A Framework for Processing Correlated Probabilistic Data

Sebastiaan Johannes van Schaik

St. Cross College

Michaelmas Term 2014

Submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy

Department of Computer Science

University of Oxford

A Framework for Processing Correlated Probabilistic Data

Sebastiaan Johannes van Schaik
St. Cross College

D.Phil. Thesis
Michaelmas Term 2014

Abstract

The amount of digitally-born data has surged in recent years. In many scenarios, this data is inherently *uncertain* (or: *probabilistic*), such as data originating from sensor networks, image and voice recognition, location detection, and automated web data extraction. Probabilistic data requires novel and different approaches to data mining and analysis, which explicitly account for the uncertainty and the correlations therein.

This thesis introduces *ENFrame*: a framework for processing and mining correlated probabilistic data. Using this framework, it is possible to express both traditional and novel algorithms for data analysis in a special *user language*, without having to explicitly address the uncertainty of the data on which the algorithms operate. The framework will subsequently execute the algorithm on the probabilistic input, and perform exact or approximate parallel probability computation. During the probability computation, correlations and provenance are succinctly encoded using probabilistic events.

This thesis contains novel contributions in several directions. An expressive *user language* – a subset of Python – is introduced, which allows a programmer to implement algorithms for probabilistic data without requiring knowledge of the underlying probabilistic model. Furthermore, an *event language* is presented, which is used for the probabilistic interpretation of the user program. The event language can succinctly encode arbitrary correlations using *events*, which are the probabilistic counterparts of deterministic user program variables. These highly interconnected events are stored in an *event network*, a probabilistic interpretation of the original user program. Multiple techniques for exact and approximate probability computation (with error guarantees) of such event networks are presented, as well as techniques for parallel computation.

Adaptations of multiple existing data mining algorithms are shown to work in the framework, and are subsequently subjected to an extensive experimental evaluation. Additionally, a use-case is presented in which a probabilistic adaptation of a clustering algorithm is used to predict faults in energy distribution networks. Lastly, this thesis presents techniques for integrating a number of different probabilistic data formalisms for use in this framework and in other applications.

To Wil and Elly

Acknowledgements

Although my name appears on the title page, this thesis is just as much the work of the people who were there for me during this four-year journey, as it is mine. Without their help and unconditional support, this thesis would be very much non-existent.

In particular, I am thankful to the following people:

My parents, Wil and Elly – for supporting me from the very beginning, for raising and curbing the geek inside me.

My supervisor, Dan Olteanu – for showing me the way, sticking to my stubborn side, and successfully fighting off a plethora of distractions along the way.

Claire – for always being there when the road looked rough ahead, and for waltzing with me on Port Meadow.

Steven – for being the best friend anyone could ask for, and for meticulously picking me up on inaccuracies and overly bold claims in this thesis (and life).

Sander, Anne, and Ton – for endless amazing memories in many cities, in various countries, on multiple continents. For offering me places to stay, and for maintaining my Dutch.

Arno Siebes and Tim Furche – for kindly agreeing to act as my D.Phil. examiners, the constructive comments on this thesis, and for the supportive words over the years.

Tim Pound – for providing invaluable advice.

Thank you all.

This thesis would not have been possible without the financial support from the European Commission (in the form of the FP7 research project 'HiPerDNO'), the Engineering and Physical Sciences Research Council (in the form of the 'ADEPT' research project), het Prins Bernhard Cultuurfonds, and het De Breed Kreiken Innovatiefonds.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

The applications of data processing and analysis are wide-ranging, with impact on physics, astronomy, engineering, medical science, climate science, and many other areas of academia. In recent years, society has come to depend on data science. From hospitals, banks, weather services, large-scale grocers, to social networks and intelligence agencies: data storage and analysis now plays an essential role in companies, governments, and research institutions of all types and sizes.

In the past few years, new data sources have emerged which produce data exhibiting large measures of uncertainty. For example, geolocation data (from various devices, including smartphones [ZB07]), handwriting recognition [PS00], data from sensor networks [DGM05], and data fusion [DMG+14]. In some cases, certain data[1] is deliberately made uncertain through aggregation or injection of noise – techniques often used for privacy-preserving data analysis of demographic data sets.

Traditional database management systems are not suitable for storing this *probabilistic data* [SORK11]. This realisation has led to a series of developments in the field of probabilistic databases. At one end of the spectrum, there have been sustained systems building efforts. These have resulted in the development of new probabilistic database management systems, such as MayBMS [AJKO08; HAKO09], ORION/U-DBMS [CSP05], and MCDB [JXW+08]. At the other end, there is extensive analysis of computational problems for rich classes of queries and probabilistic data models of varying expressivity [SORK11]. As a result, our understanding of the space of possible data models and their implication on query tractability has improved, and newly developed probabilistic database management systems offer a variety of ways to explicitly annotate data with some notion of

[1] In this thesis, the term *certain data* (consisting of *certain data points*) is used to refer to data which does not exhibit any uncertainty. Also see the glossary in Appendix A.

uncertainty. These range from simple probabilities on a per-tuple basis (assuming probabilistic independence), to more advanced continuous or categorical probability distributions over individual tuple values, and detailed lineage specifications using propositional formulae [GT06].

At the same time, researchers in data analytics and mining have become interested in probabilistic data. This has lead to the development of various new algorithms for data analysis, as well as adaptations of existing algorithms. However, despite the uncertainty in the probabilistic input, these algorithms often produce a deterministic output, discarding the probabilistic nature of the data in the process. This frustrates further processing using probabilistic methods [Agg09a]. On top of that, most state-of-the-art probabilistic data mining approaches do not extend beyond the restricted probabilistic model in which the input is tuple-independent [AY09]. Not only does this simplified approach result in significant errors [VRH+09], the misalignment between progress in the field of probabilistic databases and the development of new data mining algorithms also hinders the development of data processing systems that integrate techniques for both fields.

There is a growing need for computing platforms that allow programmers to build applications which use uncertain data, without having to be concerned with the underlying uncertain nature of such data, or the attendant computationally hard inference task [DAR13; GHNR14]. Existing systems offer support for querying probabilistic data, but more complex tasks (such as data mining) require a high level of expertise in probabilistic databases and probability theory. This hinders the adoption of new technologies developed in the fields of databases and data analytics.

This thesis introduces ENFrame[2]: a framework for processing and mining probabilistic data with arbitrary correlations. The goal of ENFrame is to bridge the gap between developments in the fields of probabilistic databases, data mining, and programming languages by providing users with an intuitive way to specify programs that query, analyse, and mine probabilistic data. Throughout this thesis, the term *processing of (uncertain) data* will be used to refer to mining and querying such data. Appendix A contains a glossary of terms.

1.1 Probabilistic data

Recent applications of sensor networks in various fields have produced data with a high degree of uncertainty [DGM+04; Agg09a], requiring specialist techniques for data analysis. A number of techniques and algorithms have been proposed to process and mine this type of probabilistic

[2]ENFrame (\in-'frām\). From 'to enframe': to enclose in, or as if in, a frame (Webster's Dictionary, 1913). Additionally, a contraction of the acronym 'EN' (for *event network*) and 'Frame' (for *framework*).

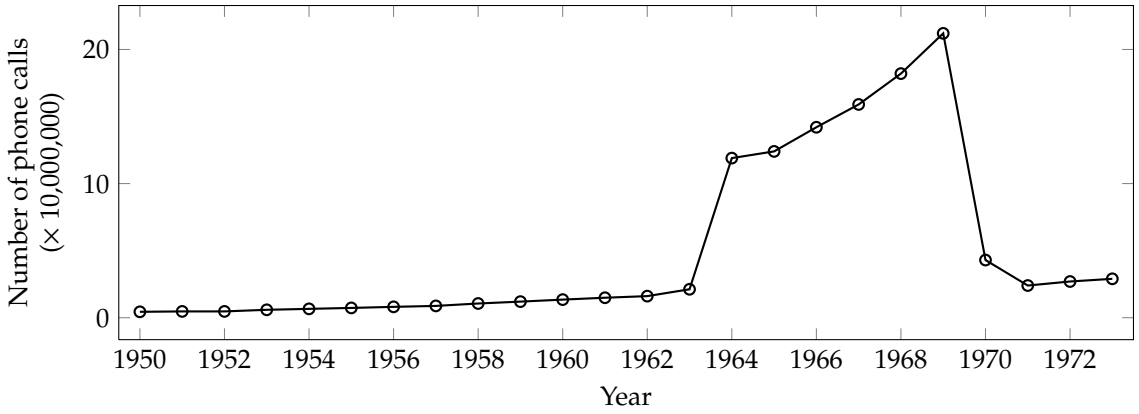


Figure 1.1: Number of international phone calls, as reported by a Belgian statistical survey (adapted from [RL87])

data, ranging from probabilistic association rule learning [CGG10; PZC+13], to probabilistic convex hull queries (*e.g.*, for tracking fish movement in the Pacific [YZNL14]), and probabilistic clustering [GPT08; CCKN06] (see Chapter 7 for a use-case [OvS12]). In the biomedical sciences, classification algorithms (including *k*-nearest neighbour classification) and probabilistic range queries are used for the analysis of the location and extent of cells in images [LS09].

One of the biggest challenges in data mining is dealing with data of poor quality which exhibits spurious patterns [WFH11]. An often-used example is the data set describing international phone calls made from Belgium between 1950 and 1973 [RL87], which is depicted in Figure 1.1. After the unusual pattern between the years of 1963 and 1970 was discovered and manually investigated, it turned out to have been caused by a human error in recording the data: during the anomalous years, the *number of minutes* spent on international phone calls was recorded, rather than the *number* of international phone calls.

Traditionally, data miners have approached the noisy data problem by using data cleansing techniques prior to mining the data. In some situations (as in the example of the Belgian international phone calls) this can lead to good results, but there is always a risk of throwing the baby out with the bath water. This is one of the main reasons probabilistic data has seen a significant rise in popularity over recent years: rather than discarding data points which are identified as *outliers*, these data points are retained, but annotated with a lower probability of being accurate. This uncertainty can be used in subsequent data processing steps.

A very recent development in the area of knowledge bases shows how probabilistic data plays a vital role in automated information integration and derivation. The Google Knowledge Vault [DMG+14] applies supervised machine learning techniques to fuse various information sources,

including the Google's own Knowledge Graph (built on top of Freebase [BEP+08]). It uses probabilistic inference to compute calibrated probabilities: every derived fact in the Knowledge Vault is stored as an RDF triple (subject, predicate, object) associated with a confidence score that expresses the likelihood that a fact is correct. This yields an enormous probabilistic knowledge base: as of October 2013, the Knowledge Vault contained 1.6 billion RDF triples. Over 300 million of these triples have confidence level ≥ 0.7, of which a third is not included in Freebase. This makes the Google Knowledge Vault about 40 times larger than DeepDive [NZRS12], the next largest knowledge base.

1.2 Effects of uncertainty on data mining

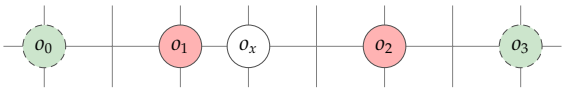
Traditional algorithms for data processing are not equipped to take data uncertainty into account. A probabilistic input asks for a probabilistic output, whereas existing algorithms that operate on certain data often produce a deterministic output. The two examples below illustrate how data uncertainty can have a significant impact on the result of classification and clustering algorithms.

1.2.1 Effects on classification

Supervised learning (mining) techniques rely on prior information to assist the data mining algorithm in its task. In classification, one or more unlabelled data points are assigned to a class using known class labels of other points. This technique has multiple applications, including credit card fraud detection, consumer profiling, and pattern recognition.

EXAMPLE 1.1: Effects of uncertainty on nearest neighbour classification

Consider the following classification problem with an unlabelled object (o_x) and four labelled objects ($o_0, \ldots, o_3$) which belong to classes *red* (solid border) and *green* (dashed border).

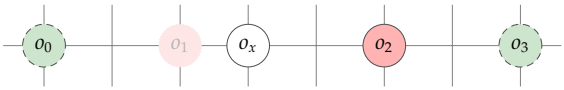


The unlabelled object is classified by looking at the class labels of its three nearest neighbouring objects: o_0, o_1, o_2 (thick borders). Two of these three nearest neighbours are red – hence, o_x is assigned class *red*.

Probabilistic data is commonly governed by the *possible worlds semantics*, under which every probabilistic database induces a probability distribution over regular (deterministic) databases. Under these semantics, tuples (data points) are only part of a limited number of

possible worlds. In other words, a data point *exists* with a certain probability. The possible worlds semantics are described in more detail in Chapter 2.

Reconsider the classification scenario, but now assume that one or more objects are *probabilistic*, *i.e.*, they only exist in a subset of the possible worlds. The presence or absence of even a single object has a significant influence on the classification result. For example, consider the possible world in which o_1 does not exist:



In this new setting, *green* is the prevailing class and o_x is therefore assigned class *green*.

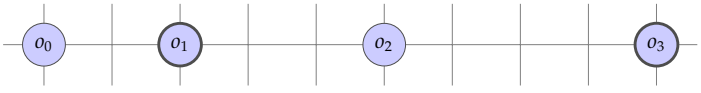
The example illustrates how the uncertainty of a single object yields two different classification scenarios, with two very different classification results. For a single data set with probabilistic data points, the number of possible scenarios grows exponentially in the number of data points.

1.2.2 Effects on clustering

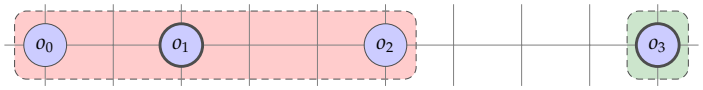
Clustering is another popular data mining technique, but unlike classification, it is typically unsupervised. Unsupervised data mining algorithms do not rely on prior information, but attempt to find patterns in unlabelled data. Clustering algorithms partition a data set into two or more clusters in such a way that data points in the same cluster are more similar to each other than to data points in other groups (according to a predefined similarity measure). Applications of clustering include image recognition [JF96], social network analysis [MSST07], and search engines [WNZ01]. As with classification, data uncertainty has a significant effect on a clustering result.

EXAMPLE 1.2: Effects of uncertainty on clustering result

Consider the following one-dimensional data set with four data points $o_0 \ldots o_3$:

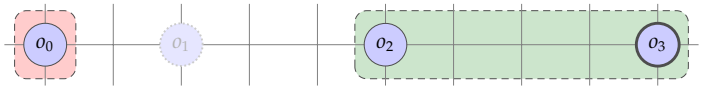


In the following image, these data points have been clustered into two clusters (red and green) using the following trivial algorithm. The procedure starts with two clusters at the two opposite ends of the feature space (at o_0 and o_3), and grows the clusters by iteratively allowing the cluster with the closest 'free' data point to absorb that data point:

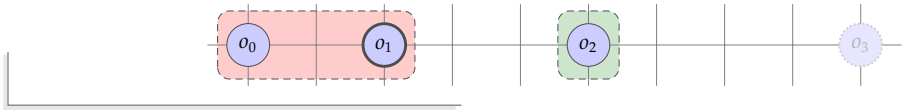


The green cluster never grows past its one and only data point o_3, because all data points are closer to the red cluster at all times.

If one or more data data points are probabilistic, the clustering result changes significantly. For example, in the possible world in which o_1 does not exist:



This small change results in a significantly different clustering result. Taking yet another object's uncertainty into account, the clustering result could change again:



A naïve approach to clustering this type of uncertain data would iterate over the (worst-case: exponentially many) possible instances, resulting in a probability distribution over (possibly equally) many clusterings. Among other things, this thesis contributes techniques that compute an output equivalent to processing all possible instances, but that attempt to mitigate the computational complexity.

1.3 The complexity of probabilistic data

In recent years, *big data* has become a popular term to describe large data sets. The term is frequently used in both industry and academia; there are multiple definitions of *big data* (*e.g.*, [Jac09; BHH+12]), most of which are based purely on the size of the data.

Probabilistic data is *big* – not necessarily because of the number of tuples in a data set, but because of its big *computational complexity*. As soon as uncertainty starts to play a serious role in any data-driven problem, its complexity increases [Agg09a]. The clustering and classification examples illustrate how uncertainty has an impact on computational complexity, even for relatively straightforward algorithms on very small data sets. Exact computation scales exponentially in the size of the data set (worst-case), because every data point can be both *in* and *out* – *i.e.*, considered 'correct' with a probability p and incorrect with probability $1 - p$.

From a programmer's point of view, data uncertainty also adds significant *conceptual complexity* to an algorithm. Regardless of the exact probabilistic data formalism (see Chapter 2), an uncertain input calls for an uncertain output, *e.g.*, a probability distribution over possible classes (classification) or partitionings (clustering). Although existing probabilistic databases offer a viable solution for simple data processing tasks such as querying, development of more complex data processing techniques currently requires a high level of expertise in probabilistic databases and probability theory.

Although the development of languages for programming with probabilistic models recently attracted a lot of attention (*e.g.*, [Roy; MWGK12]), integration of such languages with probabilistic databases is still inadequate. Furthermore, using probabilistic languages requires a thorough understanding of probability theory, which hinders wide-spread adoption.

1.4 ENFrame: a framework for processing probabilistic data

The work described in this thesis tries to bridge the gap between developments in probabilistic databases, data mining, and programming languages in order to reduce the computational and conceptual complexity of processing probabilistic data. To that end, *ENFrame* was developed: a novel framework for processing probabilistic data.

The framework reduces the conceptual complexity of processing probabilistic data by allowing users to specify programs to query, analyse, and mine such data using a *user language*. The framework interprets these programs probabilistically, and produces a probabilistic output governed by the well-defined and established *possible worlds semantics*. By making it easier for researchers and programmers to perform sound analysis of probabilistic data, the framework strives for a more wide-spread adoption of this new type of data and its analysis.

The semantics of ENFrame programs is based on a unified probabilistic interpretation of the entire processing pipeline, from the input data, through the program computation, all the way to the program output. Under the possible worlds semantics, the input is a probability distribution over a finite set of possible worlds, with each world defining a deterministic database. Consequently, the input to ENFrame can consist of arbitrarily correlated and independent probabilistic events. Existing probabilistic data mining algorithms scarcely provide support for correlated data. This feature is in line with the latest developments in probabilistic databases [SORK11], and allows for integration of ENFrame into data pipelines that combine various techniques for analysing and querying probabilistic data.

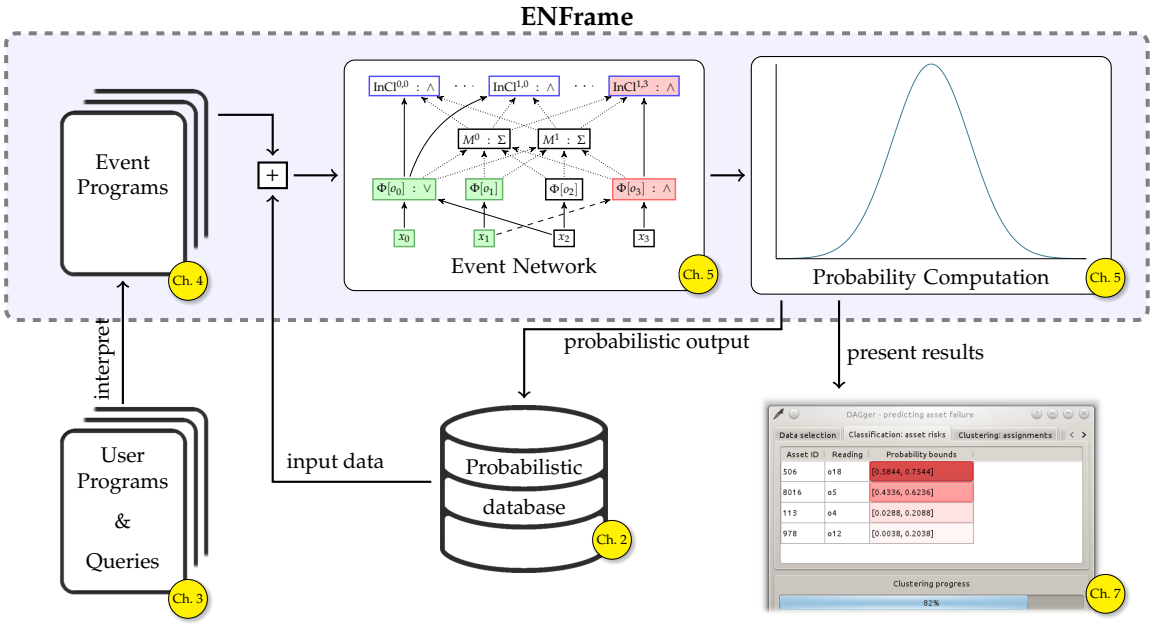


Figure 1.2: Diagram of the architecture of ENFrame

The framework features an expressive set of programming constructs (such as assignments, bounded-range loops, list comprehension, aggregate operations on lists, and calls to database engines). Its language is based on Python [vRos97], coupled with aspects of probabilistic databases – such as a strong foundation in possible worlds semantics, support for arbitrary data correlations, and exact and approximate parallel probability computation with error guarantees. ENFrame integrates seamlessly with probabilistic databases and exploits existing work in the field [OHK09]. The workings of the framework are exemplified in this thesis using adaptations of various clustering and classification algorithms, which are also used for an experimental evaluation.

The Python-based language allows the user (programmer) to write programs without the need for experience in probability theory or with probabilistic databases. In fact, the programmer need not be aware of the probabilistic nature of data or the possibly different probabilistic formalisms used by the input data sources. ENFrame will interpret the program probabilistically, and construct probability distributions for variables which represent the program output.

1.4.1 ENFrame architecture (thesis outline)

A diagram of ENFrame's architecture is presented in Figure 1.2. This thesis is predominantly organised around the different components of the framework's architecture, as is indicated by the chapter references in the diagram.

The programmer's workflow starts with writing a program in ENFrame's user language (described in Chapter 3). The framework constructs a probabilistic interpretation of these programs in the form of event programs (Chapter 4). Based on input data from probabilistic databases (Chapter 2), the event programs yield a graph data structure of highly interconnected (correlated) probabilistic events: an event network (Chapter 5). After the probabilities of nodes in the network have been computed, the results can be stored in a probabilistic database for further processing, or presented to the user.

The full outline of this thesis is as follows. Chapter 2 opens with an introduction to probabilistic databases and data formalisms. It sets up the context and describes the foundation of this thesis, as well as describing novel work on integrating different formalisms.

The framework's user language is introduced in Chapter 3, which contains a specification of three clustering algorithms (*k*-medoids, *k*-means, and Markov clustering) and a classification algorithm (*k*-nearest neighbour) as ENFrame user programs, followed by a description of the language's grammar.

Chapter 4 describes ENFrame's event language and shows probabilistic translations of the user programs from Chapter 3, alongside the probabilistic semantics of events and the grammar of the event language.

In Chapter 5, the event networks generated by event programs are described, with algorithms for exact and approximate probability computation using both parallel and sequential approaches.

Chapter 6 contains an experimental evaluation of the framework with the *k*-medoids clustering and *k*-nearest neighbour classification algorithms, using the various algorithms for probability computation. It also introduces a novel technique for comparing probabilistic results and evaluating the performance of probabilistic algorithms in terms of output quality.

A real-world use-case of the framework is presented in Chapter 7, which describes how clustering of probabilistic data can be used to predict faults in electricity distribution networks. Additionally, the chapter demonstrates techniques for querying multiple data sources with different formalisms.

The thesis closes with a discussion, related work, and suggested directions for future research in Chapter 8.

1.5 Contributions

This thesis' main contribution is ENFrame, which was researched, designed, and implemented from scratch. The development of the framework required novel contributions in several directions, which are outlined below.

A *user language*. The language (introduced in Chapter 3) is designed to provide an easily accessible, familiar, and intuitive interface to programming for probabilistic data. The syntax and semantics of the user language are formally defined in this thesis, and the language is illustrated with example programs for the k-nearest neighbour classification algorithm, as well as the k-medoids, k-means, and Markov clustering algorithms.

A probabilistic interpretation through an *event language*. ENFrame interprets the user programs probabilistically, translating regular variables in the user program into random variables (*events*) using an *event language* (introduced in Chapter 4). The syntax and semantics are defined in this thesis, and are illustrated using example data mining algorithms.

A data structure to store events: the *event network*. The interconnected events generated by the event program are stored in a novel graph data structure: the *event network* (introduced in Chapter 5). This network can be seen as the graph equivalent of a Boolean expression tree which can be compiled into a decomposition tree. The event network forms a trace of the original user program, which allows for sensitivity analysis and result explanation.

Algorithms for probability computation. To avoid having to iterate over all possible worlds, ENFrame features advanced heuristics for grouping similar worlds, as well a series of algorithms for efficient probability computation of events in the event network. To further speed up probability computation, various approximation strategies (with error bounds) are proposed, combined with algorithms for parallel computation (Chapter 5).

Experimental evaluation. The framework is experimentally evaluated by adapting the k-medoids clustering and k-nearest neighbour classification algorithms (introduced in Chapter 6). The algorithms were run on a real-world data set with readings from sensors in energy distribution networks, as well as on large synthetic data sets.

Additional contributions. In addition to the contributions that are directly related to ENFrame, this thesis also contributes the following:

- A series of techniques for integrating and querying various probabilistic data formalisms (first described in [OPvS13]). These techniques can be used for a lossless translation of data formalisms prior to processing the data using ENFrame.
- Generic techniques for comparing the outcome of various algorithms for probabilistic data, which can be used to evaluate the quality of an algorithm against a gold standard.

1.6 Dissemination

This thesis is a fusion of work that has been previously published in several conference proceedings and journals:

[OPvS13] Dan Olteanu, Lampros Papageorgiou and Sebastiaan J. van Schaik, 'ΠIgora: An integration system for probabilistic data', in *29th IEEE International Conference on Data Engineering, ICDE 2013, Brisbane, Australia, April 8-12, 2013*, Christian S. Jensen, Christopher M. Jermaine and Xiaofang Zhou, Eds., IEEE Computer Society, 2013, pages 1324–1327, ISBN: 978-1-4673-4909-3. DOI: 10.1109/ICDE.2013.6544935.

[OvS12] Dan Olteanu and Sebastiaan J. van Schaik, 'DAGger: clustering correlated uncertain data (to predict asset failure in energy networks)', in *The 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2012, Beijing, China, August 12-16, 2012*, Qiang Yang, Deepak Agarwal and Jian Pei, Eds., ACM, 2012, pages 1504–1507, ISBN: 978-1-4503-1462-6. DOI: 10.1145/2339530.2339766.

[OvS14a] Dan Olteanu and Sebastiaan J. van Schaik, 'ENFrame = (Programs + Queries) / Probabilistic Data', in *Big Uncertain Data workshop (BUDA) 2014, in conjunction with SIGMOD/PODS*, 2014.

[OvS14b] Dan Olteanu and Sebastiaan J. van Schaik, 'Probabilistic data programming with ENFrame', *IEEE Data Engineering Bulletin*, volume 37, number 3, pages 18–25, 2014.

[vSOF14] Sebastiaan J. van Schaik, Dan Olteanu and Robert Fink, 'ENFrame: a platform for processing probabilistic data.', in *Proceedings of the 17th International Conference on Extending Database Technology (EDBT), Athens, Greece, March 24-28, 2014*, Sihem Amer-Yahia, Vassilis Christophides, Anastasios Kementsietsidis, Minos N. Garofalakis, Stratos Idreos and Vincent Leroy, Eds., OpenProceedings.org, 2014, pages 355–366. DOI: 10.5441/002/edbt.2014.33.

Furthermore, the following invited paper has been accepted will soon be published:

[vSO16] Sebastiaan J. van Schaik and Dan Olteanu, 'ENFrame: a framework for processing correlated probabilistic data', *ACM Transactions on Database Systems*, 2016, invited paper, accepted (December 2015), to be published.

References to these articles can be found throughout this thesis. The primary product of my doctorate is the research described in this thesis. However, the number of lines of English is greatly exceeded by the number of lines of C++ code that was written for a fully functioning prototype of the framework, which was used to support the research with empirical data. Refer to Appendix B for details on availability and licensing.

More information regarding ENFrame, its ongoing development, and the contributions made can be found on the ENFrame project website at the Department of Computer Science at the University of Oxford: `www.cs.ox.ac.uk/projects/ENFrame`.

Chapter 2

Probabilistic Data and Formalisms

Recent years have witnessed a solid body of work in probabilistic data. At one end of the field, there have been sustained systems building efforts, resulting in a large number of (open and closed source) probabilistic database management systems, such as MayBMS [AJKO08; HAKO09], ORION/U-DBMS [CSP05], and MCDB [JXW+08]. At the other end, there has been extensive analysis of computational problems for various classes of queries and probabilistic data models of various expressiveness.

This chapter comprises a brief introduction to probabilistic data, and different formalisms for storing such data. The goal is to provide an overview of the current field of probabilistic data management, and sketch a context for ENFrame and its constructs, which are introduced in later chapters. Lastly, this chapter presents novel techniques for integrating different data formalisms, first published in [OPvS13].

2.1 Introduction to probabilistic data

Traditional relational database management systems store *deterministic* data: every tuple is considered to be correct and valid, all information held in the database is assumed to be *certain*. With a foundation in first order logic, languages like SQL can be used to query the information in such databases. These queries are processed by the database management system (DBMS), which is responsible for the required query planning and optimisation. For example, a SQL `select` query describes a (possibly empty) set of constraints, and returns those tuples that match the constraints, excluding those that do not match. A tuple is either *in* or *out* – it is not possible for the DBMS to

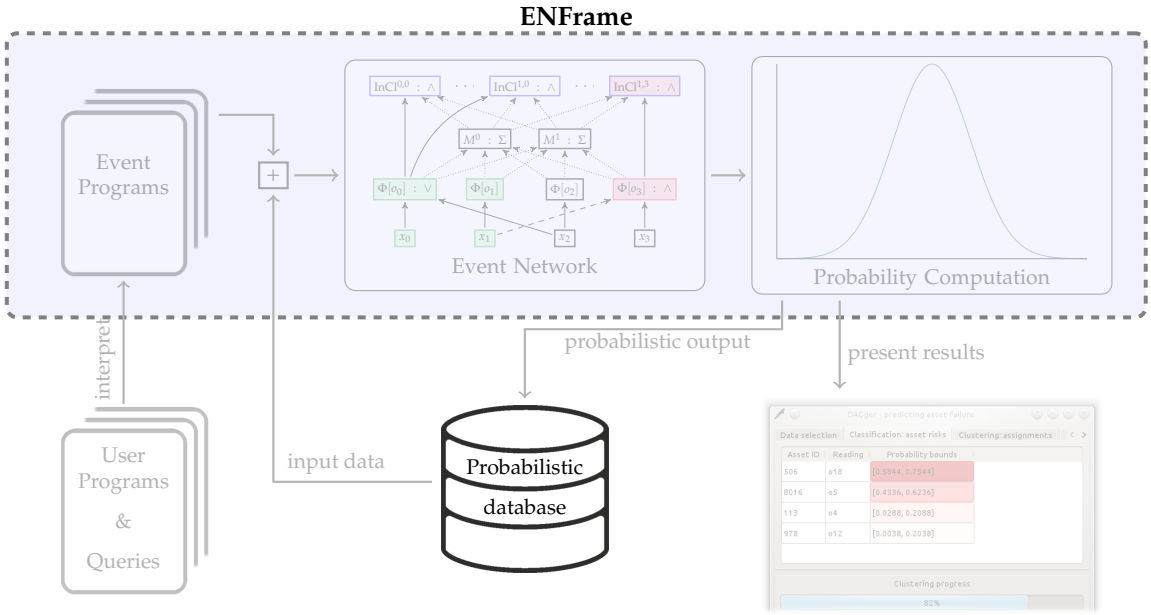


Figure 2.1: ENFrame architecture: chapter 2

return tuples that *might* occur in the query result, as the underlying data formalism does not allow for expressing the uncertainty in a query answer.

In the 1970s, when the first relational databases were introduced [Cod70], data was often (correctly) assumed to be of a certain nature. Examples include accounting data, warehousing data, and data for enterprise resource planning. With the introduction of the World Wide Web, relational databases also play a major role in content management systems and e-commerce solutions. Such traditional relational databases are able to store missing data (*i.e.*, *NULL* values), but are not able to express any notion of uncertainty. However, in recent years, new data sources have emerged that produce data which is inherently uncertain, such as sensor networks [DGM05], location (GPS) sensors [ZB07; MCA06], social networks [AR07], optical character recognition (OCR), handwriting recognition [PS00], and data extracted and integrated using automated processes [DBS09; BN08]. This shift in the nature of the data calls for more advanced ways of storing and processing such data.

Probabilistic databases (or: uncertain databases) record a notion of uncertainty together with the contained data, thereby providing a method of storing data in a way that more closely matches its nature and origin. More formally, a probabilistic database is defined as follows:

Definition 2.1: probabilistic database (adapted from [AY09; GT06])

A probabilistic-information database is a finite probability space whose outcomes are all possible database instances consistent with a given schema. This can be represented as the pair $(\mathcal{X}, \Pr)$, where $\mathcal{X}$ is a finite set of possible database instances consistent with a given schema, and $\Pr(I)$ is the probability associated with any instance $I \in \mathcal{X}$. Note that since $\Pr(\cdot)$ represents the probability vector over all instances in $\mathcal{X}$:

$$\sum_{I \in \mathcal{X}} \Pr(I) = 1$$

Instances $I \in \mathcal{X}$ are often referred to as a *possible worlds*.

This definition is restricted to probabilistic databases with discrete probability distributions (*i.e.*, yielding a finite probability space). Although the literature describes various probabilistic databases and data formalisms that utilise continuous probability distributions, the work described in this thesis follows the prevalent definition quoted in Definition 2.1; its scope is therefore restricted to finite discrete probability spaces.

2.2 Modelling uncertainty

Explicitly representing every single one of the (worst-case exponential number of) possible worlds is prohibitive. For this reason, various formalisms for expressing uncertainty have been proposed in the literature. Different methods for modelling uncertainty in probabilistic databases have their own (dis)advantages and better fit particular scenarios. In general, these formalisms can be categorised into two distinct groups [AY09]:

- tuple-level uncertainty: the uncertainty is expressed on the tuple as a whole rather than on the distinct tuple attributes. Sometimes referred to as *existential uncertainty*;
- attribute-level uncertainty: the uncertainty is expressed on the individual attribute values, *i.e.* the tuple is certain, but its attributes are uncertain. Sometimes referred to as *value uncertainty*.

Example 2.1: Tuple-level and attribute-level uncertainty in data extraction.

Automated extraction of data from on-line resources often encounters incomplete, conflicting, and uncertain information. For example, information regarding William Shakespeare's works is often not known with complete certainty (*e.g.*, Henry VIII is probably the result of a

t	work	author(s)	prob.
t_1	Henry VIII	Shakespeare & Fletcher	0.9
t_2	Romeo & J.	William Shakespeare	0.94
t_3	Romeo & J.	Christopher Marlowe	0.05
t_4	Romeo & J.	Francis Bacon	0.01

(a) Tuple-level uncertainty

t	work	author(s)
t_1	Henry VIII	{Shakespeare & Fletcher : 0.9;}
t_2	Romeo & J.	{Shakespeare : 0.94; Marlowe : 0.05; Bacon: 0.01}

(b) Attribute-level uncertainty

Table 2.1: Example of probabilistic tables with tuple-level and attribute-level uncertainty

collaboration between Shakespeare and John Fletcher), whereas other information is conflicting (*e.g.*, some sources state that all of Shakespeare's work was written by either Christopher Marlowe or Francis Bacon, others reject these theories).

Table 2.1a shows a table with tuple-level uncertainty containing information regarding the possible authors of *Henry VIII* and *Romeo & Juliet*: the former was most likely (with probability $p = 0.9$) a collaboration between Shakespeare and Fletcher, the latter was either written by Shakespeare ($p = 0.94$), Marlowe ($p = 0.05$), or Bacon ($p = 0.01$). Table 2.1b shows similar information stored in a probabilistic table with attribute-level uncertainty.

Although both tables contain the same data and express the same probabilities, there is one significant difference between the two: the tuples in Table 2.1a are independent of each other (*i.e.*, there exists a possible world which contains both t_2 and t_3), whereas the values of t_2 in Table 2.1b are mutually exclusive (*i.e.*, Shakespeare and Marlowe cannot both be responsible for *Romeo and Juliet*).

Representing uncertainty in a database is not the primary challenge faced by probabilistic database management systems – after all, this can be achieved by adding an extra column containing probabilities (for tuple-level uncertainty). The main challenge of such systems is to provide efficient methods to *query* such data [SORK11]. A discussion of query complexity of various probabilistic data models and queries is outside the scope of this thesis, but has been the subject of extensive research (*e.g.*, [DS04; OW12]).

This chapter discusses formalisms with both tuple-level and attribute-level uncertainty. Graphical models, a third type of formalism not often associated with relational databases, are also introduced briefly.

2.2.1 Tuple-level uncertainty

Various formalisms that provide tuple-level uncertainty exist, each with different levels of expressiveness in terms of the ability to encode correlations.

Probabilistic ?-tables (p-?-tables)

Probabilistic ?-tables[1] (or: p-?-tables, tuple-independent databases) are possibly the most trivial formalism for modelling tuple-level uncertainty [GT06; DS04; Agg09a; SORK11], and are sometimes referred to as the "independent tuples representation" [LLS+01]. Each probabilistic table $\mathcal{T}$ consists of probabilistically independent tuples $t_1, \ldots t_n$ which are annotated with a probability $\Pr[t_i]$. Table 2.1a (described in Example 2.1) is a typical example of a probabilistic ?-table.

Conceptually, every tuple is either present or absent. Therefore, a tuple-independent database $\mathcal{T}$ with n tuples $t_1, \ldots, t_n$ (and probabilities $\Pr[t_1], \ldots, \Pr[t_n]$) induces a probability distribution over powerset $2^{\mathcal{T}}$ yielding 2^n possible worlds. Each possible world $\mathcal{W}_i \in 2^{\mathcal{T}}$ consists of a subset of the original tuple set $\mathcal{T}$. The probability of a world is computed based on the probabilities of the tuples that appear in that world. Following the independence assumption, the probability of each of the possible worlds $\mathcal{W}_i$ is defined as:

$$\Pr[\mathcal{W}_i] = \prod_{t_j \in \mathcal{W}_i} \Pr[t_j] \quad \cdot \prod_{t_j \in (\mathcal{T} \setminus \mathcal{W}_i)} (1 - \Pr[t_j])$$

As the name suggests, tuple-independent databases lack support for storing correlations between tuples. For example, it would be desirable to be able to express that tuples t_2 and t_3 in Table 2.1a are mutually exclusive: both tuples are uncertain and both might be incorrect, but it is impossible that both records are correct in a single possible world. Other, more expressive formalisms do allow for encoding such correlations.

Prime examples of systems that are built upon tuple-independent databases are the Never Ending Language Learning System (NELL, [CBK+10]) and the Google Knowledge Vault [DMG+14].

Block-independent-disjoint databases

Block-independent disjoint (BID) databases are more expressive than the tuple-independent p-?-tables [SORK11]: a block-independent-disjoint table $\mathcal{T}$ consists of tuples $t_1, \ldots, t_n$ that are partitioned into disjoint sets (blocks) $B_1, \ldots, B_m$. A tuple t_i in block B_l is disjoint (mutually exclusive) with all

[1] The question mark in 'probabilistic ?-tables' is intentional and not a typesetting error.

other tuples $t_j \in B_l$. That is, there is no possible world in which two tuples $t_i, t_j \in B_l$ $(t_i \neq t_j)$ occur. Tuples $t_i \in B_a, t_j \in B_b$ from distinct blocks B_a, B_b are independent.

Observe how any p-?-table $\mathcal{T}_p$ can be expressed as a BID-table $\mathcal{T}_b$ by storing every tuple $t_i \in \mathcal{T}_p$ using a distinct block in $\mathcal{T}_b$. A well-known example of a system that uses this formalism to model uncertainty is Google Squared Tables [Cro10; FHOR11].

Probabilistic conditional tables (pc-tables)

Probabilistic conditional tables are a probabilistic extension of traditional *conditional tables* (c-tables), a formalism used to represent incomplete data [IJ84]. Tuples in c-tables are annotated with propositional formulae (conditions) over Boolean variables; pc-tables extend this representation by specifying a probability space over the assignments of the variables [SORK11]. This formalism allows for arbitrary correlations in the input data and is sufficiently expressive to subsume both p-?-tables and block-independent-disjoint tables. In fact, [GT06] provides a completeness result, showing that pc-tables can be used to represent *any* probabilistic database. Furthermore, it shows that pc-tables are closed under relational operations, *i.e.*, the result of any relational operation on a pc-table can be represented in the same formalism.

In a pc-table $\mathcal{T}$, every tuple t_i is associated with a propositional formula $\Phi[t_i]$ over Boolean random variables $x_i \in \mathbf{X}$ (often referred to as the *lineage* of t_i).

The total number of worlds induced by a pc-table depends on the number of Boolean random variables $v = |\mathbf{X}|$ and is bounded by 2^v. The probability of a world $\mathcal{W}_i$ depends on the lineage of the tuples that exist in that world, and is therefore defined as:

$$\Pr[\mathcal{W}_i] = \Pr\left[\underbrace{\left(\bigwedge_{t_j \in \mathcal{W}_i} \Phi[t_j]\right)}_{\substack{\text{lineage of} \\ \text{tuples in } \mathcal{W}_i}} \wedge \neg \underbrace{\left(\bigwedge_{t_j \in (\mathcal{T} \setminus \mathcal{W}_i)} \Phi[t_j]\right)}_{\substack{\text{lineage of} \\ \text{tuples not in } \mathcal{W}_i}}\right]$$

This definition uses the conjunction of (1) the lineage of all tuples present in $\mathcal{W}_i$, and (2) the negated lineage of the tuples not present in the world.

Example 2.2: Authorship of Shakespeare's works in a pc-table

Table 2.2 shows an example of a pc-table containing the previously introduced data on the authorship of Shakespeare's works. The uncertainty of (and correlations between) the various tuples is specified by the propositional formulae in the *lineage* column. For example:

t	work	author(s)	$\Phi[t_i]$	(prob.)
t_1	Henry VIII	Shakespeare & Fletcher	x_1	0.9
t_2	Romeo & J.	William Shakespeare	$x_2 \wedge \neg x_3$	0.94
t_3	Romeo & J.	Christopher Marlowe	$\neg x_2 \wedge \neg x_3$	0.05
t_4	Romeo & J.	Francis Bacon	x_3	0.01

x_i	prob.
x_1	0.9
x_2	0.95
x_3	0.01

Table 2.2: Example of tuple-level uncertainty in a pc-table

$$\begin{aligned}\Pr[t_3] = \Pr[\Phi[t_3]] &= \Pr[\neg x_2 \wedge \neg x_3] \\ &= (1 - \Pr[x_2]) \cdot (1 - \Pr[x_3]) \\ &= 0.05 \cdot 0.99 = 0.0495\end{aligned}$$

The tuples t_2, t_3, t_4 are mutually exclusive as a result of their lineage definitions: there is no possible world in which all three tuples occur. *I.e.*:

$$\Phi[t_2] \wedge \Phi[t_3] \wedge \Phi[t_4] = \bot$$

Consequently:

$$\begin{aligned}\Pr[\{t_2, t_3, t_4\}] &= \Pr[\neg\Phi[t_1] \wedge \Phi[t_2] \wedge \Phi[t_3] \wedge \Phi[t_4]] \\ &= 0\end{aligned}$$

The probability of the possible world with tuples $\{t_1, t_2\}$ is:

$$\begin{aligned}\Pr[\{t_1, t_2\}] &= \Pr[\Phi[t_1] \wedge \Phi[t_2] \wedge \neg\Phi[t_3] \wedge \neg\Phi[t_4]] \\ &= \Pr[x_1 \wedge (x_2 \wedge \neg x_3) \wedge \neg(\neg x_2 \wedge \neg x_3) \wedge \neg x_3] \\ &= \Pr[x_1 \wedge x_2 \wedge \neg x_3] = 0.9 \cdot 0.95 \cdot 0.99 \approx 0.85\end{aligned}$$

Probabilistic value-conditional tables (pvc-tables)

Although pc-tables have been shown to be complete and closed under relational operations [GT06], the size of the lineage expressions can grow exponentially (*e.g.*, after queries with aggregates [LSV02]). Probabilistic value-conditional tables (or: pvc-tables) can represent any finite probability distribution over relational databases, and results of queries with aggregates can be stored in polynomial

space [FHO12]. This more succinct representation is achieved by using *semimodule* and *semiring* expressions [GKT07; ADT11].

Using such semimodule expressions, it becomes possible to represent values (*e.g.*, results from an aggregation) *conditioned* on the value of a semiring expression. For example, the $\mathbb{B}[\mathbf{X}] \otimes \mathbb{R}$ semimodule can be used to condition a real number on a Boolean expression.

EXAMPLE 2.3: number of works by William Shakespeare

Using the data from Table 2.2, the number of works written by William Shakespeare can be expressed by:

$$\Phi = \underbrace{(x_1 \otimes 1)}_{\text{Henry VIII}} + \underbrace{((x_2 \wedge \neg x_3) \otimes 1)}_{\text{Romeo \& Juliet}}$$

Which can evaluate to 2 (in a possible world that satisfies both x_1 and $x_2 \wedge \neg x_3$), 1 (in a possible world that satisfies either x_1 or $x_2 \wedge \neg x_3$), or 0 (when neither is satisfied).

Although pvc-tables make for a richer formalism than pc-tables, the lack of support for negations (which are not captured by the Boolean semiring) and the distributive property of the semimodules pose limits to the fitness of this formalism for use in ENFrame. A more detailed description of these differences is provided in Section 4.2.1. As a consequence, ENFrame uses a formalism inspired by pvc-tables, but extends this formalism in a number of ways. Most importantly and significantly, ENFrame extends propositional expressions with so-called *conditinal values* (or: *c-values*) that allow for conditioning arbitrary values (*e.g.*, numbers in the domain of reals, vectors in the feature space) on a propositional formula. This construct enables succinct representations of random variables with an arbitrary (discrete) range. The formalism is introduced in Section 2.3, and discussed in more detail in Chapter 4.

2.2.2 Attribute-level uncertainty

This section discusses attribute-level uncertainty – a category of formalisms that expresses uncertainty in tuple attributes, rather than at the tuple level.

Probabilistic or-set-tables

Probabilistic or-set-tables (short: p-or-set-tables) can be seen as the attribute-level uncertainty equivalent of block-independent-disjoint tables: although the tuples $t_1, \ldots, t_n$ are fully probabilistically

independent, the tuple attributes $a_i^1, \ldots, a_i^m$ of tuple t_i each specify a categorical probability distribution over alternative values. Different attributes are considered to be probabilistically independent and by definition, $\sum_v \Pr[a_i^j = v] = 1 \quad (1 \leq i \leq n \text{ and } 1 \leq j \leq m)$. That is, the sum of the probabilities of the possible values v of attribute a_i^j is one. Furthermore, following the attribute-independence within a tuple, the probability of a tuple t_i having attribute values $v_1, \ldots, v_m$ can be computed by multiplying the probabilities of the distinct attribute values: $\Pr[t_i = \langle v_1, \ldots, v_m \rangle] = \prod_{j=1}^{m} \Pr[a_i^j = v_j]$.

Table 2.1b (described in Example 2.1) is a typical example of a probabilistic or-set-table (with only a single uncertain attribute: *author(s)*).

Continuous probability distributions in attributes

Some data originates from sources that are best modelled using a continuous probability distribution. A prime example is GPS location information: such data is often inaccurate and can be expressed as a continuous probability distribution over a three-dimensional space [OM96]. The accuracy of the information, and therefore the variance of the distribution, depends on factors like the position and the number of satellites in view, the quality of the client hardware (clock and radio receiver), and the quality of the error correction algorithms.

Example 2.4: Animal tracking using GPS

A probabilistic database management system storing time-series data from an experiment that tracks various animal species in their natural habitat might express the uncertainty of the tuples by storing the parameters of the probability distribution. Queries such as "how many platypuses spend most of their time on the shores of Lake Burragorang, rather than submerged in the water" would then return a probability distribution based on the distributions of the individual tuples in the probabilistic database.

Due to the continuous nature of the probability distributions that are associated with the tuples in this type of probabilistic database, the probability distribution over the possible worlds becomes continuous. Various techniques based on model fitting allow translation of discrete distributions into continuous ones [Cra46], and sampling techniques provide a way to achieve the opposite. The latter can be used in combination with correlations to enable processing of data with continuous probability distributions in ENFrame.

