from (Prob-Space)} \\
&\begin{cases} 1 & \text{if } y = f(x_1, x_2) \\ 0 & \text{if } y \neq f(x_1, x_2) \end{cases}
\end{aligned}$$

In the first case of the enumeration, the proposition $Y(\omega){=}y$ follows from $X_1(\omega){=}x_1$, $X_2(\omega){=}x_2$, the definition of Y and $y{=}f(x_1, x_2)$; hence, it can be left out of the set comprehension $\{\,\omega \mid \ldots\,\}$ above the fraction bar. In the second case, due to $y{\neq}f(x_1, x_2)$ and the definition of Y, the three propositions in the set comprehension can never be satisfied and are equivalent to *false*.

To write down this cpd succinctly, we can use the *Iverson bracket*[42] which is defined as follows (we use angled brackets instead of square ones to avoid confusion with our other notation):

$$\langle \mathit{true} \rangle \stackrel{\text{def}}{=} 1$$
$$\langle \mathit{false} \rangle \stackrel{\text{def}}{=} 0$$

The above cpd is then given by

$$\mathsf{P}(Y{=}y|X_1{=}x_1, X_2{=}x_2) = \langle y = f(x_1, x_2)\rangle$$

Moreover, by the same reasoning as above, it also holds that

$$\mathsf{P}(Y{=}y|X_1{=}x_1, X_2{=}x_2, V{=}v) = \langle y = f(x_1, x_2)\rangle$$

for any variable V and value v; in other words, Y is conditionally independent of V given X_1 and X_2. Again, the caveat that $\mathsf{P}(\ldots|\theta)$ is undefined if $\mathsf{P}(\theta){=}0$ applies.

In the above, we assumed an explicit definition of Y in terms of the sample space; we will now show that the same effect can be obtained when Y is a variable in a Bayesian network with parents X_1 and X_2 and the cpd defined above:

$$c_Y(y, x_1, x_2) \stackrel{\text{def}}{=} \langle y = f(x_1, x_2)\rangle$$

We assume that the network, besides these three variables, consists of other variables $\bar{W}$; we will use the shorthand notation for probabilistic events again, e.g. $\mathsf{P}(\bar{w})$ for $\mathsf{P}(\bar{W}{=}\bar{w})$.

If we use the implicit definition of the probability space from section 2.1.2, the sample space consists of all tuples of the form $\omega = (y, x_1, x_2, \bar{w})$; also those for which $y \neq f(x_1, x_2)$. In this respect, the situation is a little bit different than before, because for these tuples $Y(\omega) \neq f(X_1(\omega), X_2(\omega))$; however, this does not make any difference in practice because $\mathsf{P}(\{\omega\}) = 0$. We can see this from the joint probability over all variables:

$$\begin{aligned}
&\mathsf{P}(y, x_1, x_2, \bar{w}) \\
= &\quad \text{by (Joint-Subset-BN)}^{\text{p. 20}} \\
&c_Y(x_1, x_2)c_{X_1}(\ldots)c_{X_2}(\ldots)\textstyle\prod_{W\in\bar{W}} c_W(\ldots) \\
= &\quad \text{definition of } c_Y \\
&\langle y = f(x_1, x_2)\rangle c_{X_1}(\ldots)c_{X_2}(\ldots)\textstyle\prod_{W\in\bar{W}} c_W(\ldots)
\end{aligned}$$

For the tuples in question, the first factor yields 0. As the probability space is only 'accessed' through events defined in terms of the variables, and never directly through the members of the sample space, one is never exposed to the fact that this zero-probability-event actually consists of $\{\omega\}$ and is not $\emptyset$ like above.

It is interesting to consider the joint distribution over all variables except Y:

$$\begin{aligned}
&\mathsf{P}(x_1, x_2, \bar{w}) \\
= &\quad \text{by (Marginalize)}^{\text{p. 15}} \\
&\textstyle\sum_y \mathsf{P}(y, x_1, x_2, \bar{w}) \\
= &\quad \text{see joint probability distribution derived above} \\
&\textstyle\sum_y \langle y = f(x_1, x_2)\rangle c_{X_1}(\ldots)c_{X_2}(\ldots)\prod_{W\in\bar{W}} c_W(\ldots) \\
= &\quad \text{there is only one nonzero term: the one for which } y = f(x_1, x_2) \\
&c_{X_1}(\ldots)c_{X_2}(\ldots)\textstyle\prod_{W\in\bar{W}} c_W(\ldots)[f(x_1, x_2)/y]
\end{aligned}$$

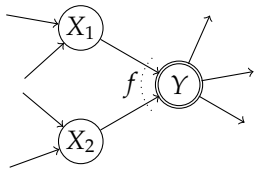


Figure 3.2: Part of a Bayesian network with *deterministic* variable Y. Its value is functionally determined (through f) by the values of X_1 and X_2.

Note the *substitution* of $f(x_1, x_2)$ for y in the c_W factors (the c_{X_i} factors cannot contain y—that would mean that Y is a parent of X_i, a cycle in the graph).

We use this to derive:

$$
\begin{aligned}
& \mathsf{P}(y, x_1, x_2, \bar{w}) \\
= \quad & \text{as above} \\
& \langle y = f(x_1, x_2)\rangle c_{X_1}(\ldots) c_{X_2}(\ldots) \textstyle\prod_{W \in \bar{W}} c_W(\ldots) \\
= \quad & \text{exhaustive case enumeration} \\
& \begin{cases} c_{X_1}(\ldots) c_{X_2}(\ldots) \prod_{W \in \bar{W}} c_W(\ldots)[f(x_1, x_2)/y] & \text{if } y = f(x_1, x_2) \\ 0 & \text{if } y \neq f(x_1, x_2) \end{cases} \\
= \quad & \text{joint distribution over all variables except } Y \text{ (derived above)} \\
& \begin{cases} \mathsf{P}(x_1, x_2, \bar{w}) & \text{if } y = f(x_1, x_2) \\ 0 & \text{if } y \neq f(x_1, x_2) \end{cases} \\
= \quad & \\
& \langle y = f(x_1, x_2)\rangle \mathsf{P}(x_1, x_2, \bar{w})
\end{aligned}
$$

Dividing both ends of this equation by $\mathsf{P}(x_1, x_2, \bar{w})$, we obtain:

$$\mathsf{P}(y | x_1, x_2, \bar{w}) = \langle y = f(x_1, x_2)\rangle$$

So, again, Y is conditionally independent of the $\bar{W}$ variables given X_1 and X_2.

In a Bayesian network, variables like Y are called *deterministic nodes*[54] or *deterministic variables* (although it is actually the *relation* between the variable and its parents that is non-probabilistic). A convention is to draw them with a double border, as in figure 3.2; we also add the name of the function.

Like we have shown above, deterministic variables lead to additional conditional independencies between the variables in a model; the graphical d-separation criterion can be extended to take this into account[28]. Deterministic variables also enable more efficient inference[58]; we show some methods for this in chapter 6.

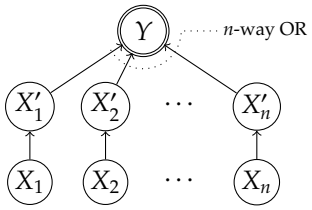


Figure 3.3: Noisy-OR relation as a Bayesian network, where Y is the 'output variable' and X_1 through X_n are the input variables.

3.3 The Noisy-OR distribution

To resolve the scalability difficulties of the MSHMM-AO model, while still being able to relate an observation A_t^c to an interval of state variables $X_{t-n}, \ldots, X_t$, we can impose a certain *structure* on the cpd $\mathsf{P}(a_t^c | x_{t-n}, \ldots, x_t)$ so it can be defined using a number of parameters that is *linear* (or constant; see the next section) w.r.t. the interval length. As we show in chapter 6, this makes inference tractable.

The particular structure we propose is a slight generalization of the Noisy-OR[48]. It defines a probabilistic relation between a binary 'output' variable Y (whose place is taken by A_t^c in our model) and n 'input' variables $X_1, \ldots, X_n$. The cpd is defined as follows:

$$\mathsf{P}(Y{=}\mathit{false} | x_1, \ldots, x_n) = \prod_{1 \le i \le n} q_i(x_i)$$

$$\mathsf{P}(Y{=}\mathit{true} | x_1, \ldots, x_n) = 1 - \prod_{1 \le i \le n} q_i(x_i) \qquad \textbf{(Noisy-OR)}$$

The reason for the name Noisy-OR is as follows. In the original formulation, the X_i variables are also defined to be binary, and $q_i(\mathit{false}) = 1$. So if all inputs are *false*, the output is certainly *false* as well, just like a conventional OR-gate. However, when an input X_i is *true*, this does not mean automatically that the output is also *true*; it has a probability of $q_i(\mathit{true})$ of being 'inhibited' by some external factor (like a noisy channel). This happens independently for each input; therefore, the probability of a *false* output given some *true* inputs is the product of the probabilities of these inputs being inhibited.

The same probabilistic relation can also be defined using a Bayesian network that contains a deterministic OR function, as shown in figure 3.3. The cpd $\mathsf{P}(y | x_1, \ldots, x_n)$ defined by this model is the same as (Noisy-OR) if we define $q_i(x_i) = c_{X_i'}(\mathit{false}, x_i)$. An advantage of the Bayesian network definition is that it 'opens up' the cpd for standard inference optimization techniques; also, it graphically displays the probabilistic independence structure.

A question that naturally arises, then, is: can a variable with a Noisy-OR cpd

in an arbitrary Bayesian network be replaced by the sub-network in figure 3.3? We will show that the answer is affirmative. However, we first observe that:

1. Besides the Noisy-OR relation, the Bayesian network in figure 3.3 formally also contains the probability distributions $\mathsf{P}(X_1), \ldots, \mathsf{P}(X_n)$. This information is not to be copied into the larger network.
2. In the larger network, the X_i variables may have additional parents and children (also among each other), and the Y variable may have children.
3. If the sub-network is copied into the larger network, the joint probability distribution is extended with the auxiliary variables X'_i.

Therefore, we rephrase the question as follows: In a network containing figure 3.3 as a sub-network—where the X_i variables can have additional children and parents, the Y variable can have children, but the X'_i variables cannot have any additional children or parents—is the joint probability over all the variables except $X'_1, \ldots, X'_n$ the same as when we remove these variables, connect Y to $X_1, \ldots, X_n$ and replace its cpd by $\mathsf{P}(y|x_1, \ldots, x_n)$?

We answer this question by considering a more general situation—a Bayesian network with an *arbitrary* subset of variables $\bar{X}$ whose children are in $\bar{X} \cup \{Y\}$. In the Noisy-OR situation above, $\bar{X}$ consists of the X'_i variables. For reasons that will shortly become clear, we divide the rest of the variables into:

- $\bar{W}$, the parents of the variables in $\bar{X}$ that are not in $\bar{X}$ themselves
- $\bar{V}$, the ancestors of the variables in $\bar{W}$ that are not in $\bar{W}$ themselves
- $\bar{Z}$, all other variables

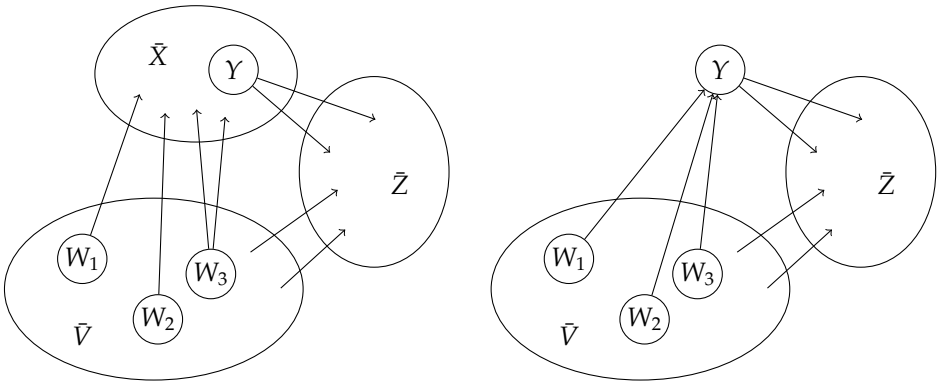
The situation is depicted in figure 3.4. The following holds:

$$
\begin{aligned}
& \textstyle\sum_{\bar{x}} \mathsf{P}(\bar{v}, \bar{w}, \bar{x}, y, \bar{z}) \\
= \quad & \text{by (Joint-Subset-BN)}^{\text{p. 20}} \\
& \textstyle\sum_{\bar{x}} \left(\prod c_V(\ldots)\right)\left(\prod c_W(\ldots)\right)\left(\prod c_X(\ldots)\right) c_Y(\ldots)\left(\prod c_Z(\ldots)\right) \\
= \quad & \text{by (Σ-Distr-R) and (Σ-Distr-L); } \bar{x} \text{ variables only occur in } c_X \text{ and } c_Y \text{ factors} \\
& \left(\prod c_V(\ldots)\right)\left(\prod c_W(\ldots)\right)\left(\textstyle\sum_{\bar{x}} \left(\prod c_X(\ldots)\right) c_Y(\ldots)\right)\left(\prod c_Z(\ldots)\right) \\
= \quad & \text{by definition of } c'_Y \text{ (below)} \\
& \left(\prod c_V(\ldots)\right)\left(\prod c_W(\ldots)\right) c'_Y(\ldots)\left(\prod c_Z(\ldots)\right)
\end{aligned}
$$

Note: to avoid clutter, we removed the subscripts in the products; $\prod c_V(\ldots)$ should be read as $\prod_{V \in \bar{V}} c_V(\ldots)$. For the last step, we define c'_Y as:

$$
c'_Y(y, \ldots) \stackrel{\text{def}}{=} \textstyle\sum_{\bar{x}} \left(\prod c_X(\ldots)\right) c_Y(\ldots)
$$

where the variables on the $c'_Y(y, \ldots)$ dots correspond to $\bar{W}$. Hence, the joint distribution over the non-$\bar{X}$ variables is of a Bayesian network form—assumed



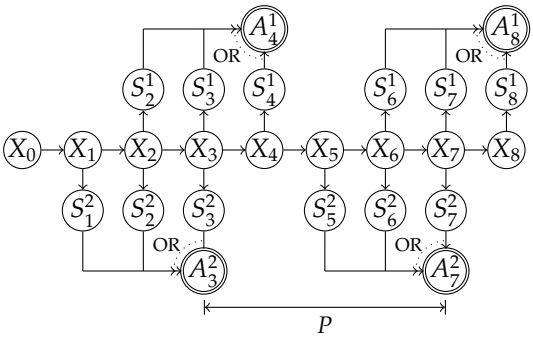
(a) Bayesian network with a sub-network $\bar{X}$. **(b)** Equivalent Bayesian network without $\bar{X}$.

Figure 3.4: A sub-network $\bar{X}$ whose children are in $\bar{X} \cup \{Y\}$ can be eliminated without changing the probability distribution over the other variables. The network is divided into disjoint sets $\bar{V}$, $\bar{W}$, $\bar{X}$, $\{Y\}$ and $\bar{Z}$ (see running text). Arrows within the ellipses can be arbitrary (without cycles); other arrows can only go in the indicated directions.

that c'_Y is of valid cpd form, i.e. $\sum_y c'_Y(y, \ldots) = 1$. And so it is; as expected, it equals:

$$
\begin{aligned}
& \mathsf{P}(y|\bar{w}) \\
= \quad & \text{definition of conditional probability} \\
& \frac{\mathsf{P}(\bar{w}, y)}{\mathsf{P}(\bar{w})} \\
= \quad & \text{by (Marginalize)}^{\text{p. 15}} \\
& \frac{\sum_{\bar{v}} \sum_{\bar{x}} \mathsf{P}(\bar{v}, \bar{w}, \bar{x}, y)}{\sum_{\bar{v}} \mathsf{P}(\bar{v}, \bar{w})} \\
= \quad & \text{by (Joint-Subset-BN)}^{\text{p. 20}} \\
& \frac{\sum_{\bar{v}} \sum_{\bar{x}} \left(\prod c_V(\ldots)\right) \left(\prod c_W(\ldots)\right) \left(\prod c_X(\ldots)\right) c_Y(\ldots)}{\sum_{\bar{v}} \left(\prod c_V(\ldots)\right) \prod c_W(\ldots)} \\
= \quad & \text{by (Σ-Distr-R) and (Σ-Distr-L)}^{\text{p. 19}} \\
& \frac{\left(\sum_{\bar{v}} \left(\prod c_V(\ldots)\right) \prod c_W(\ldots)\right) \sum_{\bar{x}} \left(\prod c_X(\ldots)\right) c_Y(\ldots)}{\sum_{\bar{v}} \left(\prod c_V(\ldots)\right) \prod c_W(\ldots)} \\
= \quad & \\
& \sum_{\bar{x}} \left(\prod c_X(\ldots)\right) c_Y(\ldots) \\
= \quad & \text{by definition of } c'_Y \\
& c'_Y(\ldots)
\end{aligned}
$$

Therefore, the Bayesian network with the $\bar{X}$ sub-network and the c_Y cpd is equivalent to the Bayesian network without the $\bar{X}$ variables and the c'_Y cpd. In the



$\text{dom}(A_t^c) = \{\mathsf{n}, \mathsf{y}\}$
$\text{dom}(S_t^c) = \{\mathsf{n}, \mathsf{y}\}$
$\text{dom}(X_t) = \{1, \ldots, L\}$
$\mathsf{P}(s_t^c | x_t)$ equal for all t
$\mathsf{P}(x_t | x_{t-1})$ equal for all t

Figure 3.5: MSHMM-NOR with $T = 8$, $P = 4$, $K = 2$, $D = 3$, $\theta^1 = 4$, $\theta^2 = 3$. The MSHMM-NOR is periodic with a period of P; all scans have a duration of D but may start at different points in time for each scanner c. The end point for scanner c within the first period of the model is denoted θ^c. Thus, S_t^c only exists if $\theta^c - t \pmod P < D$, and A_t^c only exists if $\theta^c - t \pmod P = 0$.

next section, we replace the A_t^c cpds from the MSHMM-AO model with Noisy-OR sub-networks.

3.4 MSHMM-NOR

In this section, we define the MSHMM-NOR (Multi-sensor HMM with Noisy-OR), a model tailored for the Bluetooth localization scenario (section 2.4.3). Like the MSHMM-AO, it can model unsynchronized interval observations for multiple sensors; however, we now require that the observation frequency for each sensor is the same. We do this because it simplifies optimization (see section 6.4), but it is easy to drop this requirement if needed. Also, we allow for gaps between observation intervals. Within an interval, the probability distribution is Noisy-OR where each discrete time point has the same probability of causing an observation.

This leads to the Bayesian network shown in figure 3.5. Its parameters are:

- The total number of time slices T.
- A transition model $\mathsf{P}(x_t | x_{t-1})$ and state prior $\mathsf{P}(x_0)$, like in the models above; each X_t variable ranges over L locations, and the transition model is equal for each t.
- The number of sensors K.
- For each sensor c ($1 \leq c \leq K$), a sensor model $\mathsf{P}(s_t^c | x_t)$, which is equal for each t. The S_t^c variables are binary and take the values $\{\mathsf{n}, \mathsf{y}\}$ (which play the role of *false* and *true* in the OR function, respectively).

- The period P after which the model repeats itself.
- The duration D of an interval, i.e. the number of time points it includes.
- For each sensor c, the phase θ^c: the last time point of its first interval.

Besides the inference scalability, advantages of the Noisy-OR distribution in the model (with equal $\mathsf{P}(s_t^c|x_t^c)$ for each t) are that it is easy to learn with controlled ground truth X_t data, and that it is easy to transform its time granularity. For example, consider how we can learn the parameter $\mathsf{P}(S_t^c{=}\mathsf{y}|X_t{=}x)$ in the location model, for a certain location x. This is done by keeping the mobile device at the fixed location x for some time, and recording the fraction of detections p_{cx} by sensor c. This fraction is used as the estimate of $\mathsf{P}(A_t^c{=}\mathsf{y}|X_{t-D+1}{=}\ldots{=}X_t{=}x)$. From there, we proceed as follows:

$$
\begin{aligned}
& \mathsf{P}(A_t^c{=}\mathsf{y}|X_{t-D+1}{=}\ldots{=}X_t{=}x) = p_{cx} \\
\equiv \quad & \text{by (Noisy-OR)}^{\text{p. 40}} \\
& 1 - \textstyle\prod_{t-D+1\le i\le t} \mathsf{P}(S_i^c{=}\mathsf{n}|X_i{=}x) = p_{cx} \\
\equiv \quad & \mathsf{P}(S_i^c{=}\mathsf{n}|X_i{=}x) \text{ is equal for all } i \\
& \mathsf{P}(S_t^c{=}\mathsf{n}|X_t{=}x) = \sqrt[D]{1-p_{cx}} \\
\equiv & \\
& \mathsf{P}(S_t^c{=}\mathsf{y}|X_t{=}x) = 1 - \sqrt[D]{1-p_{cx}}
\end{aligned}
$$

Might we want to change the model's time granularity later on, no new learning is needed; the estimate p_{cx} still holds, and in the above formula, the value for D is simply changed.

A final remark about the MSHMM-NOR model is that it is not a dynamic Bayesian network in the formal sense of section 2.3.3; the A_t^c variables have parents in D time slices, and furthermore, not all time slices are identical. The first concern can be remedied by a *sequential decomposition* of the D-way OR variable into D binary OR variables that only have parents in the previous and current time slices; this is explained in section 6.3.2. The second concern is inherent to sensor data models with different frequencies.

3.5 Complex queries as part of the Bayesian network

The inference queries considered in section 2.5 have been of a simple form: *given the evidence collected by the sensors, what is the probability distribution over a variable or set of variables in the model?* We can expand the query possibilities by considering *functions* over probabilistic variables, such as:

- Has the location remained the same within the interval $\{1, 2, 3\}$?
  $f(X_1, X_2, X_3) = (X_1 = X_2) \wedge (X_2 = X_3)$

- Have person P and Q (whose locations are modeled by X_t and X'_t, respectively) met within interval $\{1, 2, 3\}$?
  $f(X_1, X_2, X_3, X'_1, X'_2, X'_3) = (X_1 = X'_1) \vee (X_2 = X'_2) \vee (X_3 = X'_3)$
- Has P or Q been in any of the locations 10–15 within the interval $\{1, 2, 3\}$?
  $f(X_1, X_2, X_3, X'_1, X'_2, X'_3) = \bigvee_{1 \le t \le 3} X_t \in \{10, \ldots, 15\} \vee X'_t \in \{10, \ldots, 15\}$?

These queries can be easily written as inference queries by introducing a deterministic query variable Q into the network with $\langle f(\ldots) \rangle$ as its cpd, as explained in section 3.2. The answer to the query then consists of the probability distribution $\mathsf{P}(q|\ldots)$ over this query variable. Thus, complex queries are reduced to simple queries by adding a variable to the model (see also [48]). Therefore, in the next chapters, we only consider simple inference queries.

Chapter 4

Local collection of transition frequencies

In the previous chapters, we have investigated how Bayesian networks provide modularity with regard to different (groups of) variables within a probabilistic model. In this chapter, we look at a different kind of modularity, namely *within* a (sparse) transition model $\mathsf{P}(x_t|x_{t-1})$. This is motivated by the way the localization model scales up: not only the number of sensors, but also the number of locations grows (see figure 2.5). As we pointed out in section 2.6, the cpd $\mathsf{P}(x_t|x_{t-1})$ can be constructed from *frequencies*; in this case, *transition frequencies*. We investigate whether this can be done in a local fashion.

Specifically, the locality assumption is that different *clusters of states* are observed by different sensors; for localization scenarios, this is very natural. Note: contrary to the scenario used before, in this chapter we assume these states are *directly* observed (i.e. with certainty) so that accurate statistics can be obtained. On these clusters, we investigate how to apply the 'divide-and-conquer' principle: collecting local statistics for each cluster and aggregating these into global statistics. Advantages of this approach could consist of reducing inter-cluster communication, allowing for identity changes of observed objects, and enabling non-synchronized statistics collection—we detail these examples below.

An example state space is shown in figure 4.1. We assume it is discrete, and partitioned into clusters C_1, C_2 and C_3. Possible transitions are shown using arrows; the goal is to collect a transition count for each arrow. If we collect these counts for one cluster, we correctly distinguish all transitions within the cluster, but we cannot correctly distinguish all transitions entering or leaving the cluster; two transitions from different external states to the same internal state are counted as the same transition, and the same goes for transitions *to* different external states. The reason for this is that from the perspective of this cluster, there is only one large external state, namely 'out of local sensor reach'. Thus, when collecting only

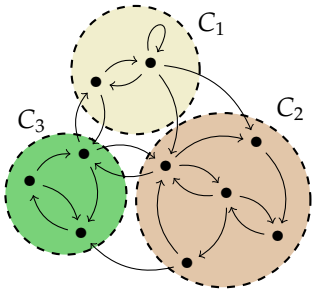


Figure 4.1: A discrete state space, partitioned into three clusters. Arrows represent the possible transitions between states.

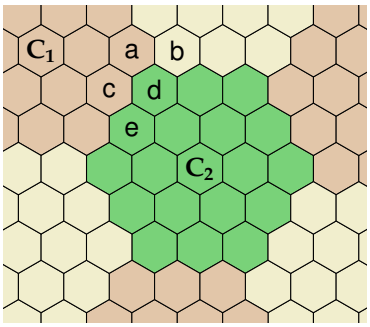


Figure 4.2: A 2D area as a discrete state space for a moving object's location. The states (a, b, ...) are called granules. Again, the state space is partitioned into clusters ($C_1, C_2, \ldots$). Transitions (not shown) are possible between adjacent granules. Self-transitions are also possible.

per-cluster statistics, we lose some information. Our research question is then: to what extent can we recover this information when combining local statistics into global statistics?

The rest of the chapter is structured as follows. In section 4.1, we give a more concrete illustration and motivation of the problem. Next, we formalize the problem statement in terms of transition counts and frequencies (section 4.2). We also briefly describe the application of our approach to the transition probabilities in a Markov model. In section 4.3, we solve the linear equations using an alternative representation of the inter-cluster transition graph. Section 4.4 describes the experiment and its results, and section 4.5 presents conclusions, related work and possible extensions.

4.1 Illustration

To illustrate the situation in a somewhat more concrete way, we will hereafter consider the state space to be the location of a moving object in a two-dimensional area. This area is split up into discrete *granules*, with clusters formed by adjacent granules; an example is shown in figure 4.2. Each granule is observed by one sensor that detects whether the object is present or not, e.g. a camera, infrared sensor or fixed Bluetooth transceiver (detecting mobile Bluetooth devices). Sensors are battery-powered and communicate wirelessly with their neighbors, forming a multi-hop network. At the center of each cluster, a hub collects the transition statistics every once in a while. Using a wired network, it sends them to a central point, where they can be combined.

We assume that acquiring transition counts in a strict per-cluster way (*local* counts) has advantages. We briefly explore three example scenarios with different advantages:

- *Avoid communication over cluster borders.* Note that each sensor only observes *presence*. Transitions are not observed explicitly, but have to be deduced by comparing logs from two adjacent sensors, after which they can be aggregated into transition counts. Due to high communication costs, it is important to do the aggregation from logs into counts close to the source sensors.
  If we were to collect global counts, this can mean two things: either the border sensors of the different clusters have to communicate with *each other* to compare logs, or they have to send their logs to their *respective hubs*, so that they can be compared using the wired network. The former option entails communication between clusters. This could cause technical/organizational difficulties due to incompatible communication standards between clusters; but even if there are none, it is (possibly costly) communication that can be avoided using our approach. The latter option means a lot of additional communication from border nodes to hub.
  Alternatively, our approach of collecting local counts means that one registers at the border when the object disappears or appears in this cluster (by comparison of the logs of the border sensors and their neighbors within the cluster); it is represented as a transition to or from the local 'out-of-reach' state, aggregated into the count, and sent to the hub at low cost. Afterward, the data from all hubs can be combined (at low cost) using the wired network, in order to deduce the counts of the border transitions using the technique we present in the following sections.

- *Allow for identity changes of objects when crossing cluster borders.* There can also be a more fundamental reason for not being able to distinguish inter-cluster transitions: objects might have a different identity in different clusters. The reasons for this can be technical (one cluster uses Bluetooth sensors, while the other uses cameras) or privacy-related (all clusters use Bluetooth, but the users of the mobile Bluetooth devices adopt a different identity in each cluster to remain anonymous). In both cases, it is not possible to collect global statistics of one specific object, but a goal can be to collect statistics of the "average object". Even then, we cannot directly infer transition counts by comparing logs from c and d (figure 4.2): the objects appearing in d at time t might not be the objects leaving from c at $t-1$; they could also have come from a. Note that we assume, in this case, that there are always multiple moving objects in the area (which is in fact a prerequisite for attaining this kind of anonymity).

- *Allow non-synchronized collection of transition statistics.* Collecting statistics locally can also be an an advantage on its own. Consider the case when we

are interested in long-term statistics, and we have already collected these for a certain observed area. Later, we extend the observed area by placing sensors in a neighboring area. Using our approach, we can simply collect the statistics for the new sensors and combine them with the existing statistics.

As we mentioned above, collecting local transition counts also has a disadvantage: the transitions that cross cluster borders cannot be counted directly. Instead, we only obtain an aggregate at both sides of the border. For example, in figure 4.2, consider the number of the transitions from c to d, written $\#(\mathsf{c},\mathsf{d})$. In cluster C_1, these transitions are observed as exit transitions $(\mathsf{c},\mathsf{away}_1)$, and in cluster C_2 as entrance transitions $(\mathsf{away}_2,\mathsf{d})$. However, we can use $\#(\mathsf{c},\mathsf{away}_1)$ and $\#(\mathsf{away}_2,\mathsf{d})$ to deduce $\#(\mathsf{c},\mathsf{d})$. From the arrangement of the granules it follows that:

$$\#(\mathsf{c},\mathsf{away}_1) = \#(\mathsf{c},\mathsf{d}) + \#(\mathsf{c},\mathsf{e})$$
$$\#(\mathsf{away}_2,\mathsf{d}) = \#(\mathsf{c},\mathsf{d}) + \#(\mathsf{a},\mathsf{d}) + \#(\mathsf{b},\mathsf{d}).$$

Together with the other transition counts, this forms a system of linear equations, whose solution space is confined because transition counts are required to be positive. Under certain conditions, the solutions lie so close together that a good approximation of $\#(\mathsf{c},\mathsf{d})$ is possible.

These conditions are mainly dependent on the form of the graph of border-crossing transitions, which we term the *inter-cluster transition graph*. In previous work[23], we have shown that there can even be one exact solution, but this poses quite restrictive conditions on this graph. In this chapter, by considering confined solution spaces, we broaden these conditions.

The technical contributions of this chapter are:

- An analysis of the solution space, relating it to the form of the inter-cluster transition graph.
- An efficient algorithm to determine the solution space.
- An experiment investigating the accuracy of an arbitrary solution from the solution space, given several cluster arrangements similar to figure 4.2.