2.2.3 Graphical models

Parallel to research in the database community, developments in artificial intelligence have lead to different perspectives on representing uncertainty. The most well-known uncertainty model is that of a Bayesian network: a probabilistic graphical model which represents a set of random variables and their conditional dependencies as a directed acyclic graph [Pea88].

Bayesian networks are often constructed using knowledge of an expert in a field and are then used to perform inference. For example, large expert-driven Bayesian networks can be used to diagnose an outbreak of classical swine fever [vGBLE10] based on a series of observations. Additionally, various techniques exist to automatically learn such networks from data (*e.g.*, [Mar03]).

Weighted finite state transducers (FST) are a different popular graphical model, often used in applications such as speech and character recognition [MPR02]. A FST is a directed graph in which the vertices represent states (*e.g.* a partially recognised word), and the outgoing edges specify a probability distribution over the possible states that can follow.

The relation between Bayesian networks and pc-tables and integration approaches for both formalisms are the subject of discussion in Section 2.4, followed by a use-case demonstration in Chapter 7. The presented integration techniques bridge the gap between ENFrame and Bayesian networks. As a result, knowledge described in such networks can be used as input to ENFrame. Other graphical models are considered beyond the scope of this work, but are discussed elsewhere in various survey papers and book chapters (*e.g.*, [DGS09]).

2.2.4 Correlations

The formalisms introduced in this chapter support expressing tuple correlations to different degrees, and do so in various ways. At one end of the spectrum, there are tuple-independent databases such as probabilistic ?-tables, which provide no support for inter-tuple correlations whatsoever. At the other end, there are pc-tables and pvc-tables, which have been proved complete – *i.e.*, are able to represent any probabilistic database. Graphical models, such as Bayesian networks, often support correlations to some degree.

Correlations occur naturally in uncertain data of various types and from various sources [SD07]. Queries over uncorrelated (*i.e.* independent) probabilistic databases often produce correlated results [SORK11]. For example, a relational `join` of two or more tables yields tuples that depend on all constituent tuples from the originating (tuple-independent) tables. Hence, the resulting tuples

exhibit correlations, which can be described as *positive*. The same type of correlations are obtained when applying constraint conditioning to probabilistic databases [KO08].

Conflicting data is another prevailing source of correlations, as has been illustrated in the running example regarding the authorship of Shakespeare's works. These *negative* correlations also occur in location data [MCA06], optical character recognition, and data integration [DBS09; BN08].

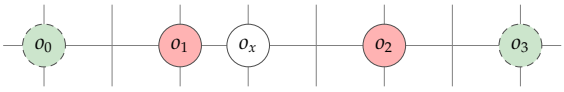
Sensor networks are known to produce a mixture of correlations: data is both spatially and temporally positively correlated, but can also be conflicting (*e.g.*, due to measurement and transmission errors as a result of hardware failure) and therefore negatively correlated [LC10].

Retaining and respecting correlations is of great importance, because they express a fundamental property of the data and its context. Most existing algorithms for data mining discard correlations (and therefore assume tuple independence) in an attempt to simplify computation. However, it has been repeatedly shown that tuple-independent data mining algorithms on correlated data yield unexpected results (*e.g.*, [VRH+09; AR14]). A more detailed overview of existing data mining algorithms for probabilistic data (and of other related work) can be found in Chapter 8.

The following example illustrates the potential consequences of ignoring correlations in a classification scenario.

EXAMPLE 2.5: Classification of correlated data

Reconsider the classification scenario first presented in Example 1.1:



Object o_x is unlabelled and will be classified by using the labels (colours) of its three nearest neighbours in all possible worlds. Assume that, by virtue of their lineage, the labelled objects are correlated as follows:

$$\Phi[o_0] = x_0 \vee x_1 \qquad \Phi[o_1] = x_0 \wedge x_1$$
$$\Phi[o_2] = \neg x_0 \wedge x_1 \qquad \Phi[o_3] = x_0 \vee x_1$$

The propositional formulae over two Boolean variables x_0, x_1 express a strong negative correlation between o_1 and o_2: the objects are mutually exclusive (*i.e.*, there is no single possible world in which both objects exist, because $\Phi[o_1] \wedge \Phi[o_2] = \bot$). Additionally, o_0 and o_3 are strongly positively correlated (with identical lineage) which results in $(\Phi[o_1] \vee \Phi[o_2]) \implies (\Phi[o_0] \wedge \Phi[o_3])$, where '$\implies$' denotes the logical binary operator that denotes the material implication.

In other words: there is no world in which o_1 and o_2 occur together, and in any world in which either exists, both o_0 and o_3 exist. Therefore, although the two closest objects to o_x are coloured red, there is no possible world in which the majority of the three nearest neighbours are red, and the unlabelled o_x will never be classified *red*.

Had correlations been ignored, then o_x would have been incorrectly classified *red* in at least half of the possible worlds.

2.3 Data in ENFrame

In probabilistic databases, discrete probability distributions over tuples or their attributes are more commonly used than continuous distributions [Agg09a; SORK11; GT06]. Of all models for describing discrete probability distributions over data, p(v)c-tables are the richest formalism. In fact, it can be shown that pc-tables can represent *any* probabilistic database [GT06]. Consequently, the pc-tables formalism subsumes the other formalisms discussed in this section: it can model tuple independence (as used in p-?-tables) and arbitrary correlations amongst tuples, including those correlations that can be expressed using block-independent-disjoint tables and p-or-set tables.

ENFrame extends propositional algebra with constructs inspired by semimodule expressions to allow for more succinct lineage expressions (Chapter 4). The output of an ENFrame program is a probability distribution over values of selected correlated program variables, optionally accompanied by a symbolic representation using either pure propositional formulae or lineage expressed in the extended propositional algebra. As a result, ENFrame integrates particularly well with pc-tables: both the input and the output of a program can be stored using this formalism.

2.4 Integrating Bayesian networks and pc-tables

N.B. this section is an edited extract of work published in the proceedings of the International Conference on Data Engineering (ICDE 2013, [OPvS13]). It is based on joint work with Dan Olteanu and Lampros Papageorgiou [Pap12]. Section 7.2 presents a use case based on this work.

Each of the formalisms described in this chapter has its own advantages and disadvantages, and is suitable for different types of data. However, when two data sources with different formalisms need to be combined, this poses an interesting challenge. The possible worlds semantics acts as

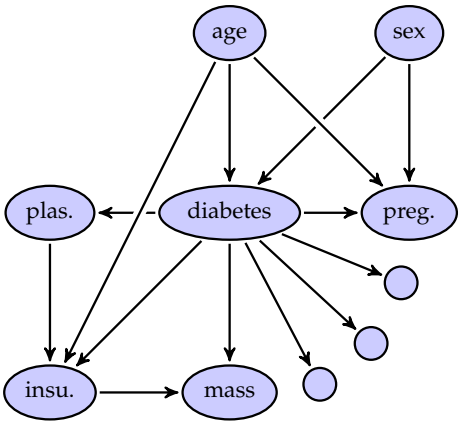


Figure 2.2: Simplified Bayesian network for diabetes from the UCI machine learning repository (the conditional probability tables at nodes are presented in Table 2.3).

sex	P
Male	0.48
Female	0.52

age	P
≤ 40	0.314
> 40	0.686

age	sex	diabetes	P
≤ 40	Female	true	0.349
≤ 40	Female	false	0.651
≤ 40	Male	true	0.255
≤ 40	Male	false	0.745
> 40	Female	true	0.355
> 40	Female	false	0.645
> 40	Male	true	0.249
> 40	Male	false	0.751

Table 2.3: Conditional probability tables for nodes *sex*, *age*, and *diabetes* in the Bayesian network shown in Figure 2.2

a bridge between the different formalisms and enables sound, equivalence-preserving translations between their instances. This section describes how Bayesian networks and pc-tables can be integrated to act as a data source for queries.

A Bayesian network is a structure $B(G, \Theta)$, where $G = (V, E)$ is a directed acyclic graph that represents a set of random variables and their conditional dependencies, and Θ is the set of parameters of all conditional probability distributions of nodes in G [Nea03]. Figure 2.2 shows an example of a simplified Bayesian network from the UCI machine learning repository [BL13]. The network models the various factors that influence the development of diabetes (*e.g.*, age) and the effects of the condition (*e.g.*, weight gain). Table 2.3 lists the conditional probability tables of the network.

2.4.1 Translating Bayesian networks

The translation from Bayesian networks to pc-tables consists of two steps:

1. Translating the nodes in the network to a series of categorical random variables;

2. Constructing propositional lineage formulae using the generated random variables.

The Bayesian network presented in Figure 2.3 will serve as a running example throughout this section. The conditional probabilities of nodes in the network are specified in Table 2.4.

Definition 2.2: Categorical random variables for nodes in Bayesian network

Let $\mathcal{B}$ be a Bayesian network consisting of a set of nodes $V = \{X_1, \ldots, X_n\}$, each with domain $\mathrm{Dom}(X_i)$ and a set of parent nodes $\mathcal{P}_{X_i}$. Furthermore, let α_{X_i} denote a set of all possible assignments of X_i to the values in its domain:

$$\alpha_{X_i} = \left\{(X_i = x_i^1), \ldots, (X_i = x_i^j)\right\} \qquad x_i^1, \ldots, x_i^j \in \mathrm{Dom}(X_i)$$

And let π_{X_i} denote the cross product (*i.e.*, all possible combinations) of value assignments of parent nodes of X_i:

$$\pi_{X_i} = \bigtimes_{P^l \in \mathcal{P}_{X_i}} \alpha_{P^l} \;=\; \alpha_{P^1} \times \ldots \times \alpha_{P^m}$$

With:

$$|\pi_{X_i}| = \prod_{P^l \in \mathcal{P}_{X_i}} |\alpha_{P^l}|$$

Then, the conditional probability distribution of $X_i \in \mathcal{B}$ can be represented using $|\pi_{X_i}|$ categorical random variables $V_{X_i}^j$:

$$\Pr\left[V_{X_i}^j = x\right] = \Pr\left[X_i = x \;\middle|\; \pi_{X_i}^j\right] \qquad x \in \mathrm{Dom}(X_i),\ 1 \le j \le |\pi_{X_i}|$$

The following example illustrates the result of the translation of a part of the example Bayesian network (Figure 2.3) into categorical random variables.

EXAMPLE 2.6: Categorical random variables for the example Bayesian network

Node C in the network depicted in Figure 2.3 is influenced by its parents A and B, as detailed in its conditional probability table (Figure 2.4c). The number of combinations of values within the domains of C's parents equals $|\pi_{X_i}| = 2 \cdot 2 = 4$, so C can be translated into 4 categorical random variables V_C^j, with $1 \leq j < 4$:

$$\Pr\left[V_C^1 = 0\right] = \Pr\left[C = 0 \,\middle|\, \pi_C^1\right] = 0.25 \qquad \Pr\left[V_C^1 = 1\right] = 0.75$$
$$\Pr\left[V_C^2 = 0\right] = \Pr\left[C = 0 \,\middle|\, \pi_C^2\right] = 0.33 \qquad \Pr\left[V_C^2 = 1\right] = 0.67$$
$$\Pr\left[V_C^3 = 0\right] = \Pr\left[C = 0 \,\middle|\, \pi_C^3\right] = 0.45 \qquad \Pr\left[V_C^3 = 1\right] = 0.55$$
$$\Pr\left[V_C^4 = 0\right] = \Pr\left[C = 0 \,\middle|\, \pi_C^4\right] = 0.50 \qquad \Pr\left[V_C^4 = 1\right] = 0.50$$

With:

$$\pi_C^1 = \{A = 0, B = 0\} \qquad \pi_C^3 = \{A = 1, B = 0\}$$
$$\pi_C^2 = \{A = 0, B = 1\} \qquad \pi_C^4 = \{A = 1, B = 1\}$$

The full list of Boolean random variables for nodes from Figure 2.3 is provided in Table 2.5.

During the second stage of the translation to pc-tables, the correlations in the Bayesian network are captured into lineage formulae:

DEFINITION 2.3: Constructing lineage formulae (continuing from Definition 2.2)

Let $X \in V$ be a node in Bayesian network $\mathcal{B} = (G, \Theta)$, with $G = (V, E)$. Recall that π_X^j is a value assignment of the parent variables of X (as defined in Definition 2.2), and let $\pi_X^j[P^i] = p_a$ denote the assignment of value $p_a \in \mathrm{Dom}(P^i)$ to parent P^i of X.

The lineage formulae $\Phi_{X=x}^j$ for node X are then recursively defined as:

$$\Phi_{X=x}^j = \left(V_X^j = x\right) \wedge \bigwedge_{P^i \in \mathcal{P}_X} \left(\bigvee_l \left(\Phi_{P^i = \pi_X^j[P^i]}^l \right) \right)$$

Note that the lineage formulae $\Phi_{X=x_a}^j$ and $\Phi_{X=x_b}^j$ are mutually exclusive. Example 2.7 illustrates the application of Definition 2.3 using the Bayesian network in Figure 2.3.

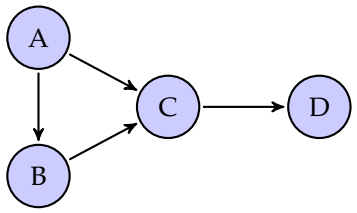


Figure 2.3: Bayesian network that serves as a running example to illustrate translation into pc-tables (see Table 2.4 for conditional probability tables)

A	Pr
0	0.55
1	0.45

(a) Node A

A	B	Pr
0	0	0.3
0	1	0.7
1	0	0.1
1	1	0.9

(b) Node B

A	B	C	Pr
0	0	0	0.25
0	0	1	0.75
0	1	0	0.33
0	1	1	0.67
1	0	0	0.45
1	0	1	0.55
1	1	0	0.50
1	1	1	0.50

(c) Node C

C	D	Pr
0	0	0.3
0	1	0.6
0	2	0.1
1	0	0.2
1	1	0.6
1	2	0.2

(d) Node D

Table 2.4: Conditional probability tables for Bayesian network presented in Figure 2.3

j	a	$\Pr\left[V_A^j = a\right]$	π_A^j
0	0	0.55	{}
0	1	0.45	{}

(a) Node A

j	c	$\Pr\left[V_C^j = c\right]$	π_C^j
0	0	0.25	$\{A = 0, B = 0\}$
0	1	0.75	$\{A = 0, B = 0\}$
1	0	0.33	$\{A = 0, B = 1\}$
1	1	0.67	$\{A = 0, B = 1\}$
2	0	0.45	$\{A = 1, B = 0\}$
2	1	0.55	$\{A = 1, B = 0\}$
3	0	0.50	$\{A = 1, B = 1\}$
3	1	0.50	$\{A = 1, B = 1\}$

(b) Node C

j	b	$\Pr\left[V_B^j = b\right]$	π_B^j
0	0	0.3	$\{A = 0\}$
0	1	0.7	$\{A = 0\}$
1	0	0.1	$\{A = 1\}$
1	1	0.9	$\{A = 1\}$

(c) Node B

j	d	$\Pr\left[V_D^j = d\right]$	π_D^j
0	0	0.3	$\{C = 0\}$
0	1	0.6	$\{C = 0\}$
0	2	0.1	$\{C = 0\}$
1	0	0.2	$\{C = 1\}$
1	1	0.6	$\{C = 1\}$
1	2	0.2	$\{C = 1\}$

(d) Node D

Table 2.5: Probability distribution of categorical random variables V_X^j generated from the Bayesian network presented in Figure 2.3

EXAMPLE 2.7: Lineage formulae for the example Bayesian network

Table 2.5 lists the categorical random variables and parent mappings for every node in the example Bayesian network (Figure 2.3). This example constructs the lineage for node D, conditioned on $C = 1$.

Assume that $\pi^1_D = \{C = 1\}$, then the lineage expression for $\Pr[D = 0 \mid C = 1]$ becomes:

$$\begin{aligned}
\Phi^1_{D=0} &= \left(V^1_D = 0\right) \wedge \bigwedge_{p^i \in \mathcal{P}_\mathcal{D}} \bigvee_l \left(\Phi^l_{p^i = \pi^1_{D=0}[p^i]}\right) && \text{(directly from Definition 2.3)} \\
&= \left(V^1_D = 0\right) \wedge \bigvee_l \left(\Phi^l_{C=\pi^1_{D=0}[C]}\right) && (C \text{ is only parent of } D) \\
&= \left(V^1_D = 0\right) \wedge \left(\Phi^0_{C=1} \vee \Phi^1_{C=1} \vee \Phi^2_{C=1} \vee \Phi^3_{C=1}\right)
\end{aligned}$$

Table 2.6 lists the lineage formulae for every node in the example network.

References to lineage of other nodes (such as to $\Phi^l_{C=1}$ in Example 2.7) can be stored either symbolically, or in expanded form. The acyclic structure of Bayesian networks ensures that no circular references can occur in the lineage expressions.

Note that, depending on the structure of the Bayesian network, parts of the lineage formulae (as exemplified in Table 2.6) can be simplified. For example, consider the full lineage $\Phi^0_{C=1}$ generated from node C in Example 2.7 (underlined expressions are part of a simplification in the following step):

$$\begin{aligned}
\Phi^0_{C=1} &= \left(V^0_C = 1\right) \wedge \Phi^0_{A=0} \wedge \left[\Phi^0_{B=0} \vee \Phi^1_{B=0}\right] \\
&= \left(V^0_C = 1\right) \wedge \left(V^0_A = 0\right) \wedge \left[\left(\left(V^0_B = 0\right) \wedge \Phi^0_{A=0}\right) \vee \left(\left(V^1_B = 0\right) \wedge \Phi^0_{A=1}\right)\right] \\
&= \left(V^0_C = 1\right) \wedge \underline{\left(V^0_A = 0\right)} \wedge \left[\left(\left(V^0_B = 0\right) \wedge \underline{\left(V^0_A = 0\right)}\right) \vee \left(\left(V^1_B = 0\right) \wedge \underline{\left(V^0_A = 1\right)}\right)\right] \\
&= \left(V^0_C = 1\right) \wedge \left(V^0_A = 0\right) \wedge \left(V^0_B = 0\right)
\end{aligned}$$

j	a	$\Phi^j_{A=a}$	$\Pr\left[\Phi^j_{A=a}\right]$
0	0	$V^0_A = 0$	0.55
0	1	$V^0_A = 1$	0.45

(a) Node A

j	a	b	$\Phi^j_{B=b}$	$\Pr\left[\Phi^j_{B=b}\right]$
0	0	0	$\left(V^0_B = 0\right) \wedge \Phi^0_{A=0}$	$0.30 \cdot 0.55 = 0.165$
0	0	1	$\left(V^0_B = 1\right) \wedge \Phi^0_{A=0}$	$0.70 \cdot 0.55 = 0.385$
1	1	0	$\left(V^1_B = 0\right) \wedge \Phi^0_{A=1}$	$0.10 \cdot 0.45 = 0.045$
1	1	1	$\left(V^1_B = 1\right) \wedge \Phi^0_{A=1}$	$0.90 \cdot 0.45 = 0.405$

(b) Node B

j	a	b	c	$\Phi^j_{C=c}$	$\Pr\left[\Phi^j_{C=c}\right]$
0	0	0	0	$\left(V^0_C = 0\right) \wedge \Phi^0_{A=0} \wedge (\Phi^0_{B=0} \vee \cancel{\Phi^1_{B=0}})$	$0.25 \cdot 0.55 \cdot 0.30 = 0.04125$
0	0	0	1	$\left(V^0_C = 1\right) \wedge \Phi^0_{A=0} \wedge (\Phi^0_{B=0} \vee \cancel{\Phi^1_{B=0}})$	$0.75 \cdot 0.55 \cdot 0.30 = 0.12375$
1	0	1	0	$\left(V^1_C = 0\right) \wedge \Phi^0_{A=0} \wedge (\Phi^0_{B=1} \vee \cancel{\Phi^1_{B=1}})$	$0.33 \cdot 0.55 \cdot 0.70 = 0.12705$
1	0	1	1	$\left(V^1_C = 1\right) \wedge \Phi^0_{A=0} \wedge (\Phi^0_{B=1} \vee \cancel{\Phi^1_{B=1}})$	$0.67 \cdot 0.55 \cdot 0.70 = 0.25795$
2	1	0	0	$\left(V^2_C = 0\right) \wedge \Phi^0_{A=1} \wedge (\cancel{\Phi^0_{B=0}} \vee \Phi^1_{B=0})$	$0.45 \cdot 0.45 \cdot 0.10 = 0.02025$
2	1	0	1	$\left(V^2_C = 1\right) \wedge \Phi^0_{A=1} \wedge (\cancel{\Phi^0_{B=0}} \vee \Phi^1_{B=0})$	$0.55 \cdot 0.45 \cdot 0.10 = 0.02475$
3	1	1	0	$\left(V^3_C = 0\right) \wedge \Phi^0_{A=1} \wedge (\cancel{\Phi^0_{B=1}} \vee \Phi^1_{B=1})$	$0.50 \cdot 0.45 \cdot 0.90 = 0.20250$
3	1	1	1	$\left(V^3_C = 1\right) \wedge \Phi^0_{A=1} \wedge (\cancel{\Phi^0_{B=1}} \vee \Phi^1_{B=1})$	$0.50 \cdot 0.45 \cdot 0.90 = 0.20250$

(c) Node C. The struck-through clauses in disjunctions conflict with other clauses in the parent conjunction, and can therefore be simplified. For example, $\Phi^1_{B=0}$ conflicts with the definition of $\Phi^0_{A=0}$.

j	c	d	$\Phi^j_{D=d}$	$\Pr\left[\Phi^j_{D=d}\right]$
0	0	0	$\left(V^0_D = 0\right) \wedge \left(\Phi^0_{C=0} \vee \Phi^1_{C=0} \vee \Phi^2_{C=0} \vee \Phi^3_{C=0}\right)$	$0.30 \cdot 0.623205 = 0.1869615$
0	0	1	$\left(V^0_D = 1\right) \wedge \left(\Phi^0_{C=0} \vee \Phi^1_{C=0} \vee \Phi^2_{C=0} \vee \Phi^3_{C=0}\right)$	$0.60 \cdot 0.623205 = 0.3739230$
0	0	2	$\left(V^0_D = 2\right) \wedge \left(\Phi^0_{C=0} \vee \Phi^1_{C=0} \vee \Phi^2_{C=0} \vee \Phi^3_{C=0}\right)$	$0.10 \cdot 0.623205 = 0.0623205$
1	1	0	$\left(V^1_D = 0\right) \wedge \left(\Phi^0_{C=1} \vee \Phi^1_{C=1} \vee \Phi^2_{C=1} \vee \Phi^3_{C=1}\right)$	$0.20 \cdot 0.376795 = 0.075359$
1	1	1	$\left(V^1_D = 1\right) \wedge \left(\Phi^0_{C=1} \vee \Phi^1_{C=1} \vee \Phi^2_{C=1} \vee \Phi^3_{C=1}\right)$	$0.60 \cdot 0.376795 = 0.226077$
1	1	2	$\left(V^1_D = 2\right) \wedge \left(\Phi^0_{C=1} \vee \Phi^1_{C=1} \vee \Phi^2_{C=1} \vee \Phi^3_{C=1}\right)$	$0.20 \cdot 0.376795 = 0.075359$

(d) Node D

Table 2.6: Full lineage formulae (and probabilities) for nodes from the Bayesian network presented in Figure 2.3

Chapter 3

Specifying Programs in ENFrame

One of the primary goals of ENFrame is to make programming for probabilistic data easier. The framework's *user language* is designed in such a way that the programmer does not need any knowledge of probabilistic databases or probability theory in order to be able to write programs for probabilistic data. Programming in the user language is effectively programming for deterministic data: probabilistic data formalisms and data uncertainty play no role in the user language. ENFrame is responsible for a probabilistic interpretation of a user program, which is the subject of discussion in Chapter 4.

This chapter contains an informal description of the user language and its constructs, and exemplifies the language using a series of data mining algorithms: k-medoids clustering, k-means clustering, Markov clustering, and k-nearest neighbour classification. User programs for these algorithms are presented in this chapter.

N.B. Throughout this thesis, listings of user programs (and extracts thereof) are typeset in a `typewriter font`. Their probabilistic interpretation (event programs) are typeset in *italics*.

3.1 Introduction to the user language

The design of ENFrame's user language is grounded in three main desiderata:

1. The language should be sufficiently expressive to allow for writing common data mining algorithms, issuing queries, and manipulating query results.

2. It must be possible for the programmer to be oblivious to the deterministic or probabilistic nature of the input data and to the probabilistic formalism.

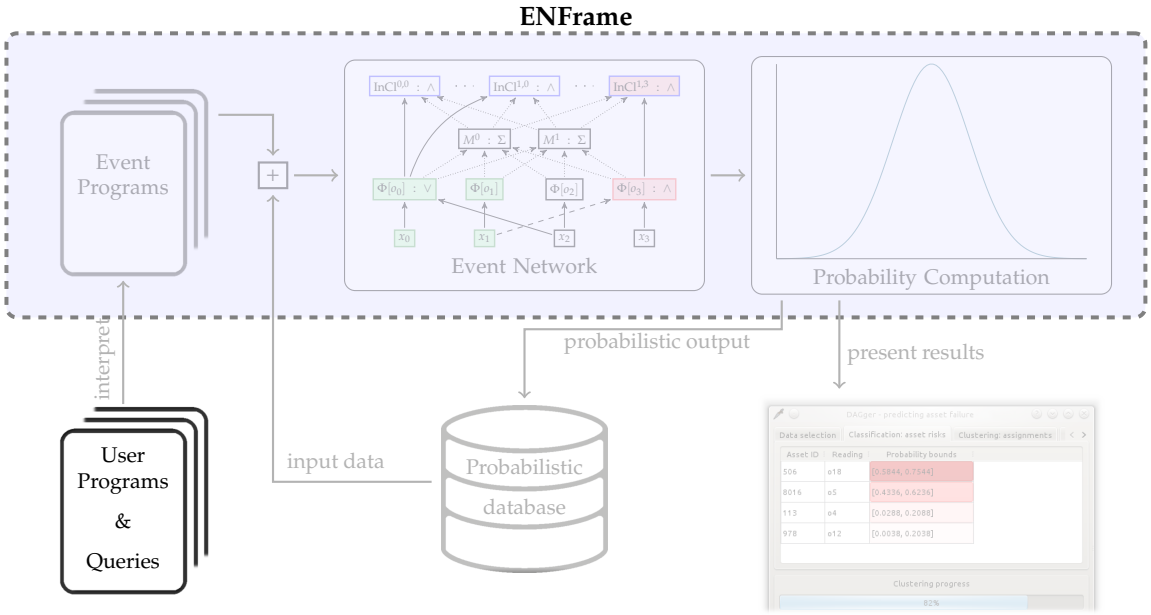


Figure 3.1: ENFrame architecture: chapter 3

3. The language should be sufficiently simple, in order to allow for an intuitive and straightforward probabilistic interpretation.

Following these requirements, the user language is based on a subset of the Python programming language [vRos97]. The language is sufficiently expressive to support a variety of data mining algorithms, including *k*-means clustering, *k*-medoids clustering, Markov clustering, and *k*-nearest neighbour classification.

3.2 Constructs of the user language

ENFrame's user language comprises the following constructs:

Variables (including arrays). A variable in the user language can be of a primitive type (real, integer, Boolean), or an array. Variables can be assigned and read multiple times. The following expressions are examples of valid variable assignments:

```
V = 2
V = W                (V becomes copy of W, assuming W is existing variable)
M[2] = True          (assigning third array element; indices start at 0)
```

Arrays are of a fixed size and are required to be initialised prior to being used. To initialise an array `M` of cardinality `k`:

```
M = [None] * k
```

Operators and functions. A variety of operators and functions can be used on variables:

`a + b`, `a * b`: computes the sum and product, respectively, of two numerical (*i.e.*, of type integer or real) variables `a` and `b`.

`a or b`, `a and b`: computes the logical disjunction and conjunction, respectively, of two Boolean variables `a` and `b`.

`pow(a,b)`: exponentiates a numerical variable `a` to a numerical variable `b`, *i.e.* a^b.

`invert(a)`: inverts a numerical variable `a`, *i.e.* $\frac{1}{a}$.

`scalar_mult(b,A)`: computes a component-wise multiplication of a numerical variable `b` with the values of a numerical array `A`. Yields a new array of the same size as `A`.

`dist(A,B)`: real-valued distance measure on the feature space between two numerical arrays (vectors) `A` and `B`. By default Euclidean, can be replaced by any other measure.

Array reductions. Given an array `M`, it can be *reduced* to a primitive value using one of the following reduction functions:

`reduce_or(M)`,`reduce_and(M)`: reduce an array `M` of Booleans to a Boolean variable by applying a logical disjunction (`reduce_or`) or conjunction (`reduce_and`) to the array values.

`reduce_sum(M)`,`reduce_mult(M)`,`reduce_count(M)`: reduce an array `M` of reals to a real by respectively summing, multiplying, or counting the values in `M`.

`select_first(b,A)`: returns the first `b` elements of `A`. The resulting array contains `b` items. If `A` initially contained fewer than `b` items, the resulting array will be padded with `None` values.

One special reduction function exists which reduces a two-dimensional numerical array to a one-dimensional array:

`reduce2_sum(M)`: reduces a two-dimensional numerical array (*i.e.*, an array of numerical arrays, also referred to as a matrix) `M` to a one-dimensional array of reals by performing a component-wise summation of the inner numerical arrays. *E.g.*, `A = [1,2,3]; B = [4,5,6];`

```
C = [7,8,9]; M = [A,B,C];
```

Then, `reduce2_sum(M)` yields `[12,15,18]`.

Loops. ENFrame's user language features bounded-range loops: for any integer `n` and a counter variable `i`, for-loops can be defined by: `for i in range (0,n)`.

List comprehensions. Python-style anonymous arrays can be defined inside a reduce function using list comprehension. For example: `reduce_sum([1 for i in range (0,n) if B[i]])` counts the number of *true* values in a Boolean array `B` of size `n`.

Input functions. The abstract primitive method `loadData()` is used to load input data for algorithms. This function can be implemented to load the objects from disk or to issue queries to a database. ENFrame supports positive relational algebra queries with aggregates via the SPROUT query engine for probabilistic data [FHO12]. The abstract methods `loadParams()` and `init()` are used to set algorithm parameters, such as the number of iterations and clusters for clustering algorithms.

All program variables have a probabilistic interpretation, which will be introduced in Chapter 4. This interpretation defines a probability distribution over variable values, which can be made available to the programmer. For instance, the programmer can define a Boolean variable that computes whether two objects belong to a distinct cluster, or co-occur in any cluster. The probability distributions of such program variables can be processed further, *e.g.* by a probabilistic database or subsequent data analysis.

3.3 User programs for data mining algorithms

This section demonstrates adaptations of four data mining algorithms: *k*-means (Algorithm 3.1), *k*-medoids (Algorithm 3.2), and Markov clustering (Algorithm 3.3), as well as *k*-nearest neighbour classification (Algorithm 3.4). Two of the algorithms (*k*-medoids clustering and *k*-nearest neighbour classification) have been used for an extensive experimental evaluation of ENFrame and its algorithms (see Chapter 6).

3.3.1 *k*-means clustering

The *k*-means clustering algorithm is possibly the most well-known unsupervised data mining algorithm for clustering a data set with n data points $o_1, \ldots, o_n$ into k disjoint partitions (clusters). The desired number of clusters k is a parameter of the clustering algorithm and therefore needs to be established beforehand; a brief discussion regarding techniques for determining k follows later in this section.

The algorithm consists of three stages, the last two of which are repeated until convergence:

1. Initialisation phase: an initial centroid M^i is determined for each cluster ($1 \leq i \leq k$). Throughout the algorithm iterations, the cluster centroid represents the centre of a cluster.

2. Assignment phase: every data point o_j ($1 \leq j \leq n$) is *assigned* to the cluster C^i of which centroid M^i is nearest.

3. Update phase: every cluster centroid M^i is recomputed to be the geometrical centre of the cluster it represents.

The assignment and update phases are repeated until convergence occurs, or a preset maximum number of iterations is reached. The location of the k initial centroids potentially has a significant influence on the clustering result; methods for centroid positioning are discussed later in this section.

User program for *k*-means clustering

The ENFrame user program for *k*-means clustering is provided in Algorithm 3.1, and will provide a *deterministic* clustering result for each of the n objects and k clusters. A probabilistic interpretation of user programs is the subject of discussion in Chapter 4.

The first two lines of the user program load the data (array `O` of `n` objects) and the algorithm parameters (number of clusters `k`, and number of algorithm iterations `it`). Line 3 computes the initial cluster centroids (to be discussed later in this section). The algorithm then runs `it` iterations (line 5) of the assignment (lines 6–13) and update (lines 15–20) phases.

During the assignment phase, the array `InCl` is initialised to hold `k` elements – one for each cluster. Each element is an array of `n` Booleans, each of which indicates whether an object `O[l]` is assigned to the cluster with index `i`. The Boolean value of `InCl[i][l]` is determined on lines 10–12: an object `O[l]` is assigned to cluster `i` if the distance from `O[l]` to cluster centroid `M[i]` (represented by `dist(O[l],M[i])`) is smaller than the distance to any other medoid `M[j]`. For the

```
1  (O, n) = loadData()              # list and number of objects
2  (k, it) = loadParams()           # number of clusters and iterations
3  M = init()                       # initialise centroids
4
5  for t in range(0,it):            # clustering iterations
6    InCl = [None] * k              # assignment phase: assign objects to clusters
7    for i in range(0,k):           # for every cluster i ...
8      InCl[i] = [None] * n
9      for l in range(0,n):         # ... and for every object l
10       InCl[i][l] = reduce_and( # l is assigned to i if i has closest centroid
11         [dist(O[l],M[i]) <= dist(O[l],M[j]) for j in range(0,k)]
12       )
13   InCl = breakTies2(InCl)        # each object is in exactly one cluster
14
15   M = [None] * k                 # update phase: compute new centroids
16   for i in range(0,k):           # for every cluster i
17     M[i] = scalar_mult(          # the new centroid is at the cluster centre
18       invert(reduce_count([1 for l in range(0,n) if InCl[i][l]])),
19       reduce2_sum([O[l] for l in range(0,n) if InCl[i][l]])
20     )
```

Algorithm 3.1: ENFrame user program for *k*-means clustering

sake of simplicity, the handling of ties (*i.e.* cases in which two cluster medoids are equidistant to an object) is not specified in the user program, and is the subject of discussion later in this section.

The *k*-medoids update phase re-initialises the vector `M` of cluster centroids (line 15), and subsequently recomputes (for every cluster `i`) the centre of the cluster (lines 17–20). The new cluster centre `M[i]` equals the summation of all vectors of objects in cluster `i` (computed using `reduce2_sum` on line 19), divided by the size of the cluster (using a component-wise multiplication with the inverse of the size, on line 18).

Determining the optimum value for *k*

Various heuristics for determining the optimum number of clusters have been proposed in the literature, *e.g.* the Caliński-Harabasz criterion [CH74], silhouette optimisation [Rou87; LOSS04], and approaches using the Bayesian information criterion or Akaike's information criterion [Aka74; PM00]. These heuristics apply to the *k*-medoids clustering algorithm as well (Section 3.3.2). A detailed discussion of heuristics for determining the parameter *k* can be found in the literature (*e.g.*

[PDN05]) and lies beyond the scope of this thesis; this work concerns itself with techniques for expressing arbitrary data mining algorithms in a framework for probabilistic programming, rather than comparing such algorithms and ways to determine their optimal parameter values.

Centroid selection during initialisation phase

The *k*-means and *k*-medoids clustering algorithms rely on an initial selection of *k* centroids (or medoids, in *k*-medoids). The initial centroid selection potentially has a significant effect on performance and the clustering result, and many approaches exist for establishing a suitable set of initial centroids [ARB+06]. Any of these techniques can be used as a pre-processing step to ENFrame by passing an initial `M` as an algorithm parameter, or by implementing the abstract `init()` function (called on line 3).

Tie breaking

When executing *k*-means clustering on deterministic data, an object can only ever be assigned to a single cluster. In the user program presented in Algorithm 3.1, line 11 does not exclude the possibility of an object being assigned to two clusters (*i.e.*, `InCl[i][l] = InCl[j][l] = true` for distinct clusters `i, j`). This situation is rectified by the call to `breakTies2`, which breaks ties in the first dimension of a two-dimensional Boolean array `M`, such that for every fixed `a`, only a single value `M[0][a],...,M[k-1][a]` has value *true*.

In fact, the need for the `breakTies2` method can be avoided altogether by replacing the definition of `InCl[i][l]` on line 10:

```
InCl[i][l] = reduce_and(
  [dist(O[l],M[i]) <= dist(O[l],M[j]) for j in range(0,k)]
)
```

By the following:

```
InCl[i][l] = reduce_and([
  (
     dist(O[l],M[i]) < dist(O[l],M[j]) or
     (i <= j and dist(O[l],M[i]) == dist(O[l],M[j]))
  ) for j in range(0,k)
])
```

This new definition of `InCl` prioritises clusters with a lower index by requiring a medoid `M[i]` to be *strictly* closer than every other medoid `M[j]`, unless `i` is smaller than `j` in which case equidistance suffices.

3.3.2 *k*-medoids clustering

The *k*-medoids clustering algorithm is very similar to the *k*-means algorithm: it consists of the same three clustering phases (initialisation, assignment, update), the first two of which are identical to *k*-means. Compared to *k*-means, the difference is the use of *medoids* rather than *centroids*. A centroid is the *mean* of the location of all data points in a cluster, and can be located at any point in the feature space. A medoid, on the other hand, is always a member of the input data set, and is defined as the data point whose dissimilarity to the other data points in the cluster is minimal [KR87]. In other words: a medoid is, from all members of a cluster, the object with the minimal total distance to all other objects in the cluster. An additional requirement is that two clusters cannot share the same medoid. This condition is only relevant during the initialisation phase: if *k* distinct medoids are chosen during this phase, there will be *k* distinct clusters (and medoids) throughout the clustering procedure.

The use of medoids (rather than centroids) allows for clustering a feature space in which a mean (centroid) cannot be defined for one or more dimensions. An often-used example is that of clustering of gene expressions [VPB03]: although it is not possible to compute the mean of a set of one or more gene expressions, dissimilarity matrices can be used as a distance function for *k*-medoids clustering.

User program for *k*-medoids clustering

The ENFrame user program for the *k*-medoids clustering algorithm is provided in Algorithm 3.2. The first thirteen lines are identical to the user program for *k*-means. On line 3, the initial cluster medoids are selected; the discussion in Section 3.3.1 regarding initial centroid selection in *k*-means applies to this algorithm as well.

The `it` iterations of the assignment and update phases start on line 5. The assignment phase starts on line 6 with the (re-)initialisation of the two-dimensional `InCl` array of Boolean values. For every cluster `i`, and every object `l`, the Boolean `InCl[i][l]` denotes whether or not the object `l` is assigned to cluster `i`. The condition for assignment is the same as the assignment condition in *k*-means: an object is assigned to a cluster `i` if the distance to that cluster's medoid is minimal.

```
(O, n) = loadData()              # list and number of objects
(k, it) = loadParams()           # number of clusters and iterations
M = init()                       # initialise medoids

for t in range(0,it):            # clustering iterations
  InCl = [None] * k              # assignment phase: assign objects to clusters
  for i in range(0,k):           # for every cluster i ...
    InCl[i] = [None] * n
    for l in range(0,n):         # ... and for every object l
      InCl[i][l] = reduce_and(   # l is assigned to i if i has closest medoid
        [(dist(O[l],M[i]) <= dist(O[l],M[j])) for j in range(0,k)]
      )
  InCl = breakTies2(InCl)        # each object is in exactly one cluster

  M = [None] * k                 # update phase: select new medoids
  for i in range(0,k):           # for every cluster i
    DistSum = [None] * n
    for l in range(0,n):         # compute total distance sum for every object l ...
      DistSum[l] = reduce_sum(   # ... to all other objects in cluster i
        [dist(O[l],O[p]) for p in range(0,n) if InCl[i][p]]
      )

    IsMedoid = [None] * n;
    for l in range(0,n):         # determine whether object is new medoid
      IsMedoid[l] = reduce_and( # object is new medoid if it has min. distance sum
        [DistSum[i][l] <= DistSum[i][p] for p in range(0,n) if InCl[i][p]]
      ) and InCl[i][l]           # and is in the cluster
    IsMedoid = breakTies(IsMedoid)

    M[i] = reduce_sum([O[l] for l in range(0,n) if IsMedoid[l]])
```

Algorithm 3.2: ENFrame user program for *k*-medoids clustering

Therefore, the code for the assignment phase in the user program (lines 6–13) is identical to the code in user program for *k*-means. Consequently, the application of tie-breaking in *k*-means clustering also applies to *k*-medoids (see Section 3.3.1).

The similarity with *k*-means ends with the update phase, during which the new cluster medoids are selected. An object `O[l]` becomes the new medoid to a cluster `i` if: (1) it was assigned to that cluster in the preceding assignment phase, and (2) the sum of distances between `O[l]` and all other objects in cluster `i` (the *total distance sum*) is minimal.

The update phase starts on line 15 with the re-initialisation of the array of medoids `M`, after which the algorithm iterates over all clusters. On lines 17–21, the `DistSum` array is initialised: it will hold, for every object `O[l]`, the sum of the distances to all other objects `O[p]` in cluster `i`. This value is computed by applying the `reduce_sum` function to an anonymous array of reals (distances to cluster members) on lines 19–21.

After the total distance sums for a cluster have been computed, the new cluster medoid is selected in the code on lines 23–28. The Boolean values in array `IsMedoid` indicate, for every object `O[l]`, whether that object is the new medoid.

Finally, on line 30, the location of the new medoid `M[i]` is set by summing over an anonymous array (generated through list comprehension) with only one member: the object which was selected as medoid.

Tie breaking

As seen in the user program for *k*-means, a possible tie in the cluster assignment is broken using a call to `breakTies2` (line 13). However, ties can occur in *k*-medoids's update phase as well: two objects can have identical total distance sums, and can therefore both be an equally suitable candidate to become a cluster medoid. This situation is resolved by a call to `breakTies` on line 28. Note that the `breakTies` method resolves a tie in a one-dimensional array, whereas `breakTies2` resolves a tie in the first dimension of a two-dimensional array.

3.3.3 Markov clustering

Markov clustering is a graph clustering algorithm based on simulation of stochastic flow [vDon00]. Clusters in a graph are characterised by the presence of many edges between vertices within a cluster and few edges between vertices in disjoint clusters.

To cluster graph vertices, the Markov clustering algorithm simulates random walks within a graph by repeatedly performing two operations: *expansion* and *inflation*. If a directed graph G is stored as an adjacency matrix A which is subsequently converted into a stochastic matrix M (*i.e.*, values are normalised such that columns sum up to 1), then the expansion and inflation operations can be described as operations on M:

- Expansion: this step corresponds to performing a step of a random walk, resulting in new probabilities for every pair of nodes (source and destination). The random step corresponds to squaring the stochastic matrix M. The result of squaring a stochastic matrix is, by definition, another stochastic matrix.
- Inflation: this step amplifies the effects of expansion. Small probabilities become smaller, large probabilities become larger. This step corresponds to taking the Hadamard power (*i.e.*, entry-wise) of M by an inflation parameter r, followed by normalisation to maintain the stochastic property.

Since higher length paths are more common within clusters than between different clusters, the probabilities associated with node pairs located in the same cluster will, in general, be relatively large – as there are many ways of going from one to the other.

User program for Markov clustering

The ENFrame user program for Markov clustering is presented in Algorithm 3.3. The program requires the following input data and parameters (lines 1–2):

- `n`: number of nodes in the graph
- `M`: $n \times n$ stochastic adjacency matrix
- `r`: the inflation parameter
- `it`: number of algorithm iterations

The columns (*i.e.*, outgoing edges) of a stochastic matrix sum up to 1. A value on row a in column b represents the probability that vertex a is visited from b when performing a random walk. A zero probability indicates that node a is not adjacent to b. A probability of one indicates that b only has a single outgoing edge: an edge to a.

The expansion phase takes place on lines 5–9. The (temporary) variable `N` holds the new stochastic matrix which is computed by a matrix self-multiplication.

```
1  (n, M) = loadData()          # M: stochastic n*n matrix of edge weights
2  (r, it) = loadParams()       # r: inflation parameter, it: number of iterations
3
4  for t in range(0,it):
5    N = [None] * n             # expansion phase
6    for i in range(0,n):
7      N[i] = [None] * n
8      for j in range(0,n):     # compute matrix square result:
9        N[i][j] = reduce_sum([M[i][k] * M[k][j] for k in range(0,n)])
10
11   M = [None] * n             # inflation phase
12   for i in range(0,n):
13     M[i] = [None] * n
14     for j in range(0,n):     # inflate and re-normalise values
15       M[i][j] = pow(N[i][j],r) *
16             invert(reduce_sum([pow(N[i][k],r) for k in range(0,n)]))
17
18 InCl = [None] * n           # interpret M to obtain clustering result
19 for i in range (0,n):
20   InCl[i] = [None] * n
21   for j in range (0,n):     # determine whether vertex j belongs to cluster i
22     InCl[i][j] = M[i][j] > 0
```

Algorithm 3.3: ENFrame user program for Markov clustering

During the inflation phase (lines 11–16), the new stochastic matrix `M` is computed by inflating every value of `N`. This value is immediately normalised on line 16 to maintain the stochastic property of `M`.

After `it` iterations, the algorithm is finished and yields the last version of the stochastic matrix `M`. If sufficient iterations were computed, this matrix will have converged to a *doubly idempotent* state: the matrix is idempotent under both self-multiplication (squaring) and inflation. In the final matrix, every column will generally have a single non-zero value, or multiple identical values [vDon00]. The interpretation of the resulting matrix is done on lines 18–22 using the Boolean matrix `InCl`: all non-zero entries (vertices) `j` in a row are assigned to the same cluster `i`. Note that this allows for up to `n` clusters, but in practice this number is much smaller.

3.3.4 *k*-nearest neighbour classification

ENFrame's user language is sufficiently expressive to go beyond the clustering algorithms discussed previously. This section presents the adaptation of a classification algorithm: *k*-nearest neighbour (or: *k*-nn). This algorithm is also included in the experimental evaluation in Chapter 6.

The *k*-nn classification algorithm attempts to classify one or more unlabelled data points using a data set consisting of ready-labelled data points. Unlike the clustering algorithms, *k*-nn is not an iterative algorithm: the class label of an unlabelled data point is determined by a plurality (or majority) vote by its *k* nearest neighbours. In case of a tie, an implementation can either choose to randomly assign a class, or take the distance to each of the *k* nearest neighbours into account. The classification algorithm that was used in Example 1.1 is, in fact, the *k*-nearest neighbour algorithm.

User program for *k*-nearest neighbour classification

The user program for *k*-nn is presented in Algorithm 3.4. The input data and required parameters are as follows (lines 5–6):

- `nu, U`: number of unlabelled data points to be classified, and their specification (*i.e.*, location).
- `nl, L`: number of labelled data points, and their specification.
- `nc, C`: number of classes, and mapping of labelled data points to classes (*e.g.* `C[a]` denotes the class of data point `L[a]`).
- `SLN[i]`: sorted list of labelled neighbours for every unlabelled object, *e.g.* `SLN[b]` is an array with `U[b]`'s neighbours in order of increasing distance.
- `k`: determines the number of neighbours considered in a plurality vote.

The two-dimensional array `Votes` is initialised on line 8 and will contain a vote for each combination of labelled object and possible class label. A labelled object `L[l]` will contribute a vote of value 2 for class `i` if the object has class label `i`. If `L[l]` does not have label `i`, the vote contributed `Votes[l][i]` will be 1.