4.2 Traces, counts, frequencies and probabilities

In this section, we formalize the problem statement. As transition statistics we will first use transition *counts*, leading to a first version of the problem statement. Then we generalize the problem statement using transition *frequencies*, and also briefly outline an approach with transition *probabilities*.

We assume that the structure of the state space, consisting of the possible *states* and *transitions*, is known. This structure is expressed as a finite directed graph $G^{\rightarrow} = (V^{\rightarrow}, E^{\rightarrow})$, of which the vertices $V^{\rightarrow}$ correspond to states and the edges $E^{\rightarrow} \subseteq V^{\rightarrow} \times V^{\rightarrow}$ to transitions. We call this graph the *global transition graph*.

Furthermore, the states are partitioned in n local clusters by a partition function $p : V^{\rightarrow} \to \{1, \ldots, n\}$. In cluster i ($1 \leq i \leq n$), a local state v with $p(v) = i$ can be observed as such, while the other states are all 'out of local sensor reach', and together form one state away_i. We express this using an observation function o_i:

$$o_i(v) \stackrel{\text{def}}{=} \begin{cases} v & \text{if } p(v) = i \\ \mathsf{away}_i & \text{if } p(v) \neq i \end{cases}$$

This induces a *local transition graph* $C_i^{\rightarrow} = (V_i^{\rightarrow}, E_i^{\rightarrow})$ for cluster i. Its vertices $V_i^{\rightarrow}$ are those states that o_i maps to, and its edges are derived from the global transitions by mapping their source and target state into their local observations. Formally:

$$V_i^{\rightarrow} = \{\, o_i(v) \mid v \in V^{\rightarrow} \,\}$$
$$E_i^{\rightarrow} = \{\, (o_i(v), o_i(w)) \mid (v, w) \in E^{\rightarrow} \,\}$$

An example is shown in figure 4.3 for a small part of the location state space in figure 4.2. The three local transition graphs in figure 4.3a are obtained from the global transition graph in figure 4.3b using partition function $p(\mathsf{a}) = p(\mathsf{c}) = 1$, $p(\mathsf{d}) = p(\mathsf{e}) = 2, p(\mathsf{b}) = 3$. The colors serve as a visual aid for this partition function.

A *trace* T is a sequence of consecutive states, where $T(\tau) \in V$ represents the state at time τ. A trace is consistent with the possible transitions, i.e. if $T(\tau) = s$ and $T(\tau + 1) = t$ then $(s, t) \in E^{\rightarrow}$. Our main assumption is that it is impossible to directly obtain a *global trace* of the system; however it is possible to obtain *local traces*. A local trace is a sequence of local observations of consecutive (global) states. The local trace observed in cluster i, derived from an unobservable global trace T, is written $o_i[T]$. It is defined by applying o_i pointwise to each state in T:

$$o_i[T](\tau) \stackrel{\text{def}}{=} o_i(T(\tau))$$

Local trace $o_i[T]$ contains states in $V_i^{\rightarrow}$ and is consistent with the transitions in $E_i^{\rightarrow}$.

If local traces $o_i[T]$ were available for all i, it would be trivial to reconstruct T (and derive global transition counts or probabilities from it). Our assumption is that, instead of local traces, only their corresponding transition counts are available. Formally, a *transition count* $\#_T(s, t)$ is the number of times that the transition (s, t) occurs in trace T (i.e. the number of distinct times τ for which $T(\tau) = s \wedge T(\tau + 1) = t$).

Thus, we arrive at a first formulation of the problem statement, in terms of transition counts:

> Assume we have, for each cluster i, a count $\#_{o_i[T]}(s, t)$ of all local transitions $(s, t) \in E_i^{\rightarrow}$. All counts are derived from the same (unobservable) trace T. Is it possible to deduce or approximate $\#_T(s, t)$ for all global transitions $(s, t) \in E^{\rightarrow}$?

Note that, in this formulation, all transition counts should be acquired over the same period. We can generalize the problem to a broader setting in which this

is not required. For this we need to make two extra assumptions, namely that the "average behavior" of the process does not change, and that the observed traces are long enough to exhibit this average behavior. Formally, in terms of Markov models, the first assumption is that the process is homogeneous (not time dependent) and ergodic (keeps returning to all states). The second assumption means that the ratios of observed transition counts (see below) approximate the model's probabilities.

When these assumptions are satisfied, the local observations can originate from different global traces, that is, they can be performed at different times (cf. the example *allow non-synchronized collection* in the introduction). The length of these traces may also vary; in order to compare transition counts regardless, we normalize them w.r.t. the trace length. These normalized counts are termed *transition frequencies* and are defined:

$$F_T(s,t) \stackrel{\text{def}}{=} \frac{\#_T(s,t)}{\sum_{x,y} \#_T(x,y)}$$

From now on, our formulation of the problem will be in terms of these more general transition frequencies:

> Assume we have, for each cluster i, a frequency $F_{o_i[T_i]}(s,t)$ of all local transitions $(s,t) \in E_i^{\rightarrow}$. These frequencies may now originate from different (unobservable) traces T_i, but the observed process should be homogeneous and ergodic, and the traces should be sufficiently long (see above). Is it possible to deduce or approximate $F_T(s,t)$ (with T an arbitrary, sufficiently long trace) for all global transitions $(s,t) \in E^{\rightarrow}$?

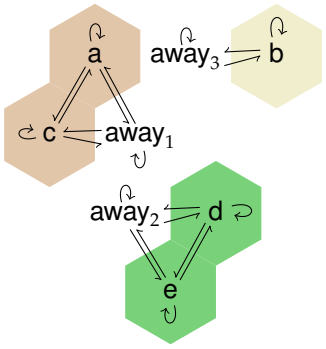
The solution method presented in section 4.3 also applies to the first problem statement, which is more restrictive in that only simultaneous observations can be combined, but more permissive w.r.t. the nature of the process and the length of the observations.

An illustration of the problem statement is shown in figure 4.3. We know the frequencies on the local transition graphs, and the goal is to find the frequencies on the global transition graph. Some of them can be directly copied, namely those between states of the same cluster. For the others, which we term *inter-cluster* frequencies, the structure of the global transition graph defines a system of linear equations. For example, the relevant equations for the (c,d) transition are:

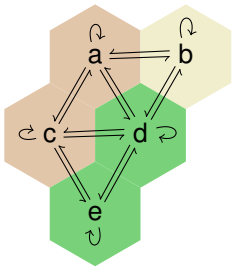
$$F_{o_1[T_1]}(\mathsf{c},\mathsf{away}_1) = F_T(\mathsf{c},\mathsf{d}) + F_T(\mathsf{c},\mathsf{e})$$
$$F_{o_2[T_2]}(\mathsf{away}_2,\mathsf{d}) = F_T(\mathsf{c},\mathsf{d}) + F_T(\mathsf{a},\mathsf{d}) + F_T(\mathsf{b},\mathsf{d})$$

In the next section, we will show how to solve the system of equations.

Closely related to transition counts and frequencies are the transition probabilities that form the parameters of a first-order Markov model. When we make the



(a) For each of the three colored clusters (C_1, C_2, C_3), we know the frequencies (not shown) on the edges of the *local transition graph*.



(b) We know the structure of the *global transition graph*, and want to obtain the frequencies on the edges. Intra-cluster frequencies like $F(\mathsf{c}, \mathsf{a})$ can be directly copied; inter-cluster frequencies like $F(\mathsf{c}, \mathsf{d})$ cannot.

Figure 4.3: Problem statement.

assumption that the observed process can be described by such a model, the observed transition counts or frequencies provide the maximum likelihood estimate of these parameters:

$$\mathsf{P}(X_{\tau+1} = t | X_\tau = s) = \frac{\#(s,t)}{\sum_y \#(s,y)} = \frac{F(s,t)}{\sum_y F(s,y)}$$

The basic use case for applying our technique with regard to Markov models is as follows. We observe local transition frequencies in the clusters, use these to derive the global transition frequencies, and use the latter to determine the parameters of a global Markov model (using the above formula). In other words, the problem is in terms of frequencies, and its solution is only translated to probabilities afterward.

However, it is also possible that, instead of starting with local transition frequencies, we want to start with local Markov models on the cluster state spaces C_i. For example, the parameters of these Markov models could have been estimated from noisy observations using expectation maximization (a *hidden* Markov model setting). Under certain restrictions, it is possible to:

1. transform these local Markov models into local frequencies of long-term average behavior;

2. then, use our technique to combine them into global frequencies of long-term average behavior;

3. finally, use the above formula to calculate the maximum likelihood estimate for the global Markov model.

To perform step 1, we can use the formula

$$F(s,t) = \pi_{C_i}(s)\mathsf{P}(X_{\tau+1} = t | X_\tau = s).$$

In this formula, a crucial role is played by the *stationary distribution* π_{C_i}, which can be derived from the collective probabilities in the local Markov model on C_i. For a further discussion on this transformation and its applicability, we refer to our previous work[23].

4.3 Solving a flow with known vertex values

In the previous section, we formulated the problem of finding global transition frequencies. In this section, we analyze this problem using a combination of linear algebra and graph theory. The frequencies form a *flow* on the global transition graph; we are looking for the unknown values of this flow, which are constrained by known sums.

We show how solution space of the corresponding system of linear equations is found by considering the structure of the inter-cluster transition graph. After this, we take the requirement into account that all frequencies should be positive, and show how it confines the solutions.

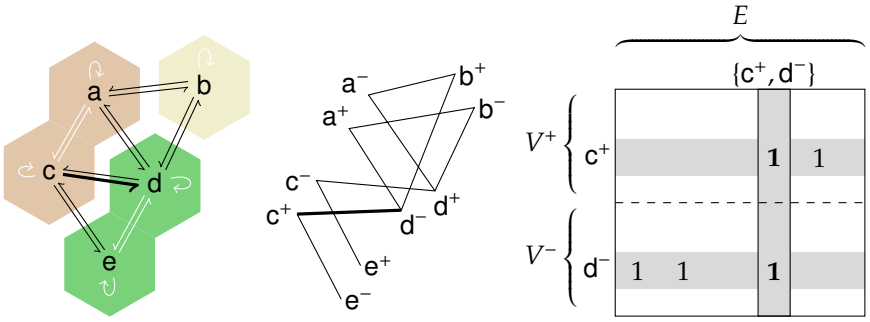
4.3.1 Problem definition

Given a directed graph $G^{\rightarrow} = (V^{\rightarrow}, E^{\rightarrow})$, a *flow* is a function $f^{\rightarrow} : E^{\rightarrow} \to \mathbb{R}$ assigning a value to each edge. A given flow $f^{\rightarrow}$ establishes, for each vertex $v \in V^{\rightarrow}$, its inflow $f^{-}(v)$ and outflow $f^{+}(v)$:

$$\begin{aligned} f^{-}(v) &= \sum_{u|(u,v)\in E^{\rightarrow}} f^{\rightarrow}(u,v) \\ f^{+}(v) &= \sum_{w|(v,w)\in E^{\rightarrow}} f^{\rightarrow}(v,w) \end{aligned} \qquad \textsc{(in-out-flow)}$$

Our problem statement is described by exactly this system of equations, but with the values of $f^{\rightarrow}$ as unknown variables, and a known inflow and outflow for each vertex.

In principle, the global transition graph from the problem statement in section 4.2 corresponds to the $G^{\rightarrow}$ graph of this system. This means that *all* frequencies are variables in the system. However, we already know a lot of these frequencies, namely those of transitions *within* a cluster. For convenience, we will disregard these in our further analysis of the problem; we remove the corresponding variables from the system. For graph $G^{\rightarrow}$, this means that, for every transition (s,t) within partition i (so $p(s) = p(t) = i$):



(a) Inter-cluster transition graph $G^{\rightarrow}$ (also shown, in white, are the left out intra-cluster transitions)

(b) G, the undirected representation of $G^{\rightarrow}$ ('uncoiled graph')

(c) $\mathbf{N}$, the incidence matrix of G. Values in the white part are not shown; the grey part has 0s except where 1s are shown.

Figure 4.4: Three representations of the inter-cluster transition graph. The transition from c to d is highlighted everywhere.

- we leave its edge out of the graph altogether. Vertices in which no transition starts or ends anymore are also removed.
- we subtract its known frequency $F_{o_i[T_i]}(s,t)$ from $f^+(s)$ and $f^-(t)$.

After doing this, the remaining graph $G^{\rightarrow} = (V^{\rightarrow}, E^{\rightarrow})$ consists only of inter-cluster transitions $E^{\rightarrow}$ and the corresponding vertices $V^{\rightarrow}$ (border granules)—see figure 4.4a. For a vertex v in cluster i, the corresponding frequencies sum up to

$$f^-(v) = F_{o_i[T_i]}(\mathsf{away}_i, v)$$
$$f^+(v) = F_{o_i[T_i]}(v, \mathsf{away}_i)$$

We term the graph that is left over the *inter-cluster transition graph*. The problem statement can then be summarized as follows: given inter-cluster transition graph $G^{\rightarrow}$ and the above values of f^+ and f^-, we are looking for solutions $f^{\rightarrow}$ that satisfy (IN-OUT-FLOW). Essentially, this amounts to solving a system of linear equations in which all coefficients are 1 or 0. Each edge corresponds to a variable, and each vertex contributes the two equations that constitute (IN-OUT-FLOW).

However, to analyze the system using graph-theoretic concepts, it is more convenient to have a one-to-one correspondence between vertices and equations. With this goal in mind, we introduce a different representation of the graph $G^{\rightarrow}$ and the corresponding equations. We term this representation the *uncoiled* system.[1]

[1]The name *uncoiled* is chosen after the effect it has on loops in the graph. In our application, however, we do not encounter loops, because a loop is a transition within a cluster, and has been filtered out.

Definition 1. Given the directed graph $G^{\rightarrow} = (V^{\rightarrow}, E^{\rightarrow})$ with n vertices labeled $\{v_1, v_2, \ldots, v_n\}$, we define its *uncoiled graph* $G = (V, E)$. This graph G is a bipartite undirected graph with partitions $V = V^+ + V^-$, where $V^+ = \{v_1^+, v_2^+, \ldots, v_n^+\}$ and $V^- = \{v_1^-, v_2^-, \ldots, v_n^-\}$. Each vertex v_i from the original graph is represented twice in V: as a source v_i^+ and as a target v_i^-. The edge set E contains an undirected edge $\{v_i^+, v_j^-\}$ iff $E^{\rightarrow}$ contains a directed edge (v_i, v_j).

For an example, see figure 4.4b. A flow $f^{\rightarrow}$ on the edges of the directed graph is represented by a flow f on the corresponding edges of G: $f\{v_i^+, v_j^-\} = f^{\rightarrow}(v_i, v_j)$. Again, this flow is assumed to be unknown, while the values of certain sums are known. In the original directed system, these values were represented by two different functions f^+ and f^- on each vertex; in the uncoiled system this role is played by one function $f^{\pm} : V \to \mathbb{R}$ on the two different types of vertex. The two representations are related as follows:

$$f^{\pm}(v_i^+) = f^+(v_i)$$
$$f^{\pm}(v_i^-) = f^-(v_i)$$

The problem statement can now be formulated in terms of the uncoiled system: given the uncoiled representation G of inter-cluster graph $G^{\rightarrow}$, and the vertex sums $f^{\pm}$ above, find a flow $f : E \to \mathbb{R}$ that satisfies, for all v,

$$f^{\pm}(v) = \sum_{w|\{v,w\}\in E} f\{v, w\}. \qquad \textbf{(VERTEX-FLOW)}$$

As in the above equation we will hereafter often refer to the vertices in V regardless of whether they are in partition V^+ or V^-. In these cases we also use the variable v_i, imposing a certain ordering: $V = \{v_1, v_2, \ldots, v_{2n}\}$. Likewise, we order the edges as $\{e_1, e_2, \ldots, e_m\}$; we keep the convention that $|V| = 2n$ and $|E| = m$.

With such orderings in place, the uncoiled system can easily be written in the matrix-vector form conventional in linear algebra. The matrix of coefficients corresponds to G's *incidence matrix* $\mathbf{N}$ (figure 4.4c):

$$\mathbf{N}_{ij} \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } v_i \in e_j \\ 0 & \text{if } v_i \notin e_j \end{cases}$$

Using the same orderings, the functions $f^{\pm}$ and f are represented as vectors:

$$\mathbf{f}^{\pm} \stackrel{\text{def}}{=} (f^{\pm}(v_1), f^{\pm}(v_2), \ldots, f^{\pm}(v_{2n}))^{\mathsf{T}}$$
$$\mathbf{f} \stackrel{\text{def}}{=} (f(e_1), f(e_2), \ldots, f(e_m))^{\mathsf{T}}$$

The matrix-vector formulation of our problem statement is then as follows: given $\mathbf{N}$ and $\mathbf{f}^{\pm}$, find a solution $\mathbf{f}$ of the system

$$\mathbf{N}\mathbf{f} = \mathbf{f}^{\pm}. \qquad \textbf{(VECTOR-FLOW)}$$

In principle, this system can be solved using basic textbook techniques such as Gauss-Jordan elimination. Instead, we exploit the special structure of $\mathbf{N}$—the equations are partitioned into two sets, and each variable participates in exactly one equation of each set—and solve parts using graph theory. This approach provides us with:

- insight in the structure of the solution space: its existence and dimensions are related to components and cycles in the uncoiled representation of the inter-cluster transition graph;
- an efficient method of solving the system.

4.3.2 Solution

A basic result in linear algebra is that every solution $\mathbf{f}$ of $\mathbf{Nf} = \mathbf{f}^{\pm}$ can be written in the form $\mathbf{f} = \mathbf{p} + \mathbf{k}$ where $\mathbf{p}$ is any fixed particular solution of the system, and $\mathbf{k}$ is a solution of $\mathbf{Nk} = \mathbf{0}$. All possible solutions of this last equation define $\mathbf{N}$'s *null space* or *kernel*:

$$\operatorname{Ker} \mathbf{N} \stackrel{\text{def}}{=} \{ \mathbf{k} \mid \mathbf{Nk} = \mathbf{0} \}$$

Conversely, all vectors $\mathbf{f}$ that can be written as $\mathbf{f} = \mathbf{p} + \mathbf{k}$ are solutions. Hence, to specify all the solutions of the system, it is enough to give one particular solution $\mathbf{p}$ and a specification of $\operatorname{Ker} \mathbf{N}$. This specification has the form of a *basis*, a minimal set of vectors of which every $\mathbf{k}$ is a linear combination.

We show that both can be found by considering an arbitrary spanning tree of G:

- A particular solution can be found by solving the system with only the edges of the tree, and assigning $f(e) = 0$ for each left-out edge e.
- The basis vectors correspond to the fundamental cycles w.r.t. this tree: cycles that are obtained by putting one of the left-out edges back in.

To find G's spanning tree S (or more accurately, as G might have multiple components, its spanning *forest* S) and simultaneously construct a particular solution $\mathbf{p}$, we present Algorithm 1. It does a depth-first traversal of the components of G; on the way back along an edge, it fills in the corresponding element of $\mathbf{p}$. For edges that would create a cycle, it fills in 0. An example is shown in figure 4.5.

Note that the algorithm constructs a *function* (a flow). Again, we represent it as a vector using the edge ordering: $\mathbf{p} \stackrel{\text{def}}{=} (p(e_1), p(e_2), \ldots, p(e_m))$. Furthermore, we assume hereafter, without loss of generality, that the left-out edges $E - S$ are last in this ordering. Each left-out edge yields one of G's *fundamental cycles* w.r.t. S:

Input: bipartite graph $G = (V, E)$ and vertex summations $f^{\pm} : V \to \mathbb{R}$
Output: solution p that matches the summations as in (VERTEX-FLOW), and spanning tree S

$p \leftarrow$ the empty (partial) function on $E \to \mathbb{R}$
$S \leftarrow \emptyset$
$visited \leftarrow \emptyset$

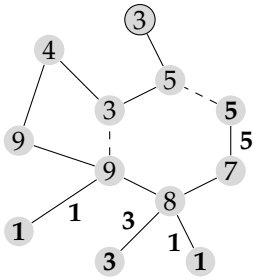
while $visited \neq V$ **do**
  $root \leftarrow$ an arbitrary element of $(V - visited)$
  SolveSubtree($root, \emptyset$)
  if $f^{\pm}(root) \neq \sum_v f\{root, v\}$ **then error** 'no solution exists'
end

procedure SolveSubtree($root$,$maybeparent$)
  // *maybeparent* records the node we came from, to prevent going back
  if $root \in visited$ **then**
    // cycle detected
    $p(maybeparent \cup \{root\}) \leftarrow 0$
  else
    $visited \leftarrow visited \cup \{root\}$
    foreach $v \in (V - maybeparent)$ **such that** $\{root, v\} \in E$ **do**
      SolveSubtree($v, \{root\}$)
    end
    if $maybeparent \neq \emptyset$ **then**
      $S \leftarrow S \cup \{maybeparent \cup \{root\}\}$
      $p(maybeparent \cup \{root\}) \leftarrow f^{\pm}(root) - \sum_v f\{root, v\}$
    end
  end
end

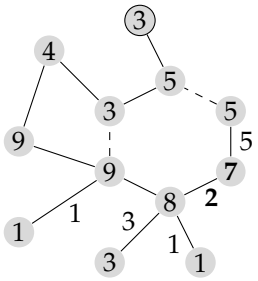
Algorithm 1: A depth-first traversal of each component of the graph, recording a particular solution in p and constructing a spanning tree S.

Definition 2. Let G be a graph with m edges $\{e_1, e_2, \ldots, e_m\}$, of which the first s edges $\{e_1, e_2, \ldots, e_s\}$ form a spanning tree S. For each q with $s < q \leq m$, the *fundamental cycle* $\mathbf{f}_q$ contains e_q and the edges in S that form a cycle with it. Instead of identifying $\mathbf{f}_q$ with this *set* of edges, we define it to be a vector of length m with alternating 1 and -1 on the positions corresponding to these edges:

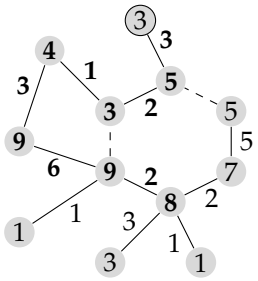
$$\begin{aligned}
\mathbf{f}_q(q) &\stackrel{\text{def}}{=} 1 \\
\mathbf{f}_q(i) &\stackrel{\text{def}}{=} 1, && \text{for } e_i \text{ on the cycle at an even distance of } e_q \\
\mathbf{f}_q(j) &\stackrel{\text{def}}{=} -1, && \text{for } e_j \text{ on the cycle at an odd distance of } e_q \\
\mathbf{f}_q(k) &\stackrel{\text{def}}{=} 0, && \text{for } e_k \text{ not on the cycle}
\end{aligned}$$



(a) Partial solution after the traversal of some one-node subtrees. These nodes connect only to one edge, so the value on this edge should equal the node value.

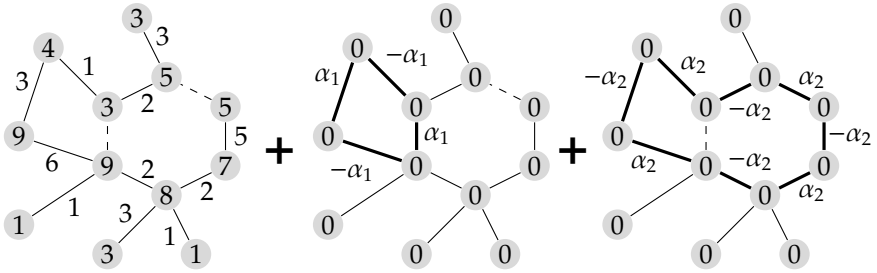


(b) Now, there is only one empty edge left for the 7 node, so we can deduce its value:
7 − 5 = **2**.



(c) Likewise, we continue up the tree. After having filled in the last edge, we check whether the sum in the root node (= 3) is correct. It is, so we have found a solution.

Figure 4.5: Finding a particular solution. The goal is to find p values at the edges that sum up to the given $f^{\pm}$ values in the nodes. This is done by a traversal of a spanning tree, here rooted in the topmost node. After traversal of each subtree, the value of the connecting edge can be deduced. Edges not in the tree (dashed) are given the value 0.



(a) Particular solution. **(b)** Fundamental cycle 1. **(c)** Fundamental cycle 2.

Figure 4.6: Each fundamental cycle leads to solutions of $\mathbf{Nf} = \mathbf{0}$. By adding them to the particular solution of the original system $\mathbf{Nf} = \mathbf{f}^{\pm}$, we find other solutions of that system. For example, the edges of the top-left node get the values $3 + \alpha_1 - \alpha_2$ and $1 - \alpha_1 + \alpha_2$, respectively. Together, these still add up to 4.

See also the examples in figure 4.6, where f_q has been multiplied by α_1 and α_2, respectively. Note that it makes sense to speak about even and odd distances because the cycles in our graph are always of even length (because the graph is bipartite). Vector $\mathbf{f}_q$ forms a solution of $\mathbf{Nf} = \mathbf{0}$, because in the vertex sums $f^{\pm}$, every vertex on the cycle has two terms that cancel each other out; all the other terms, also for the other vertices, are zero.

So far, we have made it plausible that Algorithm 1 provides us with a $\mathbf{p}$ for

which $\mathbf{Np} = \mathbf{f}^{\pm}$, and vectors $\mathbf{f}_q$ for which $\mathbf{Nf}_q = \mathbf{0}$. When we construct a $m \times (m-s)$ matrix $\mathbf{B} \stackrel{\text{def}}{=} [\mathbf{f}_{s+1}, \ldots, \mathbf{f}_m]$ containing the fundamental cycles, we get a succinct characterization of the solutions to (VECTOR-FLOW) found until now:

$$\{\, \mathbf{p} + \mathbf{B}\alpha \mid \alpha \in \mathbb{R}^{m-s} \,\}$$

In the remainder of this section, we will give proof sketches that

- if a solution exists, Algorithm 1 will find one (we also provide the conditions for existence);
- the fundamental cycles $\mathbf{f}_q$ form a basis of $\operatorname{Ker} \mathbf{N}$;

so the solutions described above are indeed *all* the solutions.

Theorem 1. *If $\mathbf{Nf} = \mathbf{f}^{\pm}$ has a solution, it also has one that is zero on all edges $E - S$.*

Proof. Suppose $\mathbf{f} = (\mathbf{f}_S; \mathbf{f}_L)$ is a solution with values $\mathbf{f}_S$ for edges S and $\mathbf{f}_L$ for $E - S$. We construct a vector $\mathbf{z}$ that is zero on $e_{s+1}, \ldots, e_m$: $\mathbf{z} \stackrel{\text{def}}{=} \mathbf{f} - \mathbf{Bf}_L$. Then $\mathbf{Nz} = \mathbf{Nf} - \mathbf{NBf}_L = \mathbf{f}^{\pm} - \mathbf{0} = \mathbf{f}^{\pm}$. □

Hereby, we justify the fact that Algorithm 1 sets the solution for the $E - S$ edges to 0; this does not prevent it from finding a solution. In other words, solving (VERTEX-FLOW) for graph G reduces to solving (VERTEX-FLOW) for its spanning forest S.

Definition 3. A *component* $C = (V_C, E_C)$ is a subgraph of G, where $V_C \subseteq V$ is a maximal set of transitively connected vertices, and $E_C = \{\{v, w\} \in E | v, w \in V_C\}$ the corresponding edges. In other words, a component consists of all the vertices and edges reachable from a certain vertex. We also define V_C^+ and V_C^-, the classes of the bipartition within C:

$$\begin{aligned} V_C^+ &\stackrel{\text{def}}{=} V_C \cap V^+ \\ V_C^- &\stackrel{\text{def}}{=} V_C \cap V^- \end{aligned}$$

Theorem 2. *A system $(G, f^{\pm})$ with G a bipartite graph has a solution for* (VERTEX-FLOW) *iff for each component C of G the following equation holds:*

$$\sum_{v \in V_C^+} f^{\pm}(v) = \sum_{v \in V_C^-} f^{\pm}(v)$$

If this is the case, Algorithm 1 will find a solution p (which is the unique solution on spanning forest S); if not, Algorithm 1 will halt with an error.

Proof (sketch). We already saw that the problem reduces to solving (VERTEX-FLOW) on spanning forest S. For each component C of S (which has the same vertices as the corresponding component of G), we can prove by induction on subtrees that it finds the unique values of p that satisfy the $f^{\pm}$ equations for all $v \in V_C$ except the root vertex. When we group together all vertices v with the same distance $\mathrm{d}(v) = n$ to the root vertex, these take the form (for $n > 0$):

$$\sum_{\mathrm{d}(v)=n} f^{\pm}(v) = \sum_{\substack{\mathrm{d}(u)=n-1 \\ \mathrm{d}(v)=n}} p\{u,v\} + \sum_{\substack{\mathrm{d}(v)=n \\ \mathrm{d}(w)=n+1}} p\{v,w\}.$$

If we subtract the equations for even $\mathrm{d}(v)$ from those for odd $\mathrm{d}(v)$, we get:

$$\sum_{\mathrm{d}(v)\text{ odd}} f^{\pm}(v) - \sum_{\mathrm{d}(v)>0\text{ and even}} f^{\pm}(v) = \sum_{\substack{\mathrm{d}(u)=0 \\ \mathrm{d}(v)=1}} p\{u,v\}.$$

Now, iff the $f^{\pm}$ equation also holds for the root vertex ($\mathrm{d}(v) = 0$), $\sum f^{\pm}(v)$ is the same for odd and even $\mathrm{d}(v)$, and since the V_C^+ and V_C^- vertices appear at alternating distances, for $v \in V_C^+$ and $v \in V_C^-$. □

Theorem 3. *Let G be a bipartite graph with incidence matrix $\mathbf{N}$, and a matrix of fundamental cycles $\mathbf{B}$ (w.r.t. spanning forest S). Then each $\mathbf{k} \in \operatorname{Ker}\mathbf{N}$ can be written as $\mathbf{B}\alpha$ for a certain α.*

Proof. Suppose $\mathbf{k} = (\mathbf{k}_S; \mathbf{k}_L)$ with values $\mathbf{k}_S$ for edges S and $\mathbf{k}_L$ for $E - S$. Again, we construct $\mathbf{z} \stackrel{\text{def}}{=} \mathbf{k} - \mathbf{B}\mathbf{k}_L$, whose elements for edges not in S are zero, and for which $\mathbf{N}\mathbf{z} = \mathbf{0}$. By Theorem 2, the solution $\mathbf{z}$ for this system is unique; so it must be the trivial $\mathbf{z} = \mathbf{0}$. Therefore, $\mathbf{k} = \mathbf{B}\mathbf{k}_L$. □

So, every $\mathbf{k} \in \operatorname{Ker}\mathbf{N}$ is a linear combination of the fundamental cycles. Furthermore, fundamental cycle $\mathbf{f}_q$ cannot be a linear combination of the other fundamental cycles, because $\mathbf{f}_q(q) = 1$, and all the other $\mathbf{f}_i$ have $\mathbf{f}_i(q) = 0$. Hence, the fundamental cycles form a basis of $\operatorname{Ker}\mathbf{N}$.

4.3.3 Inequalities

Until now, we have disregarded the fact that a meaningful solution to our original problem can only consist of positive frequencies. So, in addition to satisfying (IN-OUT-FLOW), the flow $f^{\rightarrow}$ should have $f^{\rightarrow}(u,v) \geq 0$ for every edge (u,v). This requirement can drastically reduce the space of possible solutions, which we can put to good use.

We write the m inequalities as follows ($\mathbf{e}_i$ is a vector of length m with $\mathbf{e}_i(i) = 1$ and $\mathbf{e}_i(j) = 0$ for $j \neq i$)):

$$\mathbf{e}_i(\mathbf{p} + \mathbf{B}\alpha) \geq 0 \quad \text{for all } 1 \leq i \leq m$$

This can be rewritten to

$$-\mathbf{B}_{i\star}\alpha \leq \mathbf{p}(i)$$

in which $\mathbf{B}_{i\star}$ denotes the ith row of matrix $\mathbf{B}$. This row contains the coefficients (either 0, 1 or -1) of all fundamental cycles on edge e_i. For example, the top-left edge in figure 4.6 yields the inequality $-(1,-1)\cdot(\alpha_1,\alpha_2)^\mathsf{T} \leq 3$.

4.4 Experiment

As we described in section 4.3.3, the solution space can be quite small due to the requirement that every frequency should be positive. In this section, we describe an experiment to test whether the space is indeed so small that an arbitrary solution from this space will be close enough to the original values.

For this experiment, we use a state space which is structured like figure 4.2: a triangular grid in which each state has six neighbors to which transitions are possible. Also like in figure 4.2, the clusters in which we partition the space are hexagonal; they consist of a center state and r concentric layers ($r = 2$ in the figure). This number r is called the *cluster radius*.

We then assign an arbitrary transition frequency to each transition; however, we do make sure that, per state, all outgoing and all incoming frequencies add up to the same number. Then, we discard the transitions within clusters, after which we are left with the inter-cluster transition graph and a flow $\mathbf{f}_0$. We calculate the aggregations $\mathbf{f}^\pm$, and use these as input to the linear system $\{\mathbf{Nf} = \mathbf{f}^\pm, \mathbf{f} \geq \mathbf{0}\}$, which we solve using the method outlined in section 4.3. (Note: the existence condition in Theorem 2 is satisfied, as we already know there is a solution $\mathbf{f}_0$).

From the solution space constrained by the inequalities, we pick an arbitrary vector $\mathbf{f}$, and measure how close it is to $\mathbf{f}_0$ using a metric defined below. Unfortunately, this step presents a problem: it is far from trivial to generate a vector satisfying all the inequalities. As we are working in MATLAB, we have tried using its linear programming algorithms to this end (i.e. we find the $\mathbf{f}$ in the solution space that minimizes an arbitrary linear function), but they often did not succeed in finding a vector. After some experimentation, we found that MATLAB's quadratic programming algorithms do succeed. We also found that picking a more or less 'balanced' vector produced better results than picking a truly random vector.

This balanced vector $\mathbf{f}$ is the vector from the solution space which is closest to a certain $\mathbf{b}$ (which is generally not in the solution space). We define $\mathbf{b}$ as the following vector: for each i, we divide both the $f^\pm(v_i^+)$ and the $f^\pm(v_i^-)$ sums evenly over the incident edges; of the two values that each edge is assigned in this way, we take the average. However, in this process we ignore every edge e_j that is not on any cycle, because it takes the same value $f(e_j) = \mathbf{p}(j)$ in every solution. So, before dividing the $f^\pm$ sums, we first subtract these fixed values from them. Finding $\mathbf{f}$ then consists of minimizing a quadratic function over the constrained

solution space, for which we have used MATLAB's quadprog library function (from the Optimization Toolbox).

After picking a $\mathbf{f}$, we evaluate its accuracy—the difference between $\mathbf{f}$ and $\mathbf{f}_0$. We used the following measure:

$$\frac{\text{median}\{\text{abs}(\mathbf{f}(i) - \mathbf{f}_0(i)) | e_i \text{ on a cycle}\}}{\text{avg } \mathbf{f}_0}$$

In other words, we take the median error of all edges that are on a cycle; to compare the different experiments, we normalize this error by dividing it by the average edge value.

4.4.1 Results

We have performed the above experiment with multiple configurations. We varied the size of the global state space from a 50×50 grid to a 150×150 grid, and the radius of the clusters from 5 to 15. In a first set of configurations, we increased these sizes simultaneously, because we wanted to investigate the effect of increasing the *state granularity* (while the number of clusters stays the same). In the second set of configurations, we keep the cluster size fixed at 10 and only vary the grid size (and hence the number of clusters). Each configuration is run 25 times to reduce the effects due to randomness of the original vector.

To give an impression about the size of the linear problems: the smallest inter-cluster graph (radius-10 clusters on a 50×50 grid) contains 900 edges, 436 vertices, and 29 fundamental cycles. The largest (radius-10 clusters on a 150×150 grid) contains 8736 edges, 4176 vertices, and 385 fundamental cycles.

For the first set of configurations, our hypothesis was that the accuracy becomes better with a higher state granularity. We expected this because the number of on-cycle edges (inequalities) increases while the number of fundamental cycles (dimensions) stays the same. Figure 4.7 shows that we were right.

For the second set of configurations, we expected the accuracy roughly to stay the same, because the number of on-cycle edges as well as the number of fundamental cycles grow linearly. Figure 4.8 shows the accuracy.

The running time of quadprog function varied from 0.17 sec for the smallest problem to 481 sec for the largest.

4.5 Conclusions and future work

In this chapter, we have investigated how to combine local transition counts into global transition counts, and whether it is worthwhile in terms of accuracy. Experiments on random data have shown that for a cluster radius of 15, half of the estimated counts fall within 4% of the real values; of course, it depends on the application whether this is good enough.

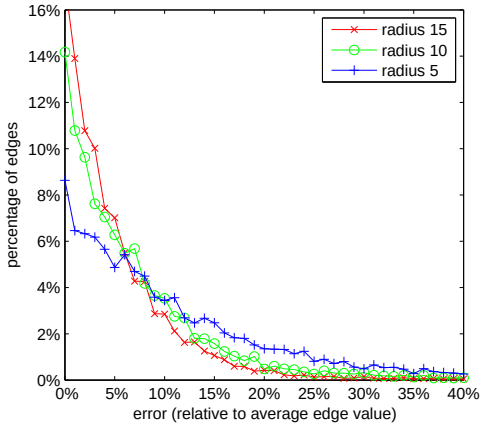


Figure 4.7: Distribution of errors on edges, first set of configurations.

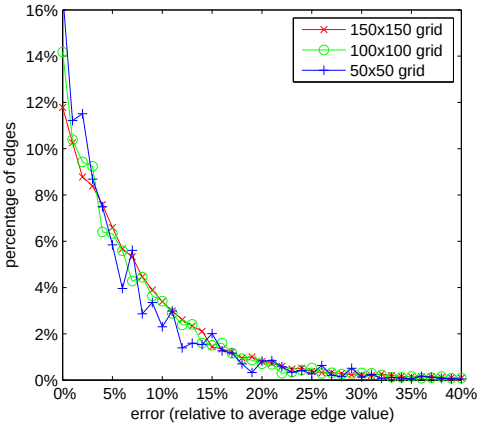


Figure 4.8: Distribution of errors on edges, second set of configurations.

For our technique to scale to large numbers of clusters, using MATLAB's quadprog algorithm is not an option because it is too slow. We need to find an algorithm that does not scale exponentially. It probably does not have to find an optimal solution like quadprog; *any* solution could be good enough. If even that is out of reach, we could settle for an approximate solution.

A related research question is whether it is possible to solve the combination of cluster frequencies *incrementally* per cluster. The idea is to start with one cluster and a large away state. Then, we position a new cluster in this away state (still leaving room for more) and solve the problem. We continue like this for the other clusters. In our localization example, this would correspond to a situation in which we gradually add sensor clusters, thereby enlarging the observed area.

Finally, we would like to have a sound method to deal with inconsistencies between the several local frequencies. In this chapter, we have assumed that the local frequencies are consistent with each other—either because they result from the exact same global trace, or because the different traces are all so long that the frequencies have converged to the transition probabilities. In both cases, the result is that $\sum_{v \in V_C^+} f^{\pm}(v) = \sum_{v \in V_C^-} f^{\pm}(v)$ for each component, and the system has an exact solution. In practice, we do not expect this to be the case, and it would be useful to be able to give the best estimate for the solution (maybe using maximum likelihood or Bayesian techniques).

Chapter 5

Probabilistic inference in a relational representation

Given a probabilistic model over a set of variables, *inference* is the task of deriving the probability distribution over a subset of query variables, given the observed values of another subset of evidence variables. As we have explained in section 2.5, this forms the basis of probabilistic sensor data processing.

The problem of how to efficiently perform inference for Bayesian networks has been extensively researched in AI literature, which has led to well-known inference procedures such as *variable elimination*[61] and *junction tree propagation*[45]. However, as we discuss in chapter 6, these conventional procedures do not suffice for our sensor data models.

In this chapter, we recast the problem into a relational algebra framework, in which the building blocks are the bulk operators $\bowtie$ and $\dot{\pi}$ on structured tables of probabilities $p[B|A]$, rather than individual multiplications and additions on single probabilities $\mathsf{P}(B{=}b|A{=}a)$. The resulting expressions are assigned a well-defined denotational semantics, which is at the same time

(a) similar to the summations over factorized probability distributions used in conventional procedures, and

(b) based on the basic relational algebra commonly used in database semantics.

To evaluate such an expression, it can either be assigned an operational semantics based on multi-dimensional array assignments (again associated with the conventional procedures), or mapped to physical database operations.

We have found this formulation to yield a lot of insight into possible optimizations for sensor data; these are presented in chapter 6. In the current chapter:

- we review the general inference problem (section 5.1),

- we introduce the relational algebra framework (section 5.2),
- we present a general outline for solving an inference query (section 5.3)
- we introduce sum-factor diagrams for visualizing solutions (section 5.4), and
- we review the conventional inference procedures using relational algebra (section 5.5).

5.1 The inference expression for Bayesian networks

An inference task, or inference query, partitions the Bayesian network nodes $\bar{V}$ into query nodes $\bar{Q}$, evidence nodes $\bar{E}$ and the remaining nodes $\bar{R}$. In this chapter, we make the distinction between nodes $\bar{V}$ and probabilistic variables $X_{\bar{V}}$ again; the latter are partitioned into $X_{\bar{Q}}$, $X_{\bar{E}}$ and $X_{\bar{R}}$. The inference task is to calculate $\mathsf{P}(x_{\bar{Q}}|x_{\bar{E}})$ for all values $x_{\bar{Q}}$, given certain values $x_{\bar{E}}$. This probability can be written in terms of the joint probability $\mathsf{P}(x_{\bar{V}}) = \mathsf{P}(x_{\bar{Q}}, x_{\bar{E}}, x_{\bar{R}})$:

$$
\begin{aligned}
& \mathsf{P}(x_{\bar{Q}}|x_{\bar{E}}) \\
= \quad & \text{definition of conditional probability} \\
& \frac{\mathsf{P}(x_{\bar{Q}}, x_{\bar{E}})}{\mathsf{P}(x_{\bar{E}})} \\
= \quad & \text{applying (MARGINALIZE)}^{\text{p. 15}} \text{ on both sides of the fraction bar} \\
& \frac{\sum_{x_{\bar{R}}} \mathsf{P}(x_{\bar{Q}}, x_{\bar{E}}, x_{\bar{R}})}{\sum_{x_{\bar{Q}}} \sum_{x_{\bar{R}}} \mathsf{P}(x_{\bar{Q}}, x_{\bar{E}}, x_{\bar{R}})}
\end{aligned}
$$

It is only necessary to calculate the outcome of the numerator for all $x_{\bar{Q}}$ 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 $\mathsf{P}(x_{\bar{Q}}, x_{\bar{E}})$:

$$
\mathsf{P}(x_{\bar{Q}}, x_{\bar{E}}) = \sum_{x_{\bar{R}}} \mathsf{P}(x_{\bar{V}}) \qquad \textbf{(INFERENCE)}
$$

The equation above suggests a naive way to perform this calculation: for each value $x_{\bar{Q}}$, determine the joint probability $\mathsf{P}(x_{\bar{Q}}, x_{\bar{E}}, x_{\bar{R}})$ at the observed $x_{\bar{E}}$ for all $x_{\bar{R}}$, and add these together. However, the time taken by this approach is exponential in $|\bar{Q} \cup \bar{R}|$, the number of unobserved variables: the sum is evaluated for each possible $x_{\bar{Q}}$ value (note: in case of multiple query variables, this means each possible combination, i.e. a $|\bar{Q}|$-tuple of values), and to each of these sums, all the $x_{\bar{R}}$ values (combinations) contribute a term. In the dynamic Bayesian networks used for sensor data processing, $|\bar{R}|$ is proportional to the number of time points taken into account, which can even grow indefinitely in a streaming setup; hence, this naive approach is out of the question.

Making use of the factorization (Fact-BN)[p. 19] of a Bayesian network's joint probability can reduce the inference cost dramatically, depending on the structure of the graph. In a dynamic Bayesian network, this structure is essentially a chain in the time dimension. As we will show in section 6.1, this allows the inference cost to be linear w.r.t. the number of time points, and therefore acceptable in a streaming setup. Substituting (Fact-BN) into (Inference) gives:

$$\mathsf{P}(x_{\bar{Q}}, x_{\bar{E}}) = \sum_{x_{\bar{R}}} \prod_{V \in \bar{V}} \mathsf{P}(x_V | x_{Parents(V)}) \qquad \textbf{(Inference-BN)}$$

Depending on the composition of the product, some factors can be pulled out of the summations due to the distributive laws (Σ-Distr-L) and (Σ-Distr-R)[p. 19]; it is sometimes suggested that this makes the expression more efficient to evaluate, and therefore forms the basis of efficient inference algorithms.

This statement alone, although not entirely untrue, 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 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, we have to introduce a notion of *sharing* or *storage* into the expressions. We rewrite the expression as a *program* consisting of assignments; each of these is evaluated only once, after which the result can be reused multiple times.

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

Both intermediate results μ_i are represented as a set-theoretic function—a set of key-value pairs $k \mapsto v$. In concrete programming language terms, a concept matching more closely to the intended semantics would be a *dictionary*; the idea is that the value v is calculated when μ_i is created, after which it can be looked up using $\mu_i(k)$.

However, in AI literature, μ_1 and μ_2 are usually implemented as *arrays*: a certain order is imposed on the keys, and the array consists only of the function values in this order. Often, these arrays are conceived as *messages* which are sent from one processing node to another.

Although the expression has an efficient evaluation when it is represented as a program, 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 the following program:

$$\begin{aligned}&\mu_3 \leftarrow \{\, b \mapsto \textstyle\sum_a \mathsf{P}(a)\mathsf{P}(b|a) \mid b \in \mathrm{dom}(B) \,\}\\&\text{return } \textstyle\sum_{b,c} \mu_3(b)\mathsf{P}(c|b)\end{aligned}$$

One first has to transform the programs back into single expressions, and then compare these.

As a better alternative, we advocate a *relational representation*, in which the basic building blocks of an expression are sets of values like μ_1, instead of single values like $\mu_1(b)$. Consider the following example:

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

This equation expresses both the above programs, and their equivalence, in one line. The two expressions have the same *denotational semantics*; we define these in terms of relational algebra, which is defined itself using set theory (section 5.2). Using well-known rewrite rules for relational algebra, the equivalence of the two expressions can be established; moreover, these rules can be used to derive new equivalent expressions.

Next to the denotational semantics, these expressions are also given an *operational semantics*. This can either be done in terms of arrays, or in terms of database operations; see section 5.2.3. This marks a clear difference compared to the initial situation described above, where the operational semantics are linked to the *programs* while denotational semantics are given to *expressions*.

5.2 Relational expressions for inference

In this section, we define our relational framework for inference expressions. We start by reviewing conventional relational algebra in section 5.2.1, followed in section 5.2.2 by a definition of the denotational semantics of the $p[\ldots]$, $\dot{\bowtie}$ and $\dot{\pi}$ operators specific for inference expressions. In section 5.2.3, operational semantics are discussed.

5.2.1 Relational algebra

As a basis for our semantics, we use a simple variant of relational algebra with an extended projection operator that can express aggregation (similar to SQL's *group by* construct). We have chosen for an algebra with $\bowtie$ as a basic operation instead of $\times$; when two relations have a common attribute, we always use it in a join condition. In fact, in our algebra it is impossible to do a cross-product in

such a case; it would require some formal renaming scheme to distinguish the two attributes in the result schema, which implies more effort than we want to impose on the reader. We use the following set-theoretic framework:

- We presuppose a set of attributes $\mathcal{A}ttr$ and a set of values $\mathcal{V}al$; each attribute $A \in \mathcal{A}ttr$ has a domain $\mathrm{dom}(A) \subseteq \mathcal{V}al$.
- A relation r consists of a schema, denoted as $\mathit{schema}(r)$, and a set of tuples.
- A relation's schema consists of a set of attributes: $\mathit{schema}(r) \in \wp\mathcal{A}ttr$.
- A relation's set of tuples is also simply denoted as r. Each tuple $t \in r$ is a function with $\mathrm{dom}(t) = \mathit{schema}(r)$, where $t(A) \in \mathrm{dom}(A)$ for each $A \in \mathit{schema}(r)$.

We define a natural join $\bowtie$:

$$\begin{aligned} & t \in (r \bowtie s) \\ \iff & \\ & \mathrm{dom}(t) = \mathit{schema}(r) \cup \mathit{schema}(s), \\ & (\mathit{schema}(r) \lhd t) \in r, \\ & (\mathit{schema}(s) \lhd t) \in s \end{aligned}$$

Here, $\lhd$ is the restriction of a function to a part of its domain:

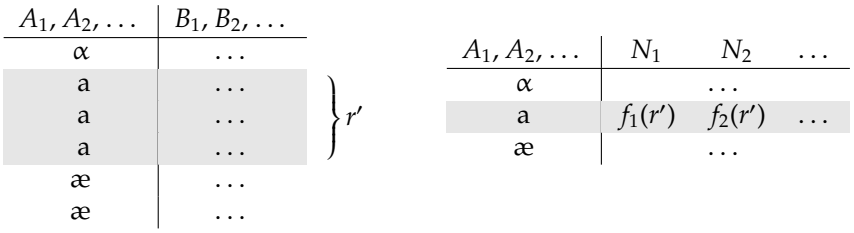
$$\bar{x} \lhd f \stackrel{\text{def}}{=} \{\, (x, y) \mid (x, y) \in f \wedge x \in \bar{x} \,\}$$

The generalized projection operator π performs projection and aggregation. It has two parameters: a list of attributes $A_1, A_2, \ldots$ from r onto which to project, and a list of aggregation functions and new attributes. An aggregation function f_i maps a relation to a value in $\mathrm{dom}(N_i)$. For each tuple in the result, it is applied to the set of tuples from r that agree with that tuple on the A values (see also figure 5.1).

$$\begin{aligned} & t \in \pi_{A_1, A_2, \ldots, f_1 \mapsto N_1, f_2 \mapsto N_2, \ldots} r \\ \iff & \\ & \exists r'.\ r' = \{\, t' \mid t' \in r,\ \{A_1, A_2, \ldots\} \lhd t' = \{A_1, A_2, \ldots\} \lhd t \,\}, \\ & \quad r' \neq \emptyset, \\ & \quad \mathrm{dom}(t) = \{A_1, A_2, \ldots, N_1, N_2, \ldots\}, \\ & \quad t(N_i) = f_i(r') \end{aligned}$$

The only aggregation function that we will use is SUM_X (where X is the attribute over which to sum):

$$\mathsf{SUM}_X(r) = \sum_{t \in r} t(X)$$



(a) relation r

(b) relation $\pi_{A_1,A_2,\ldots,f_1 \mapsto N_1, f_2 \mapsto N_2,\ldots} r$

Figure 5.1: The π operator in our algebra is extended with aggregation functionality. From the input relation r, each set of tuples $r' \subseteq r$ that agrees on the A_i attributes corresponds to one tuple in the result relation. Under the N_i attributes in this tuple, the aggregation functions f_i are applied to r'. If the $f_i \mapsto N_i$ list is empty, the operator corresponds to conventional projection.

The rename operator ρ replaces a tuple's attribute A_i with N_i (and requires that $N_i \notin \mathit{schema}(r)$ and $\mathrm{dom}(N_i) = \mathrm{dom}(A_i)$):

$$
\begin{aligned}
& t \in \rho_{A_1 \mapsto N_1, A_2 \mapsto N_2, \ldots} r \\
\iff & \\
& \exists t'.\ t' \in r, \\
& \quad t'(A_i) = t(N_i) \\
& \quad \forall A \in (\mathit{schema}(r) \setminus \{A_1, A_2, \ldots\}).\ t'(A) = t(A), \\
& \quad \mathrm{dom}(t) = (\mathit{schema}(r) \setminus \{A_1, A_2, \ldots\}) \cup \{N_1, N_2, \ldots\}
\end{aligned}
$$

To define a selection operator σ_θ, we need to model θ: a predicate such as $A_1 + A_2 = A_3$, in which attributes take the place of values. We model these predicates as functions of type $(\mathcal{A}ttr \to \mathcal{V}al) \to \mathbb{B}$: given a certain *binding* of type $\mathcal{A}ttr \to \mathcal{V}al$, they produce a Boolean value. We refer to the set of attributes in such an expression as its schema, and denote it $\mathit{schema}(\theta)$. We require that $\mathit{schema}(\theta) \subseteq \mathit{schema}(r)$.

$$
\begin{aligned}
& t \in \sigma_\theta r \\
\iff & \\
& t \in r,\ \theta(t)
\end{aligned}
$$

We will also need some less conventional operators. We define the *embodiment* of θ, denoted $[\![\theta]\!]$, which is the relation consisting of all the bindings that satisfy θ. Suppose $\mathit{schema}(\theta) = \{A_1, A_2, \ldots\}$, then:

$$
\begin{aligned}
& t \in [\![\theta]\!] \\
\iff & \\
& \mathrm{dom}(t) = \{A_1, A_2, \ldots\}, \theta(t),\ t(A_i) \in \mathrm{dom}(A_i)
\end{aligned}
$$

Likewise, we define the embodiment of a set of attributes as the set of all possible bindings of these attributes:

$$t \in [\![A_1, A_2, \ldots]\!] \iff \mathrm{dom}(t) = \{A_1, A_2, \ldots\},\ t(A_i) \in \mathrm{dom}(A_i)$$

Finally, we introduce $\bowtie_\theta$, a variant on the join operator that produces only tuples for which θ holds. It is not a primitive of the relational algebra and its denotational semantics are simply given by:

$$r \bowtie_\theta s \stackrel{\text{def}}{=} \sigma_\theta (r \bowtie s)$$

We include it because it can be implemented by very efficient 'physical operators'; see section 5.2.3. If θ is selective, these operators avoid creating a large intermediate result. We use it in the following rewrite rule:

$$r \bowtie [\![\theta]\!] = r \bowtie_\theta [\![\mathit{schema}(\theta) \setminus \mathit{schema}(r)]\!]$$

5.2.2 Operators for the inference expression

As we have mentioned in section 2.2, the cpd $\mathsf{P}(b|a)$ is actually a function (say, f): given a value b for probabilistic variable B and a value a for A, it produces a function value $f(b, a)$, which is a value between 0 and 1. It is commonly implemented (cf. section 5.1) as a multi-dimensional array in which all the function values are stored; application of the function f to values a and b corresponds to looking up the element of the array at indices a and b. In inference procedures, an often used operation is the 'multiplication' of such an array with another one, say $g(c, b) = \mathsf{P}(C = c|B = b)$. This multiplication operation $\circledast$ should be defined such that

$$(f \circledast g)(a, b, c) = f(b, a) * g(c, b) = \mathsf{P}(B = b|A = a) * \mathsf{P}(C = c|B = b)$$

Using the above function representations, it is impossible to define $\circledast$ in a generic way; in the above definition, we used the fact that the argument corresponding to variable B is the *first* one of f and the *second* one of g, and this 'meta-information' is not present in the functions (or arrays) themselves. Using the relational representation from the previous section, such a generic operator *is* possible, because relations are joined using the identity of the attributes instead of their order.

Before representing a probability distribution as a relation, we first represent a general expression ϕ as a relation. In this expression, attributes are again assumed to take the place of values: a possible expression might be $(A + B) * (A + C)$. Similar to the embodiment $[\![\ldots]\!]$ of predicates, we define an embodiment $[\![\ldots]\!]_{\mathsf{val}}$ of expressions, such that relation $[\![(A + B) * (A + C)]\!]_{\mathsf{val}}$ contains every possible binding of A, B, and C, together with the resulting value of the expression under a dedicated attribute val—see figure 5.2c for an example. In general:

$$[\![\phi]\!]_{\mathsf{val}} \stackrel{\text{def}}{=} [\![\phi = \mathsf{val}]\!] \qquad \textbf{(Repr-Expr)}$$

A	B	val
1	2	3
1	3	4

(a) $[\![A + B]\!]_{\mathsf{val}}$

A	C	val
1	10	11
1	100	101
1	1000	1001

(b) $[\![A + C]\!]_{\mathsf{val}}$

A	B	C	val
1	2	10	33
1	2	100	303
1	2	1000	3003
1	3	10	44
1	3	100	404
1	3	1000	4004

(c) $[\![(A + B) * (A + C)]\!]_{\mathsf{val}}$
$= [\![A + B]\!]_{\mathsf{val}} \overset{*}{\bowtie} [\![A + C]\!]_{\mathsf{val}}$

A	B	val
1	2	3339
1	3	4452

(d) $[\![\sum_{c\in\{10,100,1000\}} (A + B) * (A + c)]\!]_{\mathsf{val}}$
$= \dot{\pi}_{-C} [\![(A + B) * (A + C)]\!]_{\mathsf{val}}$
$= \dot{\pi}_{-C} ([\![A + B]\!]_{\mathsf{val}} \overset{*}{\bowtie} [\![A + C]\!]_{\mathsf{val}})$

Figure 5.2: Left: the embodiment of the expressions $A + B$ and $B + C$, with $\mathrm{dom}(A) = \{1\}$, $\mathrm{dom}(B) = \{2, 3\}$, and $\mathrm{dom}(C) = \{10, 100, 1000\}$. Middle: the embodiment of their product $(A+B)*(B+C)$ illustrates the equality (Repr-Mult). Right: the embodiment of a summation over this product illustrates (Repr-Sum).

A lot of the relations that we define hereafter will represent an expression and its possible values; we will refer to the non-val attributes of such a relation r as *regular* attributes, and use the notation $\mathit{regattr}(r)$:

$$\mathit{regattr}(r) \overset{\text{def}}{=} \mathit{schema}(r) \setminus \{\mathsf{val}\}$$

For every binding of ϕ, the expression has exactly one value val, so the regular attributes of $[\![\phi]\!]_{\mathsf{val}}$ form a *key* of this relation. Without proof, we mention that this holds for every other relation r in this thesis: $\mathit{regattr}(r)$ is a key.

We now consider the multiplication of two expressions. We need an operator $\overset{*}{\bowtie}$ such that

$$[\![\phi]\!]_{\mathsf{val}} \overset{*}{\bowtie} [\![\psi]\!]_{\mathsf{val}} = [\![\phi * \psi]\!]_{\mathsf{val}} \qquad \textbf{(Repr-Mult)}$$

In terms of the above relational algebra, this can be achieved by joining $[\![\phi]\!]_{\mathsf{val}}$ and $[\![\psi]\!]_{\mathsf{val}}$ on their common *regular* attributes, and multiply the values in their respective val attributes. Hence, the operator is defined as follows:

$$r \overset{*}{\bowtie} s \overset{\text{def}}{=} \pi_{\mathit{regattr}(r)\cup\mathit{regattr}(s),\, m\mapsto\mathsf{val}}(r \bowtie \rho_{\mathsf{val}\mapsto\mathsf{val}'} s)$$
$$m(\{t\}) = t(\mathsf{val}) * t(\mathsf{val}')$$

Note that we 'misuse' the extended projection π here to multiply the two values. The aggregation function m is only applied to 'groups' of one tuple, because the projection attributes $\mathit{regattr}(r) \cup \mathit{regattr}(s)$ form a key. As an illustration that (Repr-Mult) holds for this $\overset{*}{\bowtie}$ operator, consider fig. 5.2a–5.2c; we skip the formal proof. By (Repr-Mult), the identity element for $\overset{*}{\bowtie}$ is $[\![1]\!]_{\mathsf{val}}$, a relation with 1

attribute (val) and 1 tuple t for which $t(\mathsf{val}) = 1$:

$$[\![1]\!]_{\mathsf{val}} \overset{*}{\bowtie} r = r$$

For later use, we also define an efficient $\overset{*}{\bowtie}_\theta$ variant that uses $\bowtie_\theta$ instead of $\bowtie$:

$$r \overset{*}{\bowtie}_\theta s \overset{\text{def}}{=} \pi_{\mathit{regattr}(r) \cup \mathit{regattr}(s),\, m \mapsto \mathsf{val}}(r \bowtie_\theta \rho_{\mathsf{val} \mapsto \mathsf{val}'} s)$$
$$m(\{t\}) = t(\mathsf{val}) * t(\mathsf{val}')$$

The following operator we define is $\overset{+}{\pi}_{-\{A_1,A_2,\ldots\}}$. Given a relation representing the expression ϕ, it should produce the sum of the values of ϕ for all possible bindings of a set of attributes $\{A_1, A_2, \ldots\}$:

$$\overset{+}{\pi}_{-\{A_1,A_2,\ldots\}} [\![\phi]\!]_{\mathsf{val}} = [\![\textstyle\sum_{a_1 \in \mathrm{dom}(A_1)} \sum_{a_2 \in \mathrm{dom}(A_2)} \cdots \phi[a_1/A_1, a_2/A_2, \ldots]]\!]_{\mathsf{val}} \quad \textbf{(Repr-Sum)}$$

We define the operator (and a variant of it) as follows:

$$\overset{+}{\pi}_{-\{A_1,A_2,\ldots\}} r \overset{\text{def}}{=} \pi_{\mathit{regattr}(r) \setminus \{A_1,A_2,\ldots\},\, \mathsf{SUM}_{\mathsf{val}} \mapsto \mathsf{val}} r$$
$$\overset{+}{\pi}_{\{A_1,A_2,\ldots\}} r \overset{\text{def}}{=} \pi_{\{A_1,A_2,\ldots\},\, \mathsf{SUM}_{\mathsf{val}} \mapsto \mathsf{val}} r$$

The second variant does not mention the attributes that are summed out (projected out), but those that remain. While the first variant bears a direct relation to the $\sum$ operator as defined in the equations above, the second variant appeals to the notion of projecting a multi-dimensional distribution onto the mentioned subset of its dimensions, i.e. deriving $\mathsf{P}(x_A, x_B)$ from $\mathsf{P}(x_A, x_B, x_C)$.