After the object votes have been gathered, the class assignment of all `nu` unlabelled objects takes place on lines 14–30. The two-dimensional array `ClassAssign` will store, for every unlabelled object `U[u]`, an array of Booleans that represent the class assignment. To obtain those Booleans, an array `NearestVotes` is constructed first, based on the sorted list of neighbours `SLN`. The first

```
# U = unlabelled points, nu = number of unlabelled points in U
# L = labelled points, nl = number of labelled points in L
# C = class of each labelled point, nc = number of classes
# SLN[u] = sorted list of labelled neighbours for object u in U
(U, nu, L, nl, C, nc, SLN) = loadData()
(k) = loadParams()                    # k: no. of nearest neighbours to consider

Votes = [None] * nl;                  # prepare votes of labelled objects
for l in range (0, nl):               # iterate over labelled objects
  Votes[l] = [None] * nc              # votes from object l
  for i in range (0, nc):             # iterate over possible classes
    Votes[l][i] = 1 + (1 if C[l] == i else 0)  # 1/2-vote for class i

ClassAssign = [None] * nu;            # stores class assignments
for u in range (0, nu):               # iterate over unlabelled points
  VoteSum = [None] * nc               # sum of votes by nearest neighbours
  for i in range (0, nc):             # iterate over possible classes
    NearestVotes = [None] * nc;       # sort votes: nearest neighbours first
    for m in range (0, nl):
      NearestVotes[m] = Votes[SLN[u][m]][i]   # vote from mth nearest neighbour

    VoteSum[i] = reduce_sum(select_first(k, NearestVotes))  # sum first k votes

  ClassAssign[u] = [None] * k         # class assignments for point u
  for i in range (0, nc):             # iterate over classes
    ClassAssign[u][i] = reduce_and(   # assignment to class i
      VoteSum[i] >= VoteSum[p] for p in range (0, nc)
    )

  ClassAssign[u] = breakTies(ClassAssign[u])   # break ties
```

Algorithm 3.4: ENFrame user program for k-nearest neighbour classification

`k` elements from the ordered list of votes are extracted using the `select_first` method, and are subsequently summed into the `VoteSum[i]` variable.

Finally, the class assignment for an unlabelled object `U[u]` is stored in the two-dimensional array `ClassAssign`: a single Boolean in `ClassAssign[u]` can have value *true*. Possible ties are broken on using a call to `breakTies`.

3.4 Syntax of the user language

The clustering and classification algorithms provide an introduction to the constructs of the user language. This section discusses the user language in a more formal way by defining the language's grammar.

Figure 3.2 defines the grammar of the user language using the Backus-Naur Form[1]. An ENFrame user program consists of a sequence of declarations (the non-terminal ⟨*decl*⟩) and loop blocks (⟨*loop*⟩). Each declaration and loop can itself contain additional loops and declarations.

The language allows for assigning expressions (⟨*expr*⟩) to variable identifiers (⟨*varid*⟩). An expression can be:

- a Boolean, integer, or floating-point literal constant (⟨*lit*⟩);
- a variable identifier (⟨*varid*⟩);
- an array declaration (⟨*ardecl*⟩);
- the Boolean result of a comparison between two expressions (⟨*comp*⟩);
- the result of an operation such as summation, multiplication, inversion, and exponentiation;
- the result of a *reduce* operation on an anonymous array created through list comprehension (⟨*lcompr*⟩); or
- a tie breaker result (⟨*brties*⟩)

In addition to the syntactic structure as defined by the grammar in Figure 3.2, programs are governed by two additional constraints on anonymous arrays (through list comprehension) and loops. In its current state, ENFrame only supports list comprehensions that construct one-dimensional arrays of primitive types (*i.e.*, Booleans, integers, floats). Furthermore, loops can only be constructed using

[1]The Backus-Naur Form (BNF) was initially named the 'Backus Normal Form' after J.W. Backus et al. in [BBG+63]. It was later renamed to 'Backus-Naur Form' after a suggestion by Donald Knuth [Knu64]. Refer to the Encyclopedia of Computer Science [MR03] for a more detailed account of the notation's history.

integer literals or immutable integer-valued variables. This restriction ensures that loops and list comprehensions are of bounded size which is known when the probabilistic interpretation of the user program takes place.

⟨*loop*⟩ ::= {⟨*decl*⟩} { for ⟨*varid*⟩ in ⟨*range*⟩: { ⟨*loop*⟩ } }

⟨*decl*⟩ ::= ⟨*varid*⟩ = ⟨*expr*⟩ | '(' {⟨*varid*⟩,} ⟨*varid*⟩ ')' = ⟨*ext*⟩

⟨*expr*⟩ ::= ⟨*lit*⟩ | ⟨*varid*⟩ | ⟨*ardecl*⟩ | ⟨*expr*⟩ ⟨*comp*⟩ ⟨*expr*⟩
| ⟨*expr*⟩ and ⟨*expr*⟩ | ⟨*expr*⟩ or ⟨*expr*⟩
| ⟨*reduce*⟩ '(' ⟨*lcompr*⟩ ')' | pow '('⟨*expr*⟩,⟨*expr*⟩')'
| invert '('⟨*expr*⟩')' | ⟨*expr*⟩ '∗' ⟨*expr*⟩ | ⟨*expr*⟩ '+' ⟨*expr*⟩
| scalar_mult '(' ⟨*expr*⟩, ⟨*expr*⟩ ')' | ⟨*brties*⟩ '(' ⟨*expr*⟩ ')'
| select_first '(' ⟨*expr*⟩, ⟨*expr*⟩ ')'

⟨*lcompr*⟩ ::= ⟨*expr*⟩ for ⟨*varid*⟩ in ⟨*range*⟩ if ⟨*expr*⟩ [else ⟨*expr*⟩]

⟨*brties*⟩ ::= breakTies '(' ⟨*expr*⟩ ')' | breakTies2 '(' ⟨*expr*⟩ ')'

⟨*reduce*⟩ ::= reduce_and | reduce_or | reduce_sum | reduce_mult
| reduce_count

⟨*range*⟩ ::= range '(' ⟨*expr*⟩, ⟨*expr*⟩ ')'

⟨*ardecl*⟩ := [None] '∗' ⟨*expr*⟩

⟨*comp*⟩ ::= '<' | '>' | '==' | '<=' | '>='

⟨*ext*⟩ ::= loadData '(' ')' | loadParam '(' ')' | init '(' ')'

⟨*varid*⟩ ::= a variable identifier

⟨*lit*⟩ ::= a literal (Boolean, integer, float)

Figure 3.2: The grammar of the user language in Backus-Naur Form

Chapter 4

Probabilistic Programs Using Events

The previous chapter introduced the user language, in which a programmer can write programs in ENFrame. This chapter introduces the probabilistic interpretation of the deterministic user programs: *event programs*. First of all, the concept of probabilistic *events* is introduced, followed by an introduction of the constructs of the event language (operators and functions), their syntax and language semantics. After that, the *probabilistic semantics* of events is introduced, which defines how an event in the event language can be interpreted as a random variable with a probability distribution over a finite range. Once all these building blocks are in place, the translations of the user programs from the previous chapter to their probabilistic interpretation are presented and explained, followed by some final notes on the translation process.

4.1 Probabilistic interpretation through events

In query languages, a Boolean query Q on a deterministic database D is a map $Q : D \rightarrow \{true, false\}$. On a probabilistic database, such a query yields a Boolean random variable with a probability distribution over the set of Boolean constants: *true* and *false*. Similarly to queries on deterministic and probabilistic data, ENFrame programs have a sound semantics for both deterministic and probabilistic input data. For deterministic data, the result of an ENFrame program is typically deterministic (*e.g.*, a classification algorithm will yield a deterministic classification); for probabilistic data, the result is a set of probability distributions over the ranges of program variables and, by extension, a probability distribution over all possible results (*e.g.*, a distribution over all possible clustering or classification results).

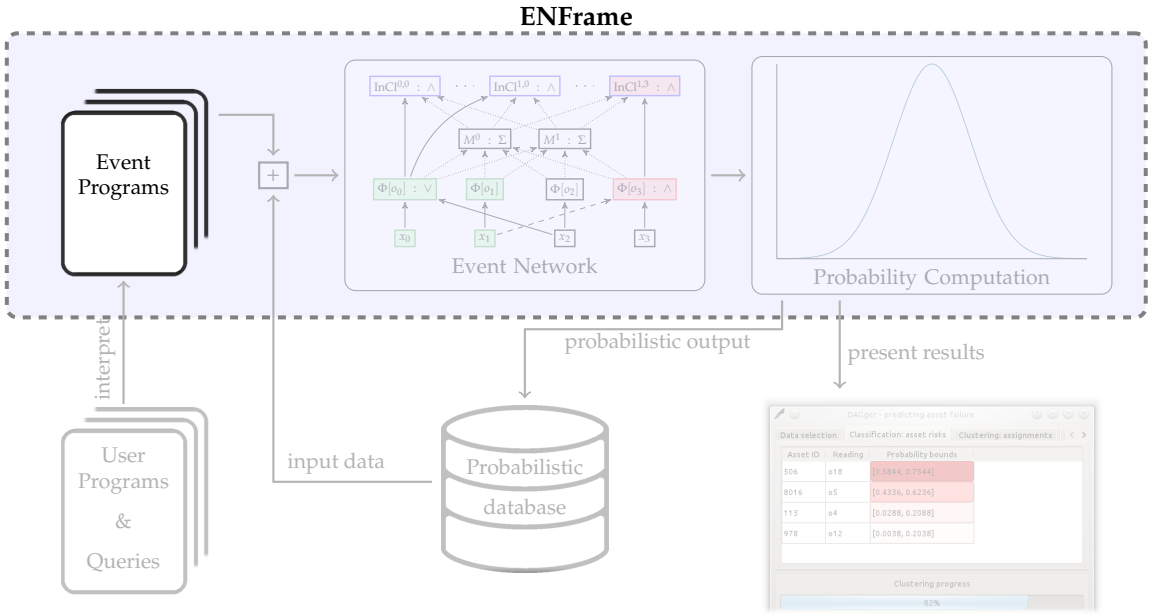


Figure 4.1: ENFrame architecture: chapter 4

At the foundation of the interpretation lie probabilistic *events*, which are the probabilistic counterpart to regular deterministic program variables. Events are a concise syntactic encoding of random variables and their probability distribution, and are generated by *event programs*. This chapter describes the syntax and semantics of events and event programs, and finally explains how ENFrame programs written in the user language can be translated to event programs. The key features of events and event programs are:

- Events can encode arbitrarily correlated discrete probability distributions over input objects. In particular, they can succinctly encode instances of formalisms such as p(v)c-tables and Bayesian networks. The input objects and their correlations can be explicitly provided, or imported via a positive relational algebra query with aggregates over p(v)c-tables [FHO12].

- Events can have a non-Boolean range, which allows for an exponentially more succinct encoding than an equivalent purely Boolean description [LSV02].

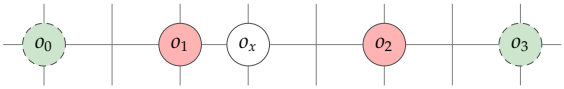
- Events have a well-defined probabilistic semantics that facilitates an interpretation as a random variable.

- Re-use of existing events leads to a concise encoding of a large number of distinct interconnected events.

In pc-tables, the tuples are annotated with propositional formulae over Boolean random variables (as described in Section 2.2.1). This formalism is fully compatible with events for objects in input data for ENFrame: every object o_l has an event $\Phi[o_l]$ over a set $\mathbf{X}$ of independent Boolean random variables. The *possible worlds* of the input objects are defined by the possible valuations $\alpha : \mathbf{X} \to \{true, false\}$. Each valuation α constitutes a world containing those objects o_l for which $\Phi[o_l]$ is satisfied by α. Due to the independence of the random variables $x \in \mathbf{X}$, the probability of a world is the product of the probabilities of the variables $x \in \mathbf{X}$ taking a truth value $\alpha(x)$.

EXAMPLE 4.1: Classification in possible worlds

Recall the classification problem introduced in Example 1.1:



Assume that each data point o_l is annotated with an event $\Phi[o_l]$ over a set of independent Boolean random variables $\mathbf{X} = \{x_0, x_1\}$ with probabilities $\Pr[x_0] = 0.6$ and $\Pr[x_1] = 0.8$:

$$\Phi[o_0] = x_0 \qquad \Phi[o_1] = x_1$$

$$\Phi[o_2] = x_0 \wedge x_1 \qquad \Phi[o_3] = \neg x_0 \vee x_1$$

The two Boolean random variables yield four possible worlds. The classification result for o_l in each of these possible worlds $\mathcal{W}$ (using the k-nn algorithm with $k = 3$):

x_0	x_1	objects	$\Pr[\mathcal{W}]$	result
$\top$	$\top$	$\{o_0, o_1, o_2, o_3\}$	$0.6 \cdot 0.8 = 0.48$	red
$\top$	$\bot$	$\{o_0\}$	$0.6 \cdot (1 - 0.8) = 0.12$	green
$\bot$	$\top$	$\{o_1, o_3\}$	$(1 - 0.6) \cdot 0.8 = 0.32$	red
$\bot$	$\bot$	$\{o_3\}$	$(1 - 0.6) \cdot (1 - 0.8) = 0.08$	green

As a result, the classification probabilities are:

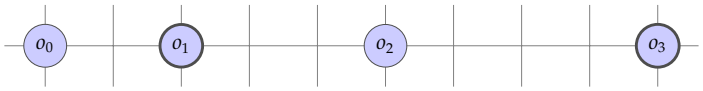
$$\Pr[o_x \text{ is classified } red] = 0.48 + 0.32 = 0.8$$

$$\Pr[o_x \text{ is classified } green] = 0.12 + 0.08 = 0.2$$

The use of events is not limited to representing uncertainty in the input data set – they are used to symbolically encode the entire computation process. This symbolic encoding using an *event language* is exemplified in Section 4.4 using the classification and clustering algorithms that were introduced in the previous chapter. The following example informally illustrates the use of events for the initialisation phase of a clustering algorithm.

EXAMPLE 4.2: Probabilistic events for initial medoid selection in the clustering process

Recall the following probabilistic clustering scenario from Example 1.2:



In a probabilistic clustering scenario, every object o_l in the data set is associated with a probabilistic event $\Phi[o_l]$ that expresses the object's existence. As a result, when applying the k-medoids clustering algorithm to find $k = 2$ clusters, the selection of the initial cluster medoids M^0 and M^1 (initialisation phase) becomes a probabilistic event. The range of such an event is the set of objects in the data set. However, the two conditions for cluster medoids still apply: a medoid is an object that is part of the data set, and two clusters cannot share the same medoid.

This is an example of the (informal) event specification for the initial medoids:

$$M^0 = \begin{cases} o_0 & \text{if } o_0 \text{ exists} \\ o_1 & \text{if } o_1 \text{ exists, and } o_0 \text{ is not the medoid of } M^0 \text{ (\textit{i.e.}, } o_0 \text{ does not exist)} \\ o_2 & \text{if } o_2 \text{ exists, and neither } o_0 \text{ nor } o_1 \text{ is the medoid of } M^0 \\ o_3 & \text{if } o_3 \text{ exists, and none of } o_0, o_1, o_2 \text{ are the medoid of } M^0 \\ \mathit{none} & \text{otherwise (\textit{i.e.}, if no objects exist)} \end{cases}$$

$$M^1 = \begin{cases} o_0 & \text{if } o_0 \text{ exists and is not the medoid of } M^0 \text{ (= contradiction)} \\ o_1 & \text{if } o_1 \text{ exists, and } o_1 \text{ is not the medoid of } M^0, \text{ and } o_0 \text{ is not the medoid of } M^1 \text{ (\textit{i.e.}, } o_0 \text{ does not exist)} \\ o_2 & \text{if } o_2 \text{ exists, and } o_2 \text{ is not the medoid of } M^0, \text{ and neither } o_0 \text{ nor } o_1 \text{ is the medoid of } M^1 \\ o_3 & \text{if } o_3 \text{ exists, and } o_3 \text{ is not the medoid of } M^0, \text{ and none of } o_0, o_1, o_2 \text{ is the medoid of } M^1 \\ \mathit{none} & \text{otherwise} \end{cases}$$

Note how these newly constructed events M^0 and M^1 reuse the events for object existence and selection of other medoids. The probabilities of each of the possible values for M^0 and M^1 are calculated using the probabilities of each of the events they depend on, as will be explained next and illustrated in Example 4.3.

4.2 Event constructs and their semantics

User programs are probabilistically interpreted as event programs. For this to be possible, every construct in the user language has a probabilistic counterpart in the event language. These probabilistic constructs and their semantics are introduced and discussed here, after which their syntax will be introduced. In Section 4.4, their use will be illustrated by showing the event programs for the data mining algorithms introduced in the previous chapter.

The semantics of event expressions is defined by extending a Boolean valuation $\alpha : \mathbf{X} \to \{true, false\}$ to a valuation expressions over any type of event. The exact behaviour of each of the event expressions under α is specified in Figure 4.3, and further detailed below.

4.2.1 Introduction to event expressions

Boolean random variables and propositional operators

Elements x_i from the set of independent Boolean random variables $\mathbf{X}$ are the most basic type of Boolean event (indicated by $\langle bevent \rangle$ in the language's grammar in Figure 4.2). Various propositional operators can be used to give rise to other Boolean events: conjunctions (operator: $\wedge$), disjunctions (operator: $\vee$), and negations (operator: $\neg$). Disjunction and conjunction events can be constructed using both binary operators (*e.g.*, $\Phi = x_1 \wedge x_2$) and n-ary operators (*e.g.*, $\Phi = \bigvee_i x_i$). The use of such operators results in a new event that depends on the operands. The following example illustrates how the events that were described informally in Example 4.2 can be constructed using these propositional operators.

Example 4.3: Probabilistic events for initial medoid selection (contd.)

Recall the probabilistic clustering scenario from Example 4.2. Assume that the objects $o_0, \ldots, o_3$ in the input data have the following propositional events over a set of independent

Boolean random variables $\mathbf{X} = \{x_0, x_1, x_2, x_3\}$:

$$\Phi[o_0] = x_0 \vee x_2 \qquad \Phi[o_1] = x_1$$
$$\Phi[o_2] = x_3 \qquad \Phi[o_3] = \neg x_1 \wedge x_3$$

The events for the selection of initial medoids M^0 and M^1 can then be defined more formally, as follows (see Appendix D for step-by-step rewritings):

$$\Phi\Big[M^0 = o_0\Big] \equiv \Phi[o_0] \;\equiv\; x_0 \vee x_2$$
$$\Phi\Big[M^0 = o_1\Big] \equiv \Phi[o_1] \wedge \neg\Phi\Big[M^0 = o_0\Big] \;\equiv\; x_1 \wedge \neg(x_0 \vee x_2)$$
$$\Phi\Big[M^0 = o_2\Big] \equiv \Phi[o_2] \wedge \neg\Phi\Big[M^0 = o_0\Big] \wedge \neg\Phi\Big[M^0 = o_1\Big] \;\equiv\; x_3 \wedge \neg(x_0 \vee x_2) \wedge \neg x_1$$
$$\Phi\Big[M^0 = o_3\Big] \equiv \Phi[o_3] \wedge \neg\Phi\Big[M^0 = o_0\Big] \wedge \neg\Phi\Big[M^0 = o_1\Big] \wedge \neg\Phi\Big[M^0 = o_2\Big] \;\equiv\; \bot$$

$$\Phi\Big[M^1 = o_0\Big] \equiv \Phi[o_0] \wedge \neg\Phi\Big[M^0 = o_0\Big] \;\equiv\; \bot$$
$$\Phi\Big[M^1 = o_1\Big] \equiv \Phi[o_1] \wedge \neg\Phi\Big[M^1 = o_0\Big] \wedge \neg\Phi\Big[M^0 = o_1\Big] \;\equiv x_1 \wedge (x_0 \vee x_2)$$
$$\Phi\Big[M^1 = o_2\Big] \equiv \Phi[o_2] \wedge \neg\Phi\Big[M^1 = o_0\Big] \wedge \neg\Phi\Big[M^1 = o_1\Big] \wedge \neg\Phi\Big[M^0 = o_2\Big] \;\equiv$$
$$\equiv x_3 \wedge \neg(x_1 \wedge (x_0 \vee x_2)) \wedge (x_0 \vee x_1 \vee x_2)$$
$$\Phi\Big[M^1 = o_3\Big] \equiv \Phi[o_3] \wedge \neg\Phi\Big[M^1 = o_0\Big] \wedge \neg\Phi\Big[M^1 = o_1\Big] \wedge \neg\Phi\Big[M^1 = o_2\Big] \wedge \neg\Phi\Big[M^0 = o_3\Big] \;\equiv$$
$$\equiv \neg x_0 \wedge \neg x_1 \wedge \neg x_2 \wedge x_3$$

Events for the assignment and update phases of the k-medoids algorithm can be constructed in a similar fashion. However, as will become clear later in this chapter, it is not possible to construct these events using purely propositional event expressions.

The probabilities of the events for initial medoid selection can be computed using the probability distributions of the independent Boolean random variables $x_i \in \mathbf{X}$. This is subject of detailed discussion in Chapter 5.

Conditional values (c-values) and real-valued events

Conditional values (or: c-values) are one of the most important constructs in ENFrame's event language. A c-value is an event that conditions some value v on a Boolean event Φ, and is denoted as follows: $\Phi \otimes v$. The value v can be a real-valued number ($v \in \mathbb{R}$; *e.g.*, a distance between two

objects), or a vector. Intuitively, the c-value evaluates to v if Φ evaluates to *true* under a valuation α; otherwise, it evaluates to a value that represents *undefined* and is denoted u. The value of u is the *identity element* (or: *neutral element*) of the monoid of v. For Boolean values $u = \mathit{false}$, and for values $v \in \mathbb{R}$ this results in $u = 0$ (*i.e.*, the identity element of the monoid of real numbers under addition).

Two or more c-values values can be summed and multiplied using the standard binary and n-ary operators to construct new real-valued events. For example, $(\Phi_1 \otimes v) + (\Phi_2 \otimes w)$ evaluates to 0, v, w, or $v + w$ in the event both Φ_1 and Φ_2 are *false*, only Φ_1 is *true*, only Φ_2 is *true*, or both Φ_1 and Φ_2 are *true*, respectively.

Conditional values make an appearance in many probabilistic interpretations of user programs. For example, consider the following extract from the update phase of the k-medoids user program (which was previously presented in Algorithm 3.2):

```
19        DistSum[l] = reduce_sum(  # ... to all other objects in cluster i
20          [dist(O[l],O[p]) for p in range(0,n) if InCl[i][p]]
21        )
```

This code computes the total distance sum `DistSum[l]` from an object `O[l]` to all other objects in the same cluster `i`. This cluster membership condition is explicitly expressed using the `InCl` array, which stores the object-to-cluster assignment. Capturing the total distance sum in a purely Boolean (propositional) event expression would incur an exponential blow-up in the number of expressions and their size [LSV02], as it would require a Boolean random variable for every possible summation result. Therefore, ENFrame uses a summation over conditional values to construct a new real-valued event $\mathrm{DistSum}^l$:

$$DistSum^l \equiv \sum_{p=0}^{n-1} \Big[InCl^{i,p} \otimes dist\big(O^l, O^p\big) \Big]$$

A c-value construct can also serve as a value to another c-value, *e.g.*: $\Phi_1 \otimes (\Phi_2 \otimes v)$, which expands to $(\Phi_1 \wedge \Phi_2) \otimes v$. Conditional values over vectors in the feature space can be used as parameter to the $\mathrm{dist}(a, b)$ function. This function evaluates to $u = 0$ if one of its parameters is u; if both parameters are defined (*i.e.*, $a, b \neq u$), the function returns the distance between the two vectors.

In this thesis, c-values are assumed to be conditioned on Boolean variables; a generalisation to variables with arbitrary finite categorical distributions is described in [vSO16] (accepted with minor revisions in December 2015).

Comparisons of real-valued events

By conditioning on a Boolean event, conditional values allow for probabilistic events with an arbitrary range, such as the *DistSum* events explained earlier. Similarly, *comparison* events allow for the combining of two real-valued events to construct a Boolean event. ENFrame supports the common relational operators for use in comparisons: $<$, $\leq$, $=$, $\geq$, $>$.

Consider the following extract from the user program for k-nearest neighbour classification (see Algorithm 3.4 for the complete program listing):

```
26      ClassAssign[u][i] = reduce_and(  # assignment to class i
27        VoteSum[i] >= VoteSum[p] for p in range (0, nc)
28      )
```

This code assigns class label `i` to an unlabelled object `u` (by setting `ClassAssign[u][i]` to *true*) if the sum of votes `VoteSum[i]` for class `i` is larger than the sum of votes `VoteSum[p]` for any other class `p`. In the probabilistic interpretation, both `VoteSum[i]` and `VoteSum[p]` are events over natural numbers that were constructed using c-values. Each of the values in the range of such an event has a certain probability of occurring, depending on the c-values it was constructed with. Section 4.3 specifies the probabilistic semantics of this and other types of events, which dictates how the probability of each of the values is computed.

The relational operator '$\geq$' can be used to compare two events over natural (or real) numbers to construct a new Boolean event $ClassAssign^{u,i}$, which represents the assignment class i to an unlabelled object o_u:

$$ClassAssign^{u,i} \equiv \bigwedge_{p=0}^{n_c-1} \left[VoteSum^i \geq VoteSum^p \right]$$

Comparisons default to *false* if both operands are u. If exactly one of the operands evaluates to u, then the comparison always results in *true*. If both operands are defined (*i.e.*, $\neq u$), the result of the comparison has the usual semantics on values in $\mathbb{R}$.

Array operations

Arrays (vectors) can be added and multiplied (both element-wise) using the '$+$' and '$\cdot$' operators. Additionally, function $\mathcal{F}_k(A)$ on an array A returns the first k elements that do not have value u. Example 4.4 in the next section illustrates probabilistic semantics using the $\mathcal{F}_k$ function.

Conditional values versus related algebraic structures

The c-value construct in the event language is inspired by work on provenance semirings [GKT07] and semimodules [ADT11; FHO12], which capture provenance for positive relational queries with aggregates on uncertain databases. The construct $\Phi \otimes v$ resembles the algebraic structure of a semimodule that is a tensor product of the Boolean semiring $\mathbb{B}[\mathbf{X}]$ freely generated by the variable set $\mathbf{X}$ with the sum monoid over the real numbers $\mathbb{R}$.

This semimodule forms the foundation for pvc-tables [FHO12], as introduced in Chapter 2. This formalism for probabilistic data allows for succinctly storing values and their lineage in a single expression. For example, the number of works written by William Shakespeare can be captured by the following semimodule expression:

$$\Psi = \underbrace{(x_1 \otimes 1)}_{\text{Henry VIII}} + \underbrace{((x_2 \wedge \neg x_3) \otimes 1)}_{\text{Romeo \& Juliet}}$$

There are, however, two key differences between c-values and these semimodules. Firstly, c-values allow for negations in events, which are not captured by the Boolean semiring. Secondly, even for positive events, the c-value $\mathbb{B}[\mathbf{X}] \otimes \mathbb{R}$ is not a semimodule since it violates the law of distributivity that governs semimodules:

$$(\Phi_1 \vee \Phi_2) \otimes v \neq \Phi_1 \otimes v + \Phi_2 \otimes v$$

In ENFrame, under an assignment that satisfies both Φ_1 and Φ_2, the left side of the equality evaluates to v, whereas the right side becomes $v + v$. In other words: ENFrame does not distribute '$\otimes$' over conjunctions and disjunctions in Boolean expressions.

4.2.2 Syntax of event expressions

The grammar for event expressions is introduced in Figure 4.2. Elements x_i from the set of independent Boolean random variables $\mathbf{X}$ are the most basic type of Boolean event ($\langle bevent \rangle$). Various propositional operators can be used to give rise to other Boolean events. When used together with a primitive value or an array ($\langle val \rangle$), the Boolean events can be used to construct conditional values ($\langle cval \rangle$). Common mathematical operations (*e.g.* addition, multiplication, exponentiation) on a conditional value can be used to give rise to a new real-valued event ($\langle revent \rangle$). A series of

$\langle prim \rangle$ ::= value of primitive type

$\langle int \rangle$ ::= an integer

$\langle inparr \rangle$ ::= an array in the input data (*e.g.*, an object feature vector)

$\langle arr \rangle$ ::= $\langle inparr \rangle \mid \langle arr \rangle$ '+' $\langle arr \rangle \mid \langle arr \rangle$ '$\cdot$' $\langle arr \rangle \mid \mathcal{F}_k$ '(' $\langle arr \rangle$ ')'

$\langle val \rangle$ ::= $\langle prim \rangle \mid \langle arr \rangle$

$\langle bevent \rangle$ ::= $x_i \in \mathbf{X} \mid \langle bevent \rangle$ '$\wedge$' $\langle bevent \rangle \mid \langle bevent \rangle$ '$\vee$' $\langle bevent \rangle \mid$ '$\neg$' $\langle bevent \rangle$
$\mid \langle comp \rangle \mid$ '$\bigwedge$' $\langle arr \rangle \mid$ '$\bigvee$' $\langle arr \rangle \mid \langle eid \rangle$

$\langle cval \rangle$ ::= $\langle bevent \rangle$ '$\otimes$' $\langle val \rangle \mid \langle val \rangle$

$\langle revent \rangle$ ::= $\langle cval \rangle \mid \langle revent \rangle^{-1} \mid \langle revent \rangle$ '+' $\langle revent \rangle \mid \langle revent \rangle^{\langle int \rangle}$
$\mid \langle revent \rangle$ '$\cdot$' $\langle revent \rangle \mid$ dist '(' $\langle cval \rangle, \langle cval \rangle$ ')'
$\mid$ '$\sum$' $\langle arr \rangle \mid$ '$\prod$' $\langle arr \rangle$

$\langle relop \rangle$ ::= $\leq \mid \geq \mid = \mid < \mid >$

$\langle comp \rangle$ ::= $\langle revent \rangle \langle relop \rangle \langle revent \rangle$

$\langle eid \rangle$ ::= event identifier

Figure 4.2: Grammar of event expressions in Backus-Naur Form

relational operators ($\langle relop \rangle$) can be used on real-valued events to construct a comparison ($\langle comp \rangle$), which itself is again a Boolean event ($\langle bevent \rangle$).

The grammar presented in Figure 4.2 does not distinguish between events over scalars and vectors for reasons of notational clarity. In this context, it is assumed that expressions are well-typed; *e.g.* the expression $x \vee y$ requires both x and y to be Booleans.

$$
\begin{aligned}
\alpha(\langle revent\rangle_1 + \langle revent\rangle_2) &= \alpha(\langle revent\rangle_1) + \alpha(\langle revent\rangle_2)\\
\alpha(\langle revent\rangle_1 \cdot \langle revent\rangle_2) &= \alpha(\langle revent\rangle_1) \cdot \alpha(\langle revent\rangle_2)\\
\alpha\Big(\langle revent\rangle^{\langle int\rangle}\Big) &= \alpha(\langle revent\rangle)^{\langle int\rangle}\\
\alpha\Big(\langle revent\rangle^{-1}\Big) &= \alpha(\langle revent\rangle)^{-1}
\end{aligned}
$$

$$
\begin{aligned}
\alpha(\langle arr\rangle_1 + \langle arr\rangle_2) &= \alpha(\langle arr\rangle_1) + \alpha(\langle arr\rangle_2)\\
\alpha(\langle arr\rangle_1 \cdot \langle arr\rangle_2) &= \alpha(\langle arr\rangle_1) \cdot \alpha(\langle arr\rangle_2)\\
\alpha(\mathcal{F}_k(\langle arr\rangle)) &= \underbrace{[v_0, v_1, \ldots]}_{\text{(up to } k \text{ items)}} \quad (\forall v_i \in \langle arr\rangle \;:\; \alpha(v_i) \neq u)
\end{aligned}
$$

$$
\begin{aligned}
\alpha(\langle bevent\rangle_1 \vee \langle bevent\rangle_2) &= \alpha(\langle bevent\rangle_1) \vee \alpha(\langle bevent\rangle_2)\\
\alpha(\langle bevent\rangle_1 \wedge \langle bevent\rangle_2) &= \alpha(\langle bevent\rangle_1) \wedge \alpha(\langle bevent\rangle_2)
\end{aligned}
$$

$$
\alpha(\langle bevent\rangle \otimes \langle val\rangle) = \begin{cases} \langle val\rangle & \text{if } \alpha(\langle bevent\rangle) = true\\ u \;\; (\boldsymbol{u} \text{ resp.}) & \text{otherwise}\end{cases}
$$

$$
\alpha\Big(\langle bevent\rangle_1 \otimes (\langle bevent\rangle_2 \otimes \langle val\rangle)\Big) = \alpha\Big((\langle bevent\rangle_1 \wedge \langle bevent\rangle_2) \otimes \langle val\rangle\Big)
$$

$$
\alpha\Big(\text{dist}(\langle cval\rangle_1, \langle cval\rangle_2)\Big) = \begin{cases} u & \text{if } \alpha(\langle cval\rangle_1) = u \;\text{ or }\; \alpha(\langle cval\rangle_2) = u\\ \text{dist}\Big(\alpha(\langle cval\rangle_1), \alpha(\langle cval\rangle_2)\Big) & \text{otherwise}\end{cases}
$$

$$
\text{for } \diamond \in \{\leq, <, =, >, \geq\}\text{:}\quad \alpha(\langle revent\rangle_1 \diamond \langle revent\rangle_2) = \begin{cases} false & \text{if } \alpha(\langle revent\rangle_1) = u \;\text{ and }\; \alpha(\langle revent\rangle_2) = u\\ true & \text{if } \alpha(\langle revent\rangle_1) = u \;\text{ or }\; \alpha(\langle revent\rangle_2) = u\\ \alpha(\langle revent\rangle_1) \diamond \alpha(\langle revent\rangle_2) & \text{otherwise}\end{cases}
$$

Figure 4.3: Semantics of event expressions specified in the event language grammar (Figure 4.2) under a valuation $\alpha : \mathbf{X} \to \{true, false\}$

4.3 Probabilistic semantics of events

The events introduced in this chapter can be interpreted as random variables. This section discusses the semantics of this interpretation, and shows how a probability distribution is defined over such events.

The Boolean random variables $x \in \mathbf{X}$ are used to construct Boolean event expressions $\langle bevent \rangle$. Conditional values are subsequently used to construct random variables over their respective range.

For every Boolean random variable $x \in \mathbf{X}$, let the constants $P_x[true]$ and $P_x[false]$ denote the probability that x is *true* or *false*, respectively. The short-hand notation P_x is used for $P_x[true]$, and $P_{\bar{x}}$ for $P_x[false]$. In line with probability theory, it is assumed that $0 < P_x < 1$ and $P_x[true] + P_x[false] = 1$.

PROPOSITION 4.1: Induced probability space

Let $\Omega = \{\alpha : \mathbf{X} \to \{true, false\}\}$ be the set of all mappings from the independent random variables $\mathbf{X}$ to *true* and *false*, and let $\alpha(x)$ denote a valuation for an $x \in \mathbf{X}$, as part of some mapping $\alpha \in \Omega$.

The probability mass function $\Pr(\alpha) = \prod_{x \in \mathbf{X}} P_x[\alpha(x)]$ for every sample $\alpha \in \Omega$, and the probability measure $\Pr(E) = \sum_{\alpha \in E} \Pr(\alpha)$ for $E \subseteq \Omega$ together form a probability space $(\Omega, 2^{\Omega}, \Pr)$ that is referred to as the *probability space induced by* $\mathbf{X}$.

PROOF (SKETCH). The values $\Pr_x[\alpha(x)]$ represent probabilities such that $0 \leq \Pr_x[\alpha(x)] \leq 1$, therefore $0 \leq \Pr(\alpha) \leq 1$. All distinct assignments $\alpha_i, \alpha_j \in \Omega$ are countable and pairwise disjoint by their very definition. Hence: $\Pr\left[\cup_{\alpha_l \in \Omega}\right] = \sum_{\alpha_l \in \Omega} \Pr[\alpha_l] = 1$.

Using Proposition 4.1, the probability distribution of any event (random variable) can be defined as follows.

DEFINITION 4.1: Probability distribution of events

Every event expression Φ is a random variable over the probability space induced by $\mathbf{X}$. Its probability is defined as follows:

$$P_{\Phi}[s] = \Pr(\{\alpha \in \Omega \mid \alpha(\Phi) = s\}) = \sum_{\substack{\alpha \in \Omega: \\ \alpha(\Phi) = s}} \Pr(\alpha).$$

By virtue of Definition 4.1, every Boolean event expression becomes a Boolean random variable, and other events become random variables over their respective ranges. The following example illustrates how this principle applies to the first-k function $\mathcal{F}_k$.

EXAMPLE 4.4: Probabilistic semantics of first-k function $\mathcal{F}_k(A)$

Consider the following array A containing integer values which are conditioned on the (Boolean) existence event of objects $o_0, \ldots, o_4$:

$$A = [\Phi[o_0] \otimes 2,\ \ \Phi[o_1] \otimes 5,\ \ \Phi[o_2] \otimes 2,\ \ \Phi[o_3] \otimes 1,\ \ \Phi[o_4] \otimes 2]$$

With:

$$\Phi[o_0] = x_0,\ \ \Phi[o_1] = x_1,\ \ \Phi[o_2] = x_2,\ \ \Phi[o_3] = x_3,\ \ \Phi[o_4] = x_4$$

Then, under valuation α:

$$\alpha = \{x_0 \mapsto \mathit{true},\ x_1 \mapsto \mathit{false},\ x_2 \mapsto \mathit{false},\ x_3 \mapsto \mathit{true},\ x_4 \mapsto \mathit{true}\}$$

only objects o_0, o_3, and o_4 exist. *I.e.*,

$$\alpha(\Phi[o_0]) = \mathit{true},\ \ \alpha(\Phi[o_1]) = \mathit{false},\ \ \alpha(\Phi[o_2]) = \mathit{false},\ \ \alpha(\Phi[o_3]) = \mathit{true},\ \ \ldots$$
$$\alpha(A) = [2, u, u, 1, 2]$$

And consequently, the list $\mathcal{F}_2(A)$ of the first $k = 2$ values of A under valuation α is:

$$\alpha(\mathcal{F}_2(A)) = [2, 1]$$

Furthermore, following Definition 4.1, the probability $\Pr[\mathcal{F}_2(A) = [2, 1]]$ is defined as the sum of the probabilities of all valuations $\alpha_j \in \Omega$ which yield $\alpha_j(\mathcal{F}_2(A)) = [2, 1]$. In this example, there are two such (partial) valuations:

$$\alpha_1 = \{x_0 \mapsto \mathit{true},\ x_1 \mapsto \mathit{false},\ x_2 \mapsto \mathit{false},\ x_3 \mapsto \mathit{true}\}$$
$$\alpha_2 = \{x_0 \mapsto \mathit{false},\ x_1 \mapsto \mathit{false},\ x_2 \mapsto \mathit{true},\ x_3 \mapsto \mathit{true}\}$$

⟨*var*⟩ ::= a variable symbol

⟨*loop*⟩ ::= { {⟨*decl*⟩} {∀ ⟨*var*⟩ in ⟨*int*⟩..⟨*int*⟩: {⟨*loop*⟩}} }

⟨*decl*⟩ ::= ⟨*eid*⟩ ≡ ⟨*event*⟩

Figure 4.4: Grammar of event programs in Backus-Naur Form. Reuses non-terminal symbols from event expression grammar (Figure 4.2).

Note how the valuations of x_4 (and, by extension, the existence of o_4) is not specified by either partial valuation α_1 nor α_2, as it is irrelevant to the probability $\Pr[\mathcal{F}_2(A) = [2,1]]$:

$$\begin{aligned}\Pr[\mathcal{F}_2(A) = [2,1]] &= \Pr[\alpha_1] + \Pr[\alpha_2] \\ &= P_{x_0} \cdot P_{\bar{x}_1} \cdot P_{\bar{x}_2} \cdot P_{x_3} \quad + \quad P_{\bar{x}_0} \cdot P_{\bar{x}_1} \cdot P_{x_2} \cdot P_{x_3}\end{aligned}$$

4.4 Event programs for data mining algorithms

Event programs are imperative specifications that define a finite sequence of named event expressions. Figure 4.4 presents the grammar for event programs, which reuses part of the grammar for event expressions (as presented in Figure 4.2). Event programs consist of a sequence of event declarations (⟨*decl*⟩) and nested loops (⟨*loop*⟩) of event declarations.

A central concept is that of event identifiers (⟨*eid*⟩); it is required that event declarations are immutable, *i.e.*, each distinct identifier may only be assigned once to an event expression. Inside a ∀*i*-loop, identifiers can be parametrised by *i* to create a distinct identifier in each iteration of the loop.

The semantics of an event program is the set of all named and *expanded* event expressions defined by the program. Expanded expressions are expressions of which all identifiers are recursively resolved and replaced by the referenced expressions. For declarations outside loops this procedure is unequivocal; each declaration inside one or more loops is instantiated for each value of the loop counter variables.

After having introduced the grammar and semantics of event programs and expressions, the remainder of this chapter will describe the translation of the user programs from Chapter 3 into event programs.

N.B. the combined size of the user and event programs prevents listing the user program and its translation side-by-side. Appendix C contains a number of pages in landscape orientation showing the two programs next to each other, with additional vertical spacing to delineate the translation.

4.4.1 *k*-means clustering

The event program for *k*-means clustering is provided in Algorithm 4.1. The algorithm starts with conditioning the object feature vectors $\vec{o_l}$ on the object event $\Phi[o_l]$, the result of which is stored in the c-value event O^i (line 1). This conditional feature vector is used throughout the algorithm's clustering phases.

On line 3, the initialisation phase takes place: k initial centroids are selected. The initial centroid are stored in events $M^0_{-1}, \ldots, M^{k-1}_{-1}$; the event subscript '$-1$' indicates that this is the value during the initialisation phase. In *k*-means (contrary to *k*-medoids), these centroids can be located anywhere in the feature space. The centroids are located at the first k objects of the data set (their probabilistic nature is irrelevant; only their location is used). This initialisation can be replaced with any type of heuristic that generates k disjoint centroids. This is further discussed in Section 3.3.1.

The iteration loop on line 5 contains the assignment and update phases for every iteration t ($0 \leq t < it$). An object o_l is assigned to cluster i in the event that centroid M^i_{t-1} (*i.e.*, as determined in the previous iteration $t-1$) is closer to o_l than any other centroid M^j_{t-1}; this gives rise to a new Boolean event $InCl^{i,l}_t$. Note how the semantics of the dist$(\cdot,\cdot)$ function and the logical operator ensure that $InCl^{i,l}_t$ is *false* in the event that object o_l does not exist. In such cases, dist$\left(O^l, y\right)$ defaults to u (for every y), and the logical operator evaluates to *false*.

For readability reasons, the encoding of the `breakTies2` function in the user program (discussed in Section 3.3.1) is omitted; a possible implementation is discussed later in this section.

After the *InCl* events have been constructed, the algorithm enters the update phase on line 12. For every cluster i, the event M^i_t stores the cluster centroid. The following expression counts the number of objects in the cluster, and inverts (*i.e.*, $\frac{1}{x}$) it:

$$\left(\sum_{l=0}^{n-1} InCl^{i,l}_t \otimes 1\right)^{-1}$$

This is then multiplied with the sum of all feature vectors of objects in the cluster. Hence, the resulting event M^i_t ranges over all possible cluster centres.

Note how the events for the assignment phase re-use (*i.e.*, depend on) the centroid events from the initialisation phase (if $t = 0$) or the previous update phase (if $t > 0$). Subsequently, the events in the

$\vec{o_0} \ldots \vec{o_{n-1}}$: n object vectors in the feature space

$\Phi[o_0] \ldots \Phi[o_{n-1}]$: n propositional object events

k, it: number of clusters and iterations

1 $\forall i \; in \; 0..n-1 : O^i \equiv \Phi[o_i] \otimes \vec{o_i}$

2

3 $M^0_{-1} \equiv \vec{o_0}; \ldots; M^{k-1}_{-1} \equiv \vec{o}_{(k-1)}$

4

5 $\forall t \; in \; 0..it-1 :$

6 $\quad \forall i \; in \; 0..k-1 :$

7 $\quad\quad \forall l \; in \; 0..n-1 :$

8 $$InCl^{i,l}_t \equiv \bigwedge_{j=0}^{k-1} \left[dist\left(O^l, M^i_{t-1}\right) \leq dist\left(O^l, M^j_{t-1}\right) \right]$$

9

10 $\quad\quad$ # Encoding of `breakTies2` omitted

11

12 $\quad \forall i \; in \; 0..k-1 :$

13 $$M^i_t \equiv \left(\sum_{l=0}^{n-1} InCl^{i,l}_t \otimes 1 \right)^{-1} \cdot \left(\sum_{l=0}^{n-1} InCl^{i,l}_t \otimes O^l \right)$$

Algorithm 4.1: ENFrame event program for k-means clustering (see Appendix C for a side-by-side listing of the user and event programs)

update phase re-use the events constructed using the assignment phase. The result is a collection of highly interconnected and therefore correlated events. Chapter 5 introduces a data structure which stores this network of events and facilitates probability computation.

Tie breaking

The importance of tie breaking in the user programs for k-means and k-medoids (as well as other algorithms) has been discussed in Section 3.3.1. The same applies for the event programs for these algorithms: some of the events should be mutually exclusive. For example, the event for cluster assignment $InCl^{i,l}_t$ should be mutually exclusive in for different clusters i: no single object l can be in two clusters in the same iteration – even if the two cluster centroids (or medoids) are equidistant to the object.

For readability reasons the encoding of `breakTies2` has been omitted from Algorithm 4.1. The solution presented in Section 3.3.1 applies here too: by prioritising clusters with a lower index, the

tie between two equidistant clusters can be broken. This requires replacing the conjunction in the event definition on line 8:

$$\bigwedge_{j=0}^{k-1}\left[\text{dist}\left(O^l, M_{t-1}^i\right) \leq \text{dist}\left(O^l, M_{t-1}^j\right)\right]$$

By the following more intricate conjunction:

$$\bigwedge_{j=0}^{k-1}\left[\left(\text{dist}\left(O^l, M_{t-1}^i\right) < \text{dist}\left(O^l, M_{t-1}^j\right)\right) \vee \left(\left(i \leq j\right) \wedge \left(\text{dist}\left(O^l, M_{t-1}^i\right) = \text{dist}\left(O^l, M_{t-1}^j\right)\right)\right)\right]$$

4.4.2 *k*-medoids clustering

The event program for *k*-medoids clustering is presented in Algorithm 4.2. The conditioning of object vectors is identical to *k*-means, but the initialisation phase (line 3) is different, and the subject of further discussion in Section 4.4.3. As is the case with the user program, the event program code for *k*-medoids' assignment phase is identical to that of *k*-means.

The update phase, however, is more intricate. The event $DistSum_t^{i,l}$ represents the total distance sum for an object o_l to all objects in cluster i at iteration t (line 14). It is constructed using a sum of the distances between o_l and all other objects o_p, where each of the distances is conditioned on the event that o_p was assigned to cluster l in the preceding assignment phase. As a result, $DistSum_t^{i,l}$ is a real-valued event of which the sample space contains all possible total distance sums. The clustering process does not require materialisation of this sample space and its probability distribution as the value of the probabilistic total distance sum is used in a comparison later in the update phase.

$IsMedoid_t^{i,l}$ represents the event that object o_l is made medoid of cluster i during iteration t. This event relies on the probabilistic distance sums and the cluster assignment events: an object o_l is the medoid of cluster i during iteration t if o_l's total distance sum is smaller than that of any other object o_p in cluster i, provided that o_l itself is a member of the cluster.

Event M_t^i represents the medoid location for cluster i in iteration t. The event is constructed using a summation over object feature vectors which are conditioned on their election as medoid (*i.e.*, $IsMedoid_t^{i,l}$). Since only a single object can ever be elected as medoid at any one time, the summation effectively sums over a single real feature vector and a series of undefined vectors.