Due to equations (Repr-Sum) and (Repr-Mult), every expression consisting of $\sum$ and $*$ operators—and in particular, the inference expression—can be represented using the bulk operators $\overset{+}{\pi}$ and $\overset{*}{\bowtie}$ instead. A consequence of this is that the same equalities apply:

- In a multi-dimensional sum $\sum_a \sum_b f(a,b,c)$ order is of no importance (we might also write it $\sum_b \sum_a f(a,b,c)$ or $\sum_{a,b} f(a,b,c)$), so this also holds for the first variant of $\overset{+}{\pi}$: $\overset{+}{\pi}_{-A} \overset{+}{\pi}_{-B} r = \overset{+}{\pi}_{-B} \overset{+}{\pi}_{-A} r = \overset{+}{\pi}_{-\{A,B\}} r$.
- The normal multiplication operator $*$ is associative and commutative, so we can unambiguously write $\phi * \chi * \psi$ and $\prod_{j \in J} \phi(j)$, where $\phi(j)$ is an expression parametrized by j. Consequently, we can also write $r \overset{*}{\bowtie} s \overset{*}{\bowtie} t$ and $\overset{*}{\bigbowtie}_{V \in \bar{V}} p[V]$.

In the example we just gave, we used a parametrized relation $p[V]$. We will use this notation to denote the relation containing the probability distribution over X_V. Likewise, we denote relations containing conditional probability distributions using $p[V|W_1, W_2, \ldots]$. These relations comprise most of the atomic components of relational expressions that we will use from now on.

They are formally defined in terms of an existing probabilistic model with variables X_V, X_{W_1}, et cetera, in conformance with our formal definition of Bayesian

$p[B|A] =$

B	A	val
false	*false*	0.12
true	*false*	0.88
false	*true*	0.25
true	*true*	0.75

Figure 5.3: The relation $p[B|A]$, with $\mathrm{dom}(A) = \mathrm{dom}(B) = \{false, true\}$ and $\mathsf{P}(X_B{=}false|X_A{=}false) = 0.12$, $\mathsf{P}(X_B{=}true|X_A{=}false) = 0.88, \ldots$

networks (section 2.2). In the relational representation, we represent a variable X_V by attribute V, with $\mathrm{dom}(V) = \mathrm{dom}(X_V)$. We define $p[V|W_1, W_2, \ldots]$ as the embodiment of the probability distribution:

$$p[V|W_1, W_2, \ldots] \stackrel{\text{def}}{=} [\![\mathsf{P}(X_V = V|X_{W_1} = W_1, X_{W_2} = W_2, \ldots)]\!]_{\mathsf{val}} \qquad \textbf{(Repr-Prob)}$$

Note: in the $[\![\ldots]\!]_{\mathsf{val}}$ expression, the attributes that have to be bound to a value are V, W_1, W_2; not the probabilistic variables. Also, when substituting a binding for attribute V, the V in the probabilistic variable name X_V is not affected! For an example of such a relation, see figure 5.3.

The relation $p[V, W]$ for unconditional probabilities is defined analogously. Most of the times, however, the atomic relations we use are the cpds $\mathsf{P}(x_V|x_{Parents(V)})$ that make up a Bayesian network. If from the context it is known which Bayesian network is meant, and what the parents of variable V are, we use the following shorthand:

$$cpd[V] \stackrel{\text{def}}{=} p[V|Parents(V)]$$

The operators introduced in this section make it possible to write probabilistic statements in terms of relational expressions. For example, the independence of X_A and X_B can be written:

$$\begin{aligned}
&\mathsf{P}(X_A{=}A, X_B{=}B) = \mathsf{P}(X_A{=}A)\mathsf{P}(X_B{=}B) \quad \text{for all } A \in \mathrm{dom}(X_A), B \in \mathrm{dom}(X_B)\\
\equiv\ &\text{by (Repr-Expr) and the definition of } [\![\ldots]\!]\\
&[\![\mathsf{P}(X_A{=}A, X_B{=}B)]\!]_{\mathsf{val}} = [\![\mathsf{P}(X_A{=}A)\mathsf{P}(X_B{=}B)]\!]_{\mathsf{val}}\\
\equiv\ &\text{by (Repr-Mult)}\\
&[\![\mathsf{P}(X_A{=}A, X_B{=}B)]\!]_{\mathsf{val}} = [\![\mathsf{P}(X_A{=}A)]\!]_{\mathsf{val}} \bowtie [\![\mathsf{P}(X_B{=}B)]\!]_{\mathsf{val}}\\
\equiv\ &\text{by (Repr-Prob)}\\
&p[A, B] = p[A] \bowtie p[B]
\end{aligned}$$

We will apply this to the inference expression in section 5.3.

5.2.3 Operational semantics

In the previous sections, we have defined a *denotational* semantics for $p[\ldots]$, $\stackrel{+}{\pi}$ and $\bowtie$ in terms of a relational algebra, itself defined in terms of set theory. The

$$
\begin{aligned}
(\!| r |\!) &= \text{return } \{\, (t(A), t(B)) \mapsto t(\mathsf{val}) \mid t \in r \,\} \\
(\!| r \overset{*}{\bowtie} s |\!) &= \mu_1 \leftarrow (\!| r |\!) \\
&\quad\ \mu_2 \leftarrow (\!| s |\!) \\
&\quad\ \text{return } \{\, (a,b,c) \mapsto v * v' \mid ((a,b) \mapsto v) \in \mu_1, ((b',c') \mapsto v') \in \mu_2, b{=}b' \,\} \\
(\!| \overset{+}{\pi}_{-B} r |\!) &= \mu \leftarrow (\!| r |\!) \\
&\quad\ \text{return } \{\, a \mapsto \textstyle\sum_b \{\, v \mid ((a,b) \mapsto v) \in \mu \,\} \mid a \in \mathrm{dom}(A) \,\} \\
(\!| \overset{+}{\pi}_{-B} (r \overset{*}{\bowtie} s) |\!) &= \mu_1 \leftarrow (\!| r |\!) \\
&\quad\ \mu_2 \leftarrow (\!| s |\!) \\
&\quad\ \text{return } \Big\{\, (a,c) \mapsto \\
&\qquad\qquad \textstyle\sum_b \{\, v * v' \mid ((a,b) \mapsto v) \in \mu_1, ((b',c') \mapsto v') \in \mu_2, b{=}b' \,\} \\
&\qquad \Big|\, a \in \mathrm{dom}(A), c \in \mathrm{dom}(C) \,\Big\}
\end{aligned}
$$

Figure 5.4: Operational semantics in terms of arrays; example for relations r and s with $\mathit{schema}(r) = \{A, B\}$ and $\mathit{schema}(s) = \{B, C\}$. The operational semantics $(\!| r |\!)$ of a relation r is a program that, when evaluated, returns a multidimensional array that can be stored, represented here by a set with key-value pairs $(k_1, k_2, \ldots) \mapsto v$ as elements; the key represents the position in the array and is not stored. The different k_i elements are the values of the (alphabetically ordered) regular attributes of a tuple in r; v is the value of its val attribute. To evaluate a composite expression, its subexpressions are evaluated first. Next, the new array is built; this is described here as a set comprehension, and can be implemented as a loop or vector operation. The last translation, for expressions of the form $\overset{+}{\pi}(\ldots \overset{*}{\bowtie} \ldots)$, is optional, and is an optimized variant that does not store the intermediate result. Note: only relations that contain a tuple for every possible key can be represented as an array.

advantage of relational expressions over numeric expressions is that they can be straightforwardly assigned an *operational* semantics—a mapping to machine instructions—without redundant calculations. By virtue of their operational semantics, two expressions that are denotationally the same, like $\overset{+}{\pi}_{-B} (p[A] \overset{*}{\bowtie} p[B|A])$ and $p[A] \overset{*}{\bowtie} \overset{+}{\pi}_{-B} p[B|A]$, can translate to different machine instructions, and have a different cost in terms of processing time or storage space. This means that an expression can be *optimized* by finding an equivalent expression with the lowest cost.

Although we do not commit to a specific operational semantics in this thesis, we indicate some general ways of providing one:

- The $p[\ldots]$ relations can be represented by *arrays*. For each tuple, at the *index* determined by the regular attributes, the *value* of the val attribute is stored. The semantics of the $\overset{+}{\pi}$ and $\overset{*}{\bowtie}$ operators are sketched in figure 5.4. With these semantics, relational expressions are translated into

programs similar to those in section 5.1. For example, the expression $\dot{\pi}_{-A}\left(p[A] \bowtie \dot{\pi}_{-B}\left(p[B|A] \bowtie \dot{\pi}_{-C} p[C|B]\right)\right)$—which can be directly obtained from $\sum_a \mathsf{P}(a) \sum_b \mathsf{P}(b|a) \sum_c \mathsf{P}(c|b)$ by applying (Repr-Mult) and (Repr-Sum)—roughly translates into the first example program in that section.

- The relations can be represented like in a conventional database. The operators $p[\ldots]$, $\dot{\pi}$ and $\bowtie$ are then translated into relational operators according to their definitions in section 5.2.2. Next, the *logical operators* from this expression are translated into *physical operators*; see any database textbook for details. The base relations $p[\ldots]$ can be stored (on disk or in memory) in a way that is optimal for these physical operators.
- As a variation on this, specific physical operators can be implemented for $\bowtie$ and $\dot{\pi}$, so the first translation can be skipped.

As a possible improvement to the database semantics, we mention that it is also possible to use a representation in which tuples with $\mathsf{val} = 0$ are not stored at all. We will discuss this in detail in section 6.2.1.

5.3 Rewriting the inference expression

Using the tools developed in section 5.2, the inference expression for a Bayesian network (Inference-BN)[p. 67] is rewritten into a relational expression:

$$\begin{aligned} & \mathsf{P}(x_{\bar{Q}}, x_{\bar{E}}) = \textstyle\sum_{x_{\bar{R}}} \prod_{V \in \bar{V}} \mathsf{P}(x_V | x_{Parents(V)}) \qquad \text{for all } x_{\bar{Q}}, x_{\bar{E}} \\ \equiv \quad & \text{by (Repr-Prob), (Repr-Mult), (Repr-Sum)} \\ & p[\bar{Q}, \bar{E}] = \dot{\pi}_{-\bar{R}} \overset{*}{\bowtie}_{V \in \bar{V}} cpd[V] \end{aligned}$$

However, when performing an inference query, we are not interested in the answer for *all* $x_{\bar{E}}$; we are interested in one particular $x_{\bar{E}}$ value. In our original discussion of inference queries, 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. This is performed by a selection $\sigma_{\bar{E}=x_{\bar{E}}}$:

$$\sigma_{\bar{E}=x_{\bar{E}}} p[\bar{Q}, \bar{E}] = \sigma_{\bar{E}=x_{\bar{E}}} \dot{\pi}_{-\bar{R}} \overset{*}{\bowtie}_{V \in \bar{V}} cpd[V]$$

It is also useful to project out the $\bar{E}$ attributes after this selection. They do not provide any information anymore, and when the expression is rewritten it simplifies intermediate schemas; as we will show shortly, the selections and projections regarding the $\bar{E}$ variables can be pushed down to the leaves of the expression.

Note that projecting out the $\bar{E}$ attributes does not reduce the cardinality of the relation; due to the preceding selection, there is only one possibility for the $\bar{E}$ values of a tuple. Because of this, the projection can also be written as $\ddot{\pi}_{-\bar{E}}$, where each summation is over a group of one tuple. Thus, we apply the operator $\dot{\pi}_{-\bar{E}}$

to both sides of the above equation. On the right hand side, it can be combined with the already present $\dot{\pi}_{-\bar{R}}$ operator into $\dot{\pi}_{\bar{Q}}$, which gives a shorter expression. On the left hand side, it can also be written as $\dot{\pi}_{\bar{Q}}$:

$$\dot{\pi}_{\bar{Q}}\sigma_{\bar{E}=x_{\bar{E}}}p[\bar{Q},\bar{E}] = \dot{\pi}_{\bar{Q}}\sigma_{\bar{E}=x_{\bar{E}}}\overset{*}{\bowtie}_{V\in\bar{V}} cpd[V] \qquad \textbf{(Inference-BN-Rel)}$$

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

> *Efficient inference in a Bayesian network boils down to rewriting the right hand side of the above equation into an efficient expression.*

Instead of applying this statement to a specific Bayesian network, we can also apply it one level of abstraction higher: then we are talking about a *procedure* that produces an efficient rewriting for *any* Bayesian network. This is the level on which the conventional inference procedures operate; however, in the way they are usually presented, these do not only produce such an expression, but also execute it.

In chapter 6, we show that variable elimination and junction tree propagation can easily be interpreted in a way in which they produce a relational expression; its execution can be deferred to a later stage, possibly after further rewriting. There are some things to say in favor of such a separation of concerns:

- The resulting expression can be 'physically' optimized by choosing efficient join algorithms, appropriate indices and table organizations—concerns which are presumably too low-level for inference procedures.
- Inference algorithms can be used on *parts* of a Bayesian network, after which the resulting expressions are connected. We show this in section 6.1.

But perhaps the largest advantage of viewing inference as 'merely' query rewriting lies not in the use of conventional inference algorithms, but in finding new optimizations. This can in principle be done without any probabilistic knowledge: any rewriting of the expression is correct as long as it respects the relational algebra equivalences. Of course, optimizations can also be guided by probabilistic intuition, and subsequently checked for correctness using relational equivalences.

In the next chapter, we discuss specific rewritings; the remainder of this chapter contains general remarks about rewriting the right hand side expression. Firstly, we note that multi-way $\overset{*}{\bowtie}$ joins usually do not exist as physical operators; to give the inference expression an operational semantics, they have to be rewritten into a parenthesized expression of binary $\bowtie$ joins. In query optimization literature, this is called *join ordering*, and as we will argue, it is the most challenging task in inference optimization. After doing this, the $\dot{\pi}_{-\bar{V}_E}$, $\sigma_{\bar{V}_E=\bar{v}_E}$ and $\dot{\pi}_{-\bar{V}_R}$ operators can be pushed into the expression, following relational algebra rewrite rules. The parenthesized expression can be imagined as a *join tree* with the root on top; thus, operators are said to be pushed *down the join tree*.

The rewrite rules relevant to an evidence variable $E \in \bar{E}$ are the following:

$$\dot{\pi}_{-E}\sigma_{E=x_E}(r \dot{\bowtie} s) = \begin{cases} \dot{\pi}_{-E}\sigma_{E=x_E} r \dot{\bowtie} s & \text{if } E \in \mathit{schema}(r),\ E \notin \mathit{schema}(s) \\ r \dot{\bowtie} \dot{\pi}_{-E}\sigma_{E=x_E} s & \text{if } E \notin \mathit{schema}(r),\ E \in \mathit{schema}(s) \\ \dot{\pi}_{-E}\sigma_{E=x_E} r \dot{\bowtie} \dot{\pi}_{-E}\sigma_{E=x_E} s & \text{if } E \in \mathit{schema}(r),\ E \in \mathit{schema}(s) \end{cases}$$

(Distr-Evidence)

As noted before, a $\dot{\pi}_{-E}$ aggregation operator after $\sigma_{E=x_E}$ is equivalent to a π_{-E} operator; with this in mind, the first two cases are basic relational algebra equivalences. The only interesting case is the third one. On the lhs, the $\dot{\bowtie}$ operator on joins tuples from r and s that agree on E, but on the rhs this attribute is projected out earlier; therefore, the rhs $\dot{\bowtie}$ operator could potentially join tuples that did not agree on E before it was projected out. However, the selection predicate on both sides makes sure they do.

In summary, these rewrite rules imply that given any join tree of an inference expression, the $\dot{\pi}_{-E}\sigma_{E=x_E}$ operators for any evidence variable $E \in \bar{E}$ can be pushed down all the way until the leaves (cpds) that contain E. Applying this for all $\bar{E}$ variables, the inference expression reads

$$\dot{\pi}_{\bar{Q}}\sigma_{\bar{E}=x_{\bar{E}}} p[\bar{Q}, \bar{E}] = \dot{\pi}_{\bar{Q}} \dot{\bigbowtie}_{V \in \bar{V}} \mathit{cpd}_{-\bar{E}}[V] = \dot{\pi}_{-\bar{R}} \dot{\bigbowtie}_{V \in \bar{V}} \mathit{cpd}_{-\bar{E}}[V]$$

in which we use the abbreviation

$$\mathit{cpd}_{-\bar{E}}[V] \stackrel{\text{def}}{=} \dot{\pi}_{-\bar{L}}\sigma_{\bar{L}=x_{\bar{L}}} \mathit{cpd}[V] \qquad \text{where } L = \bar{E} \cap \mathit{schema}(\mathit{cpd}[V]).$$

We will use this expression as starting point for the conventional inference algorithms in section 5.5; by default, we assume that pushing down these operators yields a more efficient expression. However, a notable exception to this is discussed in section 6.1.

Regarding the variables $R \in \bar{R}$, the $\dot{\pi}_{-R}$ operators can also be pushed down, but generally not as far as for the $\bar{E}$ variables. The distributivity properties (Σ-Distr-L) and (Σ-Distr-R)[p. 19] can be written in a relational representation using (Repr-Sum) and (Repr-Mult)[p. 72], which provides us with the relevant equivalences:[1]

$$\dot{\pi}_{-A}(r \dot{\bowtie} s) = \dot{\pi}_{-A} r \dot{\bowtie} s \qquad \text{if } A \notin \mathit{schema}(s) \qquad \textbf{(Distr-L)}$$

$$\dot{\pi}_{-A}(r \dot{\bowtie} s) = r \dot{\bowtie} \dot{\pi}_{-A} s \qquad \text{if } A \notin \mathit{schema}(r) \qquad \textbf{(Distr-R)}$$

When r and s both contain A, in general $\dot{\pi}_{-A}(r \dot{\bowtie} s) = \dot{\pi}_{-A} r \dot{\bowtie} \dot{\pi}_{-A} s$ does not hold. For example, consider the case where $r = s = [\![A]\!]_{\mathsf{val}}$ with $\mathrm{dom}(A) = \{1, 2\}$: the lhs would evaluate to a single tuple with $\mathsf{val} = 1^2 + 2^2$ and the rhs to one with $\mathsf{val} = (1 + 2)^2$.

For an illustration of (Distr-R), consider the relation in figure 5.2d, which corresponds to the expression $\dot{\pi}_{-C}([\![A + B]\!]_{\mathsf{val}} \dot{\bowtie} [\![A + C]\!]_{\mathsf{val}})$. Applying (Distr-R)

[1] These also hold when the conventional relational operators π and $\bowtie$ are used instead.

to this expression says that it equals $[\![A + B]\!]_{\mathsf{val}} \bowtie^{*} \dot{\pi}_{-C} [\![A + C]\!]_{\mathsf{val}}$:

A	B	val
1	2	3339
1	3	4452

$=$

A	B	val
1	2	3
1	3	4

$\bowtie^{*}$

A	val
1	1113

Coming back to the inference expression $\dot{\pi}_{-\bar{R}} \bowtie^{*}_{V \in \bar{V}} cpd_{-\bar{E}}[V]$, we can again make the simplifying assumption that it is generally desirable to push down $\dot{\pi}$ operators as far as possible, because they always reduce the number of tuples. However, in making this assumption we ignore, for example, the fact that the cost of $\dot{\pi}_{-R} r$ depends on the physical *order* of r, that the cost of later operators is affected by the order that $\dot{\pi}_{-R}$ produces, and that it may save costs to join $\dot{\pi}_{-A}$ and $\dot{\pi}_{-B}$ into $\dot{\pi}_{-AB}$. In this thesis, we do not give any further thought to these considerations.

Thus, for all $R \in \bar{R}$, we push down $\dot{\pi}_{-R}$ until we encounter a join $r \bowtie s$ for which $R \in schema(r) \wedge R \in schema(s)$. The expression tree obtained in that way can also be formed by adding 'project onto' operators to the tree instead of pushing down 'project away' operators. For each subtree T:

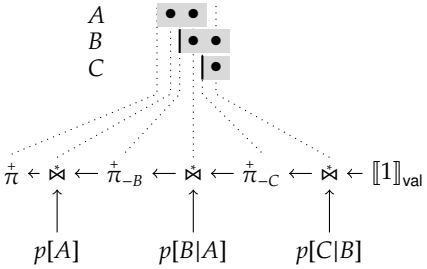
- Let $v(T)$ be the set of variables in the schemas of the $cpd_{-\bar{E}}[\ldots]$ relations in T's leaves (note: these do not contain $\bar{E}$ variables anymore).
- Let $w(T)$ be the union of $\bar{Q}$ and the variables in the $cpd_{-\bar{E}}[\ldots]$ relations in the rest of the tree.
- Replace T with $\dot{\pi}_{v(T) \cap w(T)}$.

However, thus far we have neglected the most challenging part: to find the join ordering. In the next section, we *visualize* this problem using sum-factor diagrams.

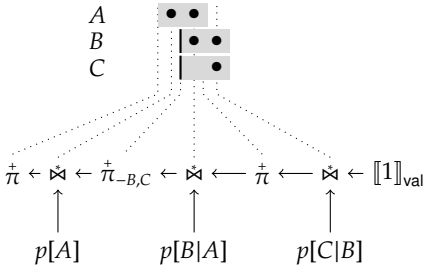
5.4 Sum-factor diagrams

The schemas of the intermediate relations in an inference expression provide important information about the cost of evaluating the expression. This is especially true if intermediate relations are stored as multi-dimensional arrays, like in the operational semantics in figure 5.4; the size of such an array relates exponentially to the number of dimensions, i.e. the size of the schema. This is the reason why the heuristics in the conventional inference procedures aim to reduce the size of the largest intermediate schema (see section 5.5.4). As a tool for visualizing these intermediate schemas and visually rewriting the expression, we introduce the *sum-factor diagram*.[2]

[2] We originally introduced sum-factor diagrams[24] for inference expressions in the conventional notation, i.e. *summations* over *factors*; now we use them for their relational representations. They can also be used to visualize and rewrite a right-deep expression of conventional $\bowtie$ and π operators.



(a) Sum-factor diagram and expression tree for $p[A] \dot{\bowtie} \dot{\pi}_{-B}(p[B|A] \dot{\bowtie} \dot{\pi}_{-C} p[C|B])$.



(b) Sum-factor diagram and expression tree for $p[A] \dot{\bowtie} \dot{\pi}_{-B,C}(p[B|A] \dot{\bowtie} p[C|B])$. The third row of the diagram shows that $\dot{\pi}_{-C}$ can be pushed one step to the right, resulting in figure 5.5a.

Figure 5.5: Sum-factor diagram (top) and expression tree (bottom) for two equivalent expressions. The vertical axis lists the variables in the expression: A,B,C. The horizontal axis corresponds to the intermediate relations in the expression, as shown by the dotted lines. The variables in the schema of a $\dot{\bowtie}$ result are shown in gray; those in the schema of the left join operand are shown as dots; those removed by a $\dot{\pi}$ operator are shown as a vertical bar. For symmetry reasons, $\dot{\pi}$ operators are added above each $\dot{\bowtie}$, and $\dot{\bowtie}\ [\![1]\!]_{\mathsf{val}}$ is added at the bottom of the tree.

5.4.1 Sum-factor diagrams for right-deep expressions

A sum-factor diagram shows all the intermediate schemas in a *right-deep* binary $\dot{\bowtie}$-tree with $\dot{\pi}$ operators above the joins. These schemas are arranged from left to right roughly in the same order as the relations occur in the expression, as is shown in figure 5.5. The regular attributes of the schemas are arranged on the vertical axis; the val attribute is not shown, but is present in each schema. To construct a sum-factor diagram:

1. List the expression's regular attributes on the vertical axis, in any order.

2. For each base relation (cpd), add a column to the diagram, in the same order as the relations occur in the textual representation of the expression. Put a dot in this column for each regular attribute (variable) that occurs in the relation.

3. For each $\dot{\pi}_{-\{A_1,A_2,\ldots\}}$ operator, add a vertical line for each attribute A_i that is summed *out*. In the textual representation, the $\dot{\pi}$ operator occurs between two base relations; the vertical lines should be placed between the two corresponding columns.

4. For each row in the diagram, color a contiguous group of cells: from *right to left*, start with the first cell that contains a dot, and end when you encounter

a vertical line. If there is none, continue coloring until the left edge of the diagram.

In sum-factor diagrams, *evidence variables are ignored*, i.e. omitted from the vertical axis. As explained above, they can always be projected out immediately, and play no role in the cost of an expression. If evidence variables are left out, the running intersection probability (see section 5.5.4) holds for each variable in the diagram, which can immediately be seen by the fact that there is an uninterrupted row of gray cells for that variable.

Sum-factor diagrams also provide a visual way to *rewrite* a right-deep expression into an equivalent one. Pushing $\dot{\pi}_{-A}$ down the join tree corresponds to moving the vertical line in row A to the right; due to (Distr-R)$^{\text{p. 78}}$, the line cannot be moved over dots. As we have mentioned before, we usually assume that it is desirable to push all $\dot{\bowtie}$ operators as far to the right as possible. For example, in figure 5.5b, the line in row C can only be moved one step to the right (resulting in figure 5.5a), but no further.

However, the challenge in rewriting an inference expression lies not in pushing down $\dot{\pi}$ operators down a tree, but in determining the right structure (join order) for that tree—one that allows a lot of $\dot{\pi}$ operators to be pushed down far, reducing the size of the intermediate schemas. We first assume trees without $\dot{\pi}$ operators; for a right-deep expression, each possible tree corresponds to a permutation of the base relations. Due to associativity and commutativity of $\dot{\bowtie}$, all permutations are valid. After picking a permutation, the $\dot{\pi}$ operators can be pushed down.

Thus, the space of right-deep plans for a given inference expression is visualized by all the permutations of the columns of dots, and vertical lines that are always to the left of the dots.

5.4.2 Extended sum-factor diagrams for bushy expressions

Although not as useful as for right-deep expressions, sum-factor diagrams can also represent bushy expressions. The horizontal axis of the diagram then (still) corresponds to the rightmost path through the expression tree; the left operand of a join on this path, which in a right-deep expression is always a base relation, can now also be a subtree. The schema of the join result is still shown using gray cells in the corresponding column; the schema of the subtree is still indicated by dots; however, the extra variables that occur in intermediate schemas within the subtree (and have been projected out) are now also shown, using struck out dots. See figure 5.6. Because the sum-factor diagram does not show the inner structure of the subtree, a sub-diagram can be drawn for it.

The rule that the vertical lines in the diagram can be pushed to the right until a dot is met, is now amended with the following clause: if this dot is the only one of that row, the line can be drawn through the dot, and is pushed into the sub-diagram.

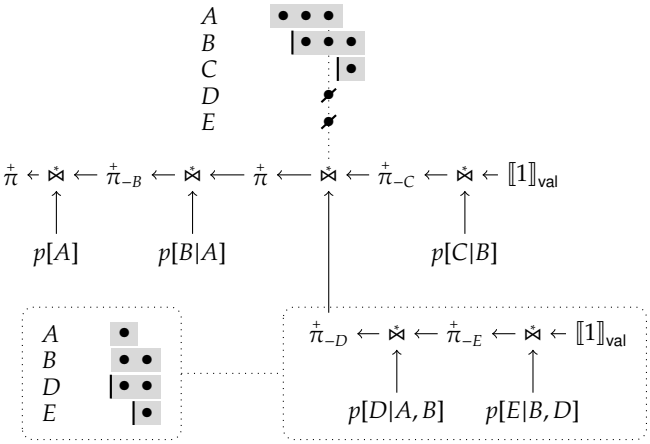


Figure 5.6: The extended sum-factor diagram (top) represents the bushy expression $p[A] \overset{*}{\bowtie} \overset{+}{\pi}_{-B}(p[B|A] \overset{*}{\bowtie} (\overset{+}{\pi}_{-D}(p[D|A,B] \overset{*}{\bowtie} \overset{+}{\pi}_{-E} p[E|D,B]) \overset{*}{\bowtie} \overset{+}{\pi}_{-C} p[C|B]))$. The dotted column of this diagram corresponds to the subtree in the dotted rectangle. This subtree can itself be represented by another sum-factor diagram, shown in the lower left corner.

5.5 Conventional inference procedures

We will review two well-known inference procedures from the AI literature in the relational framework, which means that we present a view that slightly differs from the conventional one. In the conventional view, an inference procedure takes as input a Bayesian network, a set of query variables X_Q and a set of evidence x_E, and outputs the probability distribution $\mathsf{P}(x_Q, x_E)$ (or, equivalently, $\mathsf{P}(x_Q|x_E)$, as we explained above). In our view, the output is a relational algebra expression evaluating to that distribution. However, this does not affect the essence of the procedures.

As we will explain in section 5.5.4, the heuristics guiding these inference procedures try to minimize the dimensionality (the number of attributes) of the intermediate relations. In chapter 6, we will mostly work with a sparse representation of relations, where dimensionality is not important. Nevertheless, we present these procedures here for a more complete view.

5.5.1 Variable elimination

The procedure known as *variable elimination*[61] or *bucket elimination*[18] produces an inference expression given an *elimination order* of the $\bar{R}$ variables. This order, which is usually derived from the structure of the Bayesian network using heuristics, determines the quality of the produced expression; heuristics for a good elimination order are discussed in section 5.5.4. In simple cases, it is also possible

to manually determine a good order.

The procedure (in relational terms) is given by Alg. 2. It processes the $\bar{R}$ variables in the given order; in each step i it joins all the relations containing R_i, making its result available for later steps.

As input for the algorithm, we use the cpd relations from (INFERENCE-BN-REL), with the $\overset{+}{\pi}_{-\bar{E}}\sigma_{\bar{E}=x_{\bar{E}}}$ operators already pushed down, as we discussed in section 5.3:

$$\bar{s} = \left\{ cpd_{-\bar{E}}[V] \;\middle|\; V \in \bar{V} \right\}$$

Equivalently, we could have used the set $\{ cpd[V] \mid V \in \bar{V} \}$ as input, and afterward push the $\overset{+}{\pi}_{-\bar{E}}\sigma_{\bar{E}=x_{\bar{E}}}$ operators down.

In a recent article[15], it has been shown that variable elimination can be improved by afterward pushing down $\overset{+}{\pi}$ operators where possible.