$\vec{o}_0 \dots \vec{o}_{n-1}$: n object vectors in the feature space

$\Phi[o_0] \dots \Phi[o_{n-1}]$: n propositional object events

k, it: number of clusters and iterations

1 $\forall i \; in \; 0..n-1 : O^i \equiv \Phi[o_i] \otimes \vec{o}_i$

2

3 $M_{-1} \equiv initialiseMedoids()$

4

5 $\forall t \; in \; 0..it-1 :$

6 $\quad \forall i \; in \; 0..k-1 :$

7 $\quad\quad \forall l \; in \; 0..n-1 :$

8 $\quad\quad\quad InCl_t^{i,l} \equiv \bigwedge_{j=0}^{k-1} \Big[dist\Big(O^l, M_{t-1}^i\Big) \leq dist\Big(O^l, M_{t-1}^j\Big) \Big]$

9

10 $\quad$ # Encoding of `breakTies2` omitted

11

12 $\quad \forall i \; in \; 0..k-1 :$

13 $\quad\quad \forall l \; in \; 0..n-1 :$

14 $\quad\quad\quad DistSum_t^{i,l} \equiv \sum_{p=0}^{n-1} \Big[InCl_t^{i,p} \otimes dist\Big(O^l, O^p\Big) \Big]$

15

16 $\quad\quad \forall l \; in \; 0..n-1 :$

17 $\quad\quad\quad IsMedoid_t^{i,l} \equiv \bigwedge_{p=0}^{n-1} \Big[DistSum_t^{i,l} \leq \Big(InCl_t^{i,p} \otimes DistSum_t^{i,p} \Big) \Big] \wedge InCl_t^{i,l}$

18

19 $\quad\quad$ # Encoding of `breakTies` omitted

20

21 $\quad\quad M_t^i \equiv \sum_{l=0}^{n-1} \Big[IsMedoid_t^{i,l} \otimes O^l \Big]$

Algorithm 4.2: ENFrame event program for k-medoids clustering (see Appendix C for a side-by-side listing of the user and event programs)

4.4.3 Initial medoid selection

As has been discussed in Section 3.3.1, the choice of initial medoids (for k-means: centroids) is of significant importance to a clustering. In a deterministic scenario, every medoid needs to be a member of the data set. In a probabilistic scenario, a medoid event depends on the existence event of the object. In other words, in every possible world, every cluster medoid should be one of the objects that actually exists in that world. This requirement makes the initial medoid selection more intricate than the initial centroid selection in probabilistic k-means clustering: the k initial medoids cannot be drawn from the first k objects from the data set. A possible solution to this was presented by Example 4.2, which can be expressed in the event language as follows:

$$
\begin{aligned}
&\forall i \; in \; 0..k-1 : \\
&\quad \forall l \; in \; 0..n-1 : \\
&\qquad IsInitMedoid^{i,l} = \Phi[o_l] \wedge \neg \left(\bigvee_{p=0}^{l-1} IsInitMedoid^{i,p} \right) \wedge \neg \left(\bigvee_{j=0}^{i-1} IsInitMedoid^{j,l} \right) \\
&M^i_{-1} = \sum_{l=0}^{n-1} \left[IsInitMedoid^{i,l}_t \otimes O^l \right]
\end{aligned}
$$

This code constructs a series of Boolean events $IsInitMedoid^{i,l}$, each of which represents the event that object o_l is the initial medoid of cluster i. An object o_l is selected as a medoid for cluster i if:

1. the object exists; and
2. no other object o_p ($p < l$) is medoid of cluster i; and
3. o_l is not a medoid of any cluster j ($j < i$).

Once these Boolean events have been constructed, the k initial medoids are stored in M^i_{-1} using a construct virtually identical to the one used in the update phase on line 21.

4.4.4 Markov clustering

The probabilistic translation of the Markov clustering user program is presented in Algorithm 4.3. Lines 1–3 are a pre-processing step. The values in the stochastic matrix $\mathcal{M}$ are conditioned on the object events $\Phi[o_l]$ (line 1), after which the conditioned values are normalised to maintain the stochastic property of $\mathcal{M}$. The resulting values in $M^{i,j}_{-1}$ are used in the expansion step on line 7, which constructs the temporary matrix N by self-multiplication (squaring). This matrix is inflated and renormalised on line 11. As is the case with the user program, the cluster assignment happens after the iterative part of the program has completed (line 15).

$\mathcal{M}$: stochastic $n \times n$ matrix with edge weights

$\Phi[o_0] \ldots \Phi[o_{n-1}]$: n propositional object (vertex) events

r, it: inflation parameter and number of iterations

1 $\forall i \; in \; 0..n-1: \quad \forall j \; in \; 0..n-1: \quad M_{-2}^{i,j} \equiv \Big(\Phi[o_i] \wedge \Phi[o_j]\Big) \otimes \mathcal{M}^{i,j}$

2

3 $\forall i \; in \; 0..n-1: \quad \forall j \; in \; 0..n-1: \quad M_{-1}^{i,j} \equiv M_{-2}^{i,j} \cdot \Big(\sum_{k=0}^{n-1} M_{-2}^{i,k}\Big)^{-1}$

4 $\forall t \; in \; 0..it-1:$

5 $\quad \forall i \; in \; 0..n-1:$

6 $\quad\quad \forall j \; in \; 0..n-1:$

7 $\quad\quad\quad N_t^{i,j} \equiv \sum_{k=0}^{n-1} \Big[M_{t-1}^{i,k} \cdot M_{t-1}^{k,j} \Big]$

8

9 $\quad \forall i \; in \; 0..n-1:$

10 $\quad\quad \forall j \; in \; 0..n-1:$

11 $\quad\quad\quad M_t^{i,j} \equiv \Big(N_t^{i,j}\Big)^r \quad \cdot \quad \left(\sum_{k=0}^{n-1} \Big(N_t^{i,k}\Big)^r\right)^{-1}$

12

13 $\forall i \; in \; 0..n-1:$

14 $\quad \forall j \; in \; 0..n-1:$

15 $\quad\quad InCl^{i,j} \equiv M_{it-1}^{i,j} > 0$

Algorithm 4.3: ENFrame event program for Markov clustering (see Appendix C for a side-by-side listing of the user and event programs). The event $M_{-2}^{i,j}$ represents the value of element (i, j) in the stochastic matrix prior to normalisation ($M_{-1}^{i,j}$).

4.4.5 *k*-nearest neighbour classification

The event program for k-nearest neighbour classification is presented in Algorithm 4.4. The object-cluster votes are gathered on line 4. The vote value is conditioned on two Boolean events in two separate c-values: the existence of the vote-casting object ($\Phi[o_l]$), and the class assignment C^l of said object l. As a result, the event $Votes^{l,i}$ has three possible values:

$$\alpha\Big(Votes^{l,i}\Big) = \begin{cases} u & \text{in the event } \Phi[o_l] \text{ is } \textit{false}, \textit{i.e.} \text{ object } l \text{ does not exist} \\ 1 & \Phi[o_l] \text{ is } \textit{true} \text{ (}\textit{i.e.}\text{, object } l \text{ exists), but object } l \text{ does not have class label } i \\ 2 & \Phi[o_l] \text{ is } \textit{true}, \text{ and object } l \text{ has class label } i \end{cases}$$

n_l = number of labelled data points

n_c = number of classes

C^l = class of labelled data point l $(0 \leq C^l < n_c)$

N_u^m = m^{th} closest neighbour of unlabelled point o_u

1

2 $\forall l \; in \; 0 \ldots (n_l - 1) :$

3 $\quad \forall i \; in \; 0 \ldots (n_c - 1) :$

4 $\quad\quad Votes^{l,i} \equiv \Phi[o_l] \otimes \Big(1 + \Big(\big(C^l = i\big) \otimes 1\Big)\Big)$

5

6 $\forall u \; in \; 0 \ldots (n_u - 1) :$

7 $\quad \forall i \; in \; 0 \ldots (n_c - 1) :$

8 $\quad\quad \forall m \; in \; 0 \ldots (n_l - 1) :$

9 $\quad\quad\quad NearestVotes_u^m \equiv Votes^{N_u^m, i}$

10

11 $\quad\quad VoteSum_u^i \equiv \sum \mathcal{F}_k(NearestVotes_u)$

12

13 $\quad \forall i \; in \; 0 \ldots (n_c - 1) :$

14 $\quad\quad ClassAssign_u^i \equiv \bigwedge_{p=0}^{n_c - 1} \Big[VoteSum_u^i \geq VoteSum_u^p\Big]$

15

16 # definition of breakTies omitted

Algorithm 4.4: ENFrame event program for k-nearest neighbour classification (see Appendix C for a side-by-side listing of the user and event programs)

On line 9, the votes are sorted by increasing distance to unlabelled object u using N_u, which is a sorted list of u's neighbours. The result is stored in the event $NearestVotes_u^m$, which constitutes a probability distribution over all possible sorted lists of votes for class m from neighbours of u.

The $VoteSum_u^i$ event collects the votes for class i from the k nearest neighbours to u by applying the first-k function $\mathcal{F}_k$ on the list of nearest votes (also see Example 4.4). Lastly, the class assignment is decided on line 14 by comparing the *VoteSum* events of the different classes.

4.5 Translating user programs to event programs

As has been shown in the translations of the classification and clustering algorithms, virtually every construct in a user program maps to a probabilistic counterpart in the event programs. However, one of the main differences that can be observed is the naming of variables versus the naming and assignment of their probabilistic counterparts – the events.

In user language programs, variables are reassigned repeatedly. For example, the changing centroids in *k*-means are stored in the same variable throughout the iterations. In contrast, events in event programs are *immutable*: they can only be assigned once, and cannot be changed or reassigned later. The rationale behind this is that events depend on each other and represent a trace of the computation of an algorithm. As will be discussed in Chapter 5, probability computation is only possible if the full trace is available. Therefore, during the translation from user to event program, the created events are annotated with the loop iterator variable. For example, the iteration counter variable t in *k*-means, *k*-medoids, and Markov clustering features in all event definitions inside the loop.

The representation of arrays also differs between the two languages. Arrays in user language programs have a known fixed size, which facilitates a straightforward translation into the event language. Any k-dimensional array $M[n_1], \ldots, [n_k]$ in the user language translates to $\prod_i n_i$ distinct events $M^{0,\ldots,0}, \ldots, M^{n_1-1,\ldots,n_k-1}$ in the event language.

4.6 Interpreting a probabilistic result

The output of an ENFrame program is twofold:

1. a symbolic trace of the program in the form of events, and

2. the probabilities of values of (a selection of) events

The programmer is responsible for constructing events (through user program variables) that represent a useful algorithm result. This requires some insight into concepts of uncertainty and probabilities, as uncertainty fundamentally changes the way an algorithm output needs to be interpreted.

For example: using a clustering algorithm to compute a series of probability distributions that represent object-to-cluster assignments is not necessarily a useful result. If the distributions of two distinct objects indicate that both have some non-zero probability of being assigned to some cluster,

this does not provide any insight on whether these objects occur in this cluster together. In fact, the objects might be mutually exclusive (or strongly negatively correlated), and still both have a non-zero probability of being assigned to some cluster. A more telling interpretation of the output would be the pairwise assignment probabilities, which can be computed using $O\left(n^2\right)$ events (one for every pair of distinct objects in the input data).

The user language allows for constructing a plethora of such queries on the events in the event network. It is up to the user to write the right query for a functional interpretation of the output of his probabilistic algorithms. Chapter 7 describes an application of a probabilistic clustering algorithm, and the interpretation of its output. The interpretation of a probabilistic result inevitably requires the programmer to have a basic understanding of probabilities and probability distributions. However, ENFrame hides the most intricate parts of the interpretation process, as well as the probability computation itself.

Chapter 5

Event Networks and Probability Computation

The previous chapters discussed deterministic user programs and their probabilistic interpretation into event programs. The event programs give rise to large numbers of highly interconnected (correlated) events that form a trace of the program for probabilistic data. The data structure that stores these events and facilitates probability computation is called an *event network*, and is introduced in this chapter. Furthermore, a variety of exact and approximate algorithms for probability computation of event networks is presented, including techniques for parallel computation.

5.1 Storing events in networks

The event programs in Chapter 4 construct large numbers of events. Rather than *expanding* every event when a program is run (*i.e.*, recursively in-lining events it depends on), ENFrame stores events in data structures called *event networks*. These networks are large directed graphs in which vertices represent events, and edges indicate their interdependencies. Every event is only stored once, and can be reused by adding additional edges to the event network.

Definition 5.1: Event network

An event network is a directed graph $G = (V, E)$ which contains events (vertices) $v \in V$ as specified by the event program. Every event $v_i \in V$ is constructed using an operator (as defined in the language's grammar in Figure 4.2). An edge $(v_a, v_b) \in E$ represents the dependency of *parent* event v_b on *child* v_a.

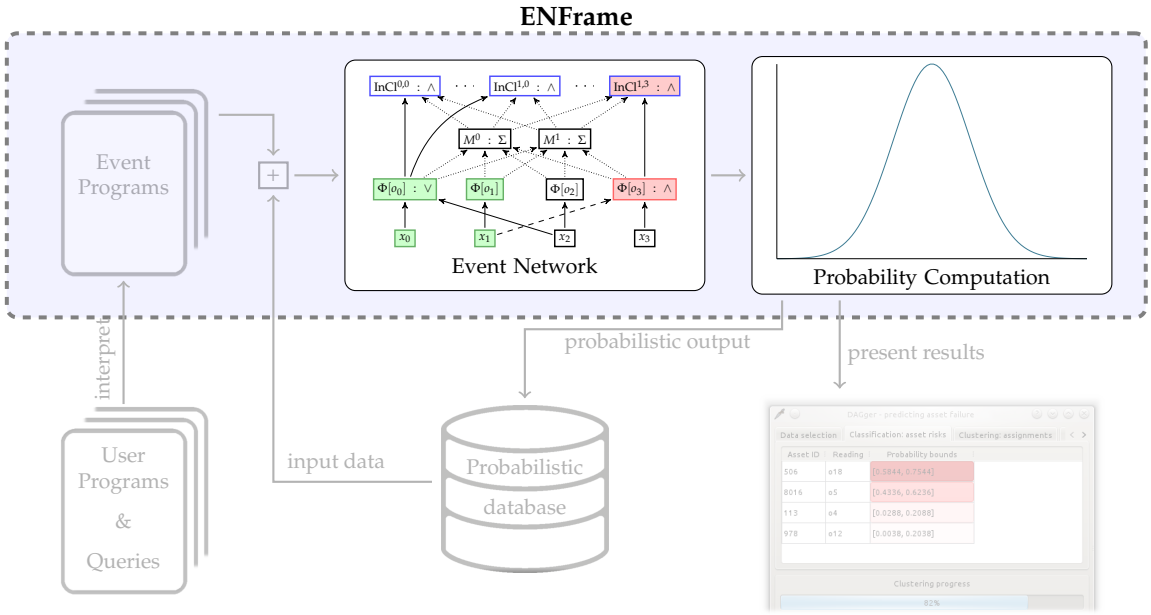
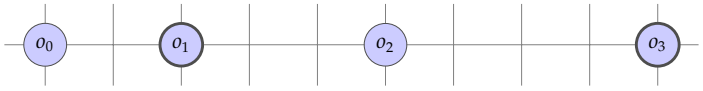


Figure 5.1: ENFrame architecture: chapter 5

The foundation of every event network consists of the Boolean random variables $x_i \in \mathbf{X}$ from the input data, and the object events that are constructed using these variables. This foundation is illustrated in the next example.

EXAMPLE 5.1: Foundation of event network for k-medoids clustering

Recall Example 1.2 and assume that the following four objects $o_0, \ldots, o_3$ comprise the input data set to a k-medoids clustering problem:

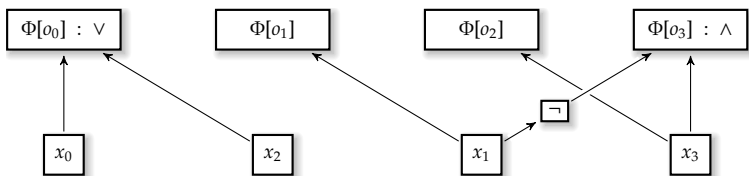


Furthermore, assume that each of the objects o_l is associated with a Boolean event $\Phi[o_l]$ over independent Boolean random variables $x_i \in \mathbf{X}$, as introduced in Example 4.3:

$$\Phi[o_0] = x_0 \vee x_2 \qquad \Phi[o_1] = x_1$$

$$\Phi[o_2] = x_3 \qquad \Phi[o_3] = \neg x_1 \wedge x_3$$

Then, the foundation of the event network will look like this:

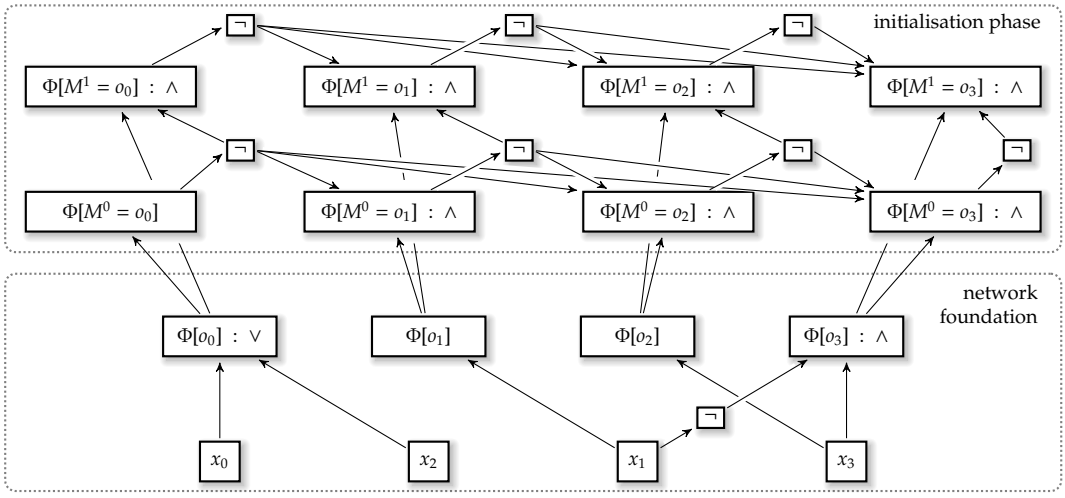


The four Boolean random variables are located at the bottom of the network, and form the basis for a negation event and the four object events with labels $\Phi[o_l]$ which are events (disjunctions and conjunctions) over the random variables. For example, node $\Phi[o_3]$ (top right corner) is a conjunction of x_3 and the negation of x_1. For sake of clarity, every object event is drawn as a distinct vertex, even if the event is equal to one of the variables (*e.g.*, $\Phi[o_1] = x_1$, but both are drawn as distinct vertices).

Using the foundation of object events, the network is built up following the event definitions in the event program. The following example illustrates how the initialisation phase of k-medoids results in new events that can be added to the network introduced in Example 5.1.

Example 5.2: Simplified event network for k-medoids clustering (contd.)

The first phase of the k-medoids clustering algorithm is the initialisation phase in which the initial medoids are selected. Following the definitions from Example 4.3, the event network from Example 5.1 can be expanded as follows to include the events for the initial medoid selection of the two clusters:



Note that a number of edges in the above network travel underneath unrelated nodes – only edges with an origin drawn close to a node actually originate from that node. Edges that appear to have no arrowhead are part of a longer edge.

In Example 4.3, the event $\Phi\left[M^1 = o_2\right]$ is defined as the following conjunction:

$$\Phi\left[M^1 = o_2\right] \equiv \Phi[o_2] \wedge \neg\Phi\left[M^1 = o_0\right] \wedge \neg\Phi\left[M^1 = o_1\right] \wedge \neg\Phi\left[M^0 = o_2\right]$$

This event is represented as a node in the event network, with four incoming edges from children: the operands of the conjunction.

The reasonably straightforward initialisation phase described in Example 5.2 (for only two clusters and four objects) makes for a complicated layer of interconnected events in the network. Adding the assignment and update phases unfortunately results in a very convoluted graph drawing, which is therefore not presented. In fact, the network that describes k-medoids clustering consists of $O(kn)$ events for the initialisation phase, $O\left(k^2 \cdot n\right)$ events for the assignment phase, and $O\left(k \cdot n^2\right)$ for the update phase. The completed network for the clustering problem described in the above example consistd of more than one hundred events; the network for a modest clustering problem for $k = 5$ and $n = 1000$ exceeds ten million events. Section 6.2.3 details an ENFrame feature called *higher-order events* which is designed to reduce the size of the network.

Event networks are used to facilitate the probability computation of (a subset of) the events generated by an event program, which is referred to as *compilation*. The last events generated by the event program often represent the probabilistic output of an algorithm. For example, $InCl^{i,l}_{\text{it}-1}$ represents the Boolean event that an object o_l is clustered into cluster i during the last iteration of k-means and k-medoids clustering, and $ClassAssign^{i}_{u}$ represents the event that an unlabelled object o_u is assigned class label i in k-nearest neighbour classification. These events are referred to as the *targets* of the *compilation*, or simply: *compilation targets*. During the compilation process, the exact distribution of the other events need not be known – a partial distribution often suffices, as will become clear in the next section. Hence, the compilation process focusses on the compilation targets, rather than all events.

5.2 Introduction to probability computation

The compilation techniques employed by ENFrame find their origins in Boole's expansion theorem [Boo54], which is sometimes referred to as *Shannon expansion* [Sha49] or more commonly *decomposition*. Given an expression Φ over Boolean random variables $\mathbf{X}$, Shannon's expansion procedure repeatedly selects a random variable $x_i \in \mathbf{X}$ and then computes two partial valuations of Φ: $\Phi|_{x_i}$ for x_i set to *true* ($\top$), and $\Phi|_{\neg x_i}$ for x_i set to *false* ($\bot$).

Using the disjunction of the (repeated) decompositions, the probability Pr[Φ] can be established as follows:

$$\Phi' = \left(x_i \wedge \Phi|_{x_i}\right) \vee \left(\neg x_i \wedge \Phi|_{\neg x_i}\right)$$

For which, notably:

$$\Pr[\Phi] = \Pr[\Phi']$$

The probability Pr [Φ] is then computed:

$$\begin{aligned}
\Pr[\Phi] &= \Pr[\Phi'] \\
&= \Pr\Big[\left(x_i \wedge \Phi|_{x_i}\right) \vee \left(\neg x_i \wedge \Phi|_{\neg x_i}\right)\Big] \\
&= \Pr\Big[x_i \wedge \Phi|_{x_i}\Big] + \Pr\Big[\neg x_i \wedge \Phi|_{\neg x_i}\Big] && \text{(following mutual exclusiveness)} \\
&= \Pr[x_i] \cdot \Pr\Big[\Phi|_{x_i}\Big] + \Pr[\neg x_i] \cdot \Pr\Big[\Phi|_{\neg x_i}\Big] && \text{(following indep. of } x_i, \Phi|_{x_i} \text{ and } \Phi|_{\neg x_i})
\end{aligned}$$

Example 5.3: Decomposition of a propositional formula

Consider the propositional formula:

$$\Phi = x_0 \vee x_1 \vee x_2$$

This expression can be decomposed into two partial valuations using x_0:

$$\Phi|_{x_0} = \top \vee x_1 \vee x_2 = \top \qquad \text{and} \qquad \Phi|_{\neg x_0} = \bot \vee x_1 \vee x_2 = x_1 \vee x_2$$

Because the variables $x_i \in \mathbf{X}$ are independent, the probability of Φ can be computed as follows:

$$\begin{aligned}
\Pr[\Phi] &= \Pr\Big[x_0 \wedge \Phi|_{x_0}\Big] \;+\; \Pr\Big[\neg x_0 \wedge \Phi|_{\neg x_0}\Big] \\
&= \Pr[x_0] \cdot \Pr\Big[\Phi|_{x_0}\Big] \;+\; \Pr[\neg x_0] \cdot \Pr\Big[\Phi|_{\neg x_0}\Big] \\
&= \Pr[x_0] \cdot 1 \;+\; \Pr[\neg x_0] \cdot \Pr\Big[\Phi|_{\neg x_0}\Big]
\end{aligned}$$

A further decomposition step (using either x_1 or x_2) is required to determine the probability $\Pr\Big[\Phi|_{\neg x_0}\Big]$ and finally obtain Pr[Φ].

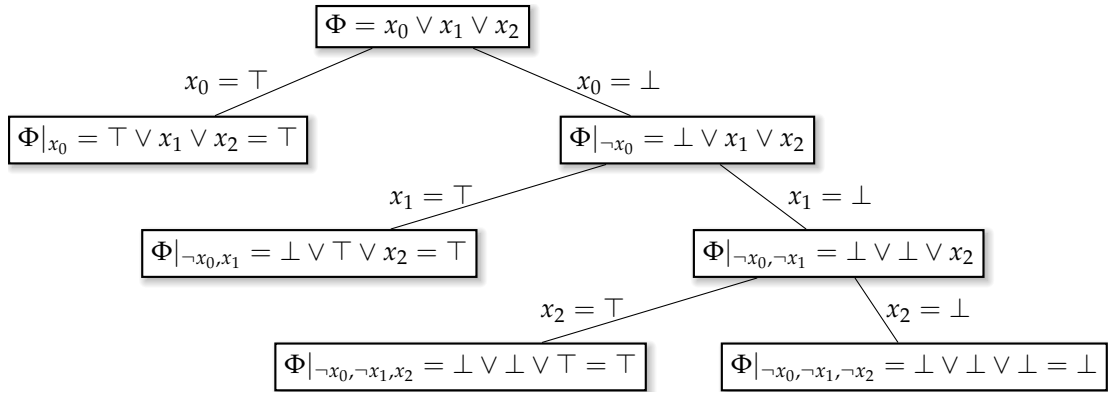


Figure 5.2: Example of a decision tree for $\Phi = x_0 \vee x_1 \vee x_2$

Every decomposition step results in two simpler expressions $\Phi|_{x_i}$ and $\Phi|_{\neg x_i}$. By repeating this procedure, all random variables in the expression Φ are eventually resolved to constants *true* and *false*. The trace of this repeated decomposition is called the *decision tree*, which is a type of binary decision diagram [Lee59]. An example of such a tree is depicted in Figure 5.2.

The height of the decision tree is bounded by the number of variables in the input data $h = |\mathbf{X}|$; the number of nodes (valuations) in the tree grows exponentially in h. However, as can be observed from Figure 5.2, not all branches necessarily reach full depth. Depending on the expression Φ and the order in which the variables are expanded on, some branches may (un)satisfy[1] Φ within fewer than h decomposition steps.

The probability of a leaf node can be computed by multiplying the probabilities of the variable valuations on the path from the root. Furthermore, the probability of Φ can be computed by summing the probabilities of all leaf nodes that satisfy Φ.

Example 5.4: Probability computation in a decision tree

Reconsider Example 5.3, and assume that the Boolean random variables x_0, x_1, x_2 have probabilities $\Pr[x_0] = 0.3$, $\Pr[x_1] = 0.4$, $\Pr[x_2] = 0.5$, respectively. Then, the probability of branch $\Phi|_{\neg x_0, \neg x_1, x_2}$ of the decision tree in Figure 5.2 (bottom) can be computed as follows:

$$\begin{aligned}\Pr\left[\Phi|_{\neg x_0, \neg x_1, x_2}\right] &= \Pr[\neg x_0] \cdot \Pr[\neg x_1] \cdot \Pr[x_2] \\ &= (1 - 0.3) \cdot (1 - 0.4) \cdot 0.5 \;=\; 0.21\end{aligned}$$

[1] Throughout this thesis, the term *unsatisfy* is used to denote a *false* evaluation of a Boolean expression Φ, rather than a non-evaluation.

Furthermore, the probability of $\Phi = x_0 \vee x_1 \vee x_2$ is obtained by summing the probabilities of the decision tree leaf nodes that satisfy Φ:

$$\begin{aligned}\Pr[\Phi] &= \Pr\left[\Phi|_{x_0}\right] + \Pr\left[\Phi|_{\neg x_0, x_1}\right] + \Pr\left[\Phi|_{\neg x_0, \neg x_1, x_2}\right] \\ &= 0.3 + (1 - 0.3) \cdot 0.4 + (1 - 0.3) \cdot (1 - 0.4) \cdot 0.5 \\ &= 0.79 \qquad (= 1 - 0.7 \cdot 0.6 \cdot 0.5)\end{aligned}$$

Note that this particular expression Φ serves as a simple example; its probability can be established without constructing a decision tree by computing $1 - \Pr[\neg x_0 \wedge \neg x_1 \wedge \neg x_2]$. However, events generated by ENFrame programs are generally significantly more complex than this example.

Although an exact computation of the probability of an arbitrary expression Φ requires a full traversal of the decision tree, the tree never needs to be fully materialised – a depth-first search suffices to collect probabilities of leaf nodes that satisfy Φ. Depending on the exact expression, the decision tree is not necessarily (in fact: often not) balanced, and the order in which the variables are processed has a significant influence on the height of the decision tree. For example: the decision tree for $\Phi = (x_0 \vee x_1 \vee x_2) \wedge x_2$ will have height three if constructed in the order in which the variables appear (from left to right, x_0, x_1, x_2). However, by prioritising x_2, the height can be restricted to just one. As will be detailed later, ENFrame employs a heuristic to find a favourable variable ordering.

By interpreting user programs using events and performing probability computation through decomposition, ENFrame's output becomes equivalent to executing the deterministic user program in every possible world. Moreover, by attempting to reduce the height of the tree (exploiting the fact that many possible worlds are alike), ENFrame can achieve a significant performance gain when compared to explicitly iterating over every single possible world.

As will be detailed in Section 5.4, ENFrame can establish an absolute ε-approximation by selectively pruning parts of this decision tree that contribute relatively little probability mass. Various approximation techniques have different strategies to decide which branches to prune.

5.2.1 Computational complexity

The problem of computing the probability of arbitrary propositional expressions over Boolean random variables is #P-hard, even for restricted cases such as positive bipartite expressions in

Problem	NP-hard decision problem variant	#P counting problem variant
Boolean satisfiability (SAT)	Given a propositional logic formula (Boolean expression) Φ over a set of Boolean variables $\mathbf{X}$, does there exist a valuation α of $\mathbf{X}$ that satisfies Φ?	How many such valuations α exist?
Subset sum	Given a set of integers S and an integer s, does a subset $S' \subseteq S$ exist such that $\sum S' = s$ (where typically, $s = 0$)?	How many such subsets S' exist?
Travelling salesman problem	Given a graph G and cost limit c, does G contain any Hamiltonian cycles with total costs $\leq c$?	How many cycles with total costs $\leq c$ does G contain?

Table 5.1: Problems in the complexity class NP and their #P counterparts

disjunctive normal form [PB83]. This inherent hardness calls for heuristics and approximation algorithms – both will be introduced later in this chapter.

The #P (pronounced "sharp P", "number P", or "hash P") complexity class was first defined in [Val79], and contains the set of counting problems that are associated with problems in NP. Table 5.1 shows a number of examples of such pairs of problems.

Exact probability computation of an expression Φ requires a full traversal of the decision tree, which is computationally equivalent to counting all valuations α that satisfy a Boolean expression (#SAT). A naïve approach would consider every compilation target (event) individually and compute the probabilities of its various possible values.

To counteract the hardness of the problem, ENFrame employs a bulk compilation technique inspired by decomposition that considers a large number of expressions (the compilation targets) at the same time. As a result, every node in the decision tree applies to a set of expressions, rather than a single one. This technique is combined with heuristics to achieve a beneficial variable order. Additionally, ENFrame provides several approximation algorithms with error guarantees. These algorithms prune branches of the decision tree which contribute a limited probability mass (Section 5.4). To further speed up the compilation process, these algorithms can utilise multiple CPU cores for parallel probability computation.

5.3 Exact probability computation

When computing the probabilities for the compilation targets in an event network using a decision tree, the partially evaluated expressions $\Phi|_x$ and $\Phi|_{\neg x}$ are not materialised for the compilation targets.

Instead, ENFrame uses a technique called *masking*.

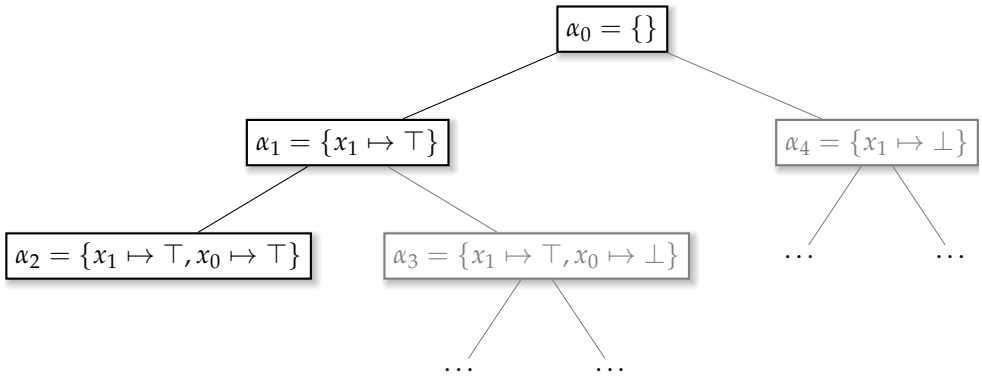
A masked network is a network under a (possibly partial) assignment α of the Boolean random variables in the input. Since each node in the network is an event, its mask under α is exactly the value of α applied to the event. The mask value can be Boolean, real, a vector, or undetermined, where the latter implies that α is partial and needs to be extended by more variable assignments in order to determine the event's mask value.

Every time a new valuation α is created by deciding on a Boolean variable x_i, its value $\alpha(x) \in \mathbb{B}$ is *propagated* into the event network using a DFS procedure starting at node x_i. From there, x_i's *mask* is propagated to its parent events, which might be masked in turn. The semantics of the events (as described in Figure 4.3) determine whether and how masks of child nodes are propagated. If the propagation procedure reaches a node that has already been masked, the DFS backtracks to the next unmasked node in the network, if any.

Whenever a compilation target is reached and masked by a mask propagation of a branch α, the probability (initially 0) of that mask value is *increased* by $\Pr[\alpha]$. For example, if a Boolean compilation target is satisfied by a valuation α, the probability of that event being true is increased by $\Pr[\alpha]$. If a mask propagation results in the masking of all compilation targets, the current decision tree branch α is considered finished and not extended any further. Otherwise, a next variable is selected, α is split into two new branches, and the masking process continues.

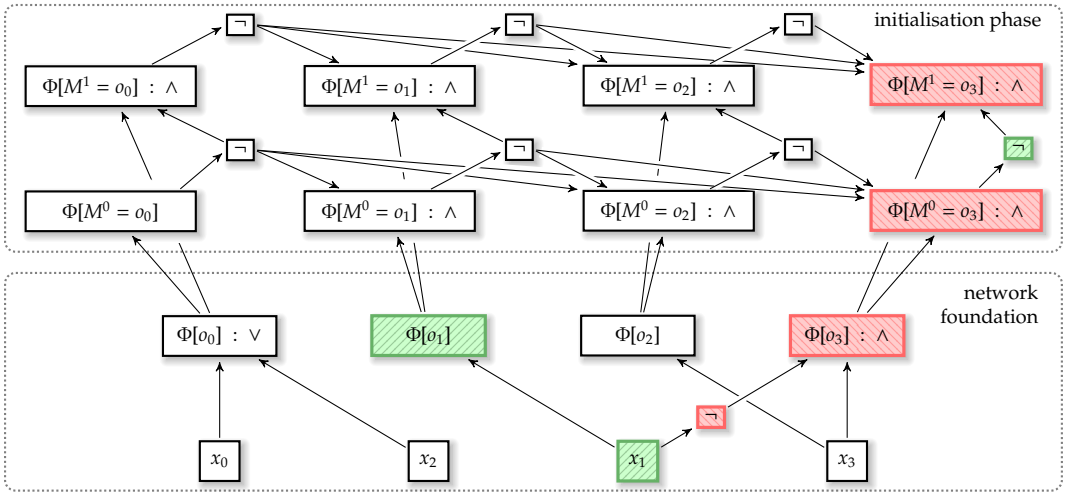
Example 5.5: Masking in an event network

Recall the event network from Example 5.2, consisting of the foundation and initialisation phase for k-medoids clustering. Assume that, for this example, the medoid selection events are compilation targets, the probabilities of which need computing. This example illustrates the consecutive masking for two valuations α_1 and α_2 from the following decision tree:



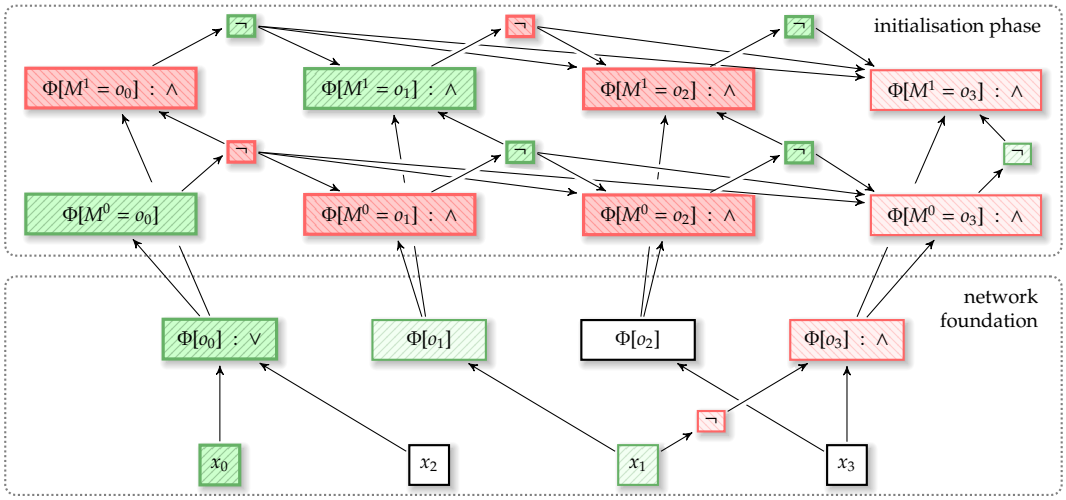
Variable x_1 is the first to be expanded on in the decision tree (node α_1). As a result, a mask propagation for $x_1 = \top$ is performed. Recall that the decision tree is explored depth-

first. Therefore, the propagation for $x_1 = \bot$ will take place after the left subtree has been investigated. The following network shows the result of the bottom-up propagation of the mask $x_1 = \top$:



The *true* mask of x_1 propagates to $\Phi[o_1]$ (masking it *true*), and will further propagate to parents $\Phi\left[M^0 = o_1\right]$ and $\Phi\left[M^1 = o_1\right]$. These conjunctions have multiple children which are yet to be masked, and therefore remain unmasked. Subsequently, x_1's *true* mask is propagated to its second parent node: a negation. This node passes on its *false* mask to its only parent $\Phi[o_3]$, which is a conjunction with x_3. The *false* mask yields a conjunction that evaluates to *false* (regardless of the mask of the other operands), hence $\Phi[o_3]$ is masked false. As a consequence, o_3 can not be an initial medoid to any cluster in this possible world.

The depth-first exploration of the decision tree continues to construct a new branch α_2 in the tree by deciding on x_0. The mask propagation for $x_0 = \top$ is illustrated below:



The propagation of x_0's *true* mask triggers a wave of new masks in the network, which results in all initial medoid nodes being masked. It is established that, in the possible world constituted by valuation $\alpha_2 = \{x_0 \mapsto \mathit{true},\ x_1 \mapsto \mathit{true}\}$, objects o_0 and o_1 are the respective initial medoids of the clusters.

Note how there is no need to further grow the branch α_2 of the decision tree: all compilation targets (*i.e.*, the initial medoid nodes) have been masked, the value of x_2 and x_3 (and, by extension, the existence of o_2) is irrelevant. From here, the depth-first exploration of the decision tree continues at $\alpha_3 = \{x_1 \mapsto \top, x_0 \mapsto \bot\}$.

Every mask propagation starts at Boolean nodes at the bottom of the event network, and potentially ends at the compilation targets. In ENFrame, all node masks for a certain valuation α are stored in a single data structure. The masks of these nodes and intermediate Boolean nodes can have three values: *unknown*, *true*, and *false*. Network nodes that represent monotonic real-valued events (*e.g.*, conditional values and sums of non-negative numbers) are masked using a lower and upper bound value which converge as more variables are decided on. Comparison events (*e.g.*, $r_1 < r_2$, where r_1, r_2 are real-valued events) can compare two real-valued events, and can decide the (Boolean) result of the comparison based on the converging bounds of the operands.

The structure of the event network itself remains unaltered during the masking process: masking acts like a literal *mask*, *i.e.* a layer being projected on top of the network. The pseudocode for the algorithm for exact probability computation of event networks is listed in Algorithm 5.1. The logic that determines the behaviour of nodes in the masking process is illustrated by the pseudocode in Algorithm 5.2. Although the presented clustering and classification user programs result in Boolean compilation targets, the pseudocode presented in Algorithm 5.1 can infer (finite) categorical probability distributions. The Bernoulli distribution induced by Boolean events in the presented event programs is a special case of such distributions.

The compilation of an event network D starts in the COMPILE procedure, which initialises the masks $M[v_i]$ of network nodes v_i to their initial *unknown* value, and sets the initial probabilities of the values in the range of the compilation targets to 0. The recursive DFS procedure is then called for network D, with masks M (all *unknown*) and the empty valuation $\alpha = \{\}$.

The DFS procedure checks whether all compilation targets have been reached. If this is the case the recursive DFS call returns, as there is no need to select a next variable to branch on. Otherwise, the procedure continues to select a next variable $x \in \mathbf{X}$ to branch on, constructs two new masks

```
1  COMPILE(network D)
2      foreach v_i ∈ D do M[v_i].val ← unknown        ▷ initialise (empty) masks for nodes in the network
3      foreach t_i ∈ targets(D) do
4          foreach y ∈ t_i.range do                   ▷ initialise zero probabilities for values y in range of target t_i
5              t_i[y].prob ← 0
6      DFS(D, M, {})                                  ▷ empty DFS branch α = {}, Pr(α) = 1
7
8  DFS(network D, masks M, branch α)
9      if ∀t_i ∈ targets(D) : M[t_i] ≠ unknown) then
10         return                                     ▷ all compilation targets reached, backtrack DFS
11
12     x ← nextVariable(α)                            ▷ abstract method for selecting next variable
13     M_l = M,   M_r = M                             ▷ copy mask M for true and false variable x
14     M_l[x] = true,   M_r[x] = false
15
16     M_l ← PROPAGATEMASK(D, M_l, x, NULL, Pr[α])    ▷ call masking procedure to propagate masks for x = true
17     M_r ← PROPAGATEMASK(D, M_r, x, NULL, Pr[α])    ▷ call masking procedure to propagate masks for x = false
18
19     DFS(D, M_l, [α, x ↦ true])                     ▷ DFS left branch
20     DFS(D, M_r, [α, x ↦ false])                    ▷ DFS right branch
```

Algorithm 5.1: Algorithm for exact probability computation of an event network

M_l (for left branch) and M_r (right branch), and subsequently sets x to *true* in the masks for the left branch and to *false* for the right branch.

Both valuations of x (on both branches) are then propagated into the event network by calling the recursive PROPAGATEMASK procedure with the following parameters:

D	the event network;
M_l or M_r	the masks of the nodes of the network (includes the masked variable x);
x	the variable that is the first target of the propagation;
NULL	the node that is the source of the propagation: NULL for the first step, but non-NULL in later (recursive) calls to PROPAGATEMASK;
$\Pr[\alpha]$	the probability of the valuation α that is being propagated

```
1  PropagateMask(network D, masks M, node v, child c, prob p)
2      M[v].val = unknown
3      switch v.nodetype do
4          case ¬   : M[v].val ← not M[c].val                                   ▷ M[c] ∈ 𝔹, M[v] ∈ 𝔹
5          case ∧                                                               ▷ M[c] ∈ 𝔹, M[v] ∈ 𝔹
6              if M[c].val = false then M[v].val ← false
7              else if (∀c_i ∈ v.children : M[c_i].val = true) then M[v].val ← true
8          case ∨                                                               ▷ M[c] ∈ 𝔹, M[v] ∈ 𝔹
9              if M[c].val = true then M[v].val ← true
10             else if (∀c_i ∈ v.children : M[c_i].val = false) then M[v].val ← false
11         case ⊗                                                               ▷ c-value: M[c] ∈ 𝔹, M[v] ∈ ℝ
12             if M[c].val = true then M[v].lower ← v.val                       ▷ update lower and upper bound of c-value (ℝ)
13             M[v].upper ← M[v].lower
14         case Σ                                                               ▷ sum of c-values: M[c] ∈ ℝ, M[v] ∈ ℝ
15             M[v].lower ← M[v].lower + M[c].lower                             ▷ l. bound possibly increases with new child (ℝ) lower bound
16             M[v].upper ← M[v].upper − (c.val − M[c].lower)                   ▷ possible u. bound decrease due to child l. bound < c.val
17         case <                                                               ▷ M[c] ∈ ℝ, M[v] ∈ 𝔹
18             if M[v.left].upper < M[v.right].lower then M[v] ← true
19             else if M[v.left].lower ≥ M[v.right].upper then M[v] ← false
20         case >
21             ...
22     if M[v] ∈ ℝ and M[v].lower = M[v].upper then M[v].val = M[v].lower
23
24     if (M[v] ∈ 𝔹 and M[v].val ≠ unknown) or M[v] ∈ ℝ then                    ▷ if mask of node v has changed
25         if v ∈ targets(D) and M[v].val ≠ unknown then                        ▷ if v is a compilation target with final value
26             v[M[v].val].prob ← v[M[v].val].prob + p                          ▷ increase prob. of this possible value of event v
27
28         foreach p_i ∈ v.parents do                                           ▷ propagate mask to yet unmasked parents of v
29             if M[p_i].val = unknown then M ←PropagateMask(D, M, p_i, v, p)
30     return M
```

Algorithm 5.2: Algorithm for masking of nodes in the event network

The PropagateMask procedure (Algorithm 5.2) determines whether a node v can be masked after a mask propagation from child c, and based on the mask of one or more of its children.

The logic in this pseudocode is directly linked to the probabilistic semantics of events, as presented in Section 4.3. For example: if v is a negation (*i.e.*, $v.nodetype = \neg$), then the mask $M[v]$ of v becomes the negation of the mask $M[c]$ of its Boolean child c. The pseudocode comments indicate assertions on the ranges of events. For example, a conditional value ($v.nodetype = \otimes$) expects a child with a Boolean mask ($M[c] \in \mathbb{B}$), and will pass on a real-valued mask (or the bounds thereof).

If the mask of a node v changes (line 24) as a result of a mask propagation, then two actions need to be performed. Firstly, if v is a compilation target and if v's value is established by the mask, the probability of that value is increased by the probability p of the current possible world (induced by valuation α of the branch in the decision tree that initiated the mask propagation) on line 26. Secondly, the mask of node v needs to be propagated to all parent nodes p_i that do not yet have a mask (line 29). The result of the probability computation is stored in $v[x]$.prob for all values y of compilation targets v.

During the compilation, the leaf nodes of every branch of the decision tree will mask all compilation targets in the event network.

Proposition 5.1: Soundness of decomposition (part 1)

Assume l is the leaf node of a branch of the decision tree, constituting a (possibly incomplete) valuation $\alpha = \{x_a \mapsto \alpha(x_a), \ldots\}$. Then, α will mask all compilation targets $t_i \in T$ (where T is the set of nodes that have been marked a compilation target, and t_i is an expression over Boolean random variables $x \in \mathbf{X}$).

Proof. Consider the situation in which l is a leaf node but does not mask one or more $t_i \in T$. Then, one of two situations would occur:

- α is not a complete valuation, and can therefore be expanded using some variable $x_b \in \mathbf{X}$, $x_i \notin \alpha$. Consequently, l_α is not a leaf node in the decision tree.
- α is a complete valuation, but does not mask some $t_i \in T$. If there exists an event t_i that cannot be masked by providing a valuation to all $x_a \in \mathbf{X}$, then t_i relies on one or more external unknown variables $x_b \notin \mathbf{X}$, and is therefore not an expression over Boolean random variables $x \in \mathbf{X}$.

Both of which result in a contradiction. □

The compilation procedure terminates if the DFS search on the decision tree has been completed. At that stage, all possible worlds will have been investigated, and the probability distributions of all compilation targets $t_i \in T$ will have been computed.

PROPOSITION 5.2: Soundness of decomposition (part 2)

The decomposition technique computes the exact probability distribution of compilation targets $t_i \in T$.

PROOF (SKETCH). By exploring all possible valuations of variables x_i in the input set of variables $\mathbf{X}$, the decision tree examines all possible worlds. Some branches of the decision tree may be cut short if a partial valuation α (un)satisfies all event expressions. By aggregating (summing) the probability mass of all worlds for every possible value of target t_i, the exact probability distribution of t_i is obtained.

A direct consequence of Proposition 5.2 is that the probability distributions computed for variables in a network generated by a translated user program are equivalent to the distributions obtained by running the deterministic user program in every of the exponentially many possible worlds. This result will be experimentally confirmed in Chapter 6.

PROPOSITION 5.3: Computational complexity of decomposition and masking

The decision tree constructed using the decomposition process contains at most $O(2^v)$ nodes for an event network built using v independent Boolean random variables. The size of the event network ($|V|$ nodes and $|E|$ edges) depends the number of variables in the user program, their dependencies (correlations). Events in loops that can be *folded* (see Section 5.3.3) are only stored once, events in non-foldable loops need to be represented multiple times. A single mask propagation takes $O(|E|)$ time and is performed exactly once for every node in the decision tree, yielding an overall computational complexity of $O(2^v \cdot |E|)$ for the decomposition process (including masking).

At any one time, only a single branch of the decision tree is kept in memory, yielding a $O(v)$ space complexity. For every node on that branch, one mask data structure needs to be kept in memory, which has a $O(|V| + |E|)$ space complexity. Therefore, the total space complexity of the decomposition process (including masking) is $O(v \cdot (|V| + |E|))$.

5.3.1 Variable order

The order in which the variables are expanded in the decision tree has a significant influence on the depth of the leaf nodes [Rud93] (refer to Example 5.5 for an example). The problem of obtaining an optimal variable order has been shown to be NP-complete [BW96], and [Sie02] shows that even approximating a variable order within a constant error factor is NP-complete.

Significant efforts have been made to construct heuristics for finding a favourable variable order for propositional expressions [RK08]. Many such heuristics rely on topological properties of the expression (circuit) to define a distance measure on variables, and subsequently try to order variables in such a way that variables at a short distance from each other are grouped together in the decision tree (*e.g.*, [AMS04; Dre96; FFM93]). Other methods attempt to identify the most influential variables for early processing in the decision tree (*e.g.*, [MWBS88; RK08]).

None of these heuristics are suitable to operate on event networks that feature non-propositional constructs, but the concepts introduced in the literature remain valid. ENFrame employs a heuristic which determines the most influential variables: those variables whose valuation will affect the largest number of compilation targets. By deciding on more influential variables first, ENFrame attempts to reduce the height of the decision tree.

Whenever a node v is reached by the PropagateMask procedure, but cannot be masked due to an insufficient partial valuation α, the node will initiate a reverse (*i.e.*, top-down) DFS search via its unmasked child nodes and the rest of the network to *promote* one or more variables $x \in \mathbf{X}$. At the foundation of the event network, this propagation results in a series of votes for a variable and its valuation. The variable with the largest number of votes is considered the most influential and will become the next variable in the decision tree, expanding the partial valuation α.

5.3.2 Certain data

Not all probabilistic data sets consist of purely probabilistic data. In some scenarios, uncertain data points occur together with traditional (*i.e.*, certain) data points. As will be shown in the experimantal evaluation (Chapter 6), depending on the user program and the degree of data (un)certainty, computation can be sped up significantly as the certain data points potentially result in an improved decidability of nodes in the event network. For example, k-medoids clustering elects a new cluster medoid by searching for a cluster member with the minimal total distance sum. If a number of objects in the cluster are certain, their contribution to the distance sums does not

depend on different variable valuations inflicted by the decision tree throughout the compilation process. Therefore, it is possible to elect a medoid earlier, *i.e.*, at a lower depth of the decision tree.

Certain data points can be annotated using events that only depend on network nodes which represent the constants *true* and *false*: $\top$ and $\bot$. The valuations of these two constants are propagated before the first variable is selected in the decision tree. As a result, the computational complexity of probability computation in a network built on top of exclusively certain data points is linear in the number of edges in the network: two mask propagations, each with complexity $O(|E|)$, where $|E|$ denotes the number of edges in the event network. The resulting 'probabilities' are all zero or one, effectively yielding a deterministic output.

5.3.3 Encoding iterations

User programs for data mining algorithms (and the resulting event programs) often contain numerous loops that construct new variables and events. Event networks offer two ways of storing such loops and the events therein. The default encoding is *unfolded*: all events that occur inside loops are explicitly stored as distinct nodes in the event network. For example, a `for`-loop that iterates over i algorithm iterations (*e.g.*, in k-means or k-medoids clustering) and generates m events in every iteration will construct $i \cdot m$ events in the resulting event network.

Alternatively, if the code contains a bounded-range loop of which the body does not depend on loop variable i, the repetitive events can be stored using a *folded* encoding. All events in the body of such a `for`-loop are captured in a single group of nodes in the event network. Compilation of event networks with such *iterative* groups requires looping during the probability computation process.

The pseudocode for probability computation (as presented in Algorithm 5.1 and Algorithm 5.2) assumes unfolded networks, and few minor modifications are required to make these procedures aware of folded event networks. Firstly, the masks data structure M becomes two-dimensional to allow for storing masks for all nodes v in any loop iteration t: $M[t][v]$. This value t becomes an additional parameter to the PropagateMask procedure. Secondly, to facilitate the transition between different iterations of a loop, an extra node type is introduced. This node requires the following additional code in the PropagateMask procedure (Algorithm 5.2):

```
case ∇                                  ▷ loop node
    M[t + 1][c] ← M[t][v]               ▷ carry mask over to next iteration
    t ← t + 1                           ▷ increase iteration counter
```

Lastly, a compilation target in an iterative group is only considered *reached* when the group has reached its final iteration.

For iterative programs, a folded event network results in a smaller memory footprint for the event network. The computational complexity of probability computation remains the same.

5.4 Approximation algorithms

Exact probability computation is a computationally expensive task. Depending on the structure of the correlations in the input data and the program under consideration, an algorithm will spend significant time exploring the decision tree consisting of a number of nodes (partial variable valuations) that grows exponentially in the number of variables in the input data.

Algorithms for approximate probability calculation can counteract the hardness of the problem and potentially result in a more efficient computation, at the cost of precision. Various approximation techniques have been proposed in the literature. Within the field of probabilistic databases, methods that look for approximable patterns in lineage expressions have been shown to be effective [OHK10; FHO13]. At the same time, research in artificial intelligence has approached the related model counting (#SAT) problem using Monte Carlo sampling, *e.g.* [WS05]. Although these methods have been shown to be effective in many cases, their computational complexity is prohibitive due to their dependency on truly uniform sampling [GSS06; GSS09]. Additionally, Monte Carlo sampling methods only provide probabilistic guarantees (*i.e.*, only with probability $p < 1$ is a sampling result an ε-approximation, where p is preset), and do not exploit the structure of events [OW12].

ENFrame features a number of approximation algorithms which approximate the probabilities of compilation targets with an absolute error guarantee ε using pruning techniques. Rather than computing an exact probability p for every compilation target, these algorithms compute a lower bound p^l and upper bound p^u, based on which an approximate probability $\hat{p}$ is calculated. By accepting a lower and upper bound rather than an exact probability, less time needs to be spent on exploring the decision tree, which should result in a speed-up. This hypothesis is empirically tested in Chapter 6.

Definition 5.2: Absolute ε-approximation for Boolean events [OHK10]

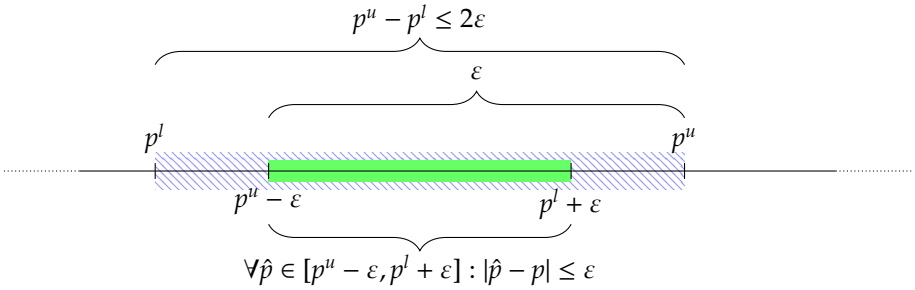
An approximated probability $\hat{p}$ of some Boolean event Φ is referred to as an absolute ε-approximation of an exact probability p if:

$$p - \varepsilon \leq \hat{p} \leq p + \varepsilon$$

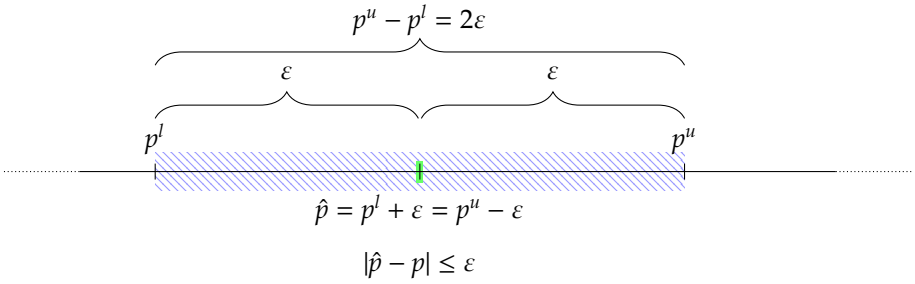
Definition 5.2 has the following implications:

- if $p^l \leq p^u$ are any lower and upper bounds (respectively) of the exact probability p, such that $p^u - \varepsilon \leq p \leq p^l + \varepsilon$, then any value $\hat{p}$ in range $[p^u - \varepsilon, p^l + \varepsilon]$ is an absolute ε-approximation of p.
- if $p^u - p^l \leq 2 \cdot \varepsilon$, then $p^u - \varepsilon \leq p^l + \varepsilon$, and any value $\hat{p}$ in range $[p^u - \varepsilon, p^l + \varepsilon]$ is an absolute ε-approximation of p.

The value $2 \cdot \varepsilon$ is referred to as the *error budget*. Definition 5.2 can be visualised as follows:



The (hashed) light blue rectangle that highlights the range $[p^l, p^u]$ represents the error budget, whereas any value within the inner green rectangle (highlighting range $[p^u - \varepsilon, p^l + \varepsilon]$) is an ε-approximation. Alternatively, when $p^u - p^l = 2 \cdot \varepsilon$:



Definition 5.2 defines an absolute ε-approximation for Boolean events, which induce a Bernoulli distribution over the range $\{true, false\}$. This approach can be extended to arbitrary finite categorical distributions.

Definition 5.3: Absolute ε-approximation for categorical probability distributions

Let Φ be an event over a finite range of n values $\{\phi_1, \ldots, \phi_n\}$, with probability distribution $P_\Phi = \{p_1, \ldots, p_n\}$. *I.e.*, $\Pr[\Phi = \phi_i] = p_i$ for $0 < i \leq n$.

The probability distribution $\hat{P}_\Phi = \{\hat{p}_1, \ldots \hat{p}_n\}$ is said to be an absolute ε-approximation of P_Φ if, for $0 < i \leq n$: $p_i - \varepsilon \leq \hat{p}_i \leq p_i + \varepsilon$.

Definition 5.2 and Definition 5.3 define the criteria for lower and upper bound probabilities to achieve an ε-approximation for the probability distribution of an event. By modifying the algorithm for exact probability computation to keep track of the lower and upper probability bounds of all values of compilation targets throughout the compilation process, it becomes possible to prune branches of the decision tree that do not need to be explored to establish an ε-approximation.

To achieve this, three additional pieces of information need to be kept throughout the compilation procedure for every compilation target $t_i \in T$:

- a lower probability bound $p^l_{t_i}[y]$ (initially 0) for each of the values y in the range of t_i;
- an upper probability bound $p^u_{t_i}[y]$ (initially 1) for each of the values in the range of t_i; and
- an error budget $B[t_i]$, initialised at $B[t_i] = 2\varepsilon$.

During the compilation procedure, the lower bound $p^l_{t_i}[y]$ of possible value y of compilation target t_i is increased by $\Pr[\alpha]$ whenever the target is reached and masked with value y during mask propagation of a valuation α (as before). However, upon establishing mask value y, the upper probability bounds $p^u_{t_i}[y']$ of all other values $y' \neq y$ in the range of t_i is decreased by that same probability. Following Definition 5.2, it is possible to terminate the compilation procedure as soon as the probability bounds for all values y of t_i have converged such that $p^u_{t_i}[y] - p^l_{t_i}[y] < 2 \cdot \varepsilon$ (for any desired $\varepsilon \geq 0$).

This approximation strategy is referred to as the POSTPONED approximation algorithm: the procedure constructs the decision tree as before, and spends the compilation targets' error budgets $B[t_i]$ by pruning the last (rightmost) branches of the decision tree. Observe how this procedure is identical to the algorithm for exact probability computation (Algorithm 5.1) when $\varepsilon = 0$.

The POSTPONED approximation algorithm works well for decision trees that are unbalanced, with relatively short branches on the left of the tree (the branches that are processed first), and longer branches at the very right of the tree. Such imbalance can be the result of the structure of correlations and the type of program represented by the event network. In those cases, the approximation algorithm can spend its error budget at the most difficult (longest) branches. However, if the decision tree is unbalanced in the opposite direction, the POSTPONED approximation algorithm is ineffective.

The second approximation algorithm designed for event networks is called GREEDY approximation. Rather than using the error budgets on the last branches, this approximation algorithm spends

the error budget as quickly as possible on the leftmost branches of the decision tree. This approximation algorithm tends to be effective on unbalanced trees with long branches on the left-hand side.

ENFrame's third approximation technique strikes a balance between the GREEDY and POSTPONED approximation algorithms by distributing the error budget evenly over the branches of the decision tree. This approach of dividing the error budget is called the BALANCED approximation algorithm and is illustrated in Figure 5.3. Starting with the root node α_0, every node α_i in the decision tree is provided with a share of the error budgets B of the compilation targets ($B_\alpha[t_i]$ indicates the budget for target t_i at node α). Every tree node then either prunes the subtree it is root of, or distributes the budget over its two children α' and α''. Half of the budget is assigned to the root of the left subtree which is processed first: $B_{\alpha'}[t_i] = \frac{B_\alpha[t_i]}{2}$. The residual error budgets $R_{\alpha'}[t_i]$ are returned to the parent α, which then assigns the other half of its budget, plus the residual budget $R_{\alpha'}[t_i]$ to the right subtree rooted by α'': $B_{\alpha''}[t_i] = \frac{B_\alpha[t_i]}{2} + R_{\alpha'}[t_i]$.

All three approximation algorithms set out above are part of the experimental evaluation presented in Chapter 6.

EXAMPLE 5.6: Error budgets for BALANCED approximation

Consider the decision tree and distribution of error budgets illustrated in Figure 5.3, which will be (partially) reused in this example. Furthermore, assume that there is one compilation target $\Phi[t_i] = (x_0 \wedge x_1) \vee \neg x_0$, and:

$$\Pr[x_0] = 0.9 \qquad \Pr[x_1] = 0.05 \qquad \varepsilon = 0.1$$

The lower and upper probability bounds of t_i are initialised to $p^l_{t_i}[true] = 0$ and $p^u_{t_i}[true] = 1$, respectively. The compilation procedure then constructs the nodes of the decision tree in the following fashion:

α_0: the root of the decision tree with the empty partial valuation $v_0 = \{\}$. The probability mass in the (sub)tree rooted by this node is, by definition, $\Pr[\alpha_0] = 1$. Budget $B_{\alpha_0}[t_i] = 2\varepsilon = 0.2$ is *not* sufficient to prune this subtree. Partial valuation $\alpha_0 = \{\}$ does not satisfy $\Phi[t_i]$ either. Expand on x_0:

- α_1: budget $B_{\alpha_1}[t_i] = \frac{B_{\alpha_0}[t_i]}{2} = 0.1$ is *not* sufficient to prune the subtree rooted by α_1, which contains a probability mass of $\Pr[\alpha_1] = \Pr[x_0] = 0.9$. The partial valuation does not satisfy the target. Decide on x_1:

α_2: budget $B_{\alpha_2}[t_i] = \frac{B_{\alpha_1}[t_i]}{2} = 0.05$ *is* sufficient to prune the subtree rooted by α_2, which contains a probability mass of $\Pr[\alpha_2] = \Pr[x_0 \wedge x_1] = 0.9 \cdot 0.05 = 0.045$. The branch is pruned, and the residual budget $R_{\alpha_2}[t_i] = B_{\alpha_2}[t_i] - \Pr[\alpha_2] = 0.005$ is returned.

α_3: budget $B_{\alpha_3}[t_i] = \frac{B_{\alpha_1}[t_i]}{2} + R_{\alpha_2} = 0.05 + 0.005 = 0.055$ is not sufficient to prune the subtree with mass $\Pr[\alpha_3] = 0.855$. The valuation α_3 unsatisfies $\Phi[t_i]$, therefore t_i's upper probability bound is decreased by $\Pr[\alpha_3]$, resulting in $p^u_{t_i}[true] = 1 - 0.855 = 0.145$. The lower bound remains unchanged: $p^l_{t_i}[true] = 0$. The residual error budget $R_{\alpha_3}[t_i] = B_{\alpha_3}[t_i] = 0.055$ is returned.

The residual error budget $R_{\alpha_1}[t_i]$ is equal to the residual budget of its right child: $R_{\alpha_1}[t_i] = R_{\alpha_3}[t_i] = 0.055$. Observe that this is equal to the original budget $B_{\alpha_1}[t_i]$ for the subtree rooted by α_1, minus the budget used in this subtree (0.045, for pruning α_2): $R_{\alpha_1}[t_i] = B_{\alpha_1}[t_i] - 0.045 = 0.1 - 0.045 = 0.055$.

α_4: budget $B_{\alpha_4}[t_i] = \frac{B_{\alpha_0}[t_i]}{2} + R_{\alpha_1}[t_i] = 0.1 + 0.055 = 0.155$ *is* sufficient to prune the subtree α_4 with mass $\Pr[\alpha_4] = \Pr[\neg x_0] = 0.1$. The branch is pruned, the residual error budget is irrelevant as this was the last branch of the tree. The compilation procedure ends here with probability bounds $p^l_{t_i}[true] = 0$ and $p^u_{t_i}[true] = 0.145$.

Following Definition 5.2, the probability bounds calculated by the DFS guarantee that any value in the interval $\left[p^u_{t_i}[true] - \varepsilon, p^l_{t_i}[true] + \varepsilon\right] = [0.045, 0.1]$ is an ε-approximation of the exact probability. In other words: the exact probability lies in the interval $[0, 0.2]$, which is correct: $\Pr[\Phi[t_i]] = 0.045 + 0.1 = 0.145$.

Note that the compilation procedure would actually have terminated after valuation α_3 resulted in a sufficiently tight pair of probability bounds: $p^u_{t_i}[true] - p^l_{t_i}[true] \leq 2 \cdot \varepsilon$.

The pseudocode of the BALANCED approximation algorithm is described in Algorithm 5.3, which is an extension of the algorithm for exact probability computation (Algorithm 5.1); the blue code and comments indicate new or changed code.

The probability bounds and error budget are initialised on lines 4–6: every possible value in the range of a compilation target is assigned a lower and upper probability bound, which are initially set to 0 and 1, respectively. Furthermore, every compilation target has a single error budget for approximation purposes.

```
1  COMPILE(network D, absolute error ε)
2      foreach v_i ∈ D do M[v_i].val ← unknown          ▷ initialise (empty) masks for nodes in the network
3      foreach t_i ∈ targets(D) do
4          foreach y ∈ t_i.range do                     ▷ initialise probability bounds for values y in range of target t_i
5              t_i[y].problower ← 0,   t_i[y].probupper ← 1
6          B[t_i] ← 2ε                                  ▷ error budget for compilation target t_i
7      DFS(D, M, { }, B)                                ▷ empty DFS branch α = { }, Pr(α) = 1
8
9  DFS(network D, masks M, branch α, error budgets B)
10     if ∀t_i ∈ targets(D) :  M[t_i] ≠ unknown) then
11         return B                                     ▷ all compilation targets reached, backtrack DFS (return residual error budget)
12
13     if ∀t_i ∈ targets(D),  ∃y ∈ t_i.range :  t_i[y].probupper − t_i[y].problower ≤ 2ε then
14         return B                                     ▷ probability of all compilation targets approximated – compilation done
15     if ∀t_i ∈ targets(D) :  B[t_i] ≥ Pr[α] then      ▷ sufficient error budget to trim branch α
16         foreach t_i ∈ T do                           ▷ decrease available budget for unmasked targets t_i
17             if M[t_i] = unknown then B'[t_i] ← B[t_i] − Pr[α]
18         return B'                                    ▷ return updated error budgets
19
20     x ← nextVariable(α)                              ▷ abstract method for selecting next variable
21     M_l = M,   M_r = M                               ▷ copy mask M for true and false variable x
22     M_l[x] = true,   M_r[x] = false
23
24     M_l ← PROPAGATEMASK(D, M_l, x, NULL, Pr[α])      ▷ call masking procedure to propagate masks for x = true
25     M_r ← PROPAGATEMASK(D, M_r, x, NULL, Pr[α])      ▷ call masking procedure to propagate masks for x = false
26
27     foreach t_i ∈ targets(D) do B_l[t_i] ← B[t_i]/2  ▷ error budget B'_l for left DFS branch
28     R_l ← DFS(D, M_l, [α, x ↦ true], B_l)            ▷ DFS left branch, storing the residual error budget R_l
29
30     foreach t_i ∈ targets(D) do B_r[t_i] ← B[t_i]/2 + R_l[t_i]   ▷ error budget B_r for right DFS branch
31     R_r ← DFS(D, M_r, [α, x ↦ false], B_r)           ▷ DFS right branch, storing the residual error budget R_r
32
33     return R_r
```

Algorithm 5.3: Algorithm for approximate probability computation of an event network. The blue parts are new or changed compared to Algorithm 5.1

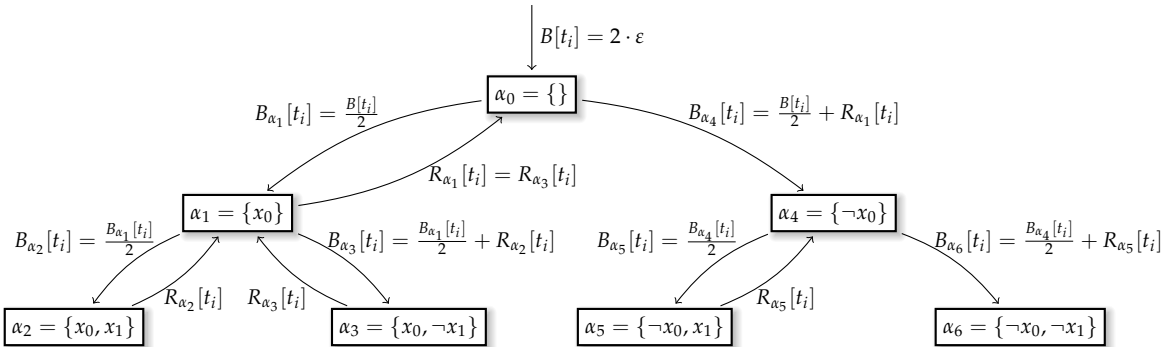


Figure 5.3: Example of error budget distribution in a decision tree for two variables in the BALANCED approximation algorithm. Forward (downwards) edges indicate budget assignment to a child node, back edges represent return of residual error budget to parent nodes.

After the initialisation, the recursive DFS procedure is called (specifying the error budgets). The DFS procedure is updated to always return the residual budget of a branch in the decision tree. If all targets have been approximated (*i.e.*, the difference between the lower and upper probability bounds is smaller than 2ε), the DFS returns (line 14). Otherwise, the procedure continues to check whether the compilation targets' error budgets are sufficient to prune the current branch of the decision tree (line 15). A branch can only be pruned if all unmasked compilation targets have budget available; otherwise, the resulting lower and upper bounds cannot be guaranteed to be tight enough to guarantee an ε-approximation. If the budgets are sufficient, they are reduced by the probability of the branch that is being pruned (line 17). Subsequently, the DFS call for branch α stops and returns the updated residual error budgets.

Note that only a single error budget is used for all values in the range of a compilation target t_i. Although the lower and upper probability bounds can be different for every value in t_i's range, the *difference* between the lower and upper bounds is identical for all values $y \in t_i.range$.

The procedure then selects a new variable to expand on, and creates the masks for the new decision tree branches. The valuation of the newly selected variable is subsequently propagated into the event network, storing the mask results in M_l and M_r. The PROPAGATEMASK procedure is largely identical to the one used for exact probability computation, with only minor changes in the code for storing the probabilities of compilation targets. The new code is provided in Algorithm 5.4.

On line 27 of Algorithm 5.3, the error budgets for the newly created left-hand branch of the decision tree are calculated: the budget for every compilation target is half the budget available to the current DFS call. A recursive call to DFS is made, specifying the network D, the newly created masks M_l, the new valuation, and the available error budget B_l. The DFS call for the left branch will

```
1  PropagateMask(network D, masks M, node v, child c, prob p)
2      ...
24     if (M[v] ∈ 𝔹 and M[v].val ≠ unknown) or M[v] ∈ ℝ then        ▷ if mask of node v has changed
25         if v ∈ targets(D) and M[v].val ≠ unknown then            ▷ if v is a compilation target with final value
26             v[M[v].val].problower ← v[M[v].val].problower + p    ▷ increase lower prob. bound of possible value
27             foreach x ∈ v.range do                               ▷ decrease upper prob. bounds of other values in v's range
28                 if x ≠ M[v].val then v[x].probupper ← v[x].probupper − p
29         ...
30     ...
```

Algorithm 5.4: Updated pseudocode for masking of nodes in the event network for ε-approximations (see Algorithm 5.2 for full listing).

return the residual error budgets R_l, which are used to establish the budgets B_r for the right-hand branch (line 30). The resulting residual error budget from a DFS call for the right branch is then returned.

After the computation finishes, the lower and upper probability bounds for every value y in the range of every compilation target $t_i \in T$ are available in $t_i[y].problower$ and $t_i[y].probupper$. Furthermore, $t_i[y].probupper - t_i[y].problower \leq 2\varepsilon$. As follows from Definition 5.3, every value $\hat{p}_y \in \Big[t_i[y].probupper - \varepsilon,\ t_i[y].problower + \varepsilon\Big]$ constitutes an ε-approximation of the exact probability $\Pr\Big[\Phi[t_i = y]\Big]$.

The performance of all three approximation algorithms is one of the main subjects of the experimental evaluation in Chapter 6.

5.5 Concurrent probability computation

The variable order heuristics and approximation algorithms all aim to reduce the time spent on probability computation. With the introduction of affordable multi-core processors and frameworks for distributed computing (*e.g.*, Apache Hadoop and MapReduce [DG04]), computation time can be further reduced through parallel probability computation. ENFrame introduces an algorithm for concurrent compilation of event networks using a shared-memory approach, which includes support for ε-approximations.

One of the major challenges when introducing concurrency to algorithms is dealing with thread synchronisation [Dij65; Pac11]: critical sections of code that require mutual exclusion amongst

the threads. Probability computation through decomposition is fundamentally very suitable for concurrent computation: every pair of branches (α_l, α_r) that are constructed by a node α can be computed independently of each other. However, the approximation algorithms introduced in Section 5.4 do introduce a dependency between such branches: the error budgets $B_{\alpha_r}[t_i]$ of α's right branch depend on the residual error budgets $R_{\alpha_l}[t_i]$ of the left branch. Synchronisation of the two threads that process a left and right subtree would completely undo any possible gain from multi-core processing, as threads would end up spending most time waiting for each other. By modifying the way error budgets are calculated, the dependency between threads can be broken – this gives rise to the BALANCED-C algorithm for concurrent approximate probability computation.

Let $\rho[t_i]$ denote the total *remaining* error budget for target t_i, *i.e.*, initially $\rho[t_i] = B[t_i] = 2\varepsilon$. The value of $\rho[t_i]$ is reduced whenever some of t_i's error budget is used ($0 \leq \rho[t_i] \leq B[t_i]$), and is therefore monotonically decreasing. Let π denote the progress of the exploration of the decision tree in terms of the fraction of leaf nodes that have been considered (pruned or visited, $0 \leq \pi \leq 1$). The value of π is updated throughout the compilation procedure, and is monotonically increasing. Lastly, let $d(\alpha)$ denote the depth of α in the tree. The size of the subtree rooted by α is then defined as:

$$\mathit{treesize}(\alpha) = 2^{|\mathbf{X}|-d(\alpha)} \qquad (= \text{number of leaves in the full subtree rooted by } \alpha)$$

The number of leaves that still need considering depends on the progress π, and is defined as:

$$\lambda = (1-\pi) \cdot 2^{|\mathbf{X}|}$$

Furthermore, σ_α expresses the size of the subtree of α relative to λ:

$$\sigma_\alpha = \frac{\mathit{treesize}(\alpha)}{\lambda}$$

Finally, the error budget $B_\alpha[t_i]$ available to a decision tree node α for approximating t_i at any time during the compilation procedure is defined as follows:

$$\begin{aligned} B_\alpha[t_i] &= \rho[t_i] \cdot \sigma_\alpha \\ &= \rho[t_i] \cdot \frac{\mathit{treesize}(\alpha)}{\lambda} \quad = \quad \rho[t_i] \cdot \frac{2^{|\mathbf{X}|-d(\alpha)}}{(1-\pi)\cdot 2^{|\mathbf{X}|}} \quad = \quad \rho[t_i] \cdot \frac{2^{-d(\alpha)}}{1-\pi} \end{aligned}$$

PROPOSITION 5.4: Error budget calculation for concurrent probability computation

The BALANCED-C algorithm which employs an error budget of $B_\alpha[t_i] = \rho[t_i] \cdot \frac{2^{-d(\alpha)}}{1-\pi}$ at a node α in the decision tree for a compilation target t_i will compute the approximate probability distributions of all compilation targets with an absolute error of ε.

The available error budget $B_\alpha[t_i]$ for a decision tree node α is now a closed-form expression, which no longer directly depends on the result (and a residual error budget) of other branches of the decision tree. Instead, the remaining budget $\rho_\alpha[t_i]$ for a compilation target t_i is always divided evenly over the decision tree. A tree node α receives a budget that depends on (1) the potential maximum size of its subtree, (2) the progress of the compilation progress, and (3) the remaining error budget. After implementing Proposition 5.4, a few small critical sections remain:

- Verifying whether sufficient budget is available to prune a branch and subsequently updating the remaining error budgets ρ.
- Updating π whenever a branch is pruned or a leaf node is identified.
- Updating probability bounds during the masking process.
- Queueing and dequeueing processing jobs.

Every single one of these critical sections are limited to updating one (or a constant number) of variables, rather than containing computationally complex (*i.e.*, of complexity beyond $O(1)$) instructions.

The BALANCED-C probability computation algorithm employs a combination of the producer-consumer pattern [Ben90] and the recursive split pattern for concurrent computing (sometimes referred to as the 'divide and conquer' pattern, *e.g.* [MMS02]). Whenever a node α in the decision tree is split by some thread $\mathcal{T}_a$ to construct tree nodes α' and α'', one of the branches α' is pushed onto a queue as an independent job to be picked up by an idle consumer thread $\mathcal{T}_b$ (note that $\mathcal{T}_a = \mathcal{T}_b$ is possible). In the meantime, $\mathcal{T}_a$ continues down the other branch α''. A thread $\mathcal{T}_i$ becomes idle when a branch is not extended further because either a node (valuation) masks all compilation targets, or a branch is pruned. When a thread becomes idle, it tries to pick up the next job (for another branch) in the queue, or waits for one to be queued. This method for concurrent probability computation is suitable for any number of threads, and the thread-scalability is evaluated in Chapter 6.

This strategy for concurrent processing is illustrated in Figure 5.4. The coloured jobs $j_0, \ldots, j_4$ are listed in the order in which they are generated and illustrate how the tree is split up into tasks that

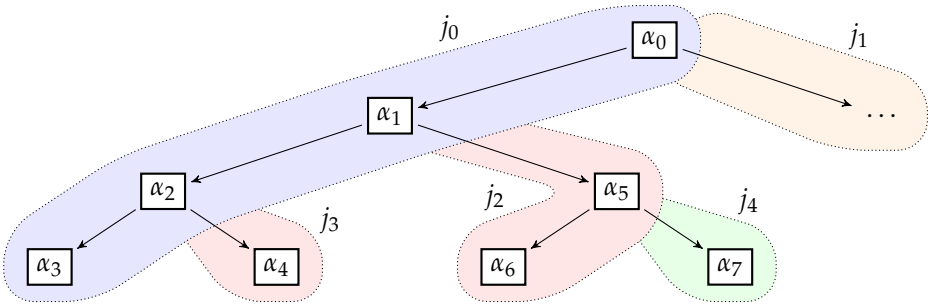


Figure 5.4: Concurrent processing of decision tree by BALANCED-C

can be processed independently. The threads can propagate masks into the event networks simultaneously; the progress and budget variables σ and ρ are updated by all threads in synchronised code blocks.

The algorithms for probability computation that were introduced in this chapter are subject of an experimental evaluation in Chapter 6.

Chapter 6

Experimental Evaluation

This chapter describes an experimental evaluation of the performance of ENFrame. The focus of this evaluation is a performance benchmark of the probability computation algorithms introduced in Chapter 5, used on the programs for k-medoids clustering and k-nearest neighbours classification.

The chapter does *not* aim to compare the quality of data mining algorithms implemented in ENFrame to other single-purpose algorithms for mining of uncertain data. Such a comparison would invariably yield discriminatory associations. Firstly, ENFrame does not aim to be an *algorithm* for processing or mining uncertain data: it is a *framework* that facilitates building such algorithms. From this perspective, ENFrame operates on the intersection of the fields of programming languages and databases. This observation is further discussed in Chapter 8.

Secondly, the framework's support for correlations and a solid foundation in possible worlds semantics make for a distorted comparison between algorithms written in ENFrame and single-purpose algorithms presented in the literature. Currently, all algorithms for probabilistic clustering and probabilistic classification rely on the (over)simplifying independence assumption (*e.g.*, [CCKN06; NKC+06], surveyed in [VRH+09] and Chapter 8). These algorithms might outperform implementations of data mining algorithms in ENFrame, at the cost of producing an output that is possibly inaccurate when compared to processing data in every possible world, due to a fundamental difference in probabilistic semantics (as explained in Section 2.2.4 and illustrated by Example 2.5).

This chapter does, however, introduce the first steps towards techniques for quality evaluation of probabilistic algorithms. These techniques are introduced in the first section, and are later used to empirically verify the claims regarding ε-approximation with error bounds, and to compare these

approximations to the quality of probability computation in the top-p most probable possible worlds. From there, after describing the experimental setup, this chapter presents general observations that apply to experimental results for probability computation for both k-medoids clustering and k-nn classification. After that, program-specific observations are presented, including experimental results using the newly introduced quality evaluators.

6.1 Quality evaluators for algorithms on probabilistic data

The literature describes many techniques for measuring the quality of clusterings and classifications of traditional *certain* data. For clustering, these methods can be divided into two categories:

- External evaluators, which compare an output (*e.g.* clustering) against an oracle that defines the desired output. Examples of external evaluators include the Rand measure [Ran71], and the F-measure.
- Internal evaluators which measure notions of *compactness* and *separation* of clusters. Examples include the Davies-Bouldin Index [DB79] and the Dunn Index [Dun73]).

The method used most often to evaluate the quality of a (supervised) classification algorithm is k-fold cross validation [DK82].

Probabilistic clustering and classification results are fundamentally different from traditional (certain) results: the output is a probability distribution over all possible results. One possible way of using the existing evaluators on a probabilistic result is by applying them in every possible world and aggregating over the exponentially many results, which is computationally prohibitive. Therefore, the techniques described in the literature are considered unfit for application in a probabilistic scenario. This thesis proposes a number of evaluators for probabilistic data algorithms.

Each of the proposed evaluators satisfies two requirements: (1) the ability to compare deterministic output from various types of probabilistic data algorithms, and (2) the ability to recognise and respect correlations in the input data. Further requirements apply to their respective application domains.

6.1.1 Evaluating probabilistic clusterings: PAPM

In addition to the general requirements for evaluators mentioned above, a couple of additional requirements apply to quality evaluators for probabilistic clusterings. A quality measure should

not make assumptions about the number of objects and clusters, as not all k clusters and n objects are guaranteed to exist in all possible worlds. Additionally, the evaluator should not rely on cluster labels, because some cluster j in some possible world w might contain the exact same set of objects as a cluster j' in another world w'.

Many of the (internal and external) clustering quality evaluators described in the literature are based on some measure of the assignment of objects to clusters. For example, such quality measures use the intra-cluster distances as a measure of separation, or the inter-cluster distances as a measure of compactness, or compare clusterings based on the cluster assignment of objects. These evaluators are incompatible with the requirements listed above. The pairwise assignment of objects to clusters, however, is a suitable measure: it is defined for both probabilistic and deterministic clusterings, it allows for taking correlations between objects into account, and it does not rely on the number of clusters or their labels.

The *Pairwise Assignment Probability Measure* (or: PAPM) is an external evaluator for uncertain clusterings, inspired by the Rand measure [Ran71]: it compares two clusterings, $\mathcal{D}_1$ and $\mathcal{D}_2$, using the pairwise assignment probabilities of objects to clusters. PAPM has two variants: PAPM_{avg} and PAPM_{max}, both of which are introduced below.

Definition 6.1: Pairwise Assignment Probability Measure

Let $\mathcal{D}_1$ and $\mathcal{D}_2$ be clustering results, with k clusters C_i $(0 \leq i < k)$ for a data set with n objects o_l $(0 \leq l < n)$. Then, the two PAPM variants are defined as follows:

$$\Pr_{\mathcal{D}}[o_a \sim o_b] = \sum_{0 \leq i < k} \Pr_{\mathcal{D}}[o_a \in C_i \wedge o_b \in C_i]$$

$$\Delta_{\mathcal{D}_1,\mathcal{D}_2}(o_a, o_b) = \left| \Pr_{\mathcal{D}_1}[o_a \sim o_b] - \Pr_{\mathcal{D}_2}[o_a \sim o_b] \right|$$

$$\text{PAPM}_{\text{avg}}(\mathcal{D}_1, \mathcal{D}_2) = \operatorname*{avg}_{o_a,o_b} \left(\frac{\beta^{\Delta_{\mathcal{D}_1,\mathcal{D}_2}(o_a,o_b)} - 1}{\beta - 1} \right)$$

$$\text{PAPM}_{\text{max}}(\mathcal{D}_1, \mathcal{D}_2) = \max_{o_a,o_b} \left(\frac{\beta^{\Delta_{\mathcal{D}_1,\mathcal{D}_2}(o_a,o_b)} - 1}{\beta - 1} \right)$$

Where $\beta > 1$ is a non-linear scaling constant.

The constant β has no influence when set close to 1, but for larger values it reduces the contribution of object pairs with small differences in assignment probabilities to the overall score. The effect of

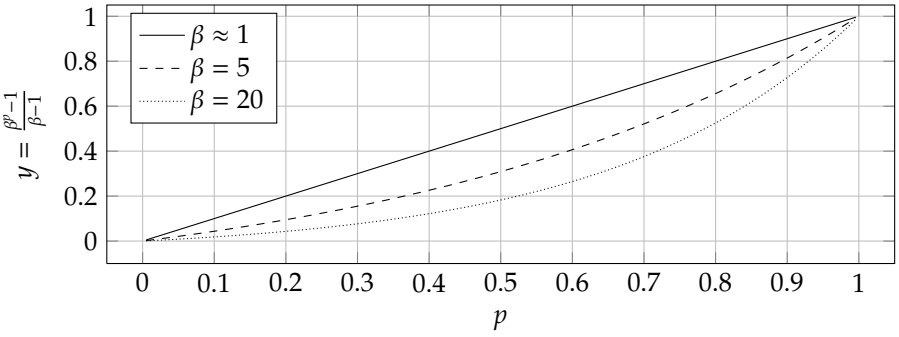


Figure 6.1: Effect of parameter β in PAPM and CPM calculations

this optional non-linear correction is illustrated in Figure 6.1. For experiments that are designed to illustrate the *difference* between two algorithms, a value of $\beta \approx 1$ is appropriate; for evaluating the performance of a heuristic against a ground truth, a larger value might be desired.

The value of both PAPM measures ranges from zero to one. A score close to zero indicates that two clustering results $\mathcal{D}_1$ and $\mathcal{D}_2$ are very similar: the average (in case of PAPM_{avg}) or maximum (PAPM_{max}) difference between pairwise assignment probabilities is small. A score close to 1 means that the clusterings are dissimilar, with large differences in pairwise assignment probabilities.

The term $\Pr[o_a \sim o_b]$ represents the pairwise assignment probability of two objects o_a and o_b to the same cluster. The events for the object-to-cluster assignment for one object o_a for multiple clusters C_i are mutually exclusive; consequently, the pairwise assignment probability can be computed as the sum of the events in which both objects o_a and o_b are assigned to the same cluster. The term $\Delta_{\mathcal{D}_1,\mathcal{D}_2}(o_a, o_b)$ represents the difference in pairwise assignment probability $\Pr[o_a \sim o_b]$ between two clusterings $\mathcal{D}_1$ and $\mathcal{D}_2$ regarding objects o_a and o_b.

Using PAPM, the quality of any probabilistic or deterministic clustering can be evaluated by comparing it to a known ground truth. In Section 6.4, explicitly clustering in every possible world is used as a ground truth for PAPM to (1) experimentally verify that ENFrame's algorithms for approximate probability computation respect their error bounds and (2) demonstrate ENFrame's superiority (in terms of both time and clustering quality) over a reference implementation that clusters in the most probable possible worlds.

6.1.2 Evaluating probabilistic classification: CPM

The *Classification Probability Measure* (or: CPM) is similar to the pairwise assignment probability measure: it compares two probabilistic classification results for an unlabelled object o_a using their respective probability distributions ($\mathcal{D}_1$ and $\mathcal{D}_2$) over all possible class assignments.

Definition 6.2: Classification Probability Measure (CPM)

Let $\mathcal{D}_1$ and $\mathcal{D}_2$ be classification results for one or more unlabelled objects $o_a, o_b, \ldots$, and let C_a denote the resulting class assignment of an object o_a. Furthermore, $\Pr_{\mathcal{D}_1}[C_a = j]$ denotes the probability of the event that o_a was assigned to class j in classification result $\mathcal{D}_1$. Then, the two CPM variants are defined as follows:

$$\mathrm{CPM}_{\mathrm{avg}}(\mathcal{D}_1, \mathcal{D}_2) = \frac{\beta^{\Delta^{\mathrm{avg}}_{\mathcal{D}_1,\mathcal{D}_2}(o_a)} - 1}{\beta - 1}$$

$$\mathrm{CPM}_{\mathrm{max}}(\mathcal{D}_1, \mathcal{D}_2) = \frac{\beta^{\Delta^{\mathrm{max}}_{\mathcal{D}_1,\mathcal{D}_2}(o_a)} - 1}{\beta - 1}$$

Where:

$$\Delta^{\mathrm{max}}_{\mathcal{D}_1,\mathcal{D}_2}(o_a) = \max_j \left| Pr_{\mathcal{D}_1}[C_a = j] - Pr_{\mathcal{D}_2}[C_a = j] \right|$$

$$\Delta^{\mathrm{avg}}_{\mathcal{D}_1,\mathcal{D}_2}(o_a) = \operatorname*{avg}_j \left| Pr_{\mathcal{D}_1}[C_a = j] - Pr_{\mathcal{D}_2}[C_a = j] \right|$$

The Classification Probability Measure can be used to evaluate a classification of an unlabelled object o_a. CPM compares the results of two classification algorithms using the probability distribution over class assignments, and reports the average or maximum difference, optionally corrected by constant β as detailed in Section 6.1.1.

6.2 Experimental setup

6.2.1 Data and correlations

The data used in the experimental evaluation was obtained from sensors in energy distribution networks, kindly provided by UK Power Networks[1] (UKPN) as part of the HiPerDNO FP7 research project [TWG+11]. The data describes network load and occurrences of *partial discharge* in energy distribution networks. Partial discharge is an electrical discharge that does not fully bridge the insulation between two conducting electrodes, and has recently been identified as one of the major causes of long-term degradation and eventual failure of cables. Figure 6.2 illustrates the effect of the phenomenon on the insulation of high-voltage cables.

[1] Formerly part of Électricité de France (EDF). http://www.ukpowernetworks.co.uk

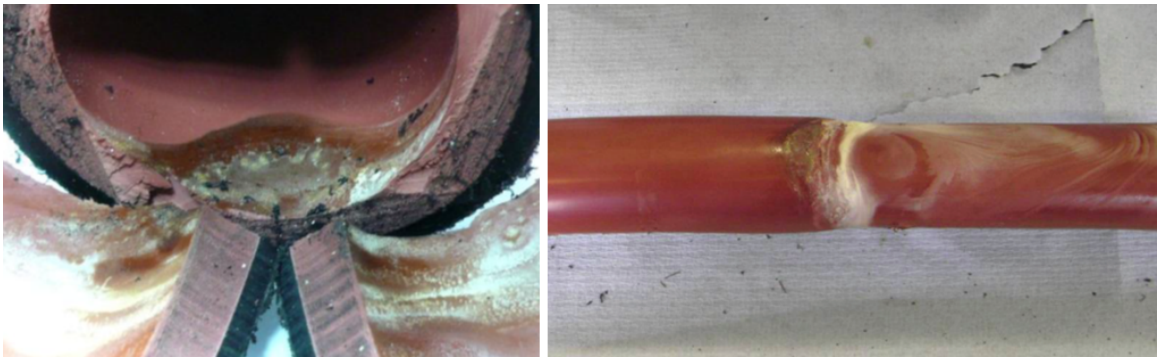


Figure 6.2: Effects of partial discharge (PD) on high-voltage cable insulation. On the left: cross-section of cable insulation affected by PD. On the right: exterior evidence of damage to the insulation. Courtesy of IPEC, [Mic07; ME11]

In order to minimise the (expensive) number of customer minutes lost, energy distribution network operators are currently deploying sensors to monitor partial discharge activity in the distribution network to be able to act preemptively [Mic07; ME11]. Unfortunately, monitoring partial discharge is not a straightforward task: the phenomenon is hard to detect, sensors often report spurious measurements and are prone to failure (as are the transmission channels).

The partial discharge occurrence count is aggregated over the duration of an hour, and subsequently paired with average network load readings during that time. The resulting data set consists of 1300 tuples with two attributes; this data is used to demonstrate the framework's ability to process non-synthetic data.

To experimentally evaluate ENFrame's ability to operate on various types of correlations, the data set has been augmented with three commonly occurring correlation patterns [ABS+06; SD07; SORK11]:

- *Positive* correlations, in which any two tuples o_a and o_b are either positively correlated or independent. Queries (especially joins and aggregates) often produce this type of correlation [KO08], as does time-series data from sensor networks [LC10]. In this correlation scheme, each event is a disjunction of l distinct positive literals. The number of variables $v = |\mathbf{X}|$ can be freely set to vary the data complexity; the experiments will present a range of different values for v.

- *Mutually exclusive* correlations, in which the data points are partitioned in *mutex* sets of cardinality (at most) m. Two points o_a, o_b within a mutex set are mutually exclusive, any other data points are independent. Any data source that produces conflicting data will yield this type of correlation, *e.g.*, geolocation data [MCA06], optical character recognition,

and automated data extraction [DBS09; BN08]. The number of variables $v = |\mathbf{X}|$ is directly dependent on the number of objects and the size of the mutex sets.

- *Conditional* correlations, which model uncertainty as a Markov chain with one node per data point. Let $\Phi[o_i]$ be the event associated with the existence of object o_l. Then, the event $\Phi[o_{i+1}]$ becomes $(\Phi[o_i] \wedge x^{\mathrm{t}}_{i+1}) \vee (\neg\Phi[o_i] \wedge x^{\mathrm{f}}_{i+1})$: this is a disjunction of two events, for the cases that o_l exists or not. Consequently, two new Boolean random variables x^{t}_{i+1} and x^{f}_{i+1} are introduced for every data point o_{i+1}. The number of variables $v = |\mathbf{X}|$ is directly dependent on the number of objects.

The Boolean random variables used in the various correlation schemes have randomly generated probabilities in the range $[0.5, 0.8]$. This range was chosen to prevent events with probabilities close to either zero or one, which are easier to approximate, resulting in an unfair advantage for approximation algorithms. To simulate a stream of probabilistic sensor readings, the data is partitioned in groups of four data points with identical lineage, *i.e.* readings from a small time window have identical correlations and uncertainty.

In addition to experiments with probabilistic data from energy distribution networks, a number of experiments were carried out on the UKPN data with varying degrees of uncertainty, and on synthetic data. The use of synthetic and/or certain data in experiments is clearly indicated.