Input:

- set of relations $\bar{s}$
- variable elimination order $R_1, \ldots, R_m$ on the $\bar{R}$ variables

Output: an expression e equivalent to $\pi_{-\bar{R}} \overset{*}{\underset{s\in\bar{s}}{\bowtie}} s$

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

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

Algorithm 2: Variable elimination in a relational representation.

5.5.2 Junction tree propagation

Junction tree propagation[45, 34] is actually a technique to do several inference queries together—originally $\sigma_{\bar{E}=x_{\bar{E}}}p[V, \bar{E}]$ for each $V \in \bar{V}$—which reuses intermediate results so that it only takes twice as much time as doing a single one of these queries. In our initial formulation, however, we take one set of query variables $\bar{Q}$ as our inference goal. In section 5.5.3, we will show how the procedure is extended to handle multiple inference queries by sharing common relational subexpressions.

Like with variable elimination, we use as input the set $\bar{s}$ of cpds with the evidence selected and evidence variables projected away. (In most versions of

junction tree propagation, the evidence is only taken into account *after* the tree has been constructed. This approach has advantages when performing multiple inference queries with *different evidence*. However, with fixed evidence, taking this into account can lead to a better triangulation[45].)

As its second input, the junction tree procedure takes an *acyclic hypergraph cover* of the hypergraph (a set of sets) formed by the schemas (regular attributes) of $\bar{s}$; we explain what this is in section 5.5.4. Equivalently[57], and most common in AI literature, one can take the maximal cliques in the *triangulation* of the primal graph of this hypergraph; this primal graph can be derived from the Bayesian network by 'marrying' the parents of each node (i.e. adding an edge between each pair of parents), dropping the direction of the arrows, and in our case also removing the evidence variables.

In the junction tree procedure, it is the triangulation that determines the quality of the inference expression and is derived heuristically. Actually, it has a lot in common with finding a variable elimination order, as we explain in section 5.5.4.

The procedure is shown in Alg. 3. It first constructs a *maximal spanning tree* between the hyperedges (cliques), using the size of their intersections as weights. From this tree (which has sets of attributes as nodes), it takes a node containing $\bar{Q}$ as root, and transforms it into an inference expression by:

- placing a $\dot{\pi}$ operator on each edge
- adding each relation $s \in \bar{s}$ as a child of a node containing its regular attributes
- replacing all clique nodes by $\overset{*}{\bowtie}$ operators
- adding $\dot{\pi}_{\bar{Q}}$ above the root

An example is shown in figure 5.7.

5.5.3 Junction tree propagation for multiple queries

In Alg. 3, whatever $\bar{Q}$ is chosen—as long $\bar{Q} \subseteq \bar{C}_q$ for some q—the same spanning tree can be used, the same $\dot{\pi}$ operators are added, and the same $\{s, \bar{C}_i\}$ edges are added. The only differences are the location of the root of the tree and the $\dot{\pi}_{\bar{Q}}$ operator. As a consequence, there are a lot of common subexpressions in the inference expressions for different $\bar{Q}$ variables.

The original junction tree algorithm[45] takes advantage of this. By storing and reusing the values of subexpressions, it can perform n inference queries (n is the number of nodes in the spanning tree) in twice the time of one query. The algorithm picks an arbitrary root of the spanning tree, and traverses each edge $\bar{C}_i \mapsto \bar{C}_j$ in the spanning tree twice: first from leaves to root (in the direction of the arrow), and then from root to leaves. The first time, it saves $\dot{\pi}_{\bar{C}_i \cap \bar{C}_j} \overset{*}{\bowtie}_{s \text{ on the } \bar{C}_i \text{ side}} s$ in S_{ij}; the second time, it saves $\dot{\pi}_{\bar{C}_i \cap \bar{C}_j} \overset{*}{\bowtie}_{s \text{ on the } \bar{C}_j \text{ side}} s$ in S_{ji}.

Input:

- set of relations $\bar{s}$
- acyclic hypergraph cover of $\{\ regattr(s) \mid s \in \bar{s}\ \}$: $\{\bar{C}_1, \ldots, \bar{C}_n\}$ with $\bar{C}_i \subseteq \mathcal{A}ttr$
- set of query variables $\bar{Q}$ and a q for which $\bar{Q} \subseteq \bar{C}_q$

Output: an expression e equivalent to $\pi_{\bar{Q}} \overset{*}{\underset{s\in\bar{s}}{\bowtie}} s$

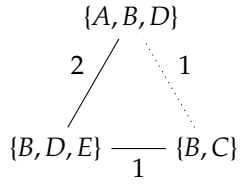
$M \leftarrow$ the complete graph with vertices $\bar{C}_i$
$w \leftarrow$ the weight function $w(\{\bar{C}_i, \bar{C}_j\}) = |\bar{C}_i \cap \bar{C}_j|$
$T \leftarrow$ a maximum weight spanning tree of M weighted with w

foreach $s \in \bar{s}$ **do**
    add a node s to T
    add an edge $\{s, \bar{C}_i\}$ to T, for some C_i with $regattr(s) \subseteq \bar{C}_i$
end
foreach edge $\{\bar{C}_i, \bar{C}_j\}$ in T **do**
    add a node $\overset{+}{\pi}_{\bar{C}_i \cap \bar{C}_j}$ on the edge (replace the old edge with two edges)
end

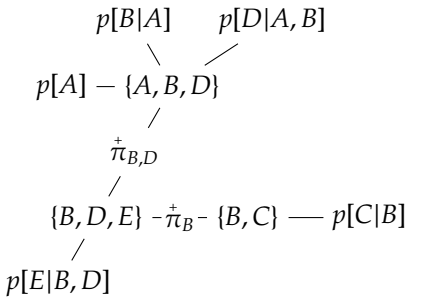
$e \leftarrow$ the expression tree obtained by replacing every C_i with $\overset{*}{\bowtie}$, taking $\bar{C}_q$ as root
$e \leftarrow \overset{+}{\pi}_{\bar{Q}} e$

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

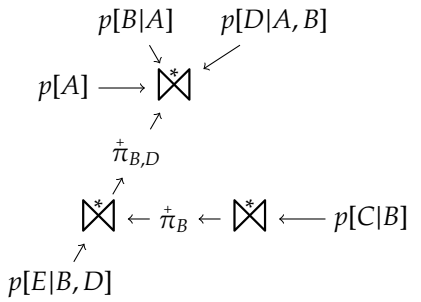
Algorithm 3: Junction tree propagation in a relational representation.



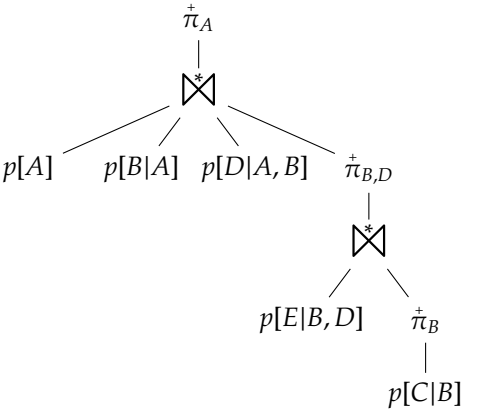
(a) Weighted complete graph M over vertices $\bar{C}_1, \bar{C}_2, \bar{C}_3$, with maximum weight spanning tree T (solid).



(b) Relations $\bar{s}$ and $\overset{+}{\pi}$ operators have been added to T.



(c) T transformed into expression tree e (arrows point towards the root).



(d) Final expression tree e in a more conventional form, with the root on top. The 1-argument join has been removed.

Figure 5.7: An example of Alg. 3, with $\bar{s} = \{p[A], p[B|A], p[C|B], p[D|A, B], p[E|B, D]\}$, $\bar{C}_1 = \{A, B, D\}$, $\bar{C}_2 = \{B, D, E\}$, $\bar{C}_3 = \{B, C\}$, $\bar{Q} = \{A\}$ and $q = 1$.

The most simple version of this algorithm is presented as Alg. 4 and known as the Shenoy-Shafer architecture[56]. There is still some redundant work in this version: to calculate an 'outgoing' S_{ab} value it joins the 'incoming' values for all the *other* edges of C_a. This join expression is different for each outgoing edge, but there is a lot of overlap. The Hugin architecture[4] described in Alg. 5 improves on this by storing the join of all incoming values in P_a, and calculating S_{ab} by dividing out S_{ba} from this. We need a new relational operator r^{-1} in order to perform this division:

$$r^{-1} = \pi_{regattr(r), f \mapsto \mathsf{val}} r$$
$$f(\{t\}) = \begin{cases} t(\mathsf{val})^{-1} & \text{if } t(\mathsf{val}) \neq 0 \\ 0 & \text{if } t(\mathsf{val}) = 0 \end{cases}$$

This 'pointwise inverse' operator replaces the val value of every tuple with its inverse.

Note that the junction tree algorithms given in the last two sections are provided mainly as a demonstration of their relational algebra formulation; they will not be used in the remainder of the thesis.

5.5.4 Acyclic hypergraphs

Both the variable elimination and junction tree techniques depend on heuristics to find a good inference expression. The aim of these heuristics is to minimize the maximum number of attributes in the schemas of the intermediate relations. The goal of this section is to give an overview of the graph-theoretic background of this problem.

In the remainder of this section, we again assume that the input to the problem consists of the set $\bar{s}$. Not the actual contents of these relations are relevant, but only their schemas, so we confine ourselves to

$$\mathcal{H} = \{\, regattr(s) \mid s \in \bar{s} \,\} = \{\, regattr(cpd[V]) \setminus \bar{E} \mid V \in \bar{V} \,\}$$

This set $\mathcal{H}$ can be interpreted as a *hypergraph*. A hypergraph is a generalization of a graph, in which edges are generalized to *hyperedges*. A hyperedge can have any nonzero number of nodes as endpoints instead of two, and is therefore modeled by a nonempty set of nodes. It is customary to define a hypergraph by just its set of hyperedges $\mathcal{H}$; the set of nodes is then implicitly defined as $\bigcup \mathcal{H}$. The *rank* of a hypergraph is defined as its maximum edge cardinality:

$$r(\mathcal{H}) \stackrel{\text{def}}{=} \max_{\bar{H} \in \mathcal{H}} |\bar{H}|$$

As we will explain later, the set of schemas of the intermediate relations is required to be an *acyclic hypergraph cover* of $\mathcal{H}$. A hypergraph $\mathcal{H}'$ is a *hypergraph cover* of $\mathcal{H}$

Input:

- tree T with nodes $\{\bar{C}_1, \ldots, \bar{C}_n\} \cup \bar{s}$
  (produced after the first foreach loop in Algorithm 3)
- query sets $\bar{Q}_1, \ldots, \bar{Q}_m$ with $\forall i.\ \exists j.\ \bar{Q}_i \subseteq \bar{C}_j$

Output: expressions $e_1, \ldots, e_m$ with e_i equivalent to $\dot{\pi}_{\bar{Q}_i} \overset{*}{\bigotimes}_{s \in \bar{s}} s$

take an arbitrary $\bar{C}_r$ as root of T and orient the edges towards it
foreach $\bar{Q}_i$ **do**
  add an edge $\bar{Q}_i \mapsto \bar{C}_j$ for some $\bar{C}_j$ containing $\bar{Q}_i$
end

foreach edge $\bar{C}_i \mapsto \bar{C}_j$, from leaves to root **do**
  $S_{ij} \leftarrow \dot{\pi}_{\bar{C}_i \cap \bar{C}_j} \left(\left(\overset{*}{\bigotimes}_{\text{for each } s \mapsto \bar{C}_i} s \right) \overset{*}{\bowtie} \overset{*}{\bigotimes}_{h \neq j,\ C_h \text{ neighbor of } \bar{C}_i} S_{hi} \right)$
end
foreach edge $\bar{C}_i \mapsto \bar{C}_j$, from root to leaves **do**
  $S_{ji} \leftarrow \dot{\pi}_{\bar{C}_i \cap \bar{C}_j} \left(\left(\overset{*}{\bigotimes}_{\text{for each } s \mapsto \bar{C}_j} s \right) \overset{*}{\bowtie} \overset{*}{\bigotimes}_{k \neq i,\ \bar{C}_k \text{ neighbor of } \bar{C}_j} S_{kj} \right)$
end
foreach edge $\bar{Q}_i \subseteq \bar{C}_j$ **do**
  $e_i \leftarrow \dot{\pi}_{\bar{Q}_i} \left(\left(\overset{*}{\bigotimes}_{\text{for each } s \mapsto \bar{C}_j} s \right) \overset{*}{\bowtie} \overset{*}{\bigotimes}_{k \text{ with } \bar{C}_k \text{ neighbor of } \bar{C}_j} S_{kj} \right)$
end

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

Algorithm 4: Multi-query junction tree propagation (Shenoy-Shafer).

Input:

- tree T with nodes $\{\bar{C}_1, \ldots, \bar{C}_n\} \cup \bar{s}$
  (produced after the first foreach loop in Algorithm 3)
- query sets $\bar{Q}_1, \ldots, \bar{Q}_m$ with $\forall i.\ \exists j.\ \bar{Q}_i \subseteq \bar{C}_j$

Output: expressions $e_1, \ldots, e_m$ with e_i equivalent to $\overset{+}{\pi}_{\bar{Q}_i} \overset{*}{\bigotimes}_{s \in \bar{s}} s$

take an arbitrary $\bar{C}_r$ as root of T and orient the edges towards it
foreach $\bar{Q}_i$ **do**
  add an edge $\bar{Q}_i \mapsto \bar{C}_j$ for some $\bar{C}_j$ containing $\bar{Q}_i$
end

foreach edge $\bar{C}_i \mapsto \bar{C}_j$, from leaves to root **do**
  $P_i \leftarrow \left(\overset{*}{\bigotimes}_{\text{for each } s \mapsto \bar{C}_i} s\right) \overset{*}{\bowtie} \overset{*}{\bigotimes}_{h \neq j,\, C_h \text{ neighbor of } \bar{C}_i} S_{hi}$
  $S_{ij} \leftarrow \overset{+}{\pi}_{\bar{C}_i \cap \bar{C}_j} P_i$
end

$P_r \leftarrow \left(\overset{*}{\bigotimes}_{\text{for each } s \mapsto \bar{C}_r} s\right) \overset{*}{\bowtie} \overset{*}{\bigotimes}_{k \text{ with } \bar{C}_k \text{ neighbor of } \bar{C}_r} S_{kr}$

foreach edge $\bar{C}_i \mapsto \bar{C}_j$, from root to leaves **do**
  $S_{ji} \leftarrow \overset{+}{\pi}_{\bar{C}_i \cap \bar{C}_j} P_j \overset{*}{\bowtie} S_{ij}^{-1}$
  $P_i \leftarrow P_i \overset{*}{\bowtie} S_{ji}$
end
foreach edge $\bar{Q}_i \mapsto \bar{C}_j$ **do**
  $e_i \leftarrow \overset{+}{\pi}_{\bar{Q}_i} P_j$
end

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

Algorithm 5: Multi-query junction tree propagation (Hugin).

if every edge of $\mathcal{H}$ is contained in some hyperedge of $\mathcal{H}'$, i.e.

$$\forall \bar{H} \in \mathcal{H}.\ \exists \bar{H}' \in \mathcal{H}'.\ \bar{H} \subseteq \bar{H}'$$

Acyclicity of a hypergraph is a somewhat more elusive concept than acyclicity of a graph. One of the possible definitions is as follows: a hypergraph is acyclic iff it has a *construction order* that always adds a hyperedge $\mathcal{H}_j$ whose intersection with the already added hyperedges is contained in (at least) one of them:

$$\forall j > 1.\ \exists i < j.\ \left(\bar{H}_1 \cup \ldots \cup \bar{H}_{j-1}\right) \cap \bar{H}_j \subseteq \bar{H}_i$$

This is known as the *running intersection property*. A function mapping each $\bar{H}_j$ (for $j > 1$) to an appropriate $\bar{H}_i$ (as defined above) is called a *branching*, and can be thought of as a tree with the $\bar{H}_i$ sets as nodes, rooted in $\bar{H}_1$. Given a tree like this, it is also possible to take any other node as a root; the result will also be a valid branching for $\mathcal{H}$, albeit with a different construction order[55]. A non-rooted tree $\mathcal{T}$ defining possible branchings for $\mathcal{H}$ is also known as a join tree, junction tree or clique tree for $\mathcal{H}$, and has a desirable property also sometimes referred to as the running intersection property, or clique intersection property:

For two arbitrary nodes $\bar{H}_j$ and $\bar{H}_k$ of $\mathcal{T}$, every node along the path connecting them contains their intersection $\bar{H}_j \cap \bar{H}_k$. Sometimes this property is formulated as follows: for every node A of the hypergraph $\mathcal{H}$ (i.e. an element of one of $\mathcal{T}$'s nodes), the subgraph $\mathcal{T}_A$ induced by the nodes containing A is connected.

The converse is also valid[55, 9]: if a tree $\mathcal{T}$ of sets has this property, its sets form an acyclic hypergraph.

Because in an inference expression, the $\stackrel{+}{\pi}_{-R}$ operators can be pushed down only as long as R occurs in *one* branch, the intermediate relations containing R will always form a connected subtree of the inference expression; therefore, the tree of intermediate schemas will satisfy the running intersection property, and the schemas themselves form an acyclic hypergraph. Furthermore, this hypergraph covers the sets in $\mathcal{H}$, because they form the base relations. Therefore, the search for good inference expressions can be formulated as: *Find an acyclic hypergraph cover C of $\mathcal{H}$ with a small $r(C)$.*

There is one problem with this formulation: not every acyclic hypergraph corresponds to an inference expression. However, using Alg. 3, we can construct an inference expression from an acyclic hypergraph; the hypergraph of intermediate schemas in this expression will be covered by the hypergraph, and hence have an equal or smaller rank.

The question remains how to find an acyclic hypergraph cover. In the literature, this question has been studied not in terms of hypergraphs, but in terms of the representation of a hypergraph called its *primal graph*. The primal graph $G(\mathcal{H})$ of hypergraph $\mathcal{H}$ has the same nodes, and contains an edge iff $\mathcal{H}$ has a hyperedge containing both its endpoints. The primal graph representation forgets some of the structure of a hypergraph: for example, the hypergraphs $\{\{A, B, C\}\}$,

$\{\{A,B,C\},\{A,B\}\}$ and $\{\{A,B\},\{B,C\},\{A,C\}\}$ all have the same primal graph. As this example shows, the same primal graph can even represent one hypergraph that is acyclic (the last one) and one that is not. This is not the case if we confine ourselves to *conformal* hypergraphs. A hypergraph $\mathcal{H}$ is conformal iff every maximal clique of $G(\mathcal{H})$ is a hyperedge of $\mathcal{H}$. When we consider only conformal hypergraphs, their primal graph does determine their acyclicity: a conformal hypergraph is acyclic iff its primal graph is chordal. A graph is *chordal* (or *triangulated*) iff every cycle of length at least four has a chord, i.e. an edge joining two nonconsecutive vertices on the cycle. Going from a primal graph G to a conformal hypergraph $\mathcal{H}$ for which $G(\mathcal{H}) = G$ is easy: just take the maximal cliques of G as hyperedges.

Thus, to find an acyclic hypergraph cover of $\mathcal{H}$, one can triangulate $G(\mathcal{H})$, i.e. add edges until one arrives at a chordal graph $CG(\mathcal{H})$. Then, its maximal cliques will form the desired acyclic hypergraph cover.

Now, the problem is reduced to finding a good triangulation of $G(\mathcal{H})$: one that keeps the maximum cliques, and therefore $r(C)$, small. For this, several heuristics are known[40]. Interestingly, these heuristics can also be used to find a good variable elimination order: the nodes of every chordal graph can be arranged into a so-called *perfect elimination order*, in which the higher-ordered neighbors of each node form a clique. If the $\bar{R}$ variables are eliminated in this order, the schema sizes of the intermediate relations will never be larger than that of the maximum clique.

The reason that heuristics are used is that the problem of finding an acyclic hypergraph cover with minimal rank is NP-hard[7]. This minimal rank minus one is also known as treewidth, and a tree of sets covering the edges of G and satisfying the running intersection property is also known as a tree decomposition of G[50].

Chapter 6

Relational inference for sensor models

In this chapter, we apply the theory from chapter 5 to the dynamic Bayesian networks presented in chapters 2 and 3. Using the relational framework, we develop general inference expressions for these sensor data models that are more efficient than those produced by conventional inference procedures. The expressions derived by the conventional procedures 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 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 contain a lot of zeros, and when they are scaled up, it is primarily this number of zeros that increases. Omitting these zeros from the relations (a *sparse representation*) is therefore crucial, and leads to totally different cost functions and optimization decisions.

Also, the conventional algorithms do not exploit the opportunity to *factorize* the OR-distribution, without which the inference time of the MSHMM-NOR model (section 3.4) grows exponentially w.r.t. the interval length D. Methods to preprocess a Bayesian network (before applying a conventional inference procedure) that factorize the OR-distribution in order to avoid this are known; we show that these methods can also be derived without any reference to probability theory, using only logic and relational algebra. Moreover, thanks to the relational representation, we can adapt the resulting expressions so they make optimal use of the sparse representation.

6.1 Exploiting dynamic Bayesian network structure

Compared to a generic Bayesian network, a dynamic Bayesian network has a special structure (section 2.3.3): 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. This structure can be exploited for query optimization in two ways.

The first, discussed in section 6.1.1, saves 'query optimization time': the same join tree is built for each t, and these are connected to each other. The basic idea is not new, e.g. see [51, section 15.5] and especially [47, section 3.4], but our presentation in relational terms is. Building on these results, the second optimization (section 6.1.2) saves 'run time' by taking advantage of shared subexpressions in the inference expression, and is novel to the best of our knowledge.

Both optimizations are shown for a generic dynamic Bayesian network, consisting of the variables $\bar{V}_t = V_t^1, \ldots, V_t^n$ for each slice t, which is instantiated for T slices. At each time t, we observe values $x_{\bar{E}_t}$ for the variables $\bar{E}_t \subset \bar{V}_t$. For simplicity of exposition, we assume for now that $\bar{Q} = \bar{I}_u$: the query variables consist of the interface between u and $u+1$ (recall from section 2.3.3 that $\bar{I}_u$ consists of those variables V_u^i that have a child in $\bar{V}_{u+1}$). Thus, the inference expression reads:

$$\dot{\pi}_{\bar{Q}} \sigma_{\bar{E}_{1..T}=x_{\bar{E}_{1..T}}} p[\bar{Q}, \bar{E}_{1..T}] = \dot{\pi}_{\bar{I}_u} \sigma_{\bar{E}_{1..T}=x_{\bar{E}_{1..T}}} \overset{*}{\bowtie}_{t=0..T} cpd[V_t^1] \bowtie \ldots \bowtie cpd[V_t^n]$$

The two optimizations consist of specific approaches to rewrite this expression.

6.1.1 Repeating structure in the inference expression

We *partly* rewrite the above inference expression using two recursive expressions f_t and b_t (which the reader familiar with the literature, e.g. [51, chapter 15], may recognize as *forward* and *backward* messages):

$$\begin{aligned}
\dot{\pi}_{\bar{Q}} \sigma_{\bar{E}_{1..T}=x_{\bar{E}_{1..T}}} p[\bar{Q}, \bar{E}_{1..T}] &= f_u \bowtie b_{u+1} \\
f_t &\overset{\text{def}}{=} \dot{\pi}_{\bar{I}_t} \sigma_{\bar{E}_t=x_{\bar{E}_t}} \left(cpd[V_t^1] \overset{*}{\bowtie} \ldots \overset{*}{\bowtie} cpd[V_t^n] \bowtie f_{t-1} \right) \\
b_t &\overset{\text{def}}{=} \dot{\pi}_{\bar{I}_{t-1}} \sigma_{\bar{E}_t=x_{\bar{E}_t}} \left(cpd[V_t^1] \overset{*}{\bowtie} \ldots \overset{*}{\bowtie} cpd[V_t^n] \bowtie b_{t+1} \right) \\
f_0 &\overset{\text{def}}{=} [\![1]\!]_{\mathsf{val}} \\
b_{T+1} &\overset{\text{def}}{=} [\![1]\!]_{\mathsf{val}} \qquad \textbf{(Recursive-DBN)}
\end{aligned}$$

Thus, we insert *some* parentheses in the expression. Also, per (Distr-Evidence)[p. 78], we push down $\sigma_{\bar{E}_t=x_{\bar{E}_t}}$ operators. Thirdly, in the f_t expressions, we apply $\dot{\pi}_{\bar{I}_t}$ in accordance with the procedure outlined in section 5.3: the interface variables $\bar{I}_t$ are the only variables from f_t that also occur somewhere in the surrounding join tree. A similar argument applies to $\bar{I}_{t-1}$ in b_t. See figure 6.1 for a sum-factor diagram of this partially rewritten expression.

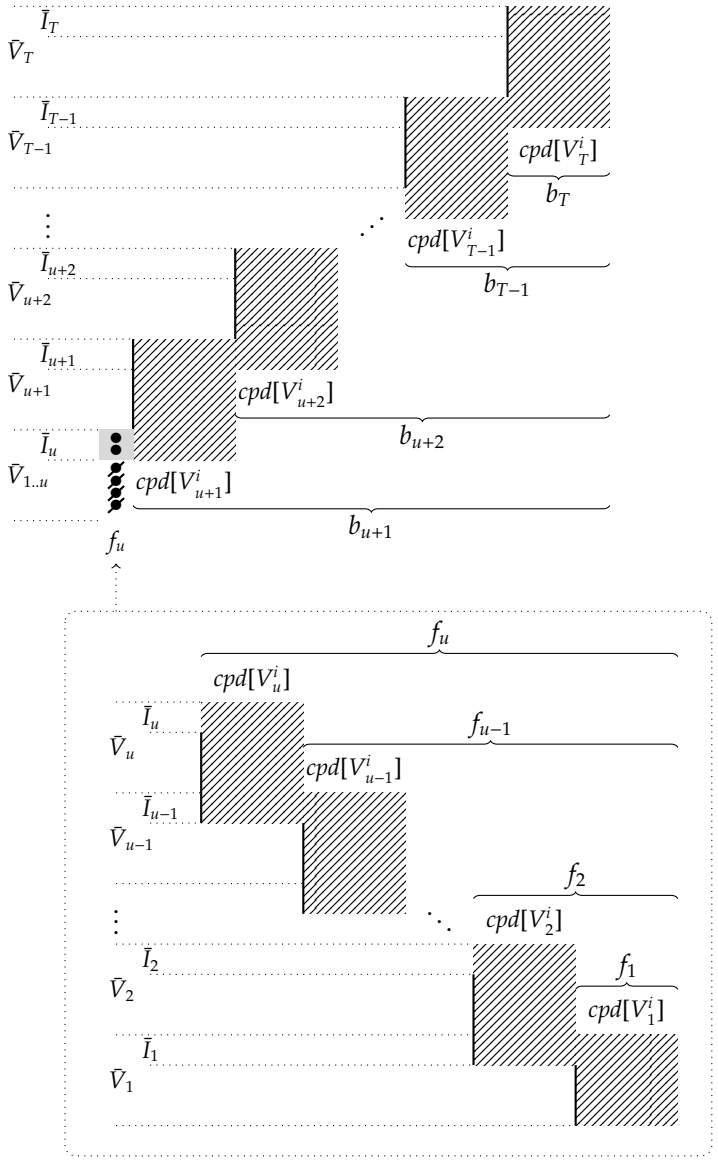


Figure 6.1: Sum-factor diagram for $f_u \bowtie b_{u+1}$, the inference expression for a dynamic Bayesian network (see text). For the leftmost relation f_u, the sum-factor diagram is given in the dotted inset. As the expressions are only partially specified, so are the diagrams: each box labeled $cpd[V_t^i]$ represents the yet unspecified join tree for the relations $cpd[V_t^1]$ through $cpd[V_t^n]$, which together contain the variables $\bar{V}_t \cup \bar{I}_{t-1}$. These cpds are the same for each t, so the same join tree (and diagram columns) can be formed for each box in the topmost diagram. This also goes for the inset diagram, except that the cpds for t=1 are different.

The f_t and b_t expressions are small inference expressions themselves, except for one difference: they contain f_{t-1} and b_{t+1} instead of a cpd. However, at least for the conventional inference procedures, there is nothing that prevents treating them in the same way: f_{t-1} is just a relation with $regattr(f_{t-1}) = \bar{I}_{t-1}$, and b_{t+1} has $regattr(b_{t+1}) = \bar{I}_t$. The evidence for this small inference expression consists of $x_{\bar{E}_t}$, and the query variables are $\bar{I}_t$ (for f_t) or $\bar{I}_{t-1}$ (for b_t).

Although the contents of f_t and b_t differ for each t, their schema is the same, and for the conventional procedures the schema is all that matters. Therefore, to optimize the inference expression for the whole dynamic Bayesian network, the following approach can be taken:

1. Optimize f_t using an inference procedure of choice.
2. Optimize f_1 using an inference procedure of choice (recall from section 2.3.3 that the cpds are different for $t = 1$).
3. Optimize b_t using an inference procedure of choice.
4. Connect the resulting expressions by recursively replacing the references to f_{t-1} and b_{t+1} by their optimized expressions.

In comparison to constructing a global join tree, this approach of chaining together local join trees can save a lot on query optimization time. Also, the repetitive structure created by the recursion shows that inference for a dynamic Bayesian network is in principle linear in T. Moreover, the inference can also be done in a streaming way; the join tree can be grown every time a batch of sensor readings arrives.

6.1.2 Sharing subexpressions

The structure of a dynamic Bayesian network also provides opportunities for saving query 'run time'. Continuing the rewriting process of the previous section, we show how to rewrite b_t such that subexpressions are *shared* among the different b_t instances; the case for f_t is analogous. Note that the *contents* of all base relations $cpd[V_t^i]$ in b_t are the same for each t; only the contents of relation b_{t+1} differ for each t. Different for each t as well is the selection predicate in $\sigma_{\bar{E}_t = x_{E_t}}$.

Hence, parts of a join tree for b_t that only consist of $cpd[V_t^i]$ relations and do not contain σ operators can be reused. The extreme case, with maximal sharing, would therefore be to join *all* the $cpd[V_t^i]$ relations before applying a σ operator or joining with b_{t+1}:

$$
\begin{aligned}
b_t &= \dot{\pi}_{\bar{I}_{t-1}} \left(\dot{\pi}_{-\bar{E}_t} \sigma_{\bar{E}_t = x_{E_t}} b'_t \mathbin{\dot{\bowtie}} b_{t+1} \right) \\
b'_t &= \rho_{V^i_{T-1} \mapsto V^i_{t-1}, V^i_T \mapsto V^i_t} b'_T \\
b'_T &= \dot{\pi}_{\bar{I}_{T-1} \cup \bar{I}_T \cup \bar{E}_T} \left(cpd[V_T^1] \mathbin{\dot{\bowtie}} \ldots \mathbin{\dot{\bowtie}} cpd[V_T^n] \right)
\end{aligned}
$$

The shared relation, whose contents are calculated only once, is b'_T; it is reused as b'_t, with the same contents, only with the attribute names changed to the current t. As explained in section 5.3, we can sum out all the attributes that do not occur anywhere else in the join tree, which leaves $\bar{I}_{t-1}$, $\bar{I}_t$ and $\bar{E}_t$. The b'_T expression is not in a tree form yet; again, this can be done by an inference procedure of choice (with no evidence variables, and $\bar{I}_{T-1} \cup \bar{I}_T \cup \bar{E}_T$ as query variables).