6.2.2 Algorithms

The experimental evaluation reports on performance benchmarks of probability computation for k-medoids clustering and k-nearest neighbour classification. This evaluation contains all five probability computation algorithms:

- The sequential EXACT algorithm for exact probability computation
- The three sequential (*i.e.*, single-threaded) algorithms for approximate probability computation: GREEDY, POSTPONED, and BALANCED
- The BALANCED-C algorithm for concurrent approximate probability computation

Furthermore, two reference implementations are used:

- The NAÏVE algorithm: k-medoids clustering in every possible world. This reference implementation consists of two iterative phases: (1) generate a (next) possible world, and (2) apply

traditional k-medoids clustering on the objects in that world. The implementation of the k-medoids algorithm is not aware of any uncertainty; it recomputes the complete deterministic clustering for the objects in every single one of the exponentially many generated possible worlds.

- The TOP-P algorithm: k-medoids clustering in the top-p most probable worlds. This reference implementation is an approximation algorithm without error guarantees which performs deterministic k-medoids (part of the NAÏVE method outlined above) in the p worlds with the highest probability.

Throughout this chapter, all algorithm names will be typeset in SMALL-CAPS. The approximation algorithms were set to compute probabilities within an (absolute) error bound of $\varepsilon = 0.1$. For k-medoids clustering, the compilation targets are the events that represent medoid selection; for k-nn, the classification events are used. To facilitate comparison, k-medoids clustering experiments were universally run with three iterations and $k = 3$ clusters. The experiments with k-medoids used a folded event network with higher-order events.

The experiments for k-nearest neighbour classification were run using various values for k have been used, as will be indicated in the figures and text.

6.2.3 Higher-order events for k-medoids clustering

Event networks contain events specified by the event programs, which are a probabilistic interpretation of the user programs. Due to the support for loops, the number of events generated by variables in the user program can potentially grow beyond practical memory limits. In those situations, it is desirable to lift the fine-grained events, built using constructs from the event language, to a higher-order realisation. Higher-order events make it possible to define events on *sets* of objects, thereby reducing the size of the event network. The prototype implementation of ENFrame used for the experimental evaluation described in this chapter contains support for such events. The results for probability computation on k-medoids clustering were obtained by using networks with higher-order events; computation for k-nearest neighbour classification was performed using regular networks.

The user and event programs for k-medoids clustering (Algorithm 3.2 and Algorithm 4.2) specify one assignment event $InCl^{i,l}$ for every combination of cluster i and object o_l. Each of the event definitions uses a conjunction over k comparisons of cluster distances, yielding $O\left(k^2 \cdot n\right)$ fine-grained

interconnected events for the assignment phase (where n denotes the number of objects, and k the number of clusters). The update phase adds another $O\left(k \cdot n^2\right)$ events to the network.

Instead, the assignment phase can be expressed using a single higher-order event per object. Such an event induces a probability distribution over all possible clusters the object can be assigned to. Similarly, the update phase can be transformed to use a single event per cluster; these events induce a probability distribution over all possible sets of cluster members. As a result, the number of objects in the network can be reduced from $O\left(k^2 \cdot n + k \cdot n^2\right)$ to $O(n + k)$.

Higher-order events transfer (part of) the complexity of a program or algorithm into native C++ code, which allows for a significant reduction of the network size, enabling more efficient probability computation. This transfer happens at the cost of flexibility: once higher-order events have been manually introduced into the event network, the network is no longer a probabilistic representation of the user program. Additionally, higher-order events hinder sensitivity analysis and query explanation – two techniques that will be further discussed in Section 8.2.

ENFrame's (C++) libraries provide support for development of higher-order events. Although additional effort is required from the programmer to integrate such events in the probability computation process, the concepts of probability computation described in Chapter 5 remain applicable.

6.2.4 Hardware and software

With the exception of Figure 6.7, all experiments were carried out on an Intel Xeon X5660 (2.80GHz, 6 cores) machine with 4GB of RAM, running Ubuntu with Linux kernel 3.5. The results of thread-scalability presented in Figure 6.7 were obtained on an Intel Xeon E5-2690 (2.90GHz, 16 cores).

The framework was implemented in C++ and compiled using GCC 4.7.2. Each of the plots depicts averages (with min/max ranges where applicable) of five runs with randomly generated event expressions and variable probabilities.

6.3 General observations on performance

6.3.1 Sequential algorithms

Figures 6.3 and 6.4 show that all of ENFrame's probability computation algorithms outperform the NAÏVE algorithm by up to six orders of magnitude for each data set with more than 10 variables. Furthermore, the BALANCED approximation can be up to four orders of magnitude faster than EXACT computation. Indeed, for a very small number of possible worlds (*i.e.*, a small number of variables),

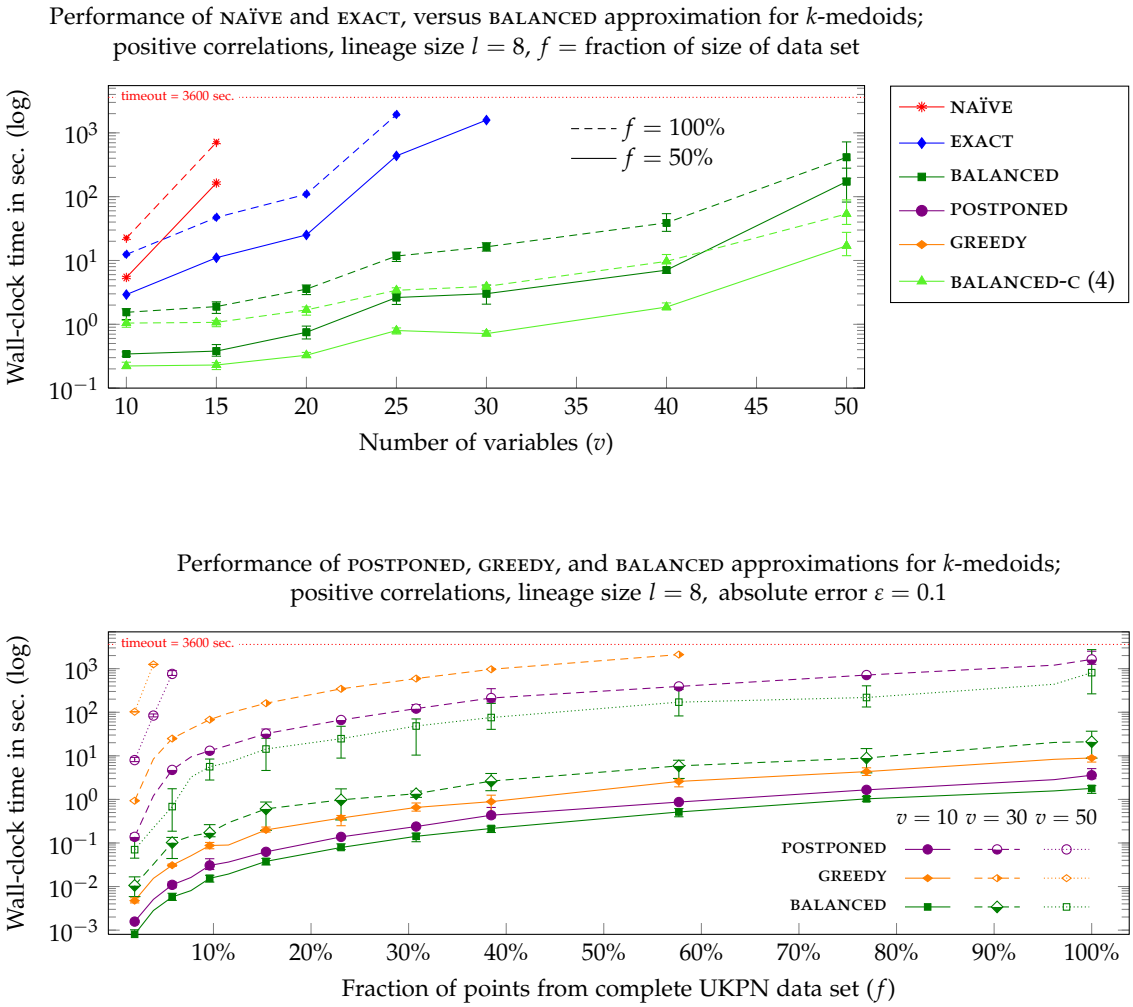


Figure 6.3: Probability computation for k-medoids on positively correlated data. Top: scalability in terms of variables, bottom: scalability of approximations in terms of size of the data set (BALANCED-C not shown for readability reasons).

it pays off to cluster individually in each world and avoid the overhead of the event networks. For a larger number of worlds, the EXACT and approximation algorithms are up to six orders of magnitude faster. The NAÏVE algorithm results in a time-out for over 20 variables ($\approx$ 1 million possible worlds), regardless of the correlation scheme.

The reason why the approximation schemes outperform EXACT is as follows. For a given depth d, there are up to 2^d nodes in the decision tree that contribute to the probability mass of values of compilation targets in the event network. The contributed mass decreases exponentially with an increase in depth, suggesting that most nodes in the decision tree only contribute a small fraction of the total mass. Depending on the desired error bound, a more shallow exploration of the decision tree is enough to obtain a sufficiently large probability mass.

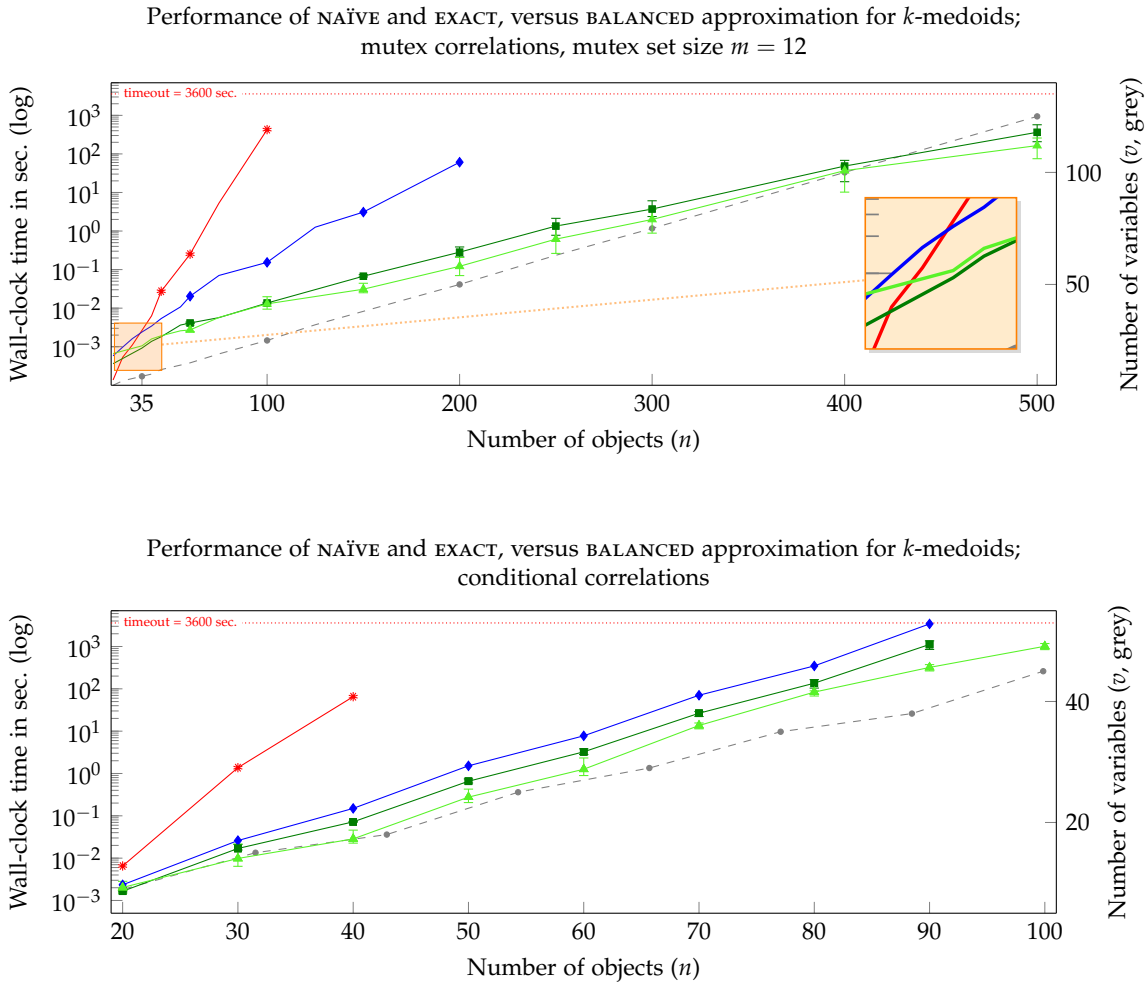


Figure 6.4: Probability computation for k-medoids on data with mutually exclusive (top) and conditional correlations (bottom). Algorithms GREEDY and POSTPONED overlap with EXACT, and are not shown. Grey dashed line indicates number of variables v on second y-axis (depends on number of objects n). Legend: see Figure 6.3

Among the approximation algorithms, BALANCED performs best; it outperforms EXACT for all correlation schemes, for both clustering and classification, by up to four orders of magnitude by performing only a shallow traversal of the decision tree. The other two methods (GREEDY and POSTPONED, only used for k-medoids clustering) use the available error budget to respectively cut the first and last branches, while exploring other branches in full depth.

The POSTPONED approximation strategy performs remarkably well for positive correlations, because the decision tree for the disjunction-based lineage is unbalanced under this scheme. The left branches of the tree correspond to variables being set to *true*, which immediately satisfies the disjunctive object events early, and thus allows for compilation targets to be reached. Further to the right of the decision tree, branches correspond to variables being set to *false*. More variables

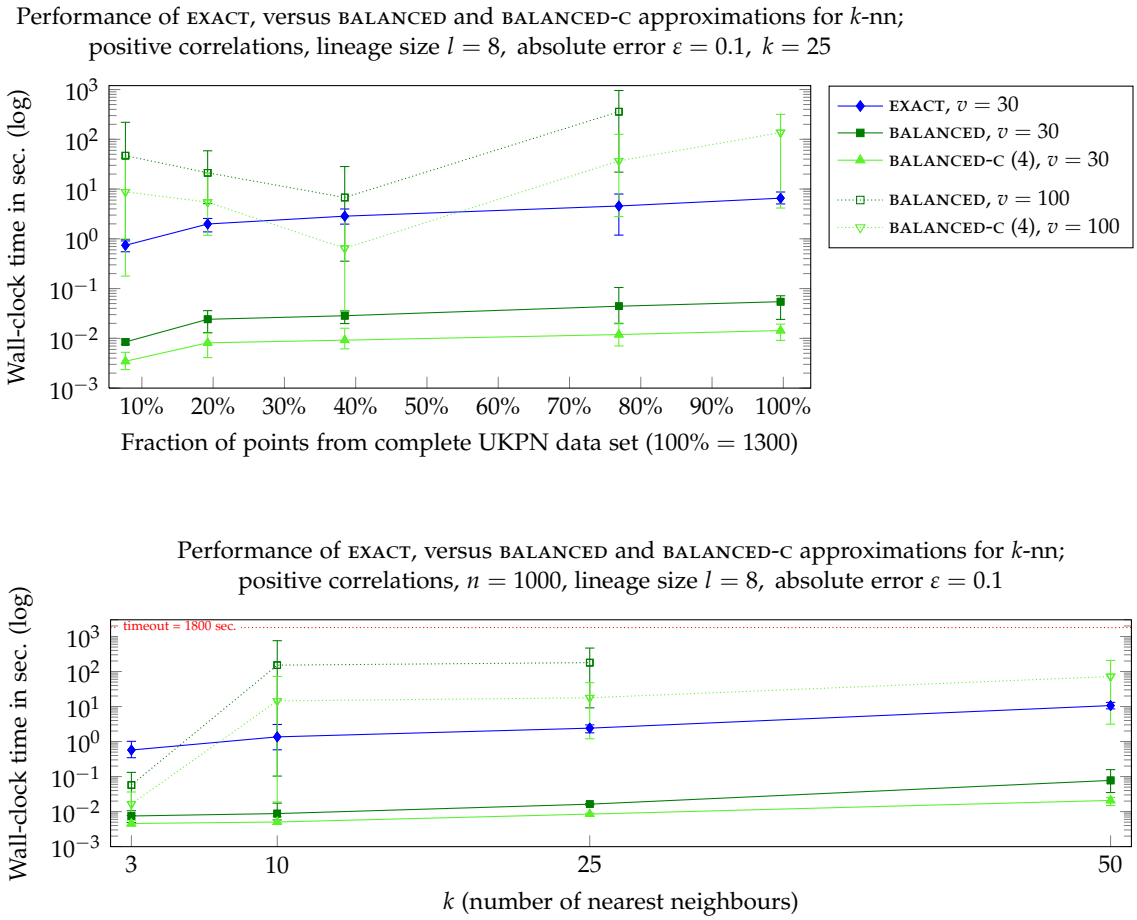


Figure 6.5: Probability computation for k-nearest neighbour on positively correlated data, and varying values for n (objects), v (variables) and k.

need to be set to (un)satisfy the disjunctive input event, thus leading to longer branches. The POSTPONED algorithm saves the error budget until the very last moment and can therefore prune the deep branches whilst maintaining the ε-approximation. The decision trees for the mutex and conditional correlation schemes are more balanced. As a result, the performance of POSTPONED and GREEDY declines and is within a factor 2 of EXACT. To improve the readability of the plots, POSTPONED and GREEDY are not shown in Figure 6.4.

6.3.2 Concurrent probability computation

By distributing the probability computation over multiple CPU cores, ENFrame's performance improves significantly. Figures 6.3, 6.4, 6.5, and 6.6 show timings for BALANCED-C, the algorithm for concurrent probability computation. Regardless of the type of program (clustering or classification), a smaller number of variables yield fewer possible worlds and hence a more shallow decision

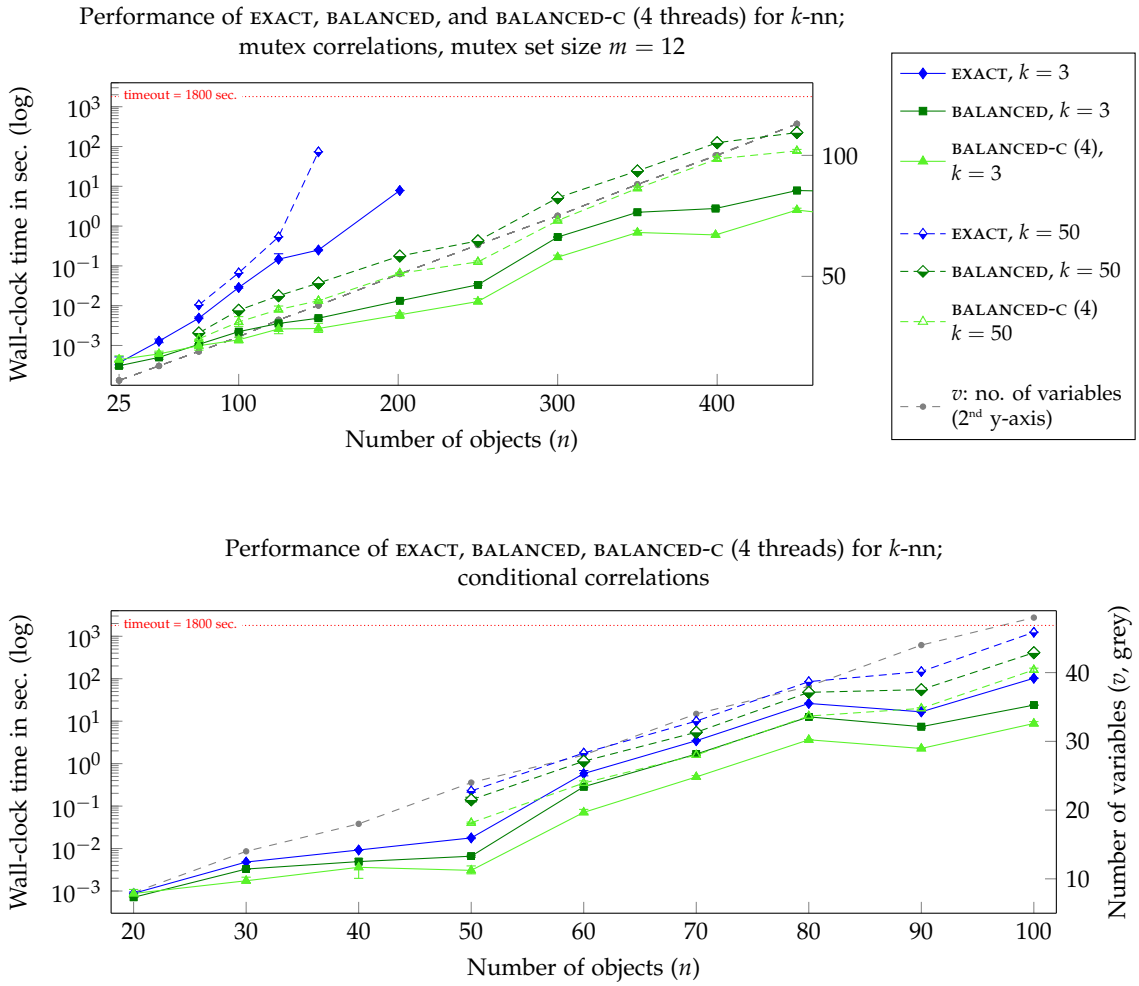


Figure 6.6: Probability computation for k-nn classification on data with mutex and conditional correlations for varying values of n (objects) and k.

tree. Under those circumstances, concurrent probability computation only yields a factor 2-3 improvement over BALANCED due to the fact that a sufficient number of jobs cannot be generated quickly enough to fully use all four available workers (threads). However, as soon as the number of variables approaches 30, the efficiency of the concurrent algorithm becomes clear: for almost all data sets and algorithms, regardless of their parameters, the improvement over BALANCED ranges from a factor 3.5 to a factor 10 (*e.g.* k-medoids in Figure 6.3 for $v = 50$, and k-nearest neighbour in Figure 6.5 for $v = 100$).

This unexpectedly large performance gain (factor 10, using only 4 threads) can be attributed to two causes. Firstly and most foremost, by investigating multiple branches of the decision tree at the same time, the available error allowance for a thread traversing a long (difficult) branch b_1 can be increased due to a simultaneous successful traversal of a shallow (easy) branch b_2 of the tree by

Performance of BALANCED-C for k-nearest neighbour, using various numbers of threads; positive correlations, $n = 10000$, lineage size $l = 8$, $\varepsilon = 0.1$

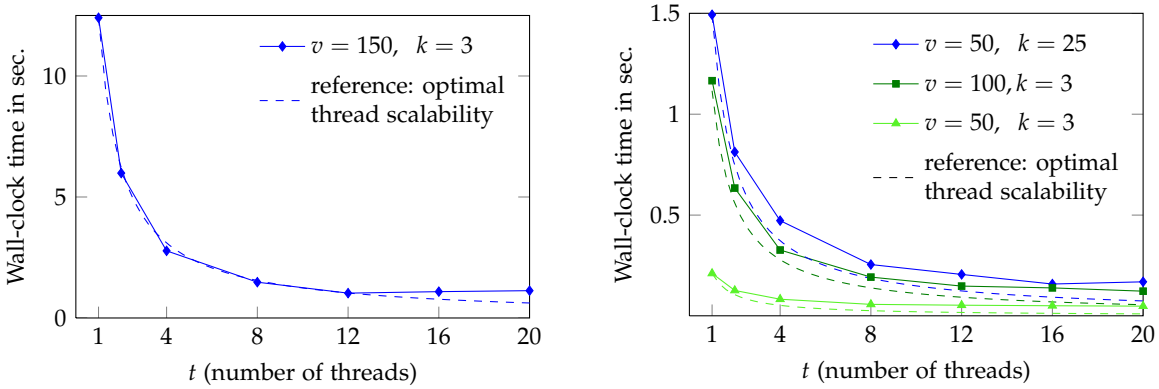


Figure 6.7: Concurrent probability computation for k-nn classification with varying v, k.

a different thread. The sequential BALANCED algorithm will spend time traversing b_1 to its full depth first, leaving some of the error budget unused after a quick traversal of b_2. This effect is especially relevant for positive correlations, which yield an unbalanced decision tree. The second cause is BALANCED's approach to keeping track of the error budgets: the sequential algorithm explicitly stores the error budgets for all compilation targets for the current branch at each depth of the tree to be able to transfer the budget from one branch to another, and spend unused budget locally. The BALANCED-C algorithm recalculates the budget at every depth using the overall progress to minimise the need for thread synchronisation. This effect is not significant for shallow decision trees, but becomes apparent for larger numbers of variables (*e.g.* $v = 100$).

The thread scalability of BALANCED-C has been evaluated using k-nn clustering, the results of which are presented in Figure 6.7. For large numbers of variables ($v = 150$, left), the actual scalability in the number of threads t is near optimal, *i.e.*, t threads yield a factor t performance improvement. For reasons explained above, the performance improvement can exceed a factor t. On the right, experiments with different values for v (number of variables) and k are shown.

6.4 Observations regarding k-medoids clustering

A large synthetically generated data set was used to investigate the influence of certain objects on the scalability of probability computation of k-medoids clustering. Figure 6.8 presents the performance of BALANCED and BALANCED-C probability computation on this data for 0% certain objects (*i.e.*, fully probabilistic data), 95% certain objects, and 100% certain objects (*i.e.*, certain data for which $v = 0$).

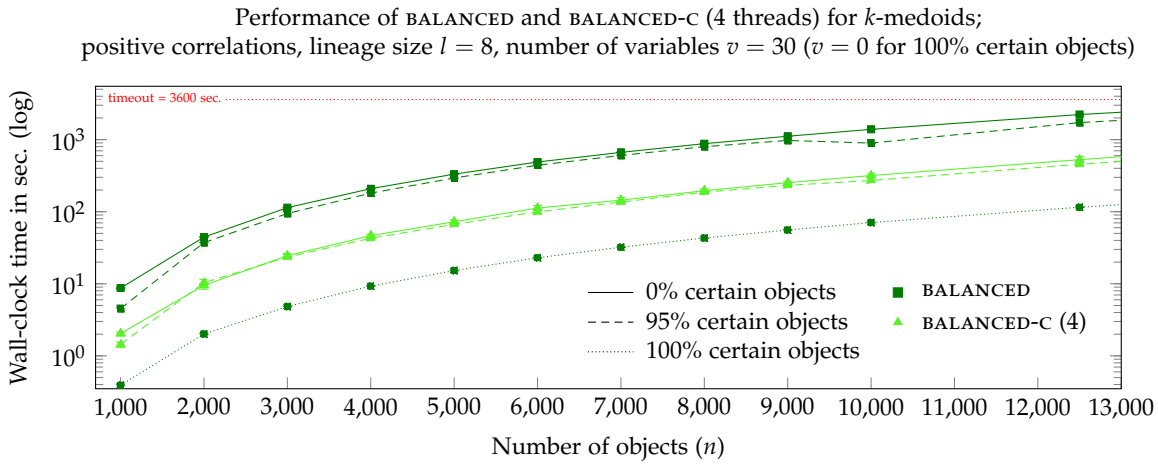


Figure 6.8: Probability computation for k-medoids with BALANCED and BALANCED-C on large-scale generated data sets with certain data points.

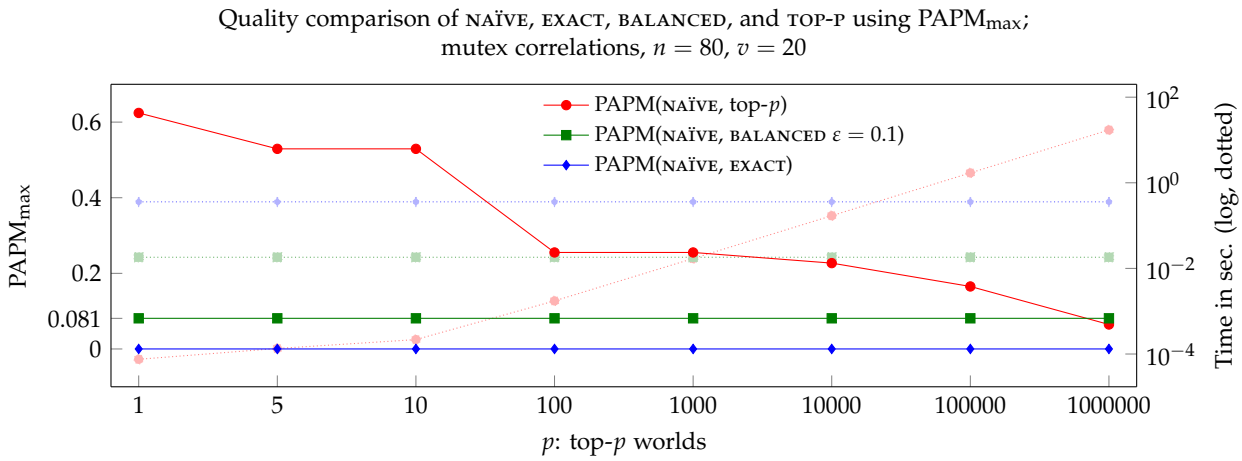


Figure 6.9: Accuracy experiment with k-medoids: comparing NAÏVE (golden standard) to EXACT, BALANCED, and TOP-P clustering (β = 1). The dotted lines indicate the performance (in seconds, on the secondary y-axis) of TOP-P, BALANCED, EXACT, respectively.

For both algorithms, the performance improves slightly when the fraction of certain objects is increased from 0% to 95%. The speed-up in such cases is explained by the fact that the distance sums of medoids to data points in a cluster become less complex and can be initialised using the distances to objects that certainly exist. Consequently, fewer variable assignments are needed to elect a cluster medoid, resulting in a shallower decision tree and an improved compilation time.

Performance of BALANCED on a data set with 100% certain objects is provided as a reference point. For fully certain data (*i.e.*, v = 0 variables), no decision tree is constructed. Therefore, only one thread is used for computation, and the performance of BALANCED and BALANCED-C is identical.

Using the Pairwise Assignment Probability Measure (PAPM) introduced in Section 6.1.1, it is possible to investigate the accuracy of ENFrame by comparing it to clustering in the TOP-P most probable worlds and NAÏVE clustering in every possible world. The latter is considered the ground truth for PAPM purposes. ENFrame's algorithms for exact probability computation are expected to have a zero error when compared to the NAÏVE clustering, as the two algorithms should yield exactly the same results. The ε-approximation algorithms (with $\varepsilon = 0.1$) are expected to have at most a 0.1 difference to the exact probability computation. To verify both claims, PAPM_{max} is used without the scaling constant (*i.e.*, $\beta = 1$).

Figure 6.9 confirms the expectations: the PAPM_{max} score of zero (blue plot line) indicates that ENFrame's EXACT clustering is equivalent to clustering in all possible worlds (NAÏVE). Furthermore, the difference between BALANCED and NAÏVE (green plot line) never exceeds 0.1: the average PAPM_{max} score over five experiment runs is approximately 0.081. On the other hand, the accuracy of clustering in the top-p most probable worlds is far off from NAÏVE, even for large p.

The dotted plot lines in Figure 6.9 indicate the performance (in seconds) of TOP-P, BALANCED, and EXACT on the second y-axis. Already for $p > 1000$, TOP-P is outperformed by ENFrame's BALANCED approximation algorithm. Only when p is almost equal to the number of worlds (*i.e.*, $p \approx 2^v \approx 1000000$), the quality of TOP-P surpasses BALANCED. However, for such large values of p, ENFrame's approximation algorithms are about five orders of magnitude faster, while providing solid error guarantees. The size of the data set for this experiment was restricted due to the (very) limited scalability of the NAÏVE algorithm.

Additionally, the influence of program-specific parameters (such as the number of dimensions, data point coordinates, the numbers of iterations) on performance has been investigated. As is the case with k-medoids on certain data, the number of dimensions has no influence on the computation time, as the algorithm uses a precomputed distance measure on the feature space. The number of clustering iterations has a linear effect on the running time of the algorithm in ENFrame.

The number of compilation targets (including those representing co-occurrence queries) has a minor influence on performance. Due to the combinatorial nature of k-medoids, events are mostly satisfied in bulk. It is thus very rare that one event alone is satisfied at any one time. This also explains why experiments with other types of compilation targets (*e.g.*, object-cluster assignment, pairwise object-cluster assignment) show no difference in performance.

The memory footprint of the event networks was monitored during probability computation, and never exceeded 1GB for the experiments presented in this chapter.

6.5 Observations regarding k-nn classification

The k-nn classification problem is a more *localised* problem than clustering: only the k nearest objects are of importance. As can be seen in Figures 6.5 and 6.6, the correlation scheme has a large influence on the objects which are considered to be possible k-nearest neighbours in an uncertain setting. For positive correlations, as the distance between the unlabelled object and a possible nearest neighbour increases, the probability of that neighbour being one of the k nearest neighbours rapidly decreases. As a result, ENFrame's BALANCED approximation performs up to four orders of magnitude better than EXACT on positive correlations (Figure 6.5 for $n = 1000, k = 10$), and both algorithms scale sub-linearly in n. The BALANCED-C algorithm (with four threads) yields yet another order of magnitude performance improvement. The initial improvement of performance in Figure 6.5 (top) for $v = 100$ can be explained by the local nature of the algorithm as well: for that particular subset of the UKPN data, the classification problem can be solved much more locally – as a result, the perofrmance improves.

As can be seen in Figure 6.5 (bottom), k has a more significant influence on performance: increasing the number of nearest neighbours increases the size of the search space for possible candidates, which in turn increases the time needed to finish the probability computation. There are many techniques to determine the right value of k for a data set [DGL96], all of which can be applied directly to k-nn in ENFrame. For our experiments we decided to use a wide range of values to investigate their influence on the performance of probability computation. Investigating the influence of k on the quality of k-nn classification results is an area of research on its own, and beyond the scope of this work.

The mutex and conditional correlation schemes yield a rather different behaviour, as can be seen in Figure 6.6. Under both schemes, the probability of a distant object being one of the k nearest neighbours is larger, which dramatically increases the number of objects that need to be considered. As a result, the BALANCED approximation scheme cannot trim as many branches of the decision tree compared to the tree and probabilities induced by positive correlations. Concurrent probability computation with four threads yields a consistently better performance with very little synchronisation overhead.

Chapter 7

Use Case Demonstrations

This chapter presents two systems for use cases that have been presented in previously published work. The first system (DAGger, Section 7.1) is an application of uncertain data clustering for sensor data from energy distribution networks. The second system (ΠQora, Section 7.2) is a query engine that integrates data from pc-tables and Bayesian networks. Both papers are included verbatim, with a few minor changes.

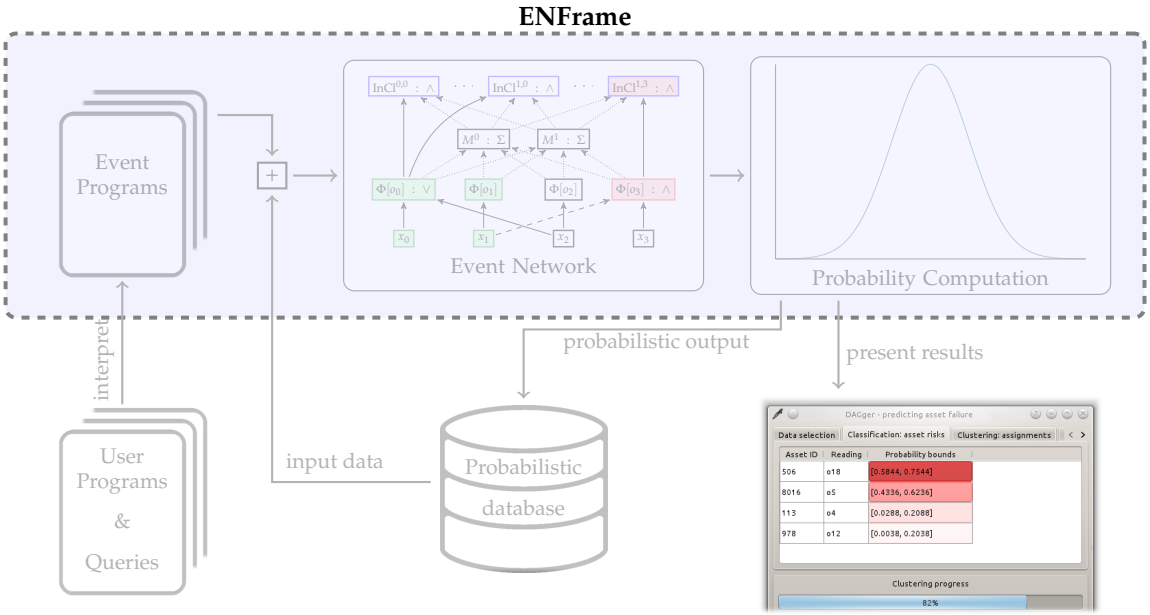


Figure 7.1: ENFrame architecture: chapter 7

7.1 DAGger: clustering correlated uncertain data (to predict asset failure in energy networks)

N.B. this section is a verbatim inclusion of a paper that was published in the proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD) in 2012 [OvS12]. The system presented in this paper (called 'DAGger') is the direct predecessor of ENFrame. Many of the techniques described in this paper have been superseded by research on ENFrame, but the use case remains relevant nonetheless.

DAGger[1] is a clustering algorithm for uncertain data. In contrast to prior work, DAGger can work on arbitrarily correlated data and can compute both exact and approximate clusterings with error guarantees. We demonstrate DAGger using a real-world scenario in which partial discharge data from UK Power Networks is clustered to predict asset failure in the energy network.

7.1.1 Clustering uncertain data

Recent years have witnessed a surge in the amount of digitally-born data. In many scenarios, this data is inherently uncertain or probabilistic, such as in automatic data extraction, image and voice detection (*e.g.*, processing handwriting, controlling mobile phones by voice), location detection, sensor networks, and measurement data [SORK11]. Uncertain data calls for new processing approaches where uncertainty is explicitly accounted for, and it has led to a solid body of work on building probabilistic databases, such as MystiQ, Trio, and MayBMS. Albeit on a smaller scale, there is effort to adapt well-known data mining tasks to uncertain data, e.g., in discovery of frequent patterns and association rules [SCCC10], clustering [GPT08], and classification [QXS+10]. However, to the best of our knowledge, prior work only considers limited probabilistic data models based on a simplifying independence assumption, and circumvents the hardness of probability computation by the use of expected values and Monte-Carlo sampling. Expected values can lead to unintuitive results, for instance when data values and their probabilities follow skewed and non-aligned distributions. In case of correlated input events, the independence assumption can lead to results that are arbitrarily off from the ground truth.

In this work we demonstrate DAGger, a novel approach to clustering correlated uncertain data. At its core, DAGger is a variant of the well-known k-medoids clustering algorithm adapted to accommodate uncertainty throughout the clustering process and probability computation.

[1] DAGger (\'da-gər\). From 'dagger': a sharp pointed knife that is used as a weapon (Merriam Webster, 2014). A loose reference to the way variable valuations are spread (*stabbed*) into the directed acyclic graph that represents an algorithm. The first three letters 'DAG' are capitalised as a reference to the same Directed Acyclic Graph.

DAGger has the following key features:

- The clustering outcome has a simple and intuitive meaning given by the possible worlds semantics: conceptually, it is equivalent to clustering in each possible world represented by the input data. This is in line with virtually all work in probabilistic databases [SORK11] and thus allows for an easy integration of query processing and mining tasks.

- It supports arbitrarily correlated input data through a symbolic representation of probabilistic events. Complex events generated during the clustering process are expressible within the same representation formalism.

- At any stage in the clustering process, DAGger computes clustering events stating the membership of an object to a cluster, and whether an object is a cluster medoid. The probability of such events can be computed exactly or approximately with absolute error guarantees using a novel compilation technique of independent interest. This technique first represents the events of all objects and clusters at all iterations in a directed acyclic graph (DAG) where common factors are represented only once; each node in this graph thus represents an event. It then bulk-compiles all events into one decision diagram to the degree required to compute their probabilities.

- In addition to the events that are intrinsic to the clustering process, DAGger supports queries over the clustering output, e.g., to compute the probability that two given objects belong to the same cluster.

The purpose of the demonstration is to show how DAGger can be effectively used to cluster and classify sensor readings of a phenomenon in energy distribution networks, called *partial discharge*. This is used to predict asset failure in energy distribution networks. We will use real (anonymised) data from UK Power Networks consisting of known readings representing asset failures and new unclassified readings. These readings are naturally uncertain due to limited sensor sensitivity, hardware failure, and unreliable transmission channels [Agg09a; CP03; DGM+04]. By using DAGger, we can improve the quality of the clustering for the set of new sensor readings and are able to distinguish spurious readings from readings that indicate an imminent failure of an asset (*e.g.*, a cable). The audience of this demonstration can explore the clustering outcome visually, as well as a ranked list of critical assets.

	date/time	class	PD	load	$\phi[o_i]$
...					
o_5	20/12 16:00	OK	5	140	$x_4 \wedge x_5 \wedge x_6$
o_6	20/12 17:00	OK	6	140	$x_5 \wedge x_6 \wedge x_7$
o_7	20/12 18:00	OK	9	150	$x_6 \wedge x_7 \wedge x_8$
o_8	20/12 19:00	OK	50	160	$x_7 \wedge x_8 \wedge x_9$
...					
o_{15}	10/01 03:00	warn	22	25	$x_{14} \wedge x_{15} \wedge x_{16}$
o_{16}	10/01 04:00	warn	20	25	$x_{15} \wedge x_{16} \wedge x_{17}$
o_{17}	10/01 05:00	warn	24	40	$x_{16} \wedge x_{17} \wedge x_{18}$
o_{18}	10/01 06:00	warn	25	50	$x_{17} \wedge x_{18} \wedge x_{19}$
...					
o_{25}	01/07 19:00	??	16	100	$x_{24} \wedge x_{25} \wedge x_{26}$
o_{26}	01/07 20:00	??	30	80	$x_{25} \wedge x_{26} \wedge x_{27}$

Table 7.1: Simplified example data set. The labelled sensor readings are prior to a fault on January 11, 2011. The last two readings can be classified by clustering them with labelled data.

In the following sections, we explain our demonstration scenario, show how DAGger clusters uncertain sensor readings, and provide details on how the audience of our demonstration can interact with the system.

7.1.2 Demonstration scenario: clustering partial discharge data

We demonstrate the clustering capability of DAGger in an application that predicts asset failure in energy networks.

Partial discharge

Partial discharge (PD) is an electrical discharge that does not fully bridge the insulation between two conducting electrodes. It has been identified as one of the major causes of long-term degradation and eventual failure of cables.

In order to minimise the customer minutes lost, energy distribution network operators (DNOs) are currently deploying sensors to monitor partial discharge activity in the distribution network, to be able to act preemptively [Mic07; ME11]. Unfortunately, monitoring partial discharge is not a straightforward task: the phenomenon is hard to detect, and sensors often report spurious measurements and are prone to failure (as are the transmission channels).

The HiPerDNO project [TWG+11] aims to show the benefits of introducing cutting edge computational techniques that improve electricity distribution network operations, in partnership with UK Power Networks and other European DNOs.

Uncertain readings of PD and load

The data used to demonstrate our system is historical data on partial discharge activities in distribution networks, as well as records of network load and asset failure. It is gathered from two different types of sensors: (1) partial discharge sensors installed on switchgear and cables in substations of the distribution network, and (2) network load sensors in substations. By aggregating the number of partial discharge occurrences over the duration of an hour and subsequently pairing this value with the average network load during that hour, a data set like the one depicted in Table 7.1 is obtained. `DAGger` can deal with data with an arbitrary number of dimensions. For this demonstration each data point has the attributes load and partial discharge as described above. A single asset typically yields up to 24 data points per day.

`DAGger` interprets this data probabilistically. The rightmost column of Table 7.1 contains probabilistic events (*i.e.*, arbitrary propositional formulas over independent Boolean random variables) that quantify the correlation and probability of readings. This probabilistic data formalism, whereby records are associated with probabilistic events, is called *probabilistic conditional tables*, or pc-tables for short, and is common in probabilistic databases [SORK11].

We associate each sensor reading with a probabilistic event. The probability of that reading being true is thus given by the probability of the event. Each load and partial discharge reading has a probability of being accurate, which is inferred from sensor specification and measurement intervals in historical data. In the events used in `DAGger`, this is captured by independent Boolean random variables $x_0, \ldots x_{m-1}$.

Inference of probabilities and correlations can be done using many techniques, *e.g.* using inference in Bayesian networks or Markov Logic Networks [RD06; JS12] and possibly based on the hardware specifications of the sensor manufacturer. In this specific application of `DAGger`, we construct a Markov chain in which each data point o_t at time t only depends on the data point o_{t-1} at time $t-1$. The conditional probabilities are then converted into events that can be processed by `DAGger`. As expected, consecutive sensor readings are strongly correlated [SJA11]. In the example in Table 7.1, we used a sliding window of size three that yields events represented by conjunctions of three literals.

The possible worlds represented by our sample data are obtained by total assignments of the event variables. For instance, the worlds in which reading o_8 is correct are defined by assignments where x_7, x_8, and x_9 are *true*. Hence, the probability of o_8 being correct is given by the product of probabilities of these Boolean random variables being *true*. The readings o_5 and o_6 are positively correlated, since they both depend on x_5 and x_6. The readings o_8 and o_{15} are independent, since their events are independent.

Predicting asset failure

In order to predict asset failure, we perform the following procedure using `DAGger`:

1. Construct a clustering using both historical data (labelled readings), and the unclassified sensor readings.
2. For each unclassified reading, query the clustering for the probability that the reading is in the same cluster as one (or more) of the readings from the `warn`-labelled set. Depending on the type of labelled readings in the same cluster with the new readings, an expert user can also understand the type of failure.

After `DAGger` has constructed a probabilistic clustering, we can query it for the event that reading o_{25} is clustered into the same cluster as at least one reading from the set $R_{\texttt{warn}}$ with readings labelled `warn`. This query is constructed using clustering events (for clusters $C_1, \ldots, C_k$):

$$\phi[o_{25} \text{ is } \texttt{warn}] = \bigvee_{1 \leq j \leq k} \left(\phi\left[o_{25} \in C_j\right] \wedge \left(\bigvee_{o_w \in R_{\texttt{warn}}} \phi\left[o_w \in C_j\right] \right) \right)$$

In this expression, $\phi\left[o_i \in C_j\right]$ denotes the event that reading o_i belongs to cluster C_j. `DAGger` can cluster the data set from Table 7.1 and perform exact classification of o_{25} within seconds. The system can thus inform the user whether new readings indicate that a fault is imminent.

7.1.3 Under DAGger's hood

At the core of `DAGger` lies the well-known k-medoids clustering algorithm [Bis06; Mac67], an unsupervised data mining technique that partitions a set of data points into k groups of similar points. It repeatedly assigns data points to clusters and re-elects cluster medoids, until convergence is reached.

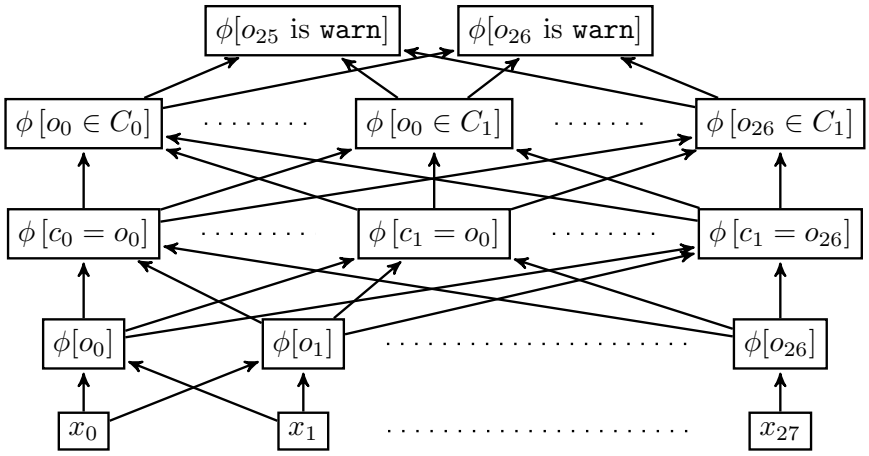


Figure 7.2: Partial DAG with five layers encoding clustering events for clusters C_0 (OK) and C_1 (warn) in our example.