One may—justly—wonder whether the above rewriting of b_t is actually an optimization. The shared b'_T relation runs the risk of becoming very large, because the $\overset{+}{\pi}_{-A}$ operators that project away $\bar{I}_t$ and $\bar{E}_t$ are not pushed down (nor are those that project away $\bar{I}_{t-1}$, but this also holds for other b_t optimizations). This large relation can take a long time to create, consume a lot of storage space, and even be slow to query (i.e. to apply the $\sigma_{\bar{E}_t = x_{\bar{E}_t}}$ operator on).

So, these costs have to be compared to the cost of calculating a separate b_t relation for each t. If T is large (so the time for the upfront calculation of b'_T can be neglected), and storage space is no issue, the question becomes which is faster: (a) the look-up in the b'_T relation which has $|\bar{I}_{T-1} \cup \bar{I}_T \cup \bar{E}_T|$ attributes (dimensions), or (b) calculating $cpd_{-\bar{E}}[V_t^i]$ relations, joining them, and applying $\overset{+}{\pi}$ operators.

The answer to this question depends on the model; in section 6.2, we show that a theoretical analysis for the MSHMM model turns out in favor of the case with shared subexpressions. A further investigation of this question is left for future research. In particular, we would like to point out that we have only discussed the case with maximal sharing above, and that the sharing of smaller subexpressions can also be considered.

As we will show in the next section, the probabilistic semantics of the shared relation (in this case b'_T) can be of use in the efficiency analysis. For completeness, we derive them here:

$$
\begin{aligned}
& \overset{+}{\pi}_{\bar{I}_{T-1} \cup \bar{I}_T \cup \bar{E}_T} \left(cpd[V_T^1] \overset{*}{\bowtie} \ldots \overset{*}{\bowtie} cpd[V_T^n] \right) \\
= \quad & \text{by (Cpd-Slice)}^{\text{p. 25}} \\
& \overset{+}{\pi}_{\bar{I}_{T-1} \cup \bar{I}_T \cup \bar{E}_T} p[\bar{V}_T | \bar{I}_{T-1}] \\
= \quad & \text{by (Marginalize)}^{\text{p. 15}} \text{ and the definition of conditional probability} \\
& p[\bar{I}_T, \bar{E}_T | \bar{I}_{T-1}]
\end{aligned}
$$

6.2 Sparseness

The MSHMM (section 2.4.2) is an example of a probabilistic model of which the cpds largely consist of zeros. In the array representation used in figure 5.4 (as well as in most explanations of inference algorithms), these zeros take up storage space as well as processing time. For a small number of zeros, this is acceptable, because the array representation induces little overhead. For larger numbers of zeros, however, it can be more efficient to use a *sparse representation* of probability

distributions, in which these zero-valued probabilities are neither stored nor used in calculations.

By excluding zero probabilities from a relation, the cardinality of this relation is no longer related to the number of attributes (i.e. the number of probabilistic variables in the expression). As we will show, this can have a profound effect on optimization decisions; we give a theoretic scalability analysis for the MSHMM, in which we assume that the processing time for certain operations scales linearly or logarithmically with the cardinality of the involved relations.

6.2.1 Sparse representation

The only thing we have to change in our relational algebra framework to sparsely represent probability distributions (and other expressions) is the way in which we embody a numeric expression. Before, this was defined in (Repr-Expr)[p. 71]:

$$[\![\phi]\!]_{\mathsf{val}} \stackrel{\text{def}}{=} [\![\phi = \mathsf{val}]\!]$$

For the sparse representation, this is changed into:

$$[\![\phi]\!]_{\mathsf{val}} \stackrel{\text{def}}{=} [\![\phi = \mathsf{val} \wedge \mathsf{val} \neq 0]\!] \qquad \textbf{(Sparse-Repr)}$$

A very important property of this new representation is that (Repr-Mult) and (Repr-Sum)[p. 73] are still valid. We prove this informally, starting with (Repr-Sum):

$$\dot{\pi}_{-A} [\![\phi]\!]_{\mathsf{val}} = [\![\textstyle\sum_{a \in \mathrm{dom}(A)} \phi[a/A]]\!]_{\mathsf{val}}$$

On the left hand side, a tuple t is included in the relation iff $[\![\phi]\!]_{\mathsf{val}}$ has at least one tuple with the same values on t's regular attributes (and some a for attribute A); i.e. there is some a for which $\phi \neq 0$. On the right hand side, however, a tuple is included in the relation iff the sum of all outcomes of ϕ for all $a \in \mathrm{dom}(A)$ is nonzero. Clearly, the latter implies the former: if the sum is nonzero, there must be at least one nonzero term. The converse also holds *if it can be assumed that ϕ cannot take negative values*, and for probabilities this is the case.

We continue with (Repr-Mult):

$$[\![\phi]\!]_{\mathsf{val}} \bowtie^{*} [\![\psi]\!]_{\mathsf{val}} = [\![\phi * \psi]\!]_{\mathsf{val}}$$

The left hand side consists of bindings for which $\phi \neq 0$ and $\psi \neq 0$; the right hand side of bindings for which $\phi * \psi \neq 0$. These conditions are equivalent.

6.2.2 Exploitation of sparseness in MSHMM

We will now analyze the scalability of the MSHMM; we compare (a) a conventional inference approach using an array representation, (b) the same inference expression using a sparse representation, and (c) an inference expression that uses

a shared subexpression (in a sparse representation). We scale up the model by expanding the detection area, i.e. 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 figure 2.5); we will analyze the effect this has on the inference time of the three approaches.

In all three approaches, we use the rewriting (RECURSIVE-DBN)[p. 94] of a dynamic Bayesian network into the recursive f_u and b_u parts. For the analysis, we restrict ourselves to one f_t expression, and assume that the total inference time is given by multiplication with T, a constant factor. For the MSHMM, the f_t factor to be optimized is:

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

Applying variable elimination gives the following result, as there is only one variable (X_{t-1}) to eliminate, which occurs in f_{t-1} and $cpd[X_t]$.

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

No matter where we add the parentheses that turn this expression into a binary tree, the inference time will scale quadratically when the array representation is used: all the probabilities $\mathsf{P}(s_t^c | x_t)$ will be taken into account, where c ranges over $1..K$ and x_t ranges over $1..L$.

Most of these sensor probabilities are irrelevant, because most locations x_t are out of the question anyway: estimates of the location at earlier times, combined with the knowledge that the mobile device can only move with a certain speed, will give these a zero probability. This probability distribution based on earlier sensor readings is represented by the relation f_{t-1}: due to (JOINT-SUBSET-BN)[p. 20], it holds that

$$f_{t-1} = \overset{+}{\pi}_{X_{t-1}} \sigma_{\bar{S}_{1..t-1}=\bar{s}_{1..t-1}} p[X_{t-1}, \bar{S}_{1..t-1}]$$

The number of X_{t-1} locations that have a nonzero joint probability with the sensor input up to $t-1$ depends on when the last y result is received. For example, if one scanner produced y at $t-1$, this number is 9: see the gray area in figure 2.5. When it has been longer ago, this number grows larger, because the mobile device could have moved further in the mean time. However, this number does not depend on K or L, so we will take it into account as a constant factor for the inference time. In the rest of this analysis, we will assume it to be 9.

Now, if we use the same optimized f_t expression as above, but with a sparse representation, f_{t-1} contains 9 tuples. After joining with $cpd[X_t]$ and projecting out X_{t-1}, the resulting relation will contain 13 tuples: all the locations reachable from this area in one step. If there is an index on X_{t-1} for relation $cpd[X_t]$, this join can be performed in $O(\log L)$ time. For the other part of the f_t expression, it

matters how we put the parentheses. Two options are

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

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

The former will still take $O(KL)$ time: almost all scans S_t^c return a n, and the relation $\dot{\pi}_{-S_t^c} \sigma_{S_t^c = \mathsf{n}} cpd[S_t^c]$ contains L tuples. These relations are joined in c order, and all these join results will contain L tuples until a $\sigma_{S_t^c = \mathsf{y}}$ is encountered for a certain c. This can be assumed to happen, on average, for $c = K/2$.[1]

The latter option is better: as the initial relation contains 13 tuples, all join results will also contain a maximum of 13 tuples. Still, K joins are performed, so the inference time will scale as $O(K \log L)$. In fact, it can be argued that a lot of redundant work is performed: 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 three results of nearby scanners can be positive. This is true for every imaginable rewriting of f_t.

If we use the subexpression sharing approach from section 6.1.2, we can avoid this work. We rewrite:

$$\begin{aligned} f_t &= \dot{\pi}_{-X_{t-1}} \left(f_{t+1} \bowtie cpd[X_t]\right) \bowtie \dot{\pi}_{-\bar{S}_t} \sigma_{\bar{S}_t = \bar{s}_t} f'_t \\ f'_t &= \rho_{S_T^c \mapsto S_t^c, X_T \mapsto X_t} f'_T \\ f'_T &= cpd[S_T^1] \bowtie \ldots \bowtie cpd[S_T^K] \end{aligned}$$

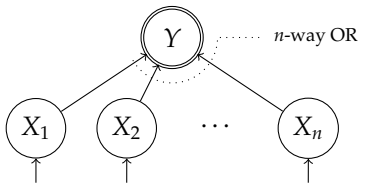
The f'_T relation is equal to the *joint sensor model* $p[\bar{S}_t | X_t]$, from which we can deduce its size: for each location x_t, 3 scanners can produce y or n while the others all produce n, which yields $2^3 = 8$ possible $(x_t, \bar{s}_t)$ combinations with nonzero probability. Hence, the relation contains $8L$ tuples; so its storage size scales linearly when we increase the detection area.

If a clever index on $\bar{S}_t$ is used, the selection $\bar{S}_t = \bar{s}_t$ can be performed in $\log K$ time if at least one scan is y. If all scans are n, it is better to postpone the selection, and first join on X_t (which takes $\log L$ time if there is an index on X_t). In conclusion, when the upfront calculation is not taken into account, the inference time can scale *sublinearly* when using a sparse representation.

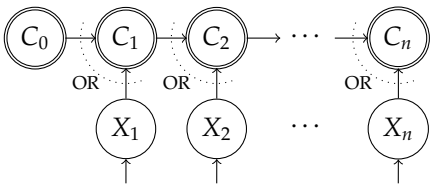
6.3 Two techniques for Noisy-OR

The relation that represents the cpd of an n-way logical OR has $n + 1$ regular attributes. When n is large, such relations become a bottlenecks for inference; in the MSHMM-NOR, this happens when the time granularity is increased. In this section, we analyze the problem, and present two methods that solve the problem

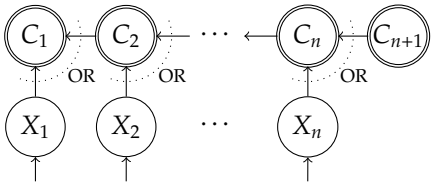
[1] A possible remedy would be to *dynamically* choose a join tree for f_t depending on the values of $\bar{s}_t$: reorder the $\sigma_{s_t^c} cpd[S_t^c]$ relations such that one with $s_t^c = \mathsf{y}$ is at the bottom.



(a) Unoptimized n-way logical OR in a Bayesian network.



(b) Sequential decomposition. C_0 is set to *false*, i.e. $\mathsf{P}(C_0 = \mathit{false}) = 1$, and $Y = C_n$.



(c) Reverse sequential decomposition; C_{n+1} is set to *false*, i.e. $\mathsf{P}(C_{n+1} = \mathit{false}) = 1$, and $Y = C_1$.

Figure 6.2: Sequential decomposition of the n-way OR variable Y into multiple binary OR variables $C_1, \ldots, C_n$, as a transformation of the Bayesian network.

by factorizing the cpd into smaller relations. The factorizations themselves stem from existing literature, but we show that both can be derived in relational terms, in a similar fashion and without probabilistic knowledge. Also, we point out that the sparse representation can again be crucial for scalability.

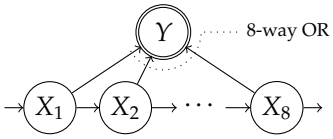
6.3.1 The problematic relation

Throughout this section we assume that the stochastic variable Y is defined as the logical OR of the variables $X_1, \ldots, X_n$. We use the Iverson notation again; recall from section 3.2 that $\langle \mathit{true} \rangle \stackrel{\text{def}}{=} 1$ and $\langle \mathit{false} \rangle \stackrel{\text{def}}{=} 0$. We show the definition of the cpd for Y, first as a conventional probabilistic definition, and then as a relation:

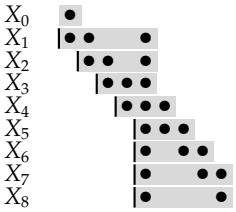
$$\mathsf{P}(y|x_1, \ldots, x_n) = \langle y = x_1 \vee \ldots \vee x_n \rangle$$
$$cpd[Y] = p[Y|X_1, \ldots, X_n] = [\![\langle Y = X_1 \vee \ldots \vee X_n \rangle]\!]_{\mathsf{val}}$$

There are two problems with this relation. Firstly, its cardinality becomes exponentially large as n is increased. It has 2^{n+1} tuples in an array representation, and 2^n in a sparse one: given certain values X_1 though X_n—for which 2^n combinations are possible—one Y value is correct (this tuple has $\mathsf{val} = 1$) and one is incorrect (the tuple has $\mathsf{val} = 0$, and is not included in the sparse representation).

Secondly, as a consequence, intermediate relations resulting from a join with this relation become large as well. As a running example, we use the Bayesian network fragment shown in figure 6.3a, where $n = 8$ and the X_i variables are



(a) Bayesian network fragment (a part of a larger model in which the X_i chain continues to the left and right).



(b) Sum-factor diagram of the optimal inference expression for the variables in this fragment, with $\bar{Q} = X_0$ and $\bar{E} = Y$. The $cpd[Y]$ relation is placed in the middle, and $cpd[X_1]$ through $cpd[X_n]$ are arranged around it from left to right.

Figure 6.3: Running example: a logical OR in a larger network with a chain.

connected in a chain. This fragment has approximately the same structure as the intervals in the MSHMM-NOR model, but is simplified for explanatory purpose. Consider the inference query with $\bar{Q} = \{X_0\}$ and $\bar{E} = \{Y\}$ (which may occur as part of a backward pass in a larger model where the X-chain extends further to the left and right):

$$\dot{\pi}_{X_0} \sigma_{Y=y} \left(cpd[Y] \bowtie cpd[X_1] \bowtie \ldots cpd[X_n] \right)$$

An optimized right-deep join tree for this query is shown as a sum-factor diagram in figure 6.3b. Note that Y is omitted on the vertical axis because it is an evidence variable. Placing the $cpd[Y]$ relation in the middle minimizes the number of intermediate attributes, but there is no way to avoid intermediate relations of up to $(n/2) + 1$ attributes.

In this way, the $cpd[Y]$ relation will always become a bottleneck in inference when n gets large. However, these problems can be solved by factoring $cpd[Y]$ into smaller relations. In the remainder of this section, we review two such factorizations known from the AI literature, again in relational terms. We show that the relational representation helps to demonstrate the correctness of the factorizations, and that the sparse physical representation is useful in the case that the 'output' variable is observed as *false* (which occurs a lot in our sensor data models): then processing the logical OR need not produce any extra tuples at all.

6.3.2 Sequential decomposition

The first factorization is known as *sequential decomposition*[31]. In the Bayesian network under consideration, it replaces the probabilistic variable Y with a sequence of variables $C_0, \ldots, C_n$ whose cpds all represent a *binary* logical OR. They are strung together so that C_i represents the cumulative OR value of X_1 through X_i (see figure 6.2b). The first variable C_0 is fixed at *false*; the last variable C_n is

equal to Y. Thus, the C_i variables have the following cpd:

$$cpd[C_i] = p[C_i|C_{i-1}, X_i] = [\![\langle C_i = C_{i-1} \vee X_i\rangle]\!]_{\mathsf{val}}$$
$$cpd[C_0] = p[C_0] = [\![\langle C_0 = \mathit{false}\rangle]\!]_{\mathsf{val}}$$

Using the relational representation, we can prove that this factorization, namely $cpd[Y] = \rho_{C_n \mapsto Y} \overset{+}{\pi}_{-\{C_0,\ldots,C_{n-1}\}} \overset{*}{\bowtie}_{0 \le i \le n} cpd[C_i]$, is valid. We first need two auxiliary theorems about representing booleans by 0 and 1, namely that $\overset{*}{\bowtie}$ functions as $\wedge$ and $\overset{+}{\pi}$ (under a certain condition) as $\exists$. The first theorem reads:

$$[\![\langle\theta\rangle]\!]_{\mathsf{val}} \overset{*}{\bowtie} [\![\langle\kappa\rangle]\!]_{\mathsf{val}} = [\![\langle\theta\rangle * \langle\kappa\rangle]\!]_{\mathsf{val}} = [\![\langle\theta \wedge \kappa\rangle]\!]_{\mathsf{val}}$$

The validity of the first equation follows from (Repr-Mult)[p. 72], and that of the second one by comparing the truth table of $\wedge$ with the multiplication table of 0 and 1. The second theorem reads:

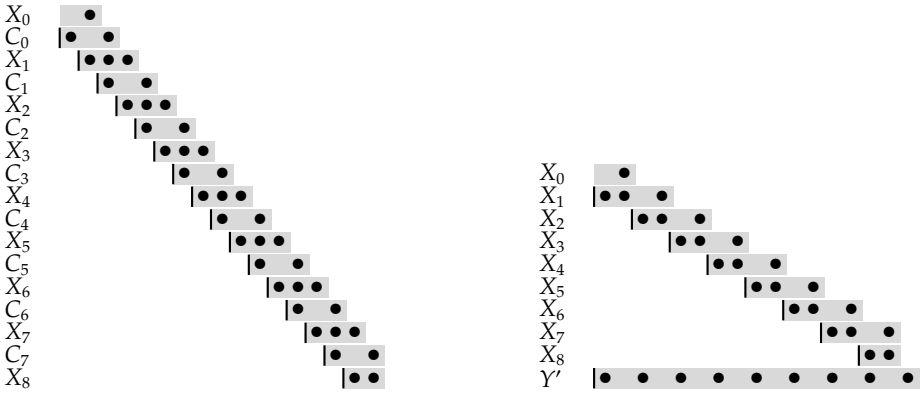
$$\overset{+}{\pi}_{-A} [\![\langle\theta\rangle]\!]_{\mathsf{val}} = [\![\textstyle\sum_a \langle\theta[a/A]\rangle]\!]_{\mathsf{val}} = [\![\langle\exists a.\theta[a/A]\rangle]\!]_{\mathsf{val}} \quad \text{if } A \text{ is a function}$$

The first equation is valid due to (Repr-Sum)[p. 73]. In the middle relation, the expression $\sum_a \langle\theta[a/A]\rangle$ yields, given a binding of the $\mathit{schema}(\theta) \setminus \{A\}$ attributes, the *number* of bindings for A that make θ true. If θ implies that A is a function of a subset of $\mathit{schema}(\theta) \setminus \{A\}$, this is at most one: if there exists such a binding it yields 1, otherwise it yields 0. Therefore, the expression equals $\langle\exists a.\theta[a/A]\rangle$.

Now, the relational factorization introduced above follows from the predicate logic factorization of an n-way OR into n binary ORs:

$$\begin{aligned}
& cpd[Y] \\
= \quad & \text{definition of } cpd[Y] \\
& [\![\langle Y = X_1 \vee \ldots \vee X_n\rangle]\!]_{\mathsf{val}} \\
= \quad & \text{renaming an attribute} \\
& \rho_{C_n \mapsto Y} [\![\langle C_n = X_1 \vee \ldots \vee X_n\rangle]\!]_{\mathsf{val}} \\
= \quad & \text{predicate logic} \\
& \rho_{C_n \mapsto Y} [\![\langle \exists c_0, \ldots, c_{n-1}.(C_n = c_{n-1} \vee X_n) \wedge \ldots \wedge (c_1 = c_0 \vee X_1\rangle \wedge (c_0 = \mathit{false})]\!]_{\mathsf{val}} \\
= \quad & \text{second auxiliary theorem} \\
& \rho_{C_n \mapsto Y} \overset{+}{\pi}_{-\{C_0,\ldots,C_{n-1}\}} [\![\langle (C_n = C_{n-1} \vee X_n) \wedge \ldots \wedge (C_1 = C_0 \vee X_1) \wedge (C_0 = \mathit{false})\rangle]\!]_{\mathsf{val}} \\
= \quad & \text{first auxiliary theorem} \\
& \rho_{C_n \mapsto Y} \overset{+}{\pi}_{-\{C_0,\ldots,C_{n-1}\}} \left([\![\langle C_n = C_{n-1} \vee X_n\rangle]\!]_{\mathsf{val}} \overset{*}{\bowtie} \ldots \overset{*}{\bowtie} [\![\langle C_1 = C_0 \vee X_1\rangle]\!]_{\mathsf{val}} \overset{*}{\bowtie} [\![\langle C_0 = \mathit{false}\rangle]\!]_{\mathsf{val}} \right) \\
= \quad & \text{definition of } cpd[C_i] \\
& \rho_{C_n \mapsto Y} \overset{+}{\pi}_{-\{C_0,\ldots,C_{n-1}\}} \overset{*}{\bowtie}_{0 \le i \le n} cpd[C_i]
\end{aligned}$$

If $cpd[Y]$ in the running example is substituted with this factorization, the intermediate relations after optimization can be kept small: by interleaving the $cpd[C_i]$



(a) Sequential decomposition.

(b) Díez-Galán decomposition.

Figure 6.4: Sum-factor diagrams of the optimized inference expressions for the running example (figure 6.3a), where $cpd[Y]$ is replaced by a factorization into smaller relations. The maximum number of attributes in the intermediate relations is not dependent on n anymore.

relations between the $cpd[X_i]$ and $cpd[X_{i+1}]$ relations in the chain, the three relations that contain attribute X_i are kept close to each other, so this attribute can be projected out as soon as possible. This optimization is written as follows:

$$\overset{+}{\pi}_{X_0}\sigma_{Y=y}\left(cpd[Y] \overset{*}{\bowtie} cpd[X_1] \overset{*}{\bowtie} \ldots \overset{*}{\bowtie} cpd[X_n]\right) = \overset{+}{\pi}_{-C_0}\left(cpd[C_0] \overset{*}{\bowtie} b_1\right)$$

$$\text{where} \quad b_i = \overset{+}{\pi}_{-X_i}\left(cpd[X_i] \overset{*}{\bowtie} \overset{+}{\pi}_{-C_i}\left(cpd[C_i] \overset{*}{\bowtie} b_{i+1}\right)\right)$$

$$b_n = \overset{+}{\pi}_{-X_n}\left(cpd[X_n] \overset{*}{\bowtie} \overset{+}{\pi}_{-C_n}\sigma_{C_n=y}cpd[C_n]\right)$$

Note that we have eliminated the renaming operator by directly using $C_n = y$ as the selection predicate. This optimized expression is shown in the sum-factor diagram in figure 6.4a. The intermediate relations now have a maximum dimensionality of 3; therefore, inference time is no longer exponential in n, but only linear.

If the *sparse representation* is used, and $y = false$, the above optimization is even more efficient. The relation $cpd[C_n]$ at the end of the expression contains only one tuple for which the selection predicate $C_n = false$ holds: the one for which C_n, C_{n-1} and X_n are all *false* (if the sparse representation is not used, the 3 tuples with other values for C_n and X_{n-1} are also included, with $\mathsf{val} = 0$). A little higher up in the expression tree, each tuple in b_n will therefore also have $C_{n-1} = false$. It will only join with the one tuple from $cpd[C_{n-1}]$ for which C_{n-1} is *false*. This process continues upwards: hence, joining with $cpd[C_i]$ relations will never cause an increase in tuples.

Besides this, the sparse representation provides more opportunities for opti-

mization. First, note that an expression of the form $[\![\langle\phi\rangle]\!]_{\mathsf{val}}$ contains only tuples with $\mathsf{val} = 1$: those for which ϕ is true. This means that it can also be written as $[\![\phi]\!] \bowtie [\![1]\!]_{\mathsf{val}}$:

$$\begin{aligned}
& [\![\langle\phi\rangle]\!]_{\mathsf{val}} \\
= & \quad \text{by (SPARSE-REPR)}^{\text{p. 98}} \\
& [\![\langle\phi\rangle = \mathsf{val} \wedge \mathsf{val} > 0]\!] \\
= & \; [\![\langle\phi\rangle = 1 \wedge \mathsf{val} = 1]\!] \\
= & \; [\![\phi \wedge \mathsf{val} = 1]\!] \\
= & \; [\![\phi]\!] \bowtie [\![\mathsf{val} = 1]\!] \\
= & \quad \text{by (SPARSE-REPR)}^{\text{p. 98}} \\
& [\![\phi]\!] \bowtie [\![1]\!]_{\mathsf{val}}
\end{aligned}$$

In the expression to be optimized, all $cpd[C_i]$ can be rewritten like this. Afterward, we apply the rewrite rule

$$(r \bowtie [\![1]\!]_{\mathsf{val}}) \overset{*}{\bowtie} s = r \bowtie s$$

(where $\mathsf{val} \notin \mathit{schema}(r), \mathsf{val} \in \mathit{schema}(s)$). Thus, the expression is rewritten as

$$\begin{aligned}
\overset{+}{\pi}_{X_0} \sigma_{Y=y} \left(cpd[Y] \overset{*}{\bowtie} cpd[X_1] \overset{*}{\bowtie} \ldots \overset{*}{\bowtie} cpd[X_n]\right) &= \overset{+}{\pi}_{-C_0} \left([\![C_0 = \mathit{false}]\!] \bowtie b_1\right) \\
\text{where} \quad b_i &= \overset{+}{\pi}_{-X_i} \left(cpd[X_i] \overset{*}{\bowtie} \overset{+}{\pi}_{-C_i} \left([\![C_i = C_{i-1} \vee X_i]\!] \bowtie b_{i+1}\right)\right) \\
b_n &= \overset{+}{\pi}_{-X_n} \left(cpd[X_n] \bowtie \overset{+}{\pi}_{-C_n} \sigma_{C_n=y} [\![C_n = C_{n-1} \vee X_n]\!]\right)
\end{aligned}$$

If the appropriate join operator $\bowtie_\theta$ is available, this can subsequently be rewritten into:

$$\begin{aligned}
\overset{+}{\pi}_{X_0} \sigma_{Y=y} \left(cpd[Y] \overset{*}{\bowtie} cpd[X_1] \overset{*}{\bowtie} \ldots \overset{*}{\bowtie} cpd[X_n]\right) &= \overset{+}{\pi}_{-C_0} \left([\![C_0 = \mathit{false}]\!] \bowtie b_1\right) \\
\text{where} \quad b_i &= \overset{+}{\pi}_{-X_i} \left(cpd[X_i] \overset{*}{\bowtie} \overset{+}{\pi}_{-C_i} \left([\![C_{i-1}]\!] \bowtie_{C_i = C_{i-1} \vee X_i} b_{i+1}\right)\right) \\
b_n &= \overset{+}{\pi}_{-X_n} \left(cpd[X_n] \bowtie_{y = C_{n-1} \vee X_n} [\![C_{n-1}]\!]\right)
\end{aligned}$$

One final note about all optimizations presented thus far: in the running example, we assumed a backwards pass over the X_n variables. Due to the direction of the sequential decomposition, the evidence variable Y corresponded to C_n, and therefore the evidence selection $\sigma_{C_n=y}$ could be inserted at the bottom of the expression. If we are performing a forward pass instead (see section 6.1), we can

better use a *reverse* sequential decomposition (figure 6.2c):

$$cpd[Y] = \rho_{C_1 \mapsto Y} \overset{+}{\pi}_{-\{C_2,\ldots,C_{n+1}\}} \overset{*}{\bowtie}_{0 \le i \le n} cpd[C_i]$$

$$cpd[C_i] = p[C_i|C_{i+1}, X_i] = [\![\langle C_i = C_{i+1} \vee X_i \rangle]\!]_{\mathsf{val}}$$

$$cpd[C_{n+1}] = p[C_{n+1}] = [\![\langle C_{n+1} = \mathit{false} \rangle]\!]_{\mathsf{val}}$$

This way, when we build a forward chain, i.e. with the X_1 variable at the bottom and X_n on top, we can insert $\sigma_{C_1=y}$ at the bottom again, and profit from the reduction of tuples throughout the whole expression when $y = \mathit{false}$ and the representation is sparse.

6.3.3 The Díez-Galán Noisy-MAX decomposition

A second factorization for the OR cpd has been proposed by Díez and Galán[20], and is similar to a factorization we have used earlier[24]. However, it is more general; it factorizes the multi-way MAX function, of which the multi-way OR is a special case when *false* is represented by 0, and *true* by 1. In this section, we show that this factorization can be derived, and therefore easily verified, relationally.