In DAGger, the assignment of data points to clusters and selection of cluster medoids are probabilistic events. Therefore, a data point belongs to a cluster or is a cluster medoid with a certain probability. Conceptually, DAGger's outcome is equivalent to applying the standard k-medoids algorithm in each possible world. However, DAGger cannot afford to enumerate the exponentially many possible worlds and perform a clustering in each of them. Instead, its computation is more symbolic as it traces the clustering events and uses them to compute probabilities of possible clusterings to any approximation degree. This symbolic computation can be orders of magnitude faster than the more extensional approach based on explicit enumeration of the possible worlds.

In this section, we give some details on how DAGger works.

Constructing events. The events $\phi[o_i]$ associated with input readings are the building blocks for events that are subsequently created by DAGger to express medoid selection and cluster assignment. At each clustering step, such events depend solely on events from the previous step. All events can be represented in a layered structure, where each layer corresponds to a clustering step and where we factor out common expressions. This layered factorisation, which is a directed acyclic graph (DAG), is key to the compact representation of the events, as it exploits the combinatorial nature of clustering computation. For instance, the event $\phi[o_i \in C_j]$ that reading o_i belongs to cluster C_j is expressed as a conjunction of the event $\phi[o_i]$ and of events for all cases in which a reading o_l is the medoid of cluster C_j, and the distance from o_i to o_l is the smallest among all distances from o_i to the other readings. Figure 7.2 partially depicts such a DAG. Clustering events are expressed using conditional expressions that involve propositional formulas and distances, since the selection of new cluster medoids depends on input events and distances between data points. They are

expressed an algebraic structure that resembles the semimodule defined by the tensor product $\mathbb{B}[\mathbf{X}] \otimes \mathbb{R}$ of the Boolean semiring $\mathbb{B}[\mathbf{X}]$ freely generated by the set $\mathbf{X}$ of input random variables, and the SUM monoid of real numbers $\mathbb{R}$. For instance, the following expression represents the total distance-sum of a reading o_i to the readings $o_0, \ldots, o_{n-1}$ in cluster C_j:

$$\Delta(C_j, o_i) = \sum_{0 \leq a < n, a \neq i} \Big(\phi[o_a \in C_j] \otimes d(o_i, o_a) \Big)$$

This expression represents a discrete probability distribution function over all possible distance-sums of o_i to readings in cluster C_j in a compact way. Indeed, for each possible truth assignment of random variables, this expression can yield a different distance-sum with a different probability. Such distance-sums are used inside inequalities to construct the events that describe medoid selection: the data point with the smallest distance-sum to the other points in the cluster is chosen as the new cluster medoid.

$$\phi[c_j = o_i] = \phi[o_i \in C_j] \wedge \bigwedge_{0 \leq a < n, a \neq i} (\Delta(C_j, o_i) \leq \Delta(C_j, o_a))$$

In the absence of the $\mathbb{B}[\mathbf{X}] \otimes \mathbb{R}$ construct, these inequalities would only be expressible as propositional events that grow exponentially in the number of objects (or readings).

Once the clustering events are constructed, classification queries such as the one described in Section 7.1.2 are added to the DAG. The DAG in Figure 7.2 includes classification queries for objects o_{25} and o_{26} from Table 7.1 in the top layer.

Probability computation. `DAGger` uses a novel bulk compilation strategy to efficiently compute the probability of the events represented in a layered DAG structure. The core idea of this compilation technique is Shannon expansion: given a Boolean random variable x, the probability $P(\Phi)$ of an event Φ is the weighted sum of probabilities of the events $\Phi|_x$ and $\Phi|_{\neg x}$ obtained by setting x to *true* and respectively to *false* in Φ, *i.e.*, $P(\Phi) = P(x) \cdot P(\Phi|_x) + P(\neg x) \cdot P(\Phi|_{\neg x})$.

The challenges faced by `DAGger` are to extend Shannon expansion (1) to work well on sets of events represented in a DAG structure and on semimodule expressions, and (2) to incrementally compute approximations to any degree.

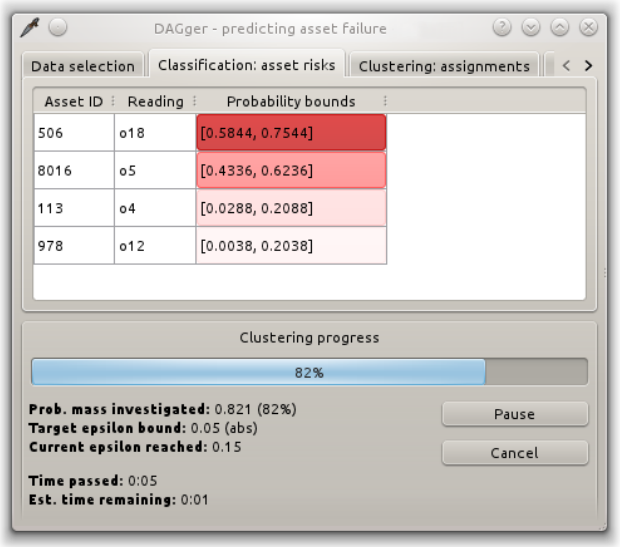
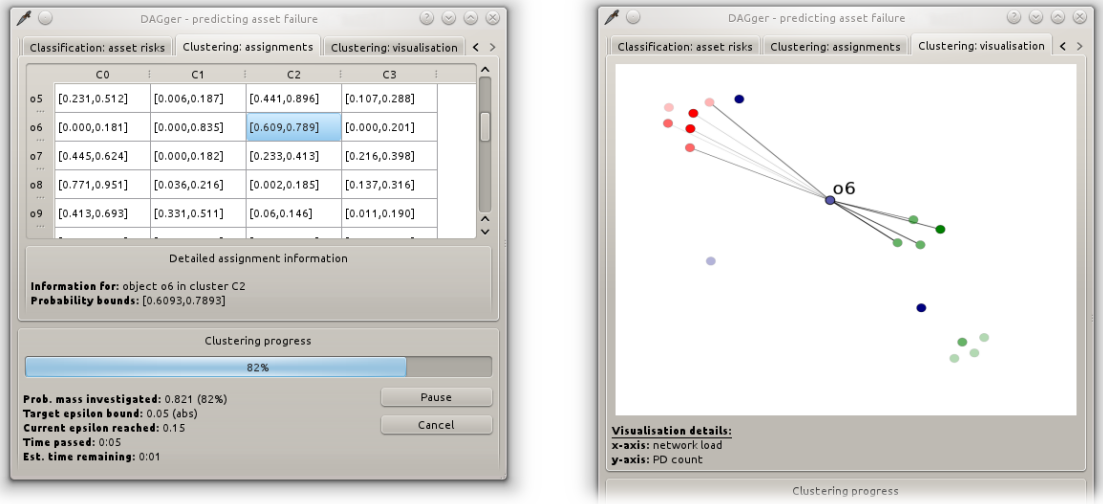


Figure 7.3: Screenshot of DAGger's user interface, showing an ordered list of assets of which sensor readings were classified as 'warning' with a high probability.



(a) Probabilistic assignment of data points to clusters (b) Visualisation of probabilistic clustering

Figure 7.4: Screenshots of DAGger's user interface, showing two different views of the clustering result.

7.1.4 Driving DAGger

DAGger has a graphical user interface to present the clustering outcome, as well as the incremental probability computation of the clustering events. Screenshots of this interface are given in Figures 7.3 and 7.4.

On the first tab 'data selection', the user can make a selection of the input data (both labelled and unlabelled data) which is to be analysed by DAGger. After the data analysis has started, the

user can monitor the progress and examine the results. On the tab "Classification: asset risks" (Figure 7.3), the system displays the results of the classification of the unlabelled data points. It lists the assets that were classified into the `warn` category in decreasing order of probability. The third tab 'Clustering: assignments' (Figure 7.4a) shows the probabilistic assignment of data points to clusters.

The last tab 'Clustering: visualisation' (Figure 7.4b) presents the user with a visual representation of the uncertain clustering. By selecting a sensor reading (in this case: o_6), the interface will show the user the probability that the data point will be clustered into the same cluster as the closest neighbouring data points: the darker the line that connects o_6 to another data point, the higher the probability that the two data points end up in the same cluster.

Throughout the interface of the system, the user will see the exact lower and upper bounds of the probabilities, while the probabilities are being established. Unless `DAGger` is configured to compute approximate probabilities, the system will present the user with the exact probabilities once the lower and upper bounds have converged.

7.1.5 DAGger versus ENFrame

`DAGger` is the direct predecessor of ENFrame: it is a probabilistic clustering algorithm, rather than a framework in which probabilistic algorithms can be expressed. The development of ENFrame required research in various other directions, including the development of a user language. Furthermore, the probability computation algorithms used in ENFrame are much more sophisticated than those introduced in `DAGger`. The latter only features a single GREEDY approximation algorithm, whereas ENFrame is powered by the much more efficient BALANCED approximation strategy. Additionally, ENFrame features heuristics for finding the most influential input random variable, as well as algorithms for concurrent probability computation.

7.2 Πgora: queries on various data formalisms

N.B. this section is an extract of work published in the proceedings of the International Conference on Data Engineering [OPvS13]. It is based on joint work with Dan Olteanu and Lampros Papageorgiou [Pap12].

The data integration techniques introduced in Section 2.4 lie at the heart of Πgora. That integration forms the foundation of a query engine that integrates pc-tables (through MayBMS [AJKO08; HAKO09]), and Bayesian networks (through SMILE [Dru99]). This section presents a use case of the Πgora system: querying of medical data from NELL (the Never Ending Language Learning system [CBK+10]) and the UCI machine learning data repository [BL13].

Real-world applications model probabilistic data using a plethora of different formalisms. These formalisms naturally support probabilistic processing to varying degrees. The pc-tables formalism supports select-project-join queries whose answers and their probabilities can be represented as pc-tables; Bayesian networks support inference queries that ask for the conditional probability of an event given another event; finite state transducers support selection queries that ask for the probability that a certain string occurs in their possible runs.

In order to harness the value of heterogeneous probabilistic data sources, it becomes imperative to provide a uniform interface to them. Such an interface would allow for their integration and enable expressive SQL-like querying across them. This is possible since their underlying formalisms have a common denominator: they all admit a sound interpretation via the possible worlds semantics [SORK11]. Under this semantics, pc-tables, Bayesian networks, and FSTs represent, respectively, finite probability distributions over sets of possible tables, sets of correlated observations, and sets of possible strings represented in an image.

Πgora (pronounced pi-gora: probabilistic agora) is a system that provides just such a uniform interface over Bayesian networks, pc-tables, and FSTs. In addition, it provides a query evaluation mechanism over the interface. The query strategies take the native querying capabilities of the underlying formalisms into account in order to devise an efficient query plan. This plan consists either of a sequence of sub-plans to be evaluated by engines for different formalisms, or of transformations of sources to one of the existing formalisms followed by evaluation using a single query engine. Further common aspects of data integration systems are not yet considered by Πgora. Examples of such aspects include: declarative specifications of the capabilities of each data source (in addition to those of their underlying formalisms); automatically rewriting the user query to use the views representing local sources [MFK+00]; and dealing with many possible rewritings in case several sources publish similar or same data.

7.2.1 The architecture of Πgora

Πgora presents a unifying relational interface over heterogeneous sources called *mediated schema*. The users phrase their queries over this mediated schema. The components of the system, as well as the data and control flow, are shown in Figure 7.5. The system works as follows. Each local source is registered to the system with a relational schema that becomes part of the mediated schema. The user queries are expressed as select-project-join SQL queries extended with an exact and approximate probability computation aggregate and with a *given* clause, which allows the user to formulate conditionals and ask for the probability of an event given another event. For instance, the user could ask for each age group and sex how probable it is that a person suffers from diabetes and their diabetes medication causes exhaustion. This query joins (1) a Bayesian network[2] expressing probabilistic relationships between, among others, the existence of diabetes, age, sex, and pregnancy, (2) a NELL[3] pc-table relating relationships between drugs and side effects, and (3) a NELL pc-table relating drugs and diseases. The user could ask for the probability that a pregnant women has a breast tumour given that she suffers from hypothyroidism. This query is a join of two (independent) Bayesian networks with relationships between age, breast cancer, pregnancy, and hypothyroidism. Next, the different components of Πgora are described.

Data sources. Each local source defines a relational schema that is part of the mediated schema. A tuple-independent NELL pc-table relating drugs and diseases (excerpt provided in Table 7.2) will be used as a running example, together with the Bayesian network shown in Figure 7.6 which contains knowledge regarding diabetes and age, sex, and pregnancy (adapted from the UCI machine learning repository).

The relational schema of the Bayesian network consists of one attribute per node in the network. The network thus corresponds to a relation *Diabetes* with attributes *age*, *diabetes*, *sex*, *pregnant*, and so on; its schema is that of the relation representing the join of all conditional probability tables (CPTs) in the network.

MayBMS is used to manage pc-tables [AJKO08; HAKO09]; Bayesian networks are represented in the XML-based BayesNets Interchange Format[4].

[2] Available from the UCI Machine Learning Repository [BL13] at http://archive.ics.uci.edu/ml/datasets.html.
[3] NELL: the Never Ending Language Learning system [CBK+10])
[4] `http://www.cs.cmu.edu/~fgcozman/Research/InterchangeFormat/`

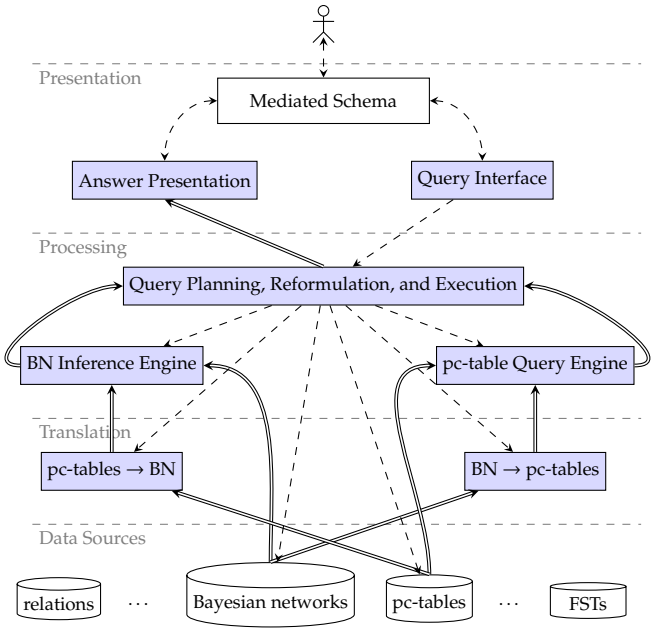


Figure 7.5: The architecture of Πgora. Double and dashed arrows denote data and control flow, respectively.

Drug	**Disease**	**Pr[Drug|Disease]**	**Φ**
avandia	diabetes	1	x_1
tamoxifen	breast_cancer	1	x_2
actos	diabetes	0.998	x_3
glucophage	diabetes	0.996	x_4

Table 7.2: Example of a NELL pc-table containing data regarding drugs and diseases

Translation layer. The possible worlds semantics acts as a bridge between the different formalisms and enables sound, equivalence-preserving translations between their instances [Pap12]. These translations are needed when Πgora's query evaluation strategy requires a translation of all sources into one formalism, and executes the query using one dedicated engine. Section 2.4 describes the translation layer in more detail and includes an example translation of the Bayesian network for diabetes.

Inference and query engines. Πgora is implemented in Java on top of the probabilistic management system MayBMS [AJKO08; HAKO09] with the SPROUT query engine [OHK10] for queries on pc-tables and SMILE [Dru99] for Bayesian inference.

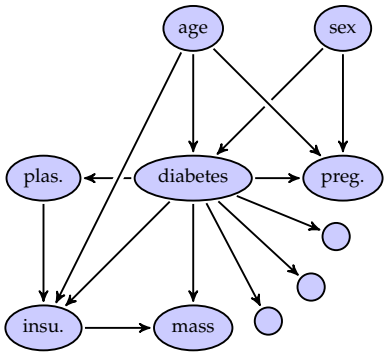


Figure 7.6: Simplified Bayesian network for diabetes from the UCI machine learning repository (the conditional probability tables at nodes are presented in Table 7.3).

sex	**P**
Male	0.48
Female	0.52

age	**P**
≤ 40	0.314
> 40	0.686

age	**sex**	**diabetes**	**P**
≤ 40	Female	true	0.349
≤ 40	Female	false	0.651
≤ 40	Male	true	0.255
≤ 40	Male	false	0.745
> 40	Female	true	0.355
> 40	Female	false	0.645
> 40	Male	true	0.249
> 40	Male	false	0.751

Table 7.3: Conditional probability tables for nodes *sex*, *age*, and *diabetes* in the Bayesian network shown in Figure 7.6

Query planning, reformulation, and execution. The task of this component is to decide how to evaluate the user query. Πgora uses two broad strategies for query evaluation, which are described below and exemplified in Section 7.2.2.

The default strategy is to identify subqueries that are naturally supported by the formalisms of the sources referenced in these subqueries, *i.e.*, select-project-join queries for pc-tables, inference queries for Bayesian networks, and selection queries for finite state transducers. After that, the engines associated with the formalism of the data sources are used to answer the subqueries. All remaining processing steps, *e.g.*, joining subqueries that are evaluated using different query engines, are supported by translation to pc-tables. Besides the identification of such subqueries, a further challenge of this strategy is thus to compute the final query result using the results of the subqueries.

Another strategy is to convert all data sources used by the query (possibly off-line) into either the pc-tables or Bayesian networks formalism, followed by evaluation using either a query or an inference engine after reformulating the query over the corresponding formalism. A possible optimisation is to specialise the query to only use those parts of the data sources that are needed

for the evaluation. For instance, if only a few nodes of a Bayesian network are needed for the evaluation, the query can be rewritten to only use their corresponding pc-tables. For relational evaluation, if the query has a conditional clause, it can be rewritten (by following the definition of conditional probabilities) into a query for the numerator and one for the denominator, and a third query that uses the results of the first two queries to compute the final result. For evaluation via Bayesian inference, the user query is rewritten into a sequence of inference queries that are optimised by identifying independence and temporary results that can be reused several times.

7.2.2 Demonstration scenario: medical data

In this section, Πgora is demonstrated using real-world data sources in the medical domain from NELL, and the UCI machine learning repository.

Let us consider the query in Figure 7.7 (top). It asks for the probability (note the construct `conf()` in the select clause) of a pregnant woman suffering from a left breast tumour, given that she also suffers from hypothyroidism. Correlations between the two medical conditions are reported in literature [Smy03]. Possible data sources are the Bayesian networks *Hypothyroid* and *Breast_cancer*, which can be joined on the common node *age*.

Πgora chooses a purely Bayesian evaluation, since both data sources are Bayesian networks. In this case, the SQL query is phrased as a sum of inference queries:

$$p = \sum_{\text{B.age}} \Pr\Big[\text{B.tumor} = \text{true} \ \wedge\ \text{B.breast} = \text{left} \ \wedge\ \text{H.tumor} = \text{true} \ \wedge\ \text{H.pregnant} = \text{true} \ \Big|\ \text{B.age} = \text{H.age} \ \wedge\ \text{H.hypothyroid} = \text{primary}\Big]$$

For a given value x for *age*, the following inference query is used:

$$p = \Pr\Big[\text{B.tumor} = \text{true} \ \wedge\ \text{B.breast} = \text{left} \ \wedge\ \text{H.tumor} = \text{true} \ \wedge\ \text{H.pregnant} = \text{true} \ \Big|\ \text{B.age} = x \ \wedge\ \text{H.age} = x \ \wedge\ \text{H.hypothyroid} = \text{primary}\Big]$$

Since the two Bayesian networks are independent, the query can be regrouped as follows:

$$p = \Pr\Big[\text{B.tumor} = \text{true} \ \wedge\ \text{B.breast} = \text{left} \ \Big|\ \text{B.age} = x\Big] \cdot \Pr\Big[\text{H.tumor} = \text{true} \ \wedge\ \text{H.pregnant} = \text{true} \ \Big|\ \text{H.age} = x \ \wedge\ \text{H.hypothyroid} = \text{primary})\Big]$$

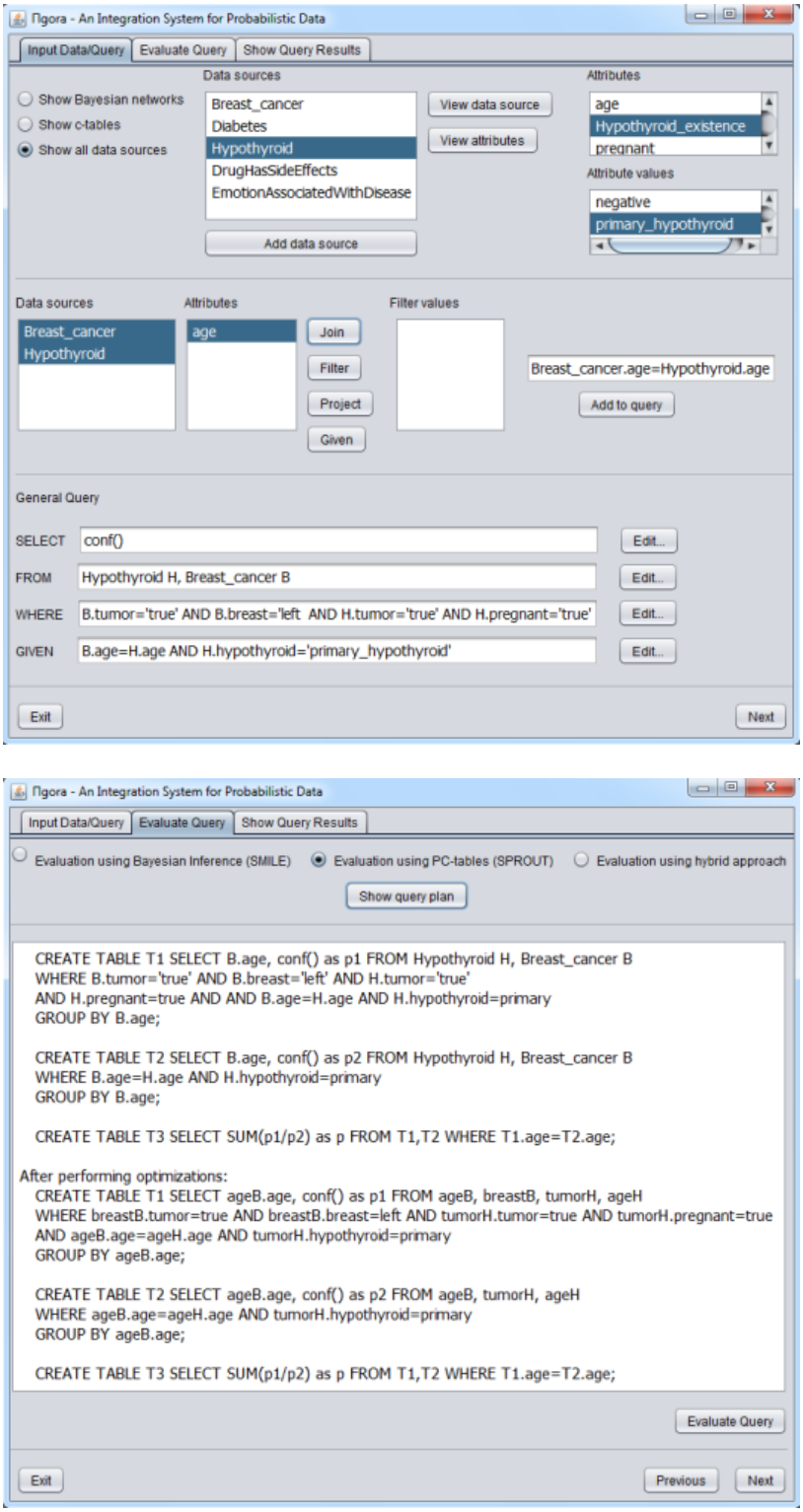


Figure 7.7: Πgora's graphical user interface: Input data and query (top) and visualisation of the chosen evaluation strategy (bottom).

Next, two further scenarios are presented. Figure 7.7 (bottom) depicts the plan for relational evaluation. First, the the conditional probability formula is applied: $P(A \mid B) = \frac{P(A \wedge B)}{P(B)}$. Expressed in SQL, this means that the following values are computed: (1) the probability of the query with a conjunction of the conditions in the `where` and `given` clauses, (2) the probability of the query representing only the `given` clause, (3) the division of the two probabilities, and finally (4) the summation of the probabilities of all *age* values. Further optimisations are applicable, such as specialising the relations *Hypothyroid* and *Breast_cancer* to those constituent pc-tables strictly needed for query evaluation.

Let us now assume that *Breast_cancer* is a pc-table. The default strategy would then split the query into the subquery referring to the *Hypothyroid* network and the subquery referring to the *Breast_cancer* relation.

For each value of x for *age* the following inference query is executed on the *Hypothyroid* network:

$$\forall x : \Pr{}_H(x) \;=\; \Pr\Big[\text{H.tumor} = \text{true} \;\wedge\; \text{H.pregnant} = \text{true} \;\Big|\; \text{H.age} \;= x \;\wedge\; \text{H.hypothyroid} = \text{primary}\Big]$$

The subquery that refers to *Breast_cancer* is now rewritten following the conditional probability formula:

create table T_1 **as select** B.age, conf() as p1
from Breast_cancer B
where B.tumor='true' **and** B.breast='left' **group by** B.age;

create table T_2 **as select** B.age, conf() as p2
from Breast_cancer B **group by** B.age;

create table T_3 **as select** T_1.age, p1/p2 as P_B
from T_1, T_2 **where** T_1.age = T_2.age;

Finally, the query answer is obtained by joining the independent intermediate results P_H and T_3: $\sum_{age} P_B(age) \cdot P_H(age)$, where $P_B(age)$ denotes P_B for the tuple (age, P_B) in T_3.

A further data source for this query could be a collection of finite state transducers modelling possible strings represented in images of scanned book pages referring to these medical conditions. Then, further evidence of correlation between these medical conditions can be signalled by multiple co-occurrences of the names of the two conditions within paragraphs on the same pages, and hence by large co-occurrence probability.

7.2.3 User interaction

The users can interact with Πgora via its graphical user interface that makes it possible to view and register data sources, compose and execute queries, inspect query evaluation strategies, and see query results.

Figure 7.7 depicts two screen shots of Πgora at work. The left screen shot corresponds to the input data and query tab: it presents a list of data sources and shows how the user can compose queries.

The right screen shot corresponds to the evaluation tab, where the user can choose one of the supported evaluation strategies, *i.e.*, evaluation via pc-tables, via Bayesian inference, or the default mixed evaluation where formalism-specific engines are used to evaluate subqueries of the input query. This tab also depicts the execution plan chosen by the system to evaluate the input query.

The right screen shot also depicts a relational plan used by a strategy based on pc-tables to evaluate the query mentioned in the previous section. The plan consists of several relational queries that together encode the conditioning expressed in the original query. It is optimised such that it only refers to those pc-tables obtained by translating the nodes in the input Bayesian networks for hypothyroidism and breast cancer that are necessary to express the query.

Chapter 8

Discussion

The main contribution of this thesis is ENFrame, a framework for processing correlated probabilistic data. The framework and its languages and algorithms have been described in the previous chapters, alongside a number of peripheral contributions to the field of probabilistic data.

This chapter places the contents of this thesis in the context of the fields of probabilistic data, data mining, and (probabilistic) programming languages. The chapter begins with a discussion of related work in those fields, including discussion of the contributions, followed by directions for future research – a number of which are already under investigation. The chapter closes with a conclusion.

8.1 ENFrame in context of related work

The research presented in this thesis is positioned on the intersection of the fields of probabilistic data, data mining, and programming languages. In recent years, multiple efforts have been made to bridge some of the gaps between these fields, yet ENFrame is the first system that aims to connect all three.

8.1.1 Data mining algorithms for probabilistic data

Although arbitrary algorithms can be expressed in ENFrame's user language, the language was primarily designed to support data mining and data processing tasks. In recent years, a large number of data mining algorithms for probabilistic data have been presented in the literature, all of which were designed for one specific data mining task. The most prevalent algorithms for

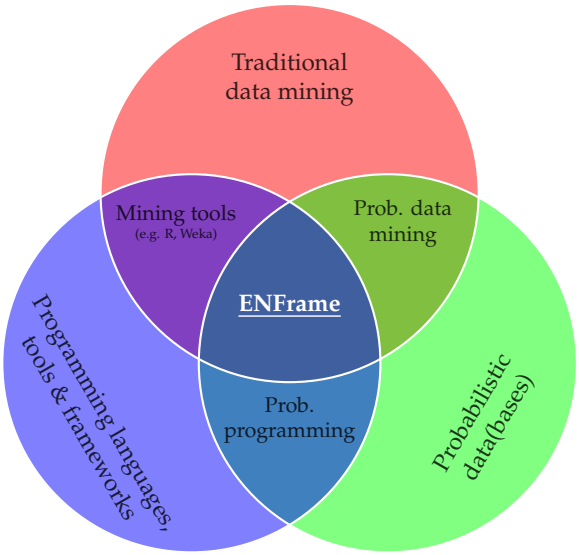


Figure 8.1: ENFrame: positioned at the intersection of probabilistic data, data mining, and programming languages.

clustering and classification of probabilistic data are discussed below. A more detailed survey of these and other types of algorithms for mining uncertain data can be found in [Agg09a].

Clustering probabilistic data

Clustering is possibly the most popular data mining algorithm for uncertain data, with a large number of articles in major publications (including surveys, *e.g.* [ZLZL14]), as well as book chapters devoted to the subject (*e.g.*, [Agg09b]). Most of the algorithms introduced for uncertain data are adaptations of algorithms for certain data.

A number of centroid-based algorithms (inspired by *k*-means) exist. Ngai et al. propose the UK-means algorithm in [NKC+06; CCKN06], using independent objects of which the location is modelled as a continuous probability distribution. The objects are subsequently assigned to the cluster of which the centroid has the closest *expected distance* to the object. Unfortunately, the algorithm does not provide support for correlated objects, and uses non-probabilistic centroids that are computed by averaging expected values of cluster members. The latter leads to information loss regarding the variance of the cluster members, resulting in an inability to detect clusters with different variance (as pointed out by [GT12]). Furthermore, a year after publication, Lee et al. showed how UK-means can be reduced to regular *k*-means [LKC07]. More recently, Kao et al. proposed various improvements to the UK-means algorithm, using Voronoi diagrams and R-tree indices [KLL+10].

In [GT12], Gullo and Tagarelli introduce the concept of uncertain centroids (U-centroids) in the context of an adaptation of *k*-means clustering for probabilistic data: the UCPC algorithm. Although this addresses one of the major shortcomings of UK-means, the presented approach lacks support for correlated objects.

Cormode and McGregor introduce various approximation techniques for clustering of uncertain data by showing how a number of clustering algorithms (*k*-means, *k*-median, *k*-center) on certain classes of data can be reduced to non-probabilistic weighted versions [CM08]. Their approach is similarly limited to tuple-independent data sets.

Further work on medoid-based algorithms for clustering uncertain data includes [GPT08], which presents a technique for clustering data with attribute-level uncertainty: UK-medoids. This approach works using an uncertain distance function that computes the *expected* distance between objects to (1) determine the closest cluster for every object, and (2) determine the new cluster medoid. Although the attribute-level uncertainty allows for a limited notion of correlations (*i.e.*, mutually exclusive attribute values), the use of the expected distance immediately eliminates such a notion. In a later paper [GT12], the same authors criticise the UK-means algorithm [NKC+06; CCKN06] for employing expected distances.

Hierarchical clustering is an alternative clustering method which constructs clusters either bottom-up by merging close objects into a cluster, or top-down by splitting up one large cluster into several smaller ones. The result is a hierarchy of clusterings. Two adaptations for probabilistic data have been proposed in the literature [GPTG08; KP05b], both of which rely on tuple independence.

At the other end of the clustering spectrum, [KP05a] proposes a density-based clustering algorithm for uncertain data: $\mathcal{F}$DBSCAN. This approach is an adaptation of the density-based DBSCAN clustering algorithm [EKSX96]. $\mathcal{F}$DBSCAN (for *fuzzy* DBSCAN) attempts to determine whether multiple data points (observations) in the data are representations of a single object: the *core object*. Using these multiple observations, a single *fuzzy object representation* is constructed. A distance function is then defined for pairs of fuzzy objects, the results of which form the input to the density-based clustering algorithm. $\mathcal{F}$DBSCAN has a very specific and limited application: clustering observations of potentially moving objects. The measurements are considered independent, so are the derived *core objects*.

Although the significance of correlations in probabilistic data is widely acknowledged in the field [VRH+09; SCCC10; AY09] and in various examples in this thesis, all of the clustering algorithms for uncertain data assume independence amongst tuples in the input data. Furthermore, of the

centroid and medoid-based approaches, only one recently introduced algorithm recognises the importance of probabilistic cluster centroids: UCPC [GT12].

It is reasonable to expect that the simplifying independence assumption results in significantly better performance and scalability of the clustering algorithms when compared to ENFrame. However, in the limited number of experiments described in the aforementioned publications, it appears that performance (on similar hardware) is in the same order of magnitude as clustering in ENFrame. For example, Ngai et al. [NKC+06] report clustering times of up to 2500 seconds for 10,000 probabilistically independent objects, using the UK-means algorithm.

Classification of probabilistic data

Classification of uncertain data is a research area that has attracted less attention than clustering. However, the same pattern emerges: the algorithms are adaptations of techniques for certain data, and all assume probabilistic independence amongst the tuples in the input data. Support vector machines for uncertain data are discussed in [BZ04], a naive Bayes classifier is introduced in [RLC+09], and a method based on nearest neighbours is discussed in [AF13].

Continuous versus discrete probability distributions

Among the algorithms for clustering and classification of uncertain data, a number of approaches consider continuous probability distributions over the location of data points in the feature space. Although ENFrame was developed from a probabilistic databases angle and does not provide native support for such distributions, sampling techniques exist to obtain finite distributions over mutually exclusive data points (samples). However, continuous distributions are often fitted using parametric methods on discrete observations [Cra46]. In those cases, it might be beneficial to use the original (discrete) observation data, rather than sample from a fitted continuous distribution.

8.1.2 Probabilistic programming and data analytics platforms

Support for iterative programs is essential in many applications including data mining, web ranking, graph analysis, and model fitting. This has recently led to a surge in data-intensive computing platforms and frameworks for certain data. For example, REX [MIG12] supports iterative distributed computation along database operations in which changes are propagated between algorithm iterations. MADlib is an open-source library for in-database analytics [HRS+12]. Similarly, Bismarck is

```
Variable<bool> firstCoin = Variable.Bernoulli(0.5);
Variable<bool> secondCoin = Variable.Bernoulli(0.5);
Variable<bool> bothHeads = firstCoin & secondCoin;
InferenceEngine ie = new InferenceEngine();
Console.WriteLine("Probability both coins are heads: "+ie.Infer(bothHeads));
```

Algorithm 8.1: Example program written for the Infer.NET framework (adapted from the framework's documentation [MWGK12]).

an architecture for in-database analytics [FKRR12]. GraphLab [LGK+10] uses graph representations for scalable parallel programming.

Interest in development of programming languages and frameworks with native support for *probabilistic* processing has (very) recently soared [GHNR14]. Infer.NET [MWGK12] is a framework for probabilistic programming in .NET, developed by Microsoft Research – it is of a closed nature with restricted availability. Other parties that have shown interest in probabilistic programming include the US Department of Defence: [DAR13] is an initiative to provide funding specifically aimed at the development of probabilistic programming techniques for advanced machine learning. Other initiatives include BUGS (Bayesian inference using Gibbs Sampling, [LTBS00]) and Church [GMR+08]. A complete overview is provided in [Roy].

Such programming languages are imperative or declarative, with the added novelties of allowing the user to express probability distributions via generative stochastic models, of drawing values at random from such distributions, and of conditioning values of program variables on observations. For example, the Infer.NET code listed in Algorithm 8.1 illustrates how to simulate flipping two coins, and inferring the probability that they both end up heads up.

Inference in such languages is often performed using Markov Chain Monte Carlo methods. This is a class of algorithms that sample from a probability distribution which is based on a Markov chain [Nea93]. The probabilistic semantics of such programming languages bear no (or little) resemblance to the semantics used in modern probabilistic databases, which lack of resemblance severely hinders integration. Furthermore, the programming languages require a thorough understanding of probability theory. ENFrame, on the other hand, uses the same semantics and probabilistic data model as recently developed probabilistic databases, in order to facilitate effortless integration. Furthermore, the framework attempts to make programming with such data more accessible by hiding the complexity of programming with probabilities through the user language. This will enable programmers without (or with little) experience in probability theory to process probabilistic data. Python is becoming an increasingly popular language for data ana-

lysis ([BDH+13], for example) due to its gentle learning curve and the large number of libraries for scientific computing. ENFrame capitalises on this popularity through its Python-based user language.

ENFrame does not provide functionality to manually define probability distributions in user or event programs: the distributions are assumed to be part of the input data. Therefore, code like the Infer.NET example cannot be expressed as such in ENFrame. Frameworks like Infer.NET are arguably more suitable for the experienced power user with a background in probability theory. Incorporating such control in ENFrame is part of future research (see Section 8.2).

Originating from the database community, the Monte Carlo Database System (MCDB) and SimSQL systems were visionary in enabling more advanced data analytics by proposing declarative SQL extensions to support loops and probability distributions [JXW+08; CVP+13]. However, the language is not expressive enough to support the writing of data mining algorithms.

8.1.3 Probability computation

Probability computation of propositional expressions and the related model counting problem (#SAT) have been an area of interest since the second half of the last century. Many different approaches have been tried and tested, and various algorithms and heuristics have been described in the literature. The most influential technique is CDP [BL99; GSS09], which is based on the seminal Davis-Putnam-Logemann-Loveland (DPLL) algorithm [DP60; DLL62]. ENFrame's technique for probability computation in event networks combines the previously introduced Shannon expansion with the DPLL-style masking algorithm for verifying satisfiability. However, ENFrame operates on significantly more expressive event networks that represent large numbers of formulae beyond propositional calculus, and introduces various approximation algorithms for probability computation.

Research in artificial intelligence has approached the #SAT problem using Monte Carlo sampling, *e.g.* [WS05]. Although these methods have been shown to be effective in many cases, their computational complexity is prohibitive due to their dependency on truly uniform sampling [GSS06; GSS09]. Additionally, sampling techniques only provide probabilistic error bounds (*i.e.*, only with probability $p < 1$ is the result an ε-approximation), and are unable to exploit the structure of correlations to speed up computation [OW12]. As a result, probability computation for expressions that allow for polynomial-time inference due to their structure (*e.g.*, a conjunction of independent clauses) takes

as much time as inference on intrinsically hard expressions [OHK10; FHO13]. However, sampling methods are an inspiration for a future research direction, as will be set out later in Section 8.2.

Techniques for approximate probability computation in probabilistic databases mainly rely on patterns that allow for simplification in the decision tree constructed through Shannon expansion [OHK10; FHO13] – for example by identifying disjunctions with mutually exclusive operands. Although such techniques have been proven effective for the correlated propositional lineage produced by queries over pc-tables, the correlations that are encoded in event networks for data processing algorithms are more intricate and rarely exhibit such patterns.

Distributed probability computation in probabilistic databases has been approached only in the context of the SimSQL/MCDB system [JXW+08; CVP+13] The query results are computed using Monte Carlo simulations on Markov chains (MCMC). This approach was not designed for exact and approximate computation with error guarantees, and does not exploit the similarity of the many possible worlds induced by pc-tables. Furthermore, like any other sampling-based approximation algorithm, MCMC methods only compute probabilistic guarantees, and do not exploit the structure of the expressions or queries during probability computation.

8.2 Directions for future research and development

At the time of writing, the ENFrame project is only four years old. It has come a long way since its inception, but as recognised throughout this thesis, there are many directions in which the framework can be researched and developed further. A selection of those directions is listed here, based on observations made regarding related work in the previous section.

Alternative data formalisms. ENFrame lacks support for any formalism other than its own extension of the propositional calculus used in pc-tables. Although this is a good starting point with a solid foundation in probabilistic databases, and is compatible with a number of other formalisms (including Bayesian networks), native support for other types of probabilistic data formalisms would accelerate mainstream adoption of the framework. Πgora (Section 2.4 and Chapter 7) aimed to fill a part of this gap through integration of formalisms with a foundation in the possible worlds semantics. However, integration of formalisms beyond that would be of great value – such as native support for continuous probability distributions (*e.g.*, through MayBMS [AJKO08]).

Supporting such radically different formalisms will require efforts in several directions. The event language needs to be developed further, and new formalisms (most notably continuous distributions) require research into alternative methods for probability computation in ENFrame.

N.B. the first steps towards support for random variables with arbitrary categorical distributions in the input data were taken after the initial submission of this thesis. Please refer to [vSO16] for more details and updated algorithms for probability computation.

Probability computation. Probabilistic inference in event networks is another area in which further development is desirable. For example, the inclusion of simplification techniques for event expressions (as described in the literature, *e.g.* [OHK10; FHO13]) could potentially result in a significant performance gain. Although the programs described in this thesis will most likely not benefit from such simplifications of the event network due to the intricate correlations amongst the generated events, the simplifications can result in tractable probability computation for simpler programs and networks.

Other possible improvements in the area of probability computation include integrating different algorithms and techniques for approximate and concurrent probability computation. There is a wealth of literature on approximate and concurrent model counting algorithms which can potentially be adapted for use on event networks, including Monte Carlo sampling approximation techniques [WS05; CVP+13; GSS09].

The concurrent probability computation algorithm introduced in this thesis relies on shared-memory hardware architectures (*i.e.*, can only work using several threads within the same machine). Research into cluster-based distributed algorithms (*e.g.*, using techniques similar to MapReduce [DG04]) would benefit ENFrame.

Sensitivity analysis and result explanation. Sensitivity analysis and result explanation are becoming increasingly important features of both probabilistic and traditional database systems [KLD11]. Query explanation techniques provide the user with reasoning as to which tuples feature (or not) in a query result. Sensitivity analysis is a related technique that helps the user understand how influential some tuples are to a query result, or, in the case of ENFrame, the output of a program.

ENFrame traces the computation of programs by means of events in the event network. By investigating the connections within the network (and the masks they propagate) more closely, both sensitivity analysis and result explanation become possible. As it stands, none of the algorithms and frameworks evaluated in Section 8.1 provide this functionality.

At the moment, ENFrame does not provide such functionality either. However, within the fields of probabilistic database systems, multiple techniques for explanation and sensitivity analysis in query results have been proposed [KLD11; CLF13a; CLF13b]. Further research on how such techniques can be applied in ENFrame is desirable.

Programming with distributions. Other frameworks and probabilistic languages that have been developed in recent years offer more control for the power user (programmer) who wants to explicitly introduce custom probability distributions in the code. ENFrame would benefit from such features, as they would make the framework a more powerful tool for programmers with a background in probability theory.

The implementation of such a feature does not require a significant effort, as long as the distributions are compatible with ENFrame's formalism. For example, a construct in the user and event languages that creates a new Bernoulli random variable with a specified distribution can be encoded in the event network as yet another Boolean random variable. Such a construct enables the user to write programs such as the Infer.NET code illustrated in Algorithm 8.1.

Trade-off performance vs. expressiveness. The constructs of the event language set out in this thesis allow for very fine-grained control over the behaviour of events. This comes at a cost: higher-level operations (such as a phase in a clustering algorithm) often require a large number of events. The masking operation during probability computation scales linearly in the size of the network. Therefore, larger networks of fine-grained events have a negative effect on memory usage and computation time.

The alternative are higher order events (as set out in Section 6.2.3), which transfer large parts of the network logic into native C++ code. This transfer happens at the cost of expressiveness, and allows for less accurate answer explanation, sensitivity analysis, and incremental maintenance. A full investigation of this functionality-performance trade-off is desirable. Furthermore, functionality to automatically generate C++ code for such exhaustive grounding would make performance gain through this feature more accessible.

Support for other types of programs. So far, the development of ENFrame has been driven by the desire to express clustering and classification algorithms for probabilistic data. Although the user and event language are sufficiently flexible to support other data mining and machine learning algorithms, it is as yet unclear where the boundaries lie. Further research into efficiently expressing

other data mining algorithms and other general-purpose algorithms is desired, possibly paired with extending the constructs of both the user and event language.

8.3 Conclusion

This thesis has introduced ENFrame – a framework for processing correlated probabilistic data. The chapters of this thesis have described the framework as separate parts of its architecture: from the data formalisms that are used throughout the framework, via ENFrame's deterministic user language and the probabilistic interpretation using the event language, to the event network and probability computation.

The advent of new data sources has introduced probabilistic data as a new player on the fields of databases and data mining. The large number of systems to store and process such data, and the amount of research done on its properties, are indicators of the interest in this new type of data – both in industry and academia. The pervasive deployment of sensors in virtually all types of electronics, combined with the growing number of other sources that produce probabilistic data, will only further accelerate development of the field.

Correlations are an integral part of the data, and cannot not be ignored during the process of data analysis. Unfortunately, this leads to complexity challenges when performing probability computation. ENFrame addresses these challenges using a combination of novel techniques for probability computation of large numbers of events, heuristics, approximation algorithms, and parallel computation.

Although the experiments with ENFrame's probability computation algorithms show a significant relative performance gain as compared to naïvely iterating over the exponentially many possible worlds, the computational complexity of processing probabilistic data is discernible. This complexity stands in the way of widespread adoption of the framework in industry and academia, as well as adoption of probabilistic data processing in general.

When comparing ENFrame to other recent developments in the fields of probabilistic databases, programming languages, and probabilistic data mining, its advantages are clear. The framework's probabilistic semantics and data model are fully compatible with recent developments in probabilistic databases, and correlations in the data are fully respected through well-defined possible worlds semantics. Programming for probabilistic data becomes easier through the well-defined

Python-based user language, with allows for an effortless probabilistic interpretation. By making programming for probabilistic data more accessible for both professional data analysts and academics with a non-scientific background, mainstream adoption of this type of data is promoted.

Bibliography

[ABS+06] Parag Agrawal, Omar Benjelloun, Anish Das Sarma, Chris Hayworth, Shubha U. Nabar, Tomoe Sugihara and Jennifer Widom, 'Trio: A system for data, uncertainty, and lineage', in *Proceedings of the 32nd International Conference on Very Large Data Bases, Seoul, Korea, September 12-15, 2006*, Umeshwar Dayal, Kyu-Young Whang, David B. Lomet, Gustavo Alonso, Guy M. Lohman, Martin L. Kersten, Sang Kyun Cha and Young-Kuk Kim, Eds., ACM, 2006, pages 1151–1154, ISBN: 1-59593-385-9 (cited on page 104).

[ADT11] Yael Amsterdamer, Daniel Deutch and Val Tannen, 'Provenance for aggregate queries', in *Proceedings of the 30th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS 2011), June 12-16, 2011, Athens, Greece*, Maurizio Lenzerini and Thomas Schwentick, Eds., ACM, 2011, pages 153–164, ISBN: 978-1-4503-0660-7. DOI: 10.1145/1989284.1989302 (cited on page 20).

[AF13] Fabrizio Angiulli and Fabio Fassetti, 'Nearest neighbor-based classification of uncertain data', *ACM Transactions on Knowledge Discovery from Data (TKDD)*, volume 7, number 1, 2013. DOI: 10.1145/2435209.2435210 (cited on page 138).

[Agg09a] Charu C. Aggarwal, Ed., *Managing and Mining Uncertain Data*, series Advances in Database Systems. Kluwer, 2009, volume 35, ISBN: 978-0-387-09689-6. DOI: 10.1007/978-0-387-09690-2 (cited on page 2).

[Agg09b] Charu C. Aggarwal, 'On clustering algorithms for uncertain data', in *Managing and Mining Uncertain Data*, series Advances in Database Systems, Charu C. Aggarwal, Ed., volume 35, Kluwer, 2009, ch. 13, pages 389–406, ISBN: 978-0-387-09689-6. DOI: 10.1007/978-0-387-09690-2 (cited on page 136).

[AJKO08] Lyublena Antova, Thomas Jansen, Christoph Koch and Dan Olteanu, 'Fast and simple relational processing of uncertain data', in *Proceedings of the 24th International Conference on Data Engineering, ICDE 2008, April 7-12, 2008, Cancún, México*, Gustavo Alonso, José A. Blakeley and Arbee L. P. Chen, Eds., IEEE, 2008, pages 983–992. DOI: 10.1109/ICDE.2008.4497507 (cited on page 1).