The cpd $\mathsf{P}(y|x_1, \ldots, x_n)$ now represents the MAX function with input variables $X_{1..n}$ and output variable Y, all with domain $0..m$ (of which the logical OR is an instance with $m = 1$):

$$cpd[Y] = p[Y|X_1, \ldots, X_n] = [\![\langle y = \mathrm{MAX}(X_1, \ldots, X_n) \rangle]\!]_{\mathsf{val}}$$

The MAX function can be characterized as follows:

$$y = \mathrm{MAX}(x_1, \ldots, x_n) \quad \equiv \quad \left(\bigwedge_{i=1..n} x_i \le y \right) \wedge \neg \left(\bigwedge_{i=1..n} x_i \le y - 1 \right)$$

The rhs is of the form $p \wedge \neg q$, for which $q \to p$ holds; indeed, if all x_i are less than $y-1$, they are also less than y. We can rewrite its numeric representation:

$$\begin{aligned}
& \langle p \wedge \neg q \rangle \\
= \quad & \langle p \rangle * \langle \neg q \rangle \\
= \quad & \langle p \rangle * (1 - \langle q \rangle) \\
= \quad & \langle p \rangle - \langle p \rangle * \langle q \rangle \\
= \quad & \langle p \rangle - \langle p \wedge q \rangle \\
= \quad & \qquad \text{because } q \to p \\
& \langle p \rangle - \langle q \rangle
\end{aligned}$$

We use this result in the following derivation, in which we rewrite the MAX function into a sum-of-products form that can be represented relationally:

$$
\begin{aligned}
& [\![\langle y = \mathrm{MAX}(X_1, \ldots, X_n)\rangle]\!]_{\mathsf{val}} \\
= \quad & \\
& [\![\langle (\textstyle\bigwedge_{i=1..n} X_i \leq Y) \wedge \neg (\bigwedge_{i=1..n} X_i \leq Y - 1)\rangle]\!]_{\mathsf{val}} \\
= \quad & \text{just derived} \\
& [\![\langle \textstyle\bigwedge_{i=1..n} X_i \leq Y\rangle - \langle \bigwedge_{i=1..n} X_i \leq Y - 1\rangle]\!]_{\mathsf{val}} \\
= \quad & \\
& [\![(\textstyle\prod_{i=1..n}\langle X_i \leq Y\rangle) - \prod_{i=1..n}\langle X_i \leq Y - 1\rangle]\!]_{\mathsf{val}} \\
= \quad & \text{see below} \\
& \left[\!\!\left[\sum_{0 \leq y' \leq m} \Delta(Y, y') * \textstyle\prod_{i=1..n}\langle X_i \leq y'\rangle \right]\!\!\right]_{\mathsf{val}} \\
= \quad & \text{by (Repr-Sum) and (Repr-Mult)}^{\text{p. 72}} \\
& \overset{+}{\pi}_{-Y'} \left([\![\Delta(Y, Y')]\!]_{\mathsf{val}} \overset{*}{\bowtie} \overset{*}{\bigotimes}_{1 \leq i \leq n} [\![\langle X_i \leq Y'\rangle]\!]_{\mathsf{val}} \right)
\end{aligned}
$$

In the equality marked 'see below', we replace a subtraction of two similar terms (differing only in Y vs. $Y-1$) by a summation. For this purpose, an auxiliary function $\Delta(y, y')$ is used:

$$
\Delta(y, y') = \begin{cases} 1 & \text{if } y' = y \\ -1 & \text{if } y' = y - 1 \\ 0 & \text{otherwise} \end{cases}
$$

In the summation, y' can start at 0, as the term $\langle X_i \leq -1\rangle$ that $y' = -1$ would contribute always equals 0 (because X_i has domain $0..m$).

Again, this factorization rewrites an n-dimensional relation into a join of low-dimensional relations, which can be interleaved between the $cpd[X_i]$ relations in the inference expression for our running example:

$$
\begin{aligned}
& \overset{+}{\pi}_{X_0} \sigma_{Y=y} \left(cpd[Y] \overset{*}{\bowtie} cpd[X_1] \overset{*}{\bowtie} \ldots \overset{*}{\bowtie} cpd[X_n] \right) = \overset{+}{\pi}_{-Y'} b_1 \\
& \qquad \text{where} \quad b_i = \overset{+}{\pi}_{-X_i} \left([\![\langle X_i \leq Y'\rangle]\!]_{\mathsf{val}} \overset{*}{\bowtie} (cpd[X_i] \overset{*}{\bowtie} b_{i+1}) \right) \\
& \qquad\qquad b_n = \overset{+}{\pi}_{-X_n} \left([\![\langle X_n \leq Y'\rangle]\!]_{\mathsf{val}} \overset{*}{\bowtie} \left(cpd[X_n] \overset{*}{\bowtie} \overset{+}{\pi}_{-Y} \sigma_{Y=y} [\![\Delta(Y, Y')]\!]_{\mathsf{val}} \right) \right)
\end{aligned}
$$

The sum-factor diagram for this expression is shown in figure 6.4b. Like with the sequential decomposition, the number of attributes in the intermediate relations is no longer dependent on n.

Using the sparse representation for this expression brings the same advantage in the $y = 0$ case as it did for the sequential decomposition expression in the $y = \mathit{false}$ case. The relation $\sigma_{Y=0} [\![\Delta(Y, Y')]\!]_{\mathsf{val}}$ contains only one tuple, with $Y = 0$,

$Y' = 0$, and $\mathsf{val} = 1$; relations higher up in the tree will therefore only have tuples with $Y' = 0$, and join with one tuple from $[\![X_i \leq Y']\!]_{\mathsf{val}}$.

Also, the optimizations for the sparse representation can be applied again. The $[\![\langle\phi\rangle]\!]_{\mathsf{val}} \bowtie s$ occurrences are replaced with $[\![\phi]\!] \bowtie s$:

$$\begin{aligned}\overset{+}{\pi}_{X_0}\sigma_{Y=y}\left(cpd[Y] \bowtie cpd[X_1] \bowtie \ldots \bowtie cpd[X_n]\right) &= \overset{+}{\pi}_{-Y'} b_1 \\ \text{where} \quad b_i &= \overset{+}{\pi}_{-X_i}\left([\![X_i \leq Y']\!] \bowtie (cpd[X_i] \bowtie b_{i+1})\right) \\ b_n &= \overset{+}{\pi}_{-X_n}\left([\![X_n \leq Y']\!] \bowtie \left(cpd[X_n] \bowtie \overset{+}{\pi}_{-Y}\sigma_{Y=y}\,[\![\Delta(Y, Y')]\!]_{\mathsf{val}}\right)\right)\end{aligned}$$

In the expressions $[\![X_i \leq Y']\!] \bowtie (s \bowtie t)$, the X_i and Y' attributes already occur in $\mathit{schema}(s \bowtie t)$, so they can be rewritten into $\sigma_{X_i \leq Y'}(s \bowtie t)$, and if the $\bowtie_\theta$ operator is available even into $s \bowtie_{X_i \leq Y'} t$:

$$\begin{aligned}\overset{+}{\pi}_{X_0}\sigma_{Y=y}\left(cpd[Y] \bowtie cpd[X_1] \bowtie \ldots \bowtie cpd[X_n]\right) &= \overset{+}{\pi}_{-Y'} b_1 \\ \text{where} \quad b_i &= \overset{+}{\pi}_{-X_i}\left(cpd[X_i] \bowtie_{X_i \leq Y'} b_{i+1}\right) \\ b_n &= \overset{+}{\pi}_{-X_n}\left(cpd[X_n] \bowtie_{X_n \leq Y'} \overset{+}{\pi}_{-Y}\sigma_{Y=y}\,[\![\Delta(Y, Y')]\!]_{\mathsf{val}}\right)\end{aligned}$$

A rewrite rule that is often useful in combination with the Díez-Galán decomposition is the following:

$$\overset{+}{\pi}_{-X}\left([\![X \leq Y]\!] \bowtie [\![\phi]\!]_{\mathsf{val}}\right) = \left[\!\!\left[\sum_{x \leq Y} \phi[x/X]\right]\!\!\right]_{\mathsf{val}}$$

Although this does not apply to our running example, it is often the case that after the factorization, $cpd[X_i]$ and $[\![X_i \leq Y']\!]$ are the only relations in the inference expression that contain X_i as an attribute, so $\overset{+}{\pi}_{-X_i}$ can be pushed down to the join of these relations. Applying the rewrite rule then yields:

$$\begin{aligned}&\overset{+}{\pi}_{-X_i}\left([\![X_i \leq Y']\!] \bowtie cpd[X_i]\right) \\ = \quad &\text{definition} \\ &\overset{+}{\pi}_{-X_i}\left([\![X_i \leq Y']\!] \bowtie [\![\mathsf{P}(X_i|\ldots)]\!]_{\mathsf{val}}\right) \\ = \quad &\text{rewrite rule} \\ &[\![\textstyle\sum_{x_i \leq Y'} \mathsf{P}(x_i|\ldots)]\!]_{\mathsf{val}}\end{aligned}$$

Thus, instead of the normal cpd, a *cumulative* cpd for X_i is used. For each value of the parents of X_i, these cumulative probabilities can be calculated using m additions. This calculation is made only once; in return, all the $\overset{+}{\pi}_{-X_i}$ operators are removed from the inference expression.

A caveat for the Díez-Galán decomposition is that it produces tuples for which $\mathsf{val} < 0$. Therefore, the condition that enables the use of (Repr-Sum) (see section 6.2.1) in the sparse representation does not hold anymore: the left-hand side

may contain tuples with $\mathsf{val} = 0$. Of course, if this becomes a problem, one can always filter these tuples out: the equation

$$\sigma_{\mathsf{val}\neq 0}\dot{\pi}_{-A}\,[\![\phi]\!]_{\mathsf{val}} = [\![\textstyle\sum_{a\in\mathrm{dom}(A)}\phi[a/A]]\!]_{\mathsf{val}}$$

does hold (for the sparse representation).

6.4 Analysis of MSHMM-NOR inference

For the MSHMM-NOR (section 3.4), the optimizations from the previous chapters can be combined. We apply the recursive query plan for Bayesian networks, the Díez-Galán decomposition, and the sharing of the sparse MSHMM sensor model. We start with the recursive query plan (Recursive-DBN)$^{\text{p. 94}}$, and consider only the backwards message b_t here for simplicity.

$$b_t = \dot{\pi}_{\bar{I}_{t-1}}\left(b_{t+1} \overset{*}{\bowtie} cpd[X_t] \overset{*}{\bowtie}\left(\overset{*}{\bigbowtie}_{\substack{1\leq c\leq K\\ \theta^c - t \pmod P < D}} cpd[S_t^c]\right) \overset{*}{\bowtie} \overset{*}{\bigbowtie}_{\theta^c - t \pmod P = 0} \sigma_{A_i^c = a_i^c} cpd[A_t^c]\right)$$

Note the restrictions on c under the $\overset{*}{\bigbowtie}$ operators; they are there because in the MSHMM-NOR model, the variables S_t^c and A_t^c exist only for certain combinations of c and t. Also, because the MSHMM-NOR is not strictly a dynamic Bayesian network, we have adopted a more general definition of the interface variables $\bar{I}_t$; these are now the $\bar{V}_i$ variables, for all $i \leq t$, that have any V_j variable as a child, for $t < j$. This is to make sure that the b_t relation, which somewhere contains a $cpd[A_j^c]$ relation with an S_i^c variable in its schema, does not project out this variable (this should happen higher in the join tree, namely at b_i).

As discussed in section 6.3.1, this expression suffers from a high-dimensional $cpd[A_t^c]$ relation and other intermediate relations. Applying the Díez-Galán optimization leads to the following relation:

$$b_t = \dot{\pi}_{\bar{I}_{t-1}}\left(b_{t+1} \overset{*}{\bowtie} cpd[X_t] \overset{*}{\bowtie}\left(\overset{*}{\bigbowtie}_{\substack{1\leq c\leq K\\ \theta^c - t \pmod P < D}} cpd[S_t^c] \bowtie [\![S_t^c \leq A_\star^c]\!]\right) \overset{*}{\bowtie} \overset{*}{\bigbowtie}_{\theta^c - t \pmod P = 0} \dot{\pi}_{-A_i^c}\sigma_{A_i^c = a_i^c}\,[\![\Delta(A_t^c, A_\star^c)]\!]_{\mathsf{val}}\right)$$

In this expression, the places of the Y and Y' variables are taken by A_t^c and $A_\star^c$, respectively. We consider $A_\star^c$ to be a part of $\bar{I}_t$ as long as t contains an S_t^c variable but no A_t^c variable. Note: the S_t^c, A_t^c and $A_\star^c$ variables have domain $\{\mathsf{n}, \mathsf{y}\}$ which is ordered $\mathsf{n} < \mathsf{y}$.

Next, we apply the sharing optimization:

$$b_t = \dot{\pi}^{+}_{\bar{I}_{t-1}} \left(b_{t+1} \overset{*}{\bowtie} cpd[X_t] \overset{*}{\bowtie} \left(p[\bar{S}_t|X_t] \bowtie \bigotimes_{\theta^c - t \pmod P < D} [\![S^c_t \leq A^c_\star]\!] \right) \overset{*}{\bowtie} \bigotimes^{*}_{\theta^c - t \pmod P = 0} \dot{\pi}^{+}_{-A^c_t} \sigma_{A^c_t = a^c_t} [\![\Delta(A^c_t, A^c_\star)]\!]_{\mathsf{val}} \right)$$

where $p[\bar{S}_t|X_t]$ is the joint sensor model, as explained in section 6.2.2; however, a difference for the MSHMM-NOR is that there are a couple of different joint sensor models, because a different combination of S^c_t variables exists for different t. As we have explained earlier, the number of tuples in $p[\bar{S}_t|X_t]$ is linear in $\mathrm{dom}(X_t)$ due to the joint sparseness of the sensor models, and thus poses no scalability problems. However, the join of all $[\![S^c_t \leq A^c_\star]\!]$ in the above expression forms a problem: it contains 3^n tuples, where n is the number of existing S^c_t variables for t. To remedy this problem, the $S^c_t \leq A^c_\star$ restrictions can be moved to a join with b_{t+1}:

$$b_t = \dot{\pi}^{+}_{\bar{I}_{t-1}} \left(cpd[X_t] \overset{*}{\bowtie} p[\bar{S}_t|X_t] \overset{*}{\bowtie}_{\bar{S}_t \leq \bar{A}_\star} \left(b_{t+1} \overset{*}{\bowtie} \bigotimes^{*}_{\theta^c - t \pmod P = 0} \dot{\pi}^{+}_{-A^c_t} \sigma_{A^c_t = a^c_t} [\![\Delta(A^c_t, A^c_\star)]\!]_{\mathsf{val}} \right) \right)$$

The reason why this is better is that, just like in the MSHMM there is a limited number of sensors for which S^c_t can be y at one time, in the MSHMM-NOR there is always a limited number of sensors with $A^c_i = \mathsf{y}$. For the far majority of sensors, $A^c_i = \mathsf{n}$ (in a large model). This produces only one tuple in $\sigma_{A^c_t = a^c_t} [\![\Delta(A^c_t, A^c_\star)]\!]_{\mathsf{val}}$, for which $A^c_\star = \mathsf{n}$ as well; later, tuples with $A^c_\star = \mathsf{n}$ will only join with tuples for which $S^c_\star = \mathsf{n}$.

As the two final optimizations, we decide on a join order (where we avoid joining the large relations $cpd[X_t]$ and $p[\bar{S}_t|X_t]$ first), and push the $\dot{\pi}^{+}_{-\bar{S}_t}$ operator down the joins:

$$b_t = \dot{\pi}^{+}_{\bar{I}_{t-1}} \left(cpd[X_t] \overset{*}{\bowtie} \dot{\pi}^{+}_{-\bar{S}_t} \left(p[\bar{S}_t|X_t] \overset{*}{\bowtie}_{\bar{S}_t \leq \bar{A}_\star} \left(b_{t+1} \overset{*}{\bowtie} \bigotimes^{*}_{\theta^c - t \pmod P = 0} \dot{\pi}^{+}_{-A^c_t} \sigma_{A^c_t = a^c_t} [\![\Delta(A^c_t, A^c_\star)]\!]_{\mathsf{val}} \right) \right) \right)$$

This leaves us with recursive b_t step which can be calculated in $O(3^n \log(K + L))$ time, where n is the number of simultaneous scans (i.e. whose intervals have an overlap at t) for which $A^c_i = \mathsf{y}$.

Chapter 7

Conclusions and future work

In this thesis, we have researched how probabilistic models can play a role in fulfilling the new requirements that sensing environments set for data management. In particular, we analyze the specific problems for sensor data in achieving the traditional data management goal of *modularity* (chapter 1), and argue that these can be solved by using probabilistic models. Then, given the fact that probabilistic models are used, we investigate the *efficiency* of query processing over these models.

This chapter gives a summary of our results in this research, and points towards interesting future directions.

7.1 Main results

Q1 How can probabilistic models be defined in a modular way?

The simple answer to this question is: by using Bayesian networks. This result is neither novel nor surprising; studying a number of articles and textbooks on artificial intelligence leaves the impression that the AI community is quite aware of this. However, this is not yet the case in the data management community, and although this modularity can be considered as the main feature of Bayesian networks, it is not often given this credit—with the quote at the start of chapter 2 as a notable exception.

In that chapter, we give an analysis of *what* kind of modularity is important for sensor data models—the ability to add, remove or replace sensors—and *why* Bayesian networks provide this modularity. These answers can be found in section 2.3. Furthermore, we make an effort to give this modularity a formal definition. Essential components in the argumentation are the formal separation of the c_V function associated with variable V in a Bayesian network and its cpd, and the theorem (Joint-Subset-BN)$^{\text{p. 20}}$ which states that the well-known factorization

of the joint probability holds for *any* subset of variables closed under *Parents*. Again, this may not come as a surprise for AI researchers, but we have never seen it formulated so explicitly, and we believe it deserves to be.

Modularity means as few dependencies as possible between different parts of a system. In chapter 3, we propose several generic models that remove the dependency arising from the synchronization of sensor observations in basic dynamic Bayesian networks. The most interesting of these models, the MSHMM-NOR (section 3.4), accommodates sensor readings that pertain to an *interval* rather than a point in time. It combines this feature with efficient learning and inference characteristics (the latter are discussed in chapter 6).

Q2 How can a transition model be constructed in a modular way?

In chapter 4, we have rephrased this question in the following way: can the transition frequencies in a global state space be derived from the transition frequencies within local clusters of states? Next, we give a mathematical analysis of this question using what we call the *uncoiled inter-cluster transition graph*. The problem can be described as a set of linear equations, which we prove has exactly one solution if this graph has no cycles; if it does, the solution space is spanned by the fundamental cycles.

Furthermore, the solution space is bounded by inequalities. We have performed an experiment, using clusters of states with a hexagonal topology and random transition frequencies, to investigate the effect of these limits: we picked a random vector from the solution space and compared it with the original frequencies. One of the results of this experiment is that the median error of these frequencies lies around 4% with clusters of diameter 15.

Q3 How can probabilistic inference be performed efficiently in a situation where the number of sensors and the domains of variables are scaled up?

This question arises mainly from our Bluetooth localization scenario and the accompanying probabilistic models MSHMM and MSHMM-NOR. For the localization scenarios, the conditional probability distributions in these models possess an inherent *sparsity*: for any given location, it is (a) only possible to move to a fixed number of other locations, and (b) only possible to be sensed by a fixed number of sensors. These fixed numbers do not change when the model is scaled up. In this context, scaling up means that the observed area is enlarged simultaneously with the number of sensors.

This sparsity is present in the conditional probability distributions, not in the graph structure of the Bayesian network. Therefore, it is not taken into account by the heuristics for classic inference algorithms like variable elimination and junction tree propagation, which assume that intermediate tables of probabilities are encoded as *arrays*, and try to minimize the dimensionality of these arrays. However, if a *sparse representation* is used, dimensionality becomes irrelevant as

an optimization criterion. We show this by presenting optimized inference procedures for the localization scenario, which scale sublinearly where conventional approaches scale quadratically. We do this for both the MSHMM and MSHMM-NOR models (in sections 6.2 and 6.4, respectively).

We derive these optimizations by the novel method of formulating them in relational algebra, which is introduced in chapter 5. This method goes far beyond the specific case of our localization models, and can be used for inference optimization on any Bayesian network. It decouples the optimization from probabilistic theory and formulates it in terms that are well-known to database researchers, thereby providing opportunities for synergistic research.

7.2 Future directions

Building on the results in this thesis, we can point to some promising future research directions and questions.

Sharing over multiple time slices. In section 6.1.2, we discussed pre-calculating the relation $p[\bar{I}_t, \bar{E}_t | \bar{I}_{t-1}]$ for one time slice of a dynamic Bayesian network, and sharing these results throughout the recursive inference calculation. An open question we already posed there is: when is it cheaper to pre-calculate and share this whole relation, and when is it cheaper to share only a part? If the balance turns out in favor for the former, then we can also look in the other direction; it may be even cheaper to share the joined relations of several time slices. For example, for three time slices, the shared relation would be $p[\bar{I}_t, \bar{E}_t, \bar{E}_{t-1}, \bar{E}_{t-2} | \bar{I}_{t-3}]$. Of course, the space taken by this relation will usually grow exponentially with the number of time slices; looking things up in such a relation will probably only be useful as long as it fits inside a fast cache.

Hybrid array/sparse representation. Although a sparse representation is crucial for large domains with many nonzeros, it is outperformed by a (conventional) array representation on small domains. This raises the question whether there is a middle road. Probability distributions are not randomly sparse, but consist of clusters of nonzeros; the idea is to store the position and size of a cluster, and use an array for its data.

Aggregation queries. Probabilistic inference queries will often be combined with some form of *aggregation*: although a localization domain model may be specified at a granularity of a m^2, it could be that users are only interested in the probability distribution at a granularity that is courser by an order of magnitude. A similar argument holds for aggregation in the time dimension. How can we optimize the data processing with these queries in mind? And can we expect other complex queries that combine probabilistic and deterministic aspects?

Approximate inference. How can approximate inference methods such as particle filters and Markov Chain Monte Carlo be integrated with the relational algebra formulation?

Evaluation framework. How do we set up an evaluation framework for sensor data processing? What kind of queries are allowed and expected? In particular: do we accommodate continuous queries, historical queries, or both? How do we evaluate non-exact results? Given the often real-time nature of sensor data queries, does *latency* play a role? See also section 2.6.

Using the probabilistic model as statistics for optimization. The probabilistic models do not only provide the data on which to operate, but also useful metadata for optimization: they tell which sensor readings are to be expected in certain situations. This could be worthwhile to exploit, for example by prefetching/precomputing relevant parts of the model, or by building dedicated indices.

Automatic optimization. In this thesis, we have manually optimized the inference queries for the MSHMM and MSHMM-NOR models with sparse conditional probability distributions. Ideally, this would be the job of a query optimizer.

7.3 Integration with streaming management

While this thesis focuses on dealing with the uncertainty in sensor data, we would finally like to point out another aspect in which sensor data poses novel requirements to data management: the *temporal* dimension. On the data supply side, sensors will flood the data management system with new information at an unprecedented rate; on the demand side, applications need to be kept up to date, causing a steady (lower bandwidth, higher quality) stream outwards.

More than before, applications will continuously be interested in the same query on the new data. This may take the form of *monitoring* (continuously update a traffic info widget on the user's desktop) or *alerting* (only contacting the application when a train service is delayed). Also, applications might be interested in *trends* rather than only current data (How is the traffic compared to yesterday? How fast is that rain cloud moving this way?).

These new temporal requirements will change how data management systems deal with new and old data. A naive *update* approach, in which a new reading from a sensor overwrites an old one, is not workable for several reasons:

- With such an approach, there is no way to detect dynamics or trends in the sensor data, or to search sensor history.
- If an application is polling the data management system, it might miss important information. However this can be solved by registering a *continuous*

query, in which case the data management system pushes the new results to the application when new data becomes available.

- In order to keep several tables consistent during the processing of a query, updates may have to wait until query processing is finished. This causes delay and planning overhead, and can become a problem when the update rate is high.

However, if sensor data is to be *appended* rather than overwritten, the question becomes when to throw it away (or: aggregate it, compress it, move it to a slower storage system); this will perhaps become the responsibility of the data management system rather than that of the applications. Also, query languages will have to include the temporal aspect. Two notable approaches are using *sliding windows* on sensor streams[6, 43] (typically for monitoring queries), or using some kind of (regular expression like) *event language*[2, 12] (typically for alert queries).

A question that is still very much open is how to integrate these approaches with the probabilistic models presented in this thesis.

Bibliography

[1] Serge Abiteboul, Rakesh Agrawal, Philip A. Bernstein, Michael J. Carey, Stefano Ceri, Bruce W. Croft, David J. DeWitt, Michael J. Franklin, Hector Garcia-Molina, Dieter Gawlick, Jim Gray, Laura M. Haas, Alon Y. Halevy, Joseph M. Hellerstein, Yannis E. Ioannidis, Martin L. Kersten, Michael J. Pazzani, Michael Lesk, David Maier, Jeffrey F. Naughton, Hans-Jörg Schek, Timos K. Sellis, Avi Silberschatz, Michael Stonebraker, Richard T. Snodgrass, Jeffrey D. Ullman, Gerhard Weikum, Jennifer Widom, and Stanley B. Zdonik. The Lowell database research self-assessment. *Commun. ACM*, 48(5):111–118, 2005.

[2] Jagrati Agrawal, Yanlei Diao, Daniel Gyllstrom, and Neil Immerman. Efficient pattern matching over event streams. In Jason Tsong-Li Wang, editor, *SIGMOD Conference*, pages 147–160. ACM, 2008.

[3] Philip E. Agre. Changing places: Contexts of awareness in computing. *Human-Computer Interaction*, 16(2, 3, 4):177–192, 2001.

[4] Stig K. Andersen, Kristian G. Olesen, Finn Verner Jensen, and Frank Jensen. Hugin - a shell for building bayesian belief universes for expert systems. In *IJCAI*, pages 1080–1085, 1989.

[5] Lyublena Antova, Thomas Jansen, Christoph Koch, and Dan Olteanu. Fast and simple relational processing of uncertain data. In *ICDE*, pages 983–992. IEEE, 2008.

[6] Arvind Arasu, Shivnath Babu, and Jennifer Widom. The CQL Continuous Query Language: Semantic foundations and query execution. *VLDB Journal*, 15(2):121–142, June 2006.

[7] 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.

[8] Magdalena Balazinska, Amol Deshpande, Michael J. Franklin, Phillip B. Gibbons, Jim Gray, Mark Hansen, Michael Liebhold, Suman Nath, Alexander S.

Szalay, and Vincent Tao. Data management in the worldwide sensor web. *IEEE Pervasive Computing*, 6(2):30–40, 2007.

[9] Catriel Beeri, Ronald Fagin, David Maier, and Mihalis Yannakakis. On the desirability of acyclic database schemes. *J. ACM*, 30(3):479–513, 1983.

[10] Omar Benjelloun, Anish Das Sarma, Alon Y. Halevy, and Jennifer Widom. Uldbs: Databases with uncertainty and lineage. In Umeshwar Dayal, Kyu-Young Whang, David B. Lomet, Gustavo Alonso, G.M. Lohman, Martin L. Kersten, Sang Kyun Cha, and Young-Kuk Kim, editors, *VLDB*, pages 953–964. ACM, 2006.

[11] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[12] Jan Carlson and Björn Lisper. An event detection algebra for reactive systems. In Giorgio C. Buttazzo, editor, *EMSOFT 2004, September 27-29, 2004, Pisa, Italy, Fourth ACM International Conference On Embedded Software, Proceedings*, pages 147–154. ACM, 2004.

[13] Surajit Chaudhuri and Umeshwar Dayal. An overview of data warehousing and OLAP technology. *SIGMOD Record*, 26(1):65–74, 1997.

[14] Guanling Chen, Ming Li, and David Kotz. Design and implementation of a large-scale context fusion network. In *MobiQuitous*, pages 246–255. IEEE Computer Society, 2004.

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

[16] A. de Keijzer. *Management of Uncertain Data - towards unattended integration*. PhD thesis, Univ. of Twente, Enschede, February 2008.

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

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

[19] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977.

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

[21] Pedro Domingos and Michael J. Pazzani. On the optimality of the simple Bayesian classifier under zero-one loss. *Machine Learning*, 29(2-3):103–130, 1997.

[22] K. Ducatel, M. Bogdanowicz, F. Scapolo, J. Leijten, and J-C. Burgelman. Scenarios for ambient intelligence in 2010. Technical report, IST Advisory Group (ISTAG), 2004.

[23] Sander Evers, Maarten Fokkinga, and Peter M. G. Apers. Composable Markov building blocks. In H. Prade and V. S. Subrahmanian, editors, *Proceedings of the 1st International Conference on Scalable Uncertainty Management (SUM 2007), Washington DC, USA*, volume 4772 of *Lecture Notes in Computer Science*, pages 131–142, Berlin, October 2007. Springer Verlag.

[24] 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.

[25] Tom Fawcett. An introduction to ROC analysis. *Pattern Recognition Letters*, 27(8):861–874, 2006.

[26] Jr. Forney. The Viterbi algorithm. *Proceedings of the IEEE*, 61(3):268–278, March 1973.

[27] Michael J. Franklin. Challenges in ubiquitous data management. In *Informatics: 10 years back, 10 years ahead*, volume 2000 of *LNCS*, pages 24–33, London, UK, 2001. Springer-Verlag.

[28] Dan Geiger, Thomas Verma, and Judea Pearl. Identifying independence in Bayesian networks. *Networks*, 20(5):507–534, 1990.

[29] N. Gershenfeld, R. Krikorian, and D. Cohen. The Internet of Things. *Scientific American*, 291(4):76–81, 2004.

[30] David L. Hall and James Llinas. An introduction to multisensor data fusion. *Proc. of the IEEE*, 85(1):6–23, January 1997.

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

[32] Joseph M. Hellerstein, Wei Hong, and Samuel R. Madden. The sensor spectrum: technology, trends, and requirements. *SIGMOD Record*, 32(4):22–27, 2003.

[33] Jeffrey Hightower and Gaetano Borriello. Particle filters for location estimation in ubiquitous computing: A case study. In Nigel Davies, Elizabeth Mynatt, and Itiro Siio, editors, *UbiComp 2004: Ubiquitous Computing: 6th International Conference*, pages 88–106. Springer-Verlag Heidelberg, 2004.

[34] Cecil Huang and Adnan Darwiche. Inference in belief networks: A procedural guide. *Int. J. Approx. Reasoning*, 15(3):225–263, 1996.

[35] Shawn R. Jeffery, Gustavo Alonso, Michael J. Franklin, Wei Hong, and Jennifer Widom. A pipelined framework for online cleaning of sensor data streams. In Ling Liu, Andreas Reuter, Kyu-Young Whang, and Jianjun Zhang, editors, *ICDE*, page 140. IEEE Computer Society, 2006.

[36] Michael I. Jordan, editor. *Learning in graphical models*. MIT Press, Cambridge, MA, USA, 1999.

[37] Marek Junghans and Hans-Joachim Jentschel. Qualification of traffic data by Bayesian network data fusion. In *10th International Conference on Information Fusion, 2007*, July 2007.

[38] Bhargav Kanagal and Amol Deshpande. Online filtering, smoothing and probabilistic modeling of streaming data. In *Proceedings of the 24th International Conference on Data Engineering (ICDE2008)*, pages 1160–1169, April 2008.

[39] M. G. Kendall. A new measure of rank correlation. *Biometrika*, 30(1-2):81–93, 1938.

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

[41] 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.

[42] Donald E. Knuth. Two notes on notation. *American Mathematical Monthly*, 99(5):403–422, May 1992.

[43] Jürgen Krämer and Bernhard Seeger. Semantics and implementation of continuous sliding window queries over data streams. *ACM Trans. Database Syst.*, 34(1), 2009.

[44] S. Kullback and RA Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, pages 79–86, 1951.

[45] 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.

[46] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill Higher Education, 1997.

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

[48] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers Inc., San Francisco, USA, 1988.

[49] Lawrence R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[50] Neil Robertson and Paul D. Seymour. Graph minors. II. Algorithmic aspects of tree-width. *J. Algorithms*, 7(3):309–322, 1986.

[51] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2nd edition, 2004.

[52] Mahadev Satyanarayanan. Pervasive computing: Vision and challenges. *IEEE Personal Communications*, 8(4):10–17, 2001.

[53] Prithviraj Sen, Amol Deshpande, and Lise Getoor. Exploiting shared correlations in probabilistic databases. In *Proceedings of the 34th International Conference on Very Large Data Bases (VLDB2008), August 24-28, 2008, Auckland, New Zealand*, pages 809–820, August 2008.

[54] Ross D. Shachter. Probabilistic inference and influence diagrams. *Operations Research*, 36(4):589–604, 1988.

[55] Glenn Shafer and Prakash P. Shenoy. Local computation in hypertrees. Technical Report 201, University of Kansas, July 1991.

[56] Prakash P. Shenoy and Glenn Shafer. Axioms for probability and belief-function proagation. In Ross D. Shachter, Tod S. Levitt, Laveen N. Kanal, and John F. Lemmer, editors, *UAI*, pages 169–198. North-Holland, 1988.

[57] Robert Endre Tarjan and Mihalis Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM J. Comput.*, 13(3):566–579, 1984.

[58] Jiří Vomlel. Exploiting functional dependence in Bayesian network inference. In Adnan Darwiche and Nir Friedman, editors, *UAI*, pages 528–535. Morgan Kaufmann, 2002.

[59] Daisy Zhe Wang, Eirinaios Michelakis, Minos Garofalakis, and Joseph M. Hellerstein. BayesStore: Managing large, uncertain data repositories with

probabilistic graphical models. In *Proceedings of the 34th International Conference on Very Large Data Bases (VL DB2008), August 24-28, 2008, Auckland, New Zealand*, pages 340–351, August 2008.

[60] Mark Weiser. The computer for the 21st century. *Scientific American*, 265(3):94–104, 1991.

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

SIKS Dissertation Series

1998-1 Johan van den Akker (CWI)
DEGAS - An Active, Temporal Database of Autonomous Objects

1998-2 Floris Wiesman (UM)
Information Retrieval by Graphically Browsing Meta-Information

1998-3 Ans Steuten (TUD)
A Contribution to the Linguistic Analysis of Business Conversations within the Language/Action Perspective

1998-4 Dennis Breuker (UM)
Memory versus Search in Games

1998-5 E.W.Oskamp (RUL)
Computerondersteuning bij Straftoemeting

1999-1 Mark Sloof (VU)
Physiology of Quality Change Modelling; Automated modeling of Quality Change of Agricultural Products

1999-2 Rob Potharst (EUR)
Classification using decision trees and neural nets

1999-3 Don Beal (UM)
The Nature of Minimax Search

1999-4 Jacques Penders (UM)
The practical Art of Moving Physical Objects

1999-5 Aldo de Moor (KUB)
Empowering Communities: A Method for the Legitimate User-Driven Specification of Network Information Systems

1999-6 Niek J.E. Wijngaards (VU)
Re-design of compositional systems

1999-7 David Spelt (UT)
Verification support for object database design

1999-8 Jacques H.J. Lenting (UM)
Informed Gambling: Conception and Analysis of a Multi-Agent Mechanism for Discrete Reallocation.

2000-1 Frank Niessink (VU)
Perspectives on Improving Software Maintenance

2000-2 Koen Holtman (TUE)
Prototyping of CMS Storage Management

2000-3 Carolien M.T. Metselaar (UVA)
Sociaal-organisatorische gevolgen van kennistechnologie; een procesbenadering en actorperspectief.

2000-4 Geert de Haan (VU)
ETAG, A Formal Model of Competence Knowledge for User Interface Design

2000-5 Ruud van der Pol (UM)

Knowledge-based Query Formulation in Information Retrieval.

2000-6 Rogier van Eijk (UU)
Programming Languages for Agent Communication

2000-7 Niels Peek (UU)
Decision-theoretic Planning of Clinical Patient Management

2000-8 Veerle Coupé (EUR)
Sensitivity Analyis of Decision-Theoretic Networks

2000-9 Florian Waas (CWI)
Principles of Probabilistic Query Optimization

2000-10 Niels Nes (CWI)
Image Database Management System Design Considerations, Algorithms and Architecture

2000-11 Jonas Karlsson (CWI)
Scalable Distributed Data Structures for Database Management

2001-1 Silja Renooij (UU)
Qualitative Approaches to Quantifying Probabilistic Networks

2001-2 Koen Hindriks (UU)
Agent Programming Languages: Programming with Mental Models

2001-3 Maarten van Someren (UvA)
Learning as problem solving

2001-4 Evgueni Smirnov (UM)
Conjunctive and Disjunctive Version Spaces with Instance-Based Boundary Sets

2001-5 Jacco van Ossenbruggen (VU)
Processing Structured Hypermedia: A Matter of Style

2001-6 Martijn van Welie (VU)
Task-based User Interface Design

2001-7 Bastiaan Schonhage (VU)
Diva: Architectural Perspectives on Information Visualization

2001-8 Pascal van Eck (VU)
A Compositional Semantic Structure for Multi-Agent Systems Dynamics.

2001-9 Pieter Jan 't Hoen (RUL)
Towards Distributed Development of Large Object-Oriented Models, Views of Packages as Classes

2001-10 Maarten Sierhuis (UvA)
Modeling and Simulating Work Practice BRAHMS: a multiagent modeling and simulation language for work practice analysis and design

2001-11 Tom M. van Engers (VUA)
Knowledge Management: The Role of Mental Models in Business Systems Design

2002-01 Nico Lassing (VU)
Architecture-Level Modifiability Analysis

2002-02 Roelof van Zwol (UT)
Modelling and searching web-based document collections

2002-03 Henk Ernst Blok (UT)
Database Optimization Aspects for Information Retrieval

2002-04 Juan Roberto Castelo Valdueza (UU)
The Discrete Acyclic Digraph Markov Model in Data Mining

2002-05 Radu Serban (VU)
The Private Cyberspace Modeling Electronic Environments inhabited by Privacy-concerned Agents

2002-06 Laurens Mommers (UL)
Applied legal epistemology; Building a knowledge-based ontology of the legal domain

2002-07 Peter Boncz (CWI)
Monet: A Next-Generation DBMS Kernel For Query-Intensive Applications

2002-08 Jaap Gordijn (VU)
Value Based Requirements Engineering: Exploring Innovative E-Commerce Ideas

2002-09 Willem-Jan van den Heuvel(KUB)
Integrating Modern Business Applications with Objectified Legacy Systems
2002-10 Brian Sheppard (UM)
Towards Perfect Play of Scrabble
2002-11 Wouter C.A. Wijngaards (VU)
Agent Based Modelling of Dynamics: Biological and Organisational Applications
2002-12 Albrecht Schmidt (Uva)
Processing XML in Database Systems
2002-13 Hongjing Wu (TUE)
A Reference Architecture for Adaptive Hypermedia Applications
2002-14 Wieke de Vries (UU)
Agent Interaction: Abstract Approaches to Modelling, Programming and Verifying Multi-Agent Systems
2002-15 Rik Eshuis (UT)
Semantics and Verification of UML Activity Diagrams for Workflow Modelling
2002-16 Pieter van Langen (VU)
The Anatomy of Design: Foundations, Models and Applications
2002-17 Stefan Manegold (UVA)
Understanding, Modeling, and Improving Main-Memory Database Performance
2003-01 Heiner Stuckenschmidt (VU)
Ontology-Based Information Sharing in Weakly Structured Environments
2003-02 Jan Broersen (VU)
Modal Action Logics for Reasoning About Reactive Systems
2003-03 Martijn Schuemie (TUD)
Human-Computer Interaction and Presence in Virtual Reality Exposure Therapy
2003-04 Milan Petkovic (UT)
Content-Based Video Retrieval Supported by Database Technology
2003-05 Jos Lehmann (UVA)
Causation in Artificial Intelligence and Law - A modelling approach
2003-06 Boris van Schooten (UT)
Development and specification of virtual environments
2003-07 Machiel Jansen (UvA)
Formal Explorations of Knowledge Intensive Tasks
2003-08 Yongping Ran (UM)
Repair Based Scheduling
2003-09 Rens Kortmann (UM)
The resolution of visually guided behaviour
2003-10 Andreas Lincke (UvT)
Electronic Business Negotiation: Some experimental studies on the interaction between medium, innovation context and culture
2003-11 Simon Keizer (UT)
Reasoning under Uncertainty in Natural Language Dialogue using Bayesian Networks
2003-12 Roeland Ordelman (UT)
Dutch speech recognition in multimedia information retrieval
2003-13 Jeroen Donkers (UM)
Nosce Hostem - Searching with Opponent Models
2003-14 Stijn Hoppenbrouwers (KUN)
Freezing Language: Conceptualisation Processes across ICT-Supported Organisations
2003-15 Mathijs de Weerdt (TUD)
Plan Merging in Multi-Agent Systems
2003-16 Menzo Windhouwer (CWI)
Feature Grammar Systems - Incremental Maintenance of Indexes to Digital Media Warehouses
2003-17 David Jansen (UT)
Extensions of Statecharts with Probability, Time, and Stochastic Timing

2003-18 Levente Kocsis (UM)
Learning Search Decisions
2004-01 Virginia Dignum (UU)
A Model for Organizational Interaction: Based on Agents, Founded in Logic
2004-02 Lai Xu (UvT)
Monitoring Multi-party Contracts for E-business
2004-03 Perry Groot (VU)
A Theoretical and Empirical Analysis of Approximation in Symbolic Problem Solving
2004-04 Chris van Aart (UVA)
Organizational Principles for Multi-Agent Architectures
2004-05 Viara Popova (EUR)
Knowledge discovery and monotonicity
2004-06 Bart-Jan Hommes (TUD)
The Evaluation of Business Process Modeling Techniques
2004-07 Elise Boltjes (UM)
Voorbeeldig onderwijs; voorbeeldgestuurd onderwijs, een opstap naar abstract denken, vooral voor meisjes
2004-08 Joop Verbeek(UM)
Politie en de Nieuwe Internationale Informatiemarkt, Grensregionale politiële gegevensuitwisseling en digitale expertise
2004-09 Martin Caminada (VU)
For the Sake of the Argument; explorations into argument-based reasoning
2004-10 Suzanne Kabel (UVA)
Knowledge-rich indexing of learning-objects
2004-11 Michel Klein (VU)
Change Management for Distributed Ontologies
2004-12 The Duy Bui (UT)
Creating emotions and facial expressions for embodied agents
2004-13 Wojciech Jamroga (UT)
Using Multiple Models of Reality: On Agents who Know how to Play
2004-14 Paul Harrenstein (UU)
Logic in Conflict. Logical Explorations in Strategic Equilibrium
2004-15 Arno Knobbe (UU)
Multi-Relational Data Mining
2004-16 Federico Divina (VU)
Hybrid Genetic Relational Search for Inductive Learning
2004-17 Mark Winands (UM)
Informed Search in Complex Games
2004-18 Vania Bessa Machado (UvA)
Supporting the Construction of Qualitative Knowledge Models
2004-19 Thijs Westerveld (UT)
Using generative probabilistic models for multimedia retrieval
2004-20 Madelon Evers (Nyenrode)
Learning from Design: facilitating multidisciplinary design teams
2005-01 Floor Verdenius (UVA)
Methodological Aspects of Designing Induction-Based Applications
2005-02 Erik van der Werf (UM)
AI techniques for the game of Go
2005-03 Franc Grootjen (RUN)
A Pragmatic Approach to the Conceptualisation of Language
2005-04 Nirvana Meratnia (UT)
Towards Database Support for Moving Object data
2005-05 Gabriel Infante-Lopez (UVA)
Two-Level Probabilistic Grammars for Natural Language Parsing
2005-06 Pieter Spronck (UM)

Adaptive Game AI

2005-07 Flavius Frasincar (TUE)
Hypermedia Presentation Generation for Semantic Web Information Systems

2005-08 Richard Vdovjak (TUE)
A Model-driven Approach for Building Distributed Ontology-based Web Applications

2005-09 Jeen Broekstra (VU)
Storage, Querying and Inferencing for Semantic Web Languages

2005-10 Anders Bouwer (UVA)
Explaining Behaviour: Using Qualitative Simulation in Interactive Learning Environments

2005-11 Elth Ogston (VU)
Agent Based Matchmaking and Clustering - A Decentralized Approach to Search

2005-12 Csaba Boer (EUR)
Distributed Simulation in Industry

2005-13 Fred Hamburg (UL)
Een Computermodel voor het Ondersteunen van Euthanasiebeslissingen

2005-14 Borys Omelayenko (VU)
Web-Service configuration on the Semantic Web; Exploring how semantics meets pragmatics

2005-15 Tibor Bosse (VU)
Analysis of the Dynamics of Cognitive Processes

2005-16 Joris Graaumans (UU)
Usability of XML Query Languages

2005-17 Boris Shishkov (TUD)
Software Specification Based on Re-usable Business Components

2005-18 Danielle Sent (UU)
Test-selection strategies for probabilistic networks

2005-19 Michel van Dartel (UM)
Situated Representation

2005-20 Cristina Coteanu (UL)
Cyber Consumer Law, State of the Art and Perspectives

2005-21 Wijnand Derks (UT)
Improving Concurrency and Recovery in Database Systems by Exploiting Application Semantics

2006-01 Samuil Angelov (TUE)
Foundations of B2B Electronic Contracting

2006-02 Cristina Chisalita (VU)
Contextual issues in the design and use of information technology in organizations

2006-03 Noor Christoph (UVA)
The role of metacognitive skills in learning to solve problems

2006-04 Marta Sabou (VU)
Building Web Service Ontologies

2006-05 Cees Pierik (UU)
Validation Techniques for Object-Oriented Proof Outlines

2006-06 Ziv Baida (VU)
Software-aided Service Bundling - Intelligent Methods & Tools for Graphical Service Modeling

2006-07 Marko Smiljanic (UT)
XML schema matching – balancing efficiency and effectiveness by means of clustering

2006-08 Eelco Herder (UT)
Forward, Back and Home Again - Analyzing User Behavior on the Web

2006-09 Mohamed Wahdan (UM)
Automatic Formulation of the Auditor's Opinion

2006-10 Ronny Siebes (VU)
Semantic Routing in Peer-to-Peer Systems

2006-11 Joeri van Ruth (UT)
Flattening Queries over Nested Data Types
2006-12 Bert Bongers (VU)
Interactivation - Towards an e-cology of people, our technological environment, and the arts
2006-13 Henk-Jan Lebbink (UU)
Dialogue and Decision Games for Information Exchanging Agents
2006-14 Johan Hoorn (VU)
Software Requirements: Update, Upgrade, Redesign - towards a Theory of Requirements Change
2006-15 Rainer Malik (UU)
CONAN: Text Mining in the Biomedical Domain
2006-16 Carsten Riggelsen (UU)
Approximation Methods for Efficient Learning of Bayesian Networks
2006-17 Stacey Nagata (UU)
User Assistance for Multitasking with Interruptions on a Mobile Device
2006-18 Valentin Zhizhkun (UVA)
Graph transformation for Natural Language Processing
2006-19 Birna van Riemsdijk (UU)
Cognitive Agent Programming: A Semantic Approach
2006-20 Marina Velikova (UvT)
Monotone models for prediction in data mining
2006-21 Bas van Gils (RUN)
Aptness on the Web
2006-22 Paul de Vrieze (RUN)
Fundaments of Adaptive Personalisation
2006-23 Ion Juvina (UU)
Development of Cognitive Model for Navigating on the Web
2006-24 Laura Hollink (VU)
Semantic Annotation for Retrieval of Visual Resources
2006-25 Madalina Drugan (UU)
Conditional log-likelihood MDL and Evolutionary MCMC
2006-26 Vojkan Mihajlović (UT)
Score Region Algebra: A Flexible Framework for Structured Information Retrieval
2006-27 Stefano Bocconi (CWI)
Vox Populi: generating video documentaries from semantically annotated media repositories
2006-28 Borkur Sigurbjornsson (UVA)
Focused Information Access using XML Element Retrieval
2007-01 Kees Leune (UvT)
Access Control and Service-Oriented Architectures
2007-02 Wouter Teepe (RUG)
Reconciling Information Exchange and Confidentiality: A Formal Approach
2007-03 Peter Mika (VU)
Social Networks and the Semantic Web
2007-04 Jurriaan van Diggelen (UU)
Achieving Semantic Interoperability in Multi-agent Systems: a dialogue-based approach
2007-05 Bart Schermer (UL)
Software Agents, Surveillance, and the Right to Privacy: a Legislative Framework for Agent-enabled Surveillance
2007-06 Gilad Mishne (UVA)
Applied Text Analytics for Blogs
2007-07 Nataša Jovanović (UT)
To Whom It May Concern - Addressee Identification in Face-to-Face Meetings
2007-08 Mark Hoogendoorn (VU)

Modeling of Change in Multi-Agent Organizations

2007-09 David Mobach (VU)
Agent-Based Mediated Service Negotiation

2007-10 Huib Aldewereld (UU)
Autonomy vs. Conformity: an Institutional Perspective on Norms and Protocols

2007-11 Natalia Stash (TUE)
Incorporating Cognitive/Learning Styles in a General-Purpose Adaptive Hypermedia System

2007-12 Marcel van Gerven (RUN)
Bayesian Networks for Clinical Decision Support: A Rational Approach to Dynamic Decision-Making under Uncertainty

2007-13 Rutger Rienks (UT)
Meetings in Smart Environments; Implications of Progressing Technology

2007-14 Niek Bergboer (UM)
Context-Based Image Analysis

2007-15 Joyca Lacroix (UM)
NIM: a Situated Computational Memory Model

2007-16 Davide Grossi (UU)
Designing Invisible Handcuffs. Formal investigations in Institutions and Organizations for Multi-agent Systems

2007-17 Theodore Charitos (UU)
Reasoning with Dynamic Networks in Practice

2007-18 Bart Orriens (UvT)
On the development an management of adaptive business collaborations

2007-19 David Levy (UM)
Intimate relationships with artificial partners

2007-20 Slinger Jansen (UU)
Customer Configuration Updating in a Software Supply Network

2007-21 Karianne Vermaas (UU)
Fast diffusion and broadening use: A research on residential adoption and usage of broadband internet in the Netherlands between 2001 and 2005

2007-22 Zlatko Zlatev (UT)
Goal-oriented design of value and process models from patterns

2007-23 Peter Barna (TUE)
Specification of Application Logic in Web Information Systems

2007-24 Georgina Ramírez Camps (CWI)
Structural Features in XML Retrieval

2007-25 Joost Schalken (VU)
Empirical Investigations in Software Process Improvement

2008-01 Katalin Boer-Sorbán (EUR)
Agent-Based Simulation of Financial Markets: A modular, continuous-time approach

2008-02 Alexei Sharpanskykh (VU)
On Computer-Aided Methods for Modeling and Analysis of Organizations

2008-03 Vera Hollink (UVA)
Optimizing hierarchical menus: a usage-based approach

2008-04 Ander de Keijzer (UT)
Management of Uncertain Data - towards unattended integration

2008-05 Bela Mutschler (UT)
Modeling and simulating causal dependencies on process-aware information systems from a cost perspective

2008-06 Arjen Hommersom (RUN)
On the Application of Formal Methods to Clinical Guidelines, an Artificial Intelligence Perspective

2008-07 Peter van Rosmalen (OU)
Supporting the tutor in the design and support of adaptive e-learning

2008-08 Janneke Bolt (UU)
Bayesian Networks: Aspects of Approximate Inference
2008-09 Christof van Nimwegen (UU)
The paradox of the guided user: assistance can be counter-effective
2008-10 Wouter Bosma (UT)
Discourse oriented summarization
2008-11 Vera Kartseva (VU)
Designing Controls for Network Organizations: A Value-Based Approach
2008-12 Jozsef Farkas (RUN)
A Semiotically Oriented Cognitive Model of Knowledge Representation
2008-13 Caterina Carraciolo (UVA)
Topic Driven Access to Scientific Handbooks
2008-14 Arthur van Bunningen (UT)
Context-Aware Querying; Better Answers with Less Effort
2008-15 Martijn van Otterlo (UT)
The Logic of Adaptive Behavior: Knowledge Representation and Algorithms for the Markov Decision Process Framework in First-Order Domains.
2008-16 Henriëtte van Vugt (VU)
Embodied agents from a user's perspective
2008-17 Martin Op 't Land (TUD)
Applying Architecture and Ontology to the Splitting and Allying of Enterprises
2008-18 Guido de Croon (UM)
Adaptive Active Vision
2008-19 Henning Rode (UT)[1]
From Document to Entity Retrieval: Improving Precision and Performance of Focused Text Search
2008-20 Rex Arendsen (UVA)
Geen bericht, goed bericht. Een onderzoek naar de effecten van de introductie van elektronisch berichtenverkeer met de overheid op de administratieve lasten van bedrijven.
2008-21 Krisztian Balog (UVA)
People Search in the Enterprise
2008-22 Henk Koning (UU)
Communication of IT-Architecture
2008-23 Stefan Visscher (UU)
Bayesian network models for the management of ventilator-associated pneumonia
2008-24 Zharko Aleksovski (VU)
Using background knowledge in ontology matching
2008-25 Geert Jonker (UU)
Efficient and Equitable Exchange in Air Traffic Management Plan Repair using Spender-signed Currency
2008-26 Marijn Huijbregts (UT)
Segmentation, Diarization and Speech Transcription: Surprise Data Unraveled
2008-27 Hubert Vogten (OU)
Design and Implementation Strategies for IMS Learning Design
2008-28 Ildiko Flesch (RUN)
On the Use of Independence Relations in Bayesian Networks
2008-29 Dennis Reidsma (UT)
Annotations and Subjective Machines - Of Annotators, Embodied Agents, Users, and Other Humans
2008-30 Wouter van Atteveldt (VU)
Semantic Network Analysis: Techniques for Extracting, Representing and Querying Media Content
2008-31 Loes Braun (UM)

[1] Thanks a lot for the LaTeX source of this list!

Pro-Active Medical Information Retrieval

2008-32 Trung H. Bui (UT)
Toward Affective Dialogue Management using Partially Observable Markov Decision Processes

2008-33 Frank Terpstra (UVA)
Scientific Workflow Design; theoretical and practical issues

2008-34 Jeroen de Knijf (UU)
Studies in Frequent Tree Mining

2008-35 Ben Torben Nielsen (UvT)
Dendritic morphologies: function shapes structure

2009-01 Rasa Jurgelenaite (RUN)
Symmetric Causal Independence Models

2009-02 Willem Robert van Hage (VU)
Evaluating Ontology-Alignment Techniques

2009-03 Hans Stol (UvT)
A Framework for Evidence-based Policy Making Using IT

2009-04 Josephine Nabukenya (RUN)
Improving the Quality of Organisational Policy Making using Collaboration Engineering

2009-05 Sietse Overbeek (RUN)
Bridging Supply and Demand for Knowledge Intensive Tasks - Based on Knowledge, Cognition, and Quality

2009-06 Muhammad Subianto (UU)
Understanding Classification

2009-07 Ronald Poppe (UT)
Discriminative Vision-Based Recovery and Recognition of Human Motion

2009-08 Volker Nannen (VU)
Evolutionary Agent-Based Policy Analysis in Dynamic Environments

2009-09 Benjamin Kanagwa (RUN)
Design, Discovery and Construction of Service-oriented Systems

2009-10 Jan Wielemaker (UVA)
Logic programming for knowledge-intensive interactive applications

2009-11 Alexander Boer (UVA)
Legal Theory, Sources of Law & the Semantic Web

2009-12 Peter Massuthe (TUE, Humboldt-Universitaet zu Berlin)
Operating Guidelines for Services

2009-13 Steven de Jong (UM)
Fairness in Multi-Agent Systems

2009-14 Maksym Korotkiy (VU)
From ontology-enabled services to service-enabled ontologies (making ontologies work in e-science with ONTO-SOA)

2009-15 Rinke Hoekstra (UVA)
Ontology Representation - Design Patterns and Ontologies that Make Sense

2009-16 Fritz Reul (UvT)
New Architectures in Computer Chess

2009-17 Laurens van der Maaten (UvT)
Feature Extraction from Visual Data

2009-18 Fabian Groffen (CWI)
Armada, An Evolving Database System

2009-19 Valentin Robu (CWI)
Modeling Preferences, Strategic Reasoning and Collaboration in Agent-Mediated Electronic Markets

2009-20 Bob van der Vecht (UU)
Adjustable Autonomy: Controling Influences on Decision Making

2009-21 Stijn Vanderlooy (UM)
Ranking and Reliable Classification

2009-22 Pavel Serdyukov (UT)
Search For Expertise: Going beyond direct evidence
2009-23 Peter Hofgesang (VU)
Modelling Web Usage in a Changing Environment
2009-24 Annerieke Heuvelink (VU)
Cognitive Models for Training Simulations
2009-25 Alex van Ballegooij (CWI)
RAM: Array Database Management through Relational Mapping
2009-26 Fernando Koch (UU)
An Agent-Based Model for the Development of Intelligent Mobile Services
2009-27 Christian Glahn (OU)
Contextual Support of social Engagement and Reflection on the Web
2009-28 Sander Evers (UT)
Sensor Data Management with Probabilistic Models
2009-29 Stanislav Pokraev (UT)
Model-Driven Semantic Integration of Service-Oriented Applications
2009-30 Marcin Zukowski (CWI)
Balancing vectorized query execution with bandwidth-optimized storage
2009-31 Sofiya Katrenko (UVA)
A Closer Look at Learning Relations from Text
2009-32 Rik Farenhorst (VU) and Remco de Boer (VU)
Architectural Knowledge Management: Supporting Architects and Auditors
2009-33 Khiet Truong (UT)
How Does Real Affect Affect Affect Recognition In Speech?
2009-34 Inge van de Weerd (UU)
Advancing in Software Product Management: An Incremental Method Engineering Approach

Summary

The increasing availability of cheap, small, low-power sensor hardware and the ubiquity of wired and wireless networks have led to the prediction that 'sensing environments' will emerge in the near future. The sensors in these environments collect detailed information about the situation people are in, which is used to enhance information-processing applications that are present on their mobile and 'ambient' devices.

Mediating between the sensor data supply and demand sides poses new requirements to data management. In this thesis, we identify and investigate the challenge of dealing with the *uncertainty* inherent in sensor data processing. This 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.

In particular, *Bayesian networks* form a good candidate for modeling sensor data in a flexible environment, because of their comprehensiveness and modularity. We provide extensive technical argumentation for this claim. As a demonstration case, we define a discrete Bayesian network for location tracking using Bluetooth transceivers.

In order to scale up sensor models, efficient probabilistic *inference* is crucial. We observe that the conventional inference methods for Bayesian networks, which have mainly been developed in the medical and hardware diagnosis domain, do not scale well for our demonstration case. We propose several optimizations, making it possible to jointly scale up the number of locations and sensors in sublinear time, and to scale up the time resolution in linear time.

More important than these optimizations themselves is the way by which we arrive at them; we define a straightforward theoretical framework for translating an inference query into relational algebra. This allows the query to be analyzed and optimized using insights and techniques from the database community; for example, using cost metrics based on *cardinality* rather than *dimensionality*, which is common in conventional inference algorithms.

A fairly orthogonal research question investigates the possibility of collecting the transition statistics (needed for acquiring the parameters of a probabilistic model) in a local, clustered fashion, in which transitions between states of different clusters cannot be directly observed. We show that this problem can be written as a constrained system of linear equations, for which we describe a specialized solution method.

Samenvatting

Door zowel de stijgende beschikbaarheid van goedkope, kleine en energiezuinige sensoren als de alomtegenwoordigheid van (draadloze) netwerken verwacht men in de nabije toekomst 'gevoelige omgevingen'. De sensoren in deze omgevingen verzamelen gedetailleerde informatie over de situatie waarin personen zich bevinden, die gebruikt wordt om informatieverwerkende toepassingen op mobiele en 'ambient' apparaten te verbeteren.

Het bij elkaar brengen van vraag en aanbod van sensor-data stelt nieuwe eisen aan data management. In dit proefschrift wordt de inherente *onzekerheid* in de verwerking van sensor data geïdentificeerd en geanalyseerd. Deze onzekerheid kent vele oorzaken: ruis in metingen, ontbrekende gegevens door uitval van sensoren of netwerkverbindingen, het 'semantische gat' tussen meetgegevens en relevante informatie, en het samenvoegen van gegevens uit meerdere bronnen. *Probabilistische modellen* beschrijven deze onzekerheden d.m.v. het modulaire, bewezen raamwerk van de kansrekening, en worden vaak toegepast voor het verwerken van sensor-data.

In het bijzonder zijn *Bayesiaanse netwerken* een goede kandidaat voor het modelleren van sensor-data in een flexibele omgeving vanwege hun veelomvattendheid en modulariteit; we voorzien deze bewering van een uitgebreide technische onderbouwing. Als voorbeeldcasus wordt een discreet Bayesiaans netwerk gedefinieerd voor localisatie d.m.v. Bluetooth-ontvangers.

Voor schaalvergroting van sensormodellen is efficiënte probabilistische *inferentie* van cruciaal belang. De conventionele methoden hiervoor, die voornamelijk ontwikkeld zijn in het domein van medische en hardware-diagnostiek, presteren slecht bij schaalvergroting van onze casus. We beschrijven een aantal optimalisaties waardoor de inferentie-tijd sublineair afhankelijk wordt van het aantal onderscheiden locaties en het aantal sensoren, en lineair van de tijdsresolutie.

Belangrijker dan deze optimalisaties zelf is de manier waarop we ertoe komen, namelijk via een eenvoudig relationele-algebraraamwerk dat we definiëren voor inferentie-queries. Hierdoor kan de query geanalyseerd en geoptimaliseerd worden d.m.v. inzichten en technieken uit de databasegemeenschap, bijvoorbeeld met kostenmetrieken die op *cardinaliteit* gebaseerd zijn i.p.v. zoals gebruikelijk op *dimensionaliteit*.

Een enigszins losstaande onderzoeksvraag behandelt de mogelijkheid om transitiestatistieken (benodigd voor de parameters van een probabilistisch model) lokaal in clusters te verzamelen, waarvoor geldt dat transities tussen verschillende clusters niet precies geobserveerd kunnen worden. We laten zien dat dit vraagstuk beschreven kan worden door een ingeperkt stelsel van lineaire vergelijkingen en geven hiervoor een gespecialiseerde oplossingsmethode.

He says, says Alan, that if you stand outside probabilistic discourse
then probability statements make no sense. That is a fair enough statement
as far as it goes. But what he forgets is that in a probabilistic universe *there is*
nowhere to stand outside probability. It is all of a piece with his idea that
numbers stand for something outside themselves, though he can't say what.
The fact is, numbers are just numbers. They don't stand for anything.
They are nuts and bolts, the nuts and bolts of mathematics. They are what we
utilize when we work with mathematics in the real world. Look around you.
Look at bridges. Look at traffic flows. Look at the movement of money.
Numbers work. Mathematics works. Probabilities work.
That is all we need to know.

J.M. Coetzee, *Diary of a Bad Year*