[Aka74] Hirotugu Akaike, 'A new look at the statistical model identification', *IEEE Transactions on Automatic Control*, volume 19, number 6, pages 716–723, December 1974, ISSN: 0018-9286. DOI: 10.1109/TAC.1974.1100705 (cited on page 36).

[AMS04] Fadi A. Aloul, Igor L. Markov and Karem A. Sakallah, 'MINCE: A static global variable-ordering heuristic for SAT search and BDD manipulation', *Journal of Universal Computer Science*, volume 10, number 12, pages 1562–1596, 2004 (cited on page 86).

[AR07] Eytan Adar and Christopher Ré, 'Managing uncertainty in social networks', *IEEE Data Engineering Bulletin*, volume 30, number 2, pages 15–22, 2007 (cited on page 14).

[AR14] Charu C. Aggarwal and Chandan K. Reddy, Eds., *Data Clustering: Algorithms and Applications*. CRC Press, 2014, ISBN: 978-1-46-655821-2 (cited on page 23).

[ARB+06] Amy Apon, Frank Robinson, Denny Brewer, Larry Dowdy, Doug Hoffman and Baochuan Lu, 'Inital starting point analysis for k-means clustering: A case study', in *Proceedings of ALAR 2006 Conference on Applied Research in Information Technology*, March 2006 (cited on page 37).

[AY09] Charu C. Aggarwal and Philip S. Yu, 'A survey of uncertain data algorithms and applications', *IEEE Transactions on Knowledge and Data Engineering*, volume 21, number 5, pages 609–623, 2009. DOI: 10.1109/TKDE.2008.190 (cited on page 2).

[BBG+63] John. W. Backus, F. L. Bauer, J. Green, C. Katz, J. McCarthy, A. J. Perlis, H. Rutishauser, K. Samelson, B. Vauquois, J. H. Wegstein, A. van Wijngaarden and M. Woodger, 'Revised report on the algorithm language ALGOL 60', *Communications of the ACM*, volume 6, number 1, P. Naur, Ed., pages 1–17, January 1963, ISSN: 0001-0782. DOI: 10.1145/366193.366201 (cited on page 45).

[BDH+13] Vineet Bafna, Alin Deutsch, Andrew Heiberg, Christos Kozanitis, Lucila Ohno-Machado and George Varghese, 'Abstractions for genomics', *Communications of the ACM*, volume 56, number 1, pages 83–93, 2013. DOI: 10.1145/2398356.2398376 (cited on page 140).

[Ben90] Mordechai Ben-Ari, *Principles of Concurrent and Distributed Programming*, 1st edition. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1990, ISBN: 0-13-711821-X (cited on page 97).

[BEP+08] Kurt D. Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge and Jamie Taylor, 'Freebase: A collaboratively created graph database for structuring human knowledge', in *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10-12, 2008*, Jason Tsong-Li Wang, Ed., ACM, 2008, pages 1247–1250, ISBN: 978-1-60558-102-6. DOI: 10.1145/1376616.1376746 (cited on page 4).

[BHH+12] Roger Barga, Laura Haas, Alon Halevy, Paul Miller and Roberto V. Zicari. (2012). Big data for good. Roberto V. Zicari, Ed., Operational Database Management Systems, [Online]. Available: http://www.odbms.org/2012/06/big-data-for-good/ (cited on page 6).

[Bis06] Christopher M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006, ISBN: 0387310738 (cited on page 122).

[BL13] K. Bache and M. Lichman, *UCI machine learning repository*, 2013 (cited on page 25).

[BL99] Elazar Birnbaum and Eliezer L. Lozinskii, 'The good old Davis-Putnam procedure helps counting models', *Journal of Artificial Intelligence Research (JAIR)*, volume 10, pages 457–477, 1999. DOI: 10.1613/jair.601 (cited on page 140).

[BN08] Jens Bleiholder and Felix Naumann, 'Data fusion', *ACM Computing Surveys*, volume 41, number 1, 2008. DOI: 10.1145/1456650.1456651 (cited on page 14).

[Boo54] George Boole, *An Investigation of the Laws of Thought: On which are Founded the Mathematical Theories of Logic and Probabilities*, series George Boole's collected logical works. Walton and Maberly, 1854 (cited on page 74).

[BW96] Beate Bollig and Ingo Wegener, 'Improving the variable ordering of OBDDs is NP-complete', *IEEE Transactions on Computers*, volume 45, number 9, pages 993–1002, Sep. 1996, ISSN: 0018-9340. DOI: 10.1109/12.537122 (cited on page 86).

[BZ04] Jinbo Bi and Tong Zhang, 'Support vector classification with input data uncertainty', in *Advances in Neural Information Processing Systems 17 [Neural Information Processing Systems, NIPS 2004, December 13-18, 2004, Vancouver, British Columbia, Canada]*, 2004 (cited on page 138).

[CBK+10] Andrew Carlson, Justin Betteridge, Bryan Kisiel, Burr Settles, Estevam R. Hruschka Jr. and Tom M. Mitchell, 'Toward an architecture for never-ending language learning', in *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*, Maria Fox and David Poole, Eds., AAAI Press, 2010 (cited on page 17).

[CCKN06] Michael Chau, Reynold Cheng, Ben Kao and Jackey Ng, 'Uncertain data mining: An example in clustering location data', in *Advances in Knowledge Discovery and Data Mining, 10th Pacific-Asia Conference, PAKDD 2006, Singapore, April 9-12, 2006, Proceedings*, Wee Keong Ng, Masaru Kitsuregawa, Jianzhong Li and Kuiyu Chang, Eds., series Lecture Notes in Computer Science, volume 3918, Springer, 2006, pages 199–204, ISBN: 3-540-33206-5. DOI: 10.1007/11731139_24 (cited on page 3).

[CGG10] Toon Calders, Calin Garboni and Bart Goethals, 'Approximation of frequentness probability of itemsets in uncertain data', in *ICDM 2010, The 10th IEEE International Conference on Data Mining, Sydney, Australia, 14-17 December 2010*, Geoffrey I. Webb, Bing Liu, Chengqi Zhang, Dimitrios Gunopulos and Xindong Wu, Eds., IEEE Computer Society, 2010, pages 749–754, ISBN: 978-0-7695-4256-0. DOI: 10.1109/ICDM.2010.42 (cited on page 3).

[CH74] T. Caliński and J. Harabasz, 'A dendrite method for cluster analysis', *Communications in Statistics*, volume 3, number 1, pages 1–27, 1974 (cited on page 36).

[CLF13a] Jianwen Chen, Yiping Li and Ling Feng, 'Sensitivity analysis of answer ordering from probabilistic databases', in *Proceedings of the 24th International Conference on Database and Expert Systems Applications, DEXA 2013, Prague, Czech Republic, August 26-29, 2013, Part II*, Hendrik Decker, Lenka Lhotská, Sebastian Link, Josef Basl and A Min Tjoa, Eds., series Lecture Notes in Computer Science, volume 8056, Springer, 2013, pages 243–258, ISBN: 978-3-642-40172-5. DOI: 10.1007/978-3-642-40173-2_21 (cited on page 143).

[CLF13b] Jianwen Chen, Yiping Li and Ling Feng, 'Sensitivity analysis of answer ordering from probabilistic databases', English, in *Database and Expert Systems Applications*, series Lecture Notes in Computer Science, Hendrik Decker, Lenka Lhotská, Sebastian Link, Josef Basl and Amin Tjoa, Eds., volume 8056, Springer Berlin Heidelberg, 2013, pages 243–258, ISBN: 978-3-642-40172-5. DOI: 10.1007/978-3-642-40173-2_21 (cited on page 143).

[CM08] Graham Cormode and Andrew McGregor, 'Approximation algorithms for clustering uncertain data', in *Proceedings of the Twenty-Seventh ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2008, June 9-11, 2008, Vancouver, BC, Canada*, Maurizio Lenzerini and Domenico Lembo, Eds., ACM, 2008, pages 191–200, ISBN: 978-1-60558-108-8. DOI: 10.1145/1376916.1376944 (cited on page 137).

[Cod70] E. F. Codd, 'A relational model of data for large shared data banks', *Communications of the ACM*, volume 13, number 6, pages 377–387, Jun. 1970, ISSN: 0001-0782. DOI: 10.1145/362384.362685 (cited on page 14).

[CP03] Reynold Cheng and Sunil Prabhakar, 'Managing uncertainty in sensor databases', *SIGMOD Record*, volume 32, number 4, pages 41–46, 2003. DOI: 10.1145/959060.959068 (cited on page 119).

[Cra46] Harold Cramér, *Mathematical Methods of Statistics*, series Princeton landmarks in mathematics and physics. Princeton University Press, 1946 (cited on page 21).

[Cro10] Dan Crow, 'Google squared: Web scale, open domain information extraction and presentation', in *European Conference on Information Retrieval, Industry Day*, 2010 (cited on page 18).

[CSP05] Reynold Cheng, Sarvjeet Singh and Sunil Prabhakar, 'U-DBMS: a database system for managing constantly-evolving data', in *VLDB*, Klemens Böhm, Christian S. Jensen, Laura M. Haas, Martin L. Kersten, Per-Åke Larson and Beng Chin Ooi, Eds., ACM, 2005, pages 1271–1274, ISBN: 1-59593-154-6, 1-59593-177-5 (cited on page 1).

[CVP+13] Zhuhua Cai, Zografoula Vagena, Luis Leopoldo Perez, Subramanian Arumugam, Peter J. Haas and Christopher M. Jermaine, 'Simulation of database-valued Markov chains using SimSQL', in *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2013, New York, NY, USA, June 22-27, 2013*, Kenneth A. Ross, Divesh Srivastava and Dimitris Papadias, Eds., ACM, 2013, pages 637–648, ISBN: 978-1-4503-2037-5. DOI: 10.1145/2463676.2465283 (cited on page 140).

[DAR13] DARPA Defense Advanced Research Projects Agency, *Probabilistic programming for advancing machine learning*, DARPA-BAA-13-31, April 2013 (cited on page 2).

[DB79] David L. Davies and Donald W. Bouldin, 'A cluster separation measure', *IEEE Transactions on Pattern Analysis and Machine Intelligence*, volume 1, number 2, 224–227, 1979. DOI: 10.1109/TPAMI.1979.4766909 (cited on page 100).

[DBS09] Xin Luna Dong, Laure Berti-Equille and Divesh Srivastava, 'Integrating conflicting data: The role of source dependence', *Proceedings of the VLDB Endowment (PVLDB)*, volume 2, number 1, pages 550–561, 2009 (cited on page 14).

[DG04] Jeffrey Dean and Sanjay Ghemawat, 'Mapreduce: Simplified data processing on large clusters', in *6th Symposium on Operating System Design and Implementation, OSDI 2004, San Francisco, California, USA, December 6-8, 2004*, Eric A. Brewer and Peter Chen, Eds., USENIX Association, 2004, pages 137–150 (cited on page 95).

[DGL96] L. Devroye, L. Györfi and G. Lugosi, *A Probabilistic Theory of Pattern Recognition*, series Applications of mathematics: stochastic modelling and applied probability. Springer, 1996, ch. Automatic Nearest Neighbor Rules, ISBN: 9780387946184 (cited on page 115).

[DGM+04] Amol Deshpande, Carlos Guestrin, Samuel Madden, Joseph M. Hellerstein and Wei Hong, 'Model-driven data acquisition in sensor networks', in *Proceedings of the Thirtieth International Conference on Very Large Data Bases, VLDB 2004, Toronto, Canada, August 31 - September 3 2004*, Mario A. Nascimento, M. Tamer Özsu, Donald Kossmann, Renée J. Miller, José A. Blakeley and K. Bernhard Schiefer, Eds., Morgan Kaufmann, 2004, pages 588–599, ISBN: 0-12-088469-0 (cited on page 2).

[DGM05] Amol Deshpande, Carlos Guestrin and Samuel Madden, 'Using probabilistic models for data management in acquisitional environments', in *Proceedings of Conference on Innovative Data Systems Research (CIDR)*, 2005, pages 317–328 (cited on page 1).

[DGS09] Amol Deshpande, Lise Getoor and Prithviraj Sen, 'Graphical models for uncertain data', in *Managing and Mining Uncertain Data*, series Advances in Database Systems, Charu C. Aggarwal, Ed., volume 35, Kluwer, 2009, ch. 4, pages 77–112, ISBN: 978-0-387-09689-6. DOI: 10.1007/978-0-387-09690-2 (cited on page 22).

[Dij65] Edsger Wiebe Dijkstra, 'Solution of a problem in concurrent programming control', *Communications of the ACM*, volume 8, number 9, Sep. 1965, ISSN: 0001-0782. DOI: 10.1145/365559.365617 (cited on page 95).

[DK82] Pierre A. Devyver and Joseph Kittler, *Pattern Recognition: A Statistical Approach*. Prentice-Hall, 1982, ISBN: 9780136542360 (cited on page 100).

[DLL62] Martin Davis, George Logemann and Donald Loveland, 'A machine program for theorem-proving', *Communications of the ACM*, volume 5, number 7, pages 394–397, Jul. 1962, ISSN: 0001-0782. DOI: 10.1145/368273.368557 (cited on page 140).

[DMG+14] Xin Luna Dong, K Murphy, E Gabrilovich, G Heitz, W Horn, N Lao, Thomas Strohmann, Shaohua Sun and Wei Zhang, 'Knowledge Vault: a web-scale approach to probabilistic knowledge fusion', in *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2014, New York, NY, USA, August 24-27, 2014*, 2014 (cited on page 1).

[DP60] Martin Davis and Hilary Putnam, 'A computing procedure for quantification theory', *Journal of the ACM*, volume 7, number 3, pages 201–215, Jul. 1960, ISSN: 0004-5411. DOI: 10.1145/321033.321034 (cited on page 140).

[Dre96] Rolf Drechsler, 'Verification of multi-valued logic networks', in *Proceedings of the 26th International Symposium on Multiple-Valued Logic, ISMVL 1996, Santiago de Compostela, Spain*, 1996, pages 10–15 (cited on page 86).

[Dru99] Marek J. Druzdzel, 'SMILE: structural modeling, inference, and learning engine and genie: A development environment for graphical decision-theoretic models', in *Proceedings of the Sixteenth National Conference on Artificial Intelligence and Eleventh Conference on Innovative Applications of Artificial Intelligence, July 18-22, 1999, Orlando, Florida, USA.*, Jim Hendler and Devika Subramanian, Eds., AAAI Press / The MIT Press, 1999, pages 902–903, ISBN: 0-262-51106-1 (cited on page 127).

[DS04] Nilesh N. Dalvi and Dan Suciu, 'Efficient query evaluation on probabilistic databases', in *Proceedings of the Thirtieth International Conference on Very Large Data Bases, VLDB 2004, Toronto, Canada, August 31 - September 3 2004*, Mario A. Nascimento, M. Tamer Özsu, Donald Kossmann, Renée J. Miller, José A. Blakeley and K. Bernhard Schiefer, Eds., Morgan Kaufmann, 2004, pages 864–875, ISBN: 0-12-088469-0 (cited on page 16).

[Dun73] J. C. Dunn, 'A fuzzy relative of the isodata process and its use in detecting compact well-separated clusters', *Journal of Cybernetics*, volume 3, number 3, pages 32–57, 1973. DOI: 10.1080/01969727308546046 (cited on page 100).

[EKSX96] Martin Ester, Hans-Peter Kriegel, Jörg Sander and Xiaowei Xu, 'A density-based algorithm for discovering clusters in large spatial databases with noise', in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining, KDD 1996, Portland, Oregon, USA*, Evangelos Simoudis, Jiawei Han and Usama M. Fayyad, Eds., AAAI Press, 1996, pages 226–231, ISBN: 1-57735-004-9 (cited on page 137).

[FFM93] Masahiro Fujita, Hisanori Fujisawa and Yusuke Matsunaga, 'Variable ordering algorithms for ordered binary decision diagrams and their evaluation', *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, volume 12, number 1, pages 6–12, 1993. DOI: 10.1109/43.184839 (cited on page 86).

[FHO12] Robert Fink, Larisa Han and Dan Olteanu, 'Aggregation in probabilistic databases via knowledge compilation', *Proceedings of the VLDB Endowment (PVLDB)*, volume 5, number 5, pages 490–501, 2012 (cited on page 20).

[FHO13] Robert Fink, Jiewen Huang and Dan Olteanu, 'Anytime approximation in probabilistic databases', *VLDB Journal*, volume 22, number 6, pages 823–848, 2013. DOI: 10.1007/s00778-013-0310-5 (cited on page 88).

[FHOR11] Robert Fink, Andrew Hogue, Dan Olteanu and Swaroop Rath, 'Sprout^2: A squared query engine for uncertain web data', in *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2011, Athens, Greece, June 12-16, 2011*, Timos K. Sellis, Renée J. Miller, Anastasios Kementsietsidis and Yannis Velegrakis, Eds., ACM, 2011, pages 1299–1302, ISBN: 978-1-4503-0661-4. DOI: 10.1145/1989323.1989481 (cited on page 18).

[FKRR12] Xixuan Feng, Arun Kumar, Benjamin Recht and Christopher Ré, 'Towards a unified architecture for in-RDBMS analytics', in *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2012, Scottsdale, AZ, USA, May 20-24, 2012*, K. Selçuk Candan, Yi Chen, Richard T. Snodgrass, Luis Gravano and Ariel Fuxman, Eds., ACM, 2012, pages 325–336, ISBN: 978-1-4503-1247-9. DOI: 10.1145/2213836.2213874 (cited on page 139).

[GHNR14] Andrew D. Gordon, Thomas A. Henzinger, Aditya V. Nori and Sriram K. Rajamani, 'Probabilistic programming', in *Proceedings of the on Future of Software Engineering, FOSE 2014, Hyderabad, India, May 31 - June 7, 2014*, James D. Herbsleb and Matthew B. Dwyer, Eds., ACM, 2014, pages 167–181, ISBN: 978-1-4503-2865-4. DOI: 10.1145/2593882.2593900 (cited on page 2).

[GKT07] Todd J. Green, Gregory Karvounarakis and Val Tannen, 'Provenance semirings', in *Proceedings of the Twenty-Sixth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 11-13, 2007, Beijing, China*, Leonid Libkin, Ed., ACM, 2007, pages 31–40, ISBN: 978-1-59593-685-1. DOI: 10.1145/1265530.1265535 (cited on page 20).

[GMR+08] Noah D. Goodman, Vikash K. Mansinghka, Daniel M. Roy, Keith Bonawitz and Joshua B. Tenenbaum, 'Church: A language for generative models', in *Proceedings of the 24th Conference in Uncertainty in Artificial Intelligence, UAI 2008, Helsinki, Finland, July 9-12, 2008*, David A. McAllester and Petri Myllymäki, Eds., AUAI Press, 2008, pages 220–229, ISBN: 0-9749039-4-9 (cited on page 139).

[GPT08] Francesco Gullo, Giovanni Ponti and Andrea Tagarelli, 'Clustering uncertain data via k-medoids', in *Scalable Uncertainty Management, Second International Conference, SUM 2008, Naples, Italy, October 1-3, 2008. Proceedings*, Sergio Greco and Thomas Lukasiewicz, Eds., series Lecture Notes in Computer Science, volume 5291, Springer, 2008, pages 229–242, ISBN: 978-3-540-87992-3. DOI: 10.1007/978-3-540-87993-0_19 (cited on page 3).

[GPTG08] Francesco Gullo, Giovanni Ponti, Andrea Tagarelli and Sergio Greco, 'A hierarchical algorithm for clustering uncertain data via an information-theoretic approach', in *Proceedings of the 8th IEEE International Conference on Data Mining (ICDM 2008), December 15-19, 2008, Pisa, Italy*, IEEE Computer Society, 2008, pages 821–826, ISBN: 978-0-7695-3502-9. DOI: 10.1109/ICDM.2008.115 (cited on page 137).

[GSS06] Carla P. Gomes, Ashish Sabharwal and Bart Selman, 'Model counting: A new strategy for obtaining good bounds', *Proceedings of the National Conference on Artificial Intelligence*, volume 21, number 1, page 54, 2006 (cited on page 88).

[GSS09] Carla P. Gomes, Ashish Sabharwal and Bart Selman, 'Model counting', in *Handbook of Satisfiability*, series Frontiers in Artificial Intelligence and Applications, Armin Biere, Marijn Heule, Hans van Maaren and Toby Walsh, Eds., volume 185, IOS Press, 2009, pages 633–654, ISBN: 978-1-58603-929-5. DOI: 10.3233/978-1-58603-929-5-633 (cited on page 88).

[GT06] Todd J. Green and Val Tannen, 'Models for incomplete and probabilistic information', *IEEE Data Engineering Bulletin*, volume 29, number 1, pages 17–24, 2006 (cited on page 2).

[GT12] Francesco Gullo and Andrea Tagarelli, 'Uncertain centroid based partitional clustering of uncertain data', *Proceedings of the VLDB Endowment (PVLDB)*, volume 5, number 7, pages 610–621, 2012 (cited on page 136).

[HAKO09] Jiewen Huang, Lyublena Antova, Christoph Koch and Dan Olteanu, 'MayBMS: a probabilistic database management system', in *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2009, Providence, Rhode Island, USA, June 29 - July 2, 2009*, Ugur Çetintemel, Stanley B. Zdonik, Donald Kossmann and Nesime Tatbul, Eds., ACM, 2009, pages 1071–1074, ISBN: 978-1-60558-551-2. DOI: 10.1145/1559845.1559984 (cited on page 1).

[HRS+12] Joseph M. Hellerstein, Christopher Ré, Florian Schoppmann, Daisy Zhe Wang, Eugene Fratkin, Aleksander Gorajek, Kee Siong Ng, Caleb Welton, Xixuan Feng, Kun Li and Arun Kumar, 'The MADlib analytics library or MAD skills, the SQL', *Proceedings of the VLDB Endowment (PVLDB)*, volume 5, number 12, pages 1700–1711, 2012 (cited on page 138).

[IJ84] Tomasz Imielinski and Witold Lipski Jr., 'Incomplete information in relational databases', *Journal of the ACM*, volume 31, number 4, pages 761–791, 1984. DOI: 10.1145/1634.1886 (cited on page 18).

[Jac09] Adam Jacobs, 'The pathologies of big data', *Communications of the ACM*, volume 52, number 8, pages 36–44, August 2009, ISSN: 0001-0782. DOI: 10.1145/1536616.1536632 (cited on page 6).

[JF96] Anil K. Jain and Patrick J. Flynn, 'Image segmentation using clustering', in *Advances in Image Understanding: A Festschrift for Azriel Rosenfeld*, Kevin Bowyer and Narendra Ahuja, Eds., Los Alamitos, CA, USA: IEEE Computer Society Press, 1996, ISBN: 0818676442 (cited on page 5).

[JS12] Abhay Kumar Jha and Dan Suciu, 'Probabilistic databases with MarkoViews', *Proceedings of the VLDB Endowment (PVLDB)*, volume 5, number 11, pages 1160–1171, 2012 (cited on page 121).

[JXW+08] Ravi Jampani, Fei Xu, Mingxi Wu, Luis Leopoldo Perez, Christopher M. Jermaine and Peter J. Haas, 'MCDB: a Monte Carlo approach to managing uncertain data', in *SIGMOD Conference*, Jason Tsong-Li Wang, Ed., ACM, 2008, pages 687–700, ISBN: 978-1-60558-102-6. DOI: 10.1145/1376616.1376686 (cited on page 1).

[KLD11] Bhargav Kanagal, Jian Li and Amol Deshpande, 'Sensitivity analysis and explanations for robust query evaluation in probabilistic databases', in *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2011, Athens, Greece, June 12-16, 2011*, Timos K. Sellis, Renée J. Miller, Anastasios Kementsietsidis and Yannis Velegrakis, Eds., ACM, 2011, pages 841–852, ISBN: 978-1-4503-0661-4. DOI: 10.1145/1989323.1989411 (cited on page 142).

[KLL+10] Ben Kao, Sau Dan Lee, Foris K.F. Lee, David Wai-lok Cheung and Wai-Shing Ho, 'Clustering uncertain data using voronoi diagrams and r-tree index', *IEEE Transactions on Knowledge and Data Engineering*, volume 22, number 9, pages 1219–1233, 2010, ISSN: 1041-4347. DOI: 10.1109/TKDE.2010.82 (cited on page 136).

[Knu64] Donald E. Knuth, 'Backus Normal Form vs. Backus Naur Form', *Communications of the ACM*, volume 7, number 12, pages 735–736, December 1964, ISSN: 0001-0782. DOI: 10.1145/355588.365140 (cited on page 45).

[KO08] Christoph Koch and Dan Olteanu, 'Conditioning probabilistic databases', *Proceedings of the VLDB Endowment (PVLDB)*, volume 1, number 1, pages 313–325, 2008 (cited on page 23).

[KP05a] Hans-Peter Kriegel and Martin Pfeifle, 'Density-based clustering of uncertain data', in *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Chicago, Illinois, USA, August 21-24, 2005*, Robert Grossman, Roberto J. Bayardo and Kristin P. Bennett, Eds., ACM, 2005, pages 672–677, ISBN: 1-59593-135-X. DOI: 10.1145/1081870.1081955 (cited on page 137).

[KP05b] Hans-Peter Kriegel and Martin Pfeifle, 'Hierarchical density-based clustering of uncertain data', in *Proceedings of the 5th IEEE International Conference on Data Mining, ICDM 2005, 27-30 November 2005, Houston, Texas, USA*, IEEE Computer Society, 2005, pages 689–692, ISBN: 0-7695-2278-5. DOI: 10.1109/ICDM.2005.75 (cited on page 137).

[KR87] Leonard Kaufman and Peter J. Rousseeuw, 'Clustering by means of medoids', *Statistical Data Analysis Based on the L_1 Norm and Related Methods*, Yadolah Dodge, Ed., pages 405–416, 1987 (cited on page 38).

[LC10] Xiang Lian and Lei Chen, 'A generic framework for handling uncertain data with local correlations', *Proceedings of the VLDB Endowment*, volume 4, number 1, pages 12–21, October 2010, ISSN: 2150-8097. DOI: 10.14778/1880172.1880174 (cited on page 23).

[Lee59] C. Y. Lee, 'Representation of switching circuits by binary-decision programs', *Bell System Technical Journal*, volume 38, number 4, pages 985–999, 1959, ISSN: 1538-7305. DOI: 10.1002/j.1538-7305.1959.tb01585.x (cited on page 76).

[LGK+10] Yucheng Low, Joseph Gonzalez, Aapo Kyrola, Danny Bickson, Carlos Guestrin and Joseph M. Hellerstein, 'GraphLab: A new framework for parallel machine learning', in *Proceedings of the Twenty-Sixth Conference on Uncertainty in Artificial Intelligence, UAI 2010, Catalina Island, CA, USA, July 8-11, 2010*, Peter Grünwald and Peter Spirtes, Eds., AUAI Press, 2010, pages 340–349, ISBN: 978-0-9749039-6-5 (cited on page 139).

[LKC07] Sau Dan Lee, Ben Kao and Reynold Cheng, 'Reducing UK-means to K-means', in *Workshops Proceedings of the 7th IEEE International Conference on Data Mining, ICDM 2007, October 28-31, 2007, Omaha, Nebraska, USA*, IEEE Computer Society, 2007, pages 483–488, ISBN: 0-7695-3033-8. DOI: 10.1109/ICDMW.2007.40 (cited on page 136).

[LLS+01] K. A. Linberg, G. P. Lewis, C. Shaaw, T. S. Rex and S. K. Fisher, 'Distribution of S- and M-cones in normal and experimentally detached cat retina', *Journal of Comparative Neurology*, volume 430, number 3, pages 343–356, February 2001 (cited on page 17).

[LOSS04] Rosa Lletí, María Cruz Ortiz, Luis Antonio Sarabia and M. Sagrario Sánchez, 'Selecting variables for k-means cluster analysis by using a genetic algorithm that optimises the silhouettes', *Analytica Chimica Acta*, volume 515, number 1, pages 87–100, 2004, Papers presented at the 5th Colloquium Chemiometricum Mediterraneum, ISSN: 0003-2670. DOI: 10.1016/j.aca.2003.12.020 (cited on page 36).

[LS09] Vebjorn Ljosa and Ambuj K. Singh, 'Probabilistic querying and mining of biological images', in *Managing and Mining Uncertain Data*, series Advances in Database Systems, Charu C. Aggarwal, Ed., volume 35, Kluwer, 2009, ch. 16, pages 461–488, ISBN: 978-0-387-09689-6. DOI: 10.1007/978-0-387-09690-2_16 (cited on page 3).

[LSV02] Jens Lechtenbörger, Hua Shu and Gottfried Vossen, 'Aggregate queries over conditional tables', *Journal of Intelligent Information Systems*, volume 19, number 3, pages 343–362, 2002. DOI: 10.1023/A:1020197923385 (cited on page 19).

[LTBS00] David J. Lunn, Andrew Thomas, Nicky Best and David Spiegelhalter, 'WinBUGS & a Bayesian modelling framework: concepts, structure, and extensibility', *Statistics and Computing*, volume 10, number 4, pages 325–337, October 2000, ISSN: 0960-3174. DOI: 10.1023/A:1008929526011 (cited on page 139).

[Mac67] J.B. MacQueen, 'Some methods for classification and analysis of multivariate observations', in *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics*, Berkeley, California: University of California Press, 1967, pages 281–297 (cited on page 122).

[Mar03] Dimitris Margaritis, 'Learning Bayesian network model structure from data', PhD thesis, University of Pittsburgh, 2003 (cited on page 22).

[MCA06] Mohamed F. Mokbel, Chi-Yin Chow and Walid G. Aref, 'The new Casper: query processing for location services without compromising privacy', in *Proceedings of the 32nd International Conference on Very Large Data Bases, VLDB 2006, Seoul, Korea, September 12-15, 2006*, Umeshwar Dayal, Kyu-Young Whang, David B. Lomet, Gustavo Alonso, Guy M. Lohman, Martin L. Kersten, Sang Kyun Cha and Young-Kuk Kim, Eds., ACM, 2006, pages 763–774, ISBN: 1-59593-385-9 (cited on page 14).

[ME11] Matthieu Michel and Carl Eastham, 'Improving the management of MV underground cable circuits using automated on-line cable partial discharge mapping', in *Proceedings of the 21st International Conference on Electricity Distribution, CIRED 2011*, 2011 (cited on page 104).

[MFK+00] Ioana Manolescu, Daniela Florescu, Donald Kossmann, Florian Xhumari and Dan Olteanu, 'Agora: Living with XML and relational', in *VLDB 2000, Proceedings of 26th International Conference on Very Large Data Bases, September 10-14, 2000, Cairo, Egypt*, Amr El Abbadi, Michael L. Brodie, Sharma Chakravarthy, Umeshwar Dayal, Nabil Kamel, Gunter Schlageter and Kyu-Young Whang, Eds., Morgan Kaufmann, 2000, pages 623–626, ISBN: 1-55860-715-3 (cited on page 127).

[Mic07] Matthieu Michel, 'Innovative asset management and targeted investments using on-line partial discharge monitoring & mapping techniques', in *Proceedings of the 19th International Conference on Electricity Distribution, CIRED 2007*, 2007 (cited on page 104).

[MIG12] Svilen R. Mihaylov, Zachary G. Ives and Sudipto Guha, 'REX: recursive, delta-based data-centric computation', *Proceedings of the VLDB Endowment (PVLDB)*, volume 5, number 11, pages 1280–1291, 2012 (cited on page 138).

[MMS02] Berna L. Massingill, Timothy G. Mattson and Beverly A. Sanders, 'Some algorithm structure and support patterns for parallel application programs', in *Proceedings of the Ninth Conference on Pattern Language of Programs 2002, PLoP 2002*, Sep. 2002 (cited on page 97).

[MPR02] Mehryar Mohri, Fernando Pereira and Michael Riley, 'Weighted finite-state transducers in speech recognition', *Computer Speech & Language*, volume 16, number 1, pages 69–88, 2002. DOI: 10.1006/csla.2001.0184 (cited on page 22).

[MR03] Daniel D. McCracken and Edwin D. Reilly, 'Backus-Naur Form (BNF)', in *Encyclopedia of Computer Science*, Chichester, UK: John Wiley and Sons Ltd., 2003, pages 129–131, ISBN: 0-470-86412-5 (cited on page 45).

[MSST07] Nina Mishra, Robert Schreiber, Isabelle Stanton and Robert Endre Tarjan, 'Clustering social networks', in *Algorithms and Models for the Web-Graph, 5th International Workshop, WAW 2007, San Diego, CA, USA, December 11-12, 2007, Proceedings*, Anthony Bonato and Fan R. K. Chung, Eds., series Lecture Notes in Computer Science, volume 4863, Springer, 2007, pages 56–67, ISBN: 978-3-540-77003-9. DOI: 10.1007/978-3-540-77004-6_5 (cited on page 5).

[MWBS88] Sharad Malik, Albert R. Wang, Robert. K. Brayton and Alberto Sangiovanni-Vincentelli, 'Logic verification using binary decision diagrams in a logic synthesis environment', in *IEEE International Conference on Computer-Aided Design, ICCAD 1988*, November 1988, pages 6–9. DOI: 10.1109/ICCAD.1988.122451 (cited on page 86).

[MWGK12] T. Minka, J.M. Winn, J.P. Guiver and D.A. Knowles, *Infer.net 2.5*, Microsoft Research Cambridge, 2012 (cited on page 7).

[Nea03] Richard E. Neapolitan, *Learning Bayesian Networks*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2003, ISBN: 0130125342 (cited on page 25).

[Nea93] Radford M. Neal, 'Probabilistic inference using Markov chain Monte Carlo methods', Department of Computer Science, University of Toronto, Tech. Rep. CRG-TR-93-1, 1993 (cited on page 139).

[NKC+06] Wang Kay Ngai, Ben Kao, Chun Kit Chui, Reynold Cheng, Michael Chau and Kevin Y. Yip, 'Efficient clustering of uncertain data', in *Proceedings of the 6th IEEE International Conference on Data Mining, ICDM 2006, 18-22 December 2006, Hong Kong, China*, IEEE Computer Society, 2006, pages 436–445, ISBN: 0-7695-2701-9. DOI: 10.1109/ICDM.2006.63 (cited on page 99).

[NZRS12] Feng Niu, Ce Zhang, Christopher Ré and Jude W. Shavlik, 'DeepDive: web-scale knowledge-base construction using statistical learning and inference', in *Proceedings of the Second International Workshop on Searching and Integrating New Web Data Sources, Istanbul, Turkey, August 31, 2012*, Marco Brambilla, Stefano Ceri, Tim Furche and Georg Gottlob, Eds., series CEUR Workshop Proceedings, volume 884, CEUR-WS.org, 2012, pages 25–28 (cited on page 4).

[OHK09] Dan Olteanu, Jiewen Huang and Christoph Koch, 'SPROUT: lazy vs. eager query plans for tuple-independent probabilistic databases', in *Proceedings of the 25th International Conference on Data Engineering, ICDE 2009, March 29 2009 - April 2 2009, Shanghai, China*, Yannis E. Ioannidis, Dik Lun Lee and Raymond T. Ng, Eds., IEEE, 2009, pages 640–651, ISBN: 978-0-7695-3545-6. DOI: 10.1109/ICDE.2009.123 (cited on page 8).

[OHK10] Dan Olteanu, Jiewen Huang and Christoph Koch, 'Approximate confidence computation in probabilistic databases', in *Proceedings of the 26th International Conference on Data Engineering, ICDE 2010, March 1-6, 2010, Long Beach, California, USA*, Feifei Li, Mirella M. Moro, Shahram Ghandeharizadeh, Jayant R. Haritsa, Gerhard Weikum, Michael J. Carey, Fabio Casati, Edward Y. Chang, Ioana Manolescu, Sharad Mehrotra, Umeshwar Dayal and Vassilis J. Tsotras, Eds., IEEE, 2010, pages 145–156, ISBN: 978-1-4244-5444-0. DOI: 10.1109/ICDE.2010.5447826 (cited on page 88).

[OM96] Thomas Owens and David McConville, 'Estimating the spatial accuracy of coordinates collected using the global positioning system', United States Geological Survey, Tech. Rep., 1996 (cited on page 21).

[OPvS13] Dan Olteanu, Lampros Papageorgiou and Sebastiaan J. van Schaik, 'Πgora: An integration system for probabilistic data', in *29th IEEE International Conference on Data Engineering, ICDE 2013, Brisbane, Australia, April 8-12, 2013*, Christian S. Jensen, Christopher M. Jermaine and Xiaofang Zhou, Eds., IEEE Computer Society, 2013, pages 1324–1327, ISBN: 978-1-4673-4909-3. DOI: 10.1109/ICDE.2013.6544935 (cited on page 11).

[OvS12] Dan Olteanu and Sebastiaan J. van Schaik, 'DAGger: clustering correlated uncertain data (to predict asset failure in energy networks)', in *The 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2012, Beijing, China, August 12-16, 2012*, Qiang Yang, Deepak Agarwal and Jian Pei, Eds., ACM, 2012, pages 1504–1507, ISBN: 978-1-4503-1462-6. DOI: 10.1145/2339530.2339766 (cited on page 3).

[OW12] Dan Olteanu and Hongkai Wen, 'Ranking query answers in probabilistic databases: Complexity and efficient algorithms', in *IEEE 28th International Conference on Data Engineering, ICDE 2012, Washington, DC, USA (Arlington, Virginia), 1-5 April, 2012*, Anastasios Kementsietsidis and Marcos Antonio Vaz Salles, Eds., IEEE Computer Society, 2012, pages 282–293, ISBN: 978-0-7695-4747-3. DOI: 10.1109/ICDE.2012.61 (cited on page 16).

[Pac11] Peter Pacheco, *An Introduction to Parallel Programming*, 1st. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2011, ISBN: 9780123742605 (cited on page 95).

[Pap12] Lampros Papageorgiou, 'Πgora: An integration system for probabilistic data', Master's thesis, University of Oxford, August 2012 (cited on page 24).

[PB83] J. Scott Provan and Michael O. Ball, 'The complexity of counting cuts and of computing the probability that a graph is connected', *SIAM Journal on Computing*, volume 12, number 4, pages 777–788, 1983. DOI: 10.1137/0212053 (cited on page 78).

[PDN05] D.T. Pham, S.S. Dimov and C.D. Nguyen, 'Selection of k in k-means clustering', *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, volume 219, number 1, pages 103–119, 2005. DOI: 10.1243/095440605X8298 (cited on page 37).

[Pea88] Judea Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, series Representation and Reasoning Series. Morgan Kaufmann, 1988, ISBN: 9781558604797 (cited on page 22).

[PM00] Dan Pelleg and Andrew W. Moore, 'X-means: Extending k-means with efficient estimation of the number of clusters', in *Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000), Stanford University, Stanford, CA, USA, June 29 - July 2, 2000*, Pat Langley, Ed., Morgan Kaufmann, 2000, pages 727–734, ISBN: 1-55860-707-2 (cited on page 36).

[PS00] Réjean Plamondon and Sargur N. Srihari, 'Online and off-line handwriting recognition: A comprehensive survey', *IEEE Transactions on Pattern Analysis and Machine Intelligence*, volume 22, number 1, pages 63–84, January 2000, ISSN: 0162-8828. DOI: 10.1109/34.824821 (cited on page 1).

[PZC+13] Bin Pei, Suyun Zhao, Hong Chen, Xuan Zhou and Dingjie Chen, 'FARP: mining fuzzy association rules from a probabilistic quantitative database', *Journal of Information Sciences*, volume 237, pages 242–260, 2013. DOI: 10.1016/j.ins.2013.02.010 (cited on page 3).

[QXS+10] Biao Qin, Yuni Xia, Rakesh Sathyesh, Sunil Prabhakar and Yicheng Tu, 'uRule: A rule-based classification system for uncertain data', in *The 10th IEEE International Conference on Data Mining Workshops, ICDMW 2010, Sydney, Australia, 13 December 2010*, Wei Fan, Wynne Hsu, Geoffrey I. Webb, Bing Liu, Chengqi Zhang, Dimitrios Gunopulos and Xindong Wu, Eds., IEEE Computer Society, 2010, pages 1415–1418, ISBN: 978-0-7695-4257-7. DOI: 10.1109/ICDMW.2010.73 (cited on page 118).

[Ran71] William M. Rand, 'Objective criteria for the evaluation of clustering methods', *Journal of American Statistical Association*, 1971 (cited on page 100).

[RD06] Matthew Richardson and Pedro Domingos, 'Markov logic networks', *Machine Learning*, volume 62, number 1-2, pages 107–136, 2006. DOI: 10.1007/s10994-006-5833-1 (cited on page 121).

[RK08] Michael Rice and Sanjay Kulhari, 'A survey of static variable ordering heuristics for efficient BDD/MDD construction', University of California, Riverside, Tech. Rep., 2008 (cited on page 86).

[RL87] P. J. Rousseeuw and A. M. Leroy, *Robust Regression and Outlier Detection*. New York, NY, USA: John Wiley & Sons, Inc., 1987, ISBN: 0-471-85233-3 (cited on page 3).

[RLC+09] Jiangtao Ren, Sau Dan Lee, Xianlu Chen, Ben Kao, Reynold Cheng and David Wai-Lok Cheung, 'Naive Bayes classification of uncertain data', in *The Ninth IEEE International Conference on Data Mining, ICDM 2009, Miami, Florida, USA, 6-9 December 2009*, Wei Wang, Hillol Kargupta, Sanjay Ranka, Philip S. Yu and Xindong Wu, Eds., IEEE Computer Society, 2009, pages 944–949, ISBN: 978-0-7695-3895-2. DOI: 10.1109/ICDM.2009.90 (cited on page 138).

[Rou87] Peter J. Rousseeuw, 'Silhouettes: A graphical aid to the interpretation and validation of cluster analysis', *Journal of Computational and Applied Mathematics*, volume 20, pages 53–65, 1987, ISSN: 0377-0427. DOI: 10.1016/0377-0427(87)90125-7 (cited on page 36).

[Roy] Daniel Roy, *Probabilistic-programming.org, a repository on probabilistic programming languages*, http://www.probabilistic-programming.org, Accessed: 2014 (cited on page 7).

[Rud93] Richard Rudell, 'Dynamic variable ordering for ordered binary decision diagrams', in *Proceedings of the 1993 IEEE/ACM International Conference on Computer-Aided Design, ICCAD 1993, Santa Clara, California, USA, November 7-11, 1993*, Michael R. Lightner and Jochen A. G. Jess, Eds., IEEE Computer Society, 1993, pages 42–47, ISBN: 0-8186-4490-7. DOI: 10.1145/259794.259802 (cited on page 86).

[SCCC10] Liwen Sun, Reynold Cheng, David W. Cheung and Jiefeng Cheng, 'Mining uncertain data with probabilistic guarantees', in *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, July 25-28, 2010*, Bharat Rao, Balaji Krishnapuram, Andrew Tomkins and Qiang Yang, Eds., ACM, 2010, pages 273–282, ISBN: 978-1-4503-0055-1. DOI: 10.1145/1835804.1835841 (cited on page 118).

[SD07] Prithviraj Sen and Amol Deshpande, 'Representing and querying correlated tuples in probabilistic databases', in *Proceedings of the 23rd International Conference on Data Engineering, ICDE 2007, Istanbul, Turkey, April 15-20, 2007*, Rada Chirkova, Asuman Dogac, M. Tamer Özsu and Timos K. Sellis, Eds., IEEE, 2007, pages 596–605. DOI: 10.1109/ICDE.2007.367905 (cited on page 22).

[Sha49] Claude. E. Shannon, 'The synthesis of two-terminal switching circuits', *Bell System Technical Journal*, volume 28, number 1, pages 59–98, 1949, ISSN: 1538-7305. DOI: 10.1002/j.1538-7305.1949.tb03624.x (cited on page 74).

[Sie02] Detlef Sieling, 'The nonapproximability of OBDD minimization', *Journal of Information and Computation*, volume 172, number 2, pages 103–138, 2002. DOI: 10.1006/inco.2001.3076 (cited on page 86).

[SJA11] Saket Sathe, Hoyoung Jeung and Karl Aberer, 'Creating probabilistic databases from imprecise time-series data', in *Proceedings of the 27th International Conference on Data Engineering, ICDE 2011, April 11-16, 2011, Hannover, Germany*, Serge Abiteboul, Klemens Böhm, Christoph Koch and Kian-Lee Tan, Eds., IEEE Computer Society, 2011, pages 327–338, ISBN: 978-1-4244-8958-9. DOI: 10.1109/ICDE.2011.5767838 (cited on page 121).

[Smy03] Peter P.A. Smyth, 'The thyroid, iodine and breast cancer', *Breast Cancer Research*, volume 5, number 5, pages 235–238, 2003 (cited on page 131).

[SORK11] Dan Suciu, Dan Olteanu, Christopher Ré and Christoph Koch, *Probabilistic Databases*, series Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2011. DOI: 10.2200/S00362ED1V01Y201105DTM016 (cited on page 1).

[TWG+11] Gareth A. Taylor, David C. H. Wallom, Sebastien Grenard, Angel Yunta Huete and Colin J. Axon, 'Recent developments towards novel high performance computing and communications solutions for smart distribution network operation', in *2nd IEEE PES International Conference and Exhibition on "Innovative Smart Grid Technologies", ISGT Europe 2011, Manchester, United Kingdom, December 5-7, 2011*, IEEE, 2011, pages 1–8, ISBN: 978-1-4577-1422-1. DOI: 10.1109/ISGTEurope.2011.6162812 (cited on page 103).

[Val79] Leslie G. Valiant, 'The complexity of computing the permanent', *Theoretical Computer Science*, volume 8, number 2, pages 189–201, 1979, ISSN: 0304-3975. DOI: 10.1016/0304-3975(79)90044-6 (cited on page 78).

[vDon00] Stijn Marinus van Dongen, 'Graph clustering by flow simulation', PhD thesis, University of Utrecht, Faculty of Mathematics and Computer Science, 2000 (cited on page 40).

[vGBLE10] Linda C. van der Gaag, Janneke Bolt, Willie Loeffen Loeffen and Armin Elbers, 'Modelling patterns of evidence in Bayesian networks: A case-study in classical swine fever', English, in *Computational Intelligence for Knowledge-Based Systems Design*, series Lecture Notes in Computer Science, Eyke Hüllermeier, Rudolf Kruse and Frank Hoffmann, Eds., volume 6178, Springer Berlin Heidelberg, 2010, pages 675–684, ISBN: 978-3-642-14048-8. DOI: 10.1007/978-3-642-14049-5_69 (cited on page 22).

[VPB03] Mark Van der Laan, Katherine Pollard and Jennifer Bryan, 'A new partitioning around medoids algorithm', *Journal of Statistical Computation and Simulation*, volume 73, number 8, pages 575–584, 2003. DOI: 10.1080/0094965031000136012 (cited on page 38).

[VRH+09] Peter Benjamin Volk, Frank Rosenthal, Martin Hahmann, Dirk Habich and Wolfgang Lehner, 'Clustering uncertain data with possible worlds', in *Proceedings of the 25th International Conference on Data Engineering, ICDE 2009, March 29 2009 - April 2 2009, Shanghai, China*, Yannis E. Ioannidis, Dik Lun Lee and Raymond T. Ng, Eds., IEEE, 2009, pages 1625–1632, ISBN: 978-0-7695-3545-6. DOI: 10.1109/ICDE.2009.174 (cited on page 2).

[vRos97] Guido van Rossum, 'A tour of the Python language', in *TOOLS 1997: 23rd International Conference on Technology of Object-Oriented Languages and Systems, July 28 - August 1, 1997, Santa Barbara, CA, USA*, IEEE Computer Society, 1997, page 370, ISBN: 0-8186-8383-X. DOI: 10.1109/TOOLS.1997.10001 (cited on page 8).

[vSO16] Sebastiaan J. van Schaik and Dan Olteanu, 'ENFrame: a framework for processing correlated probabilistic data', *ACM Transactions on Database Systems*, 2016, invited paper, accepted (December 2015), to be published (cited on page 53).

[WFH11] Ian H. Witten, Eibe Frank and Mark A. Hall, *Data Mining: Practical Machine Learning Tools and Techniques*, 3rd edition. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2011, ISBN: 0123748569, 9780123748560 (cited on page 3).

[WNZ01] Ji-Rong Wen, Jian-Yun Nie and Hong-Jiang Zhang, 'Clustering user queries of a search engine', in *Proceedings of the Tenth International World Wide Web Conference, WWW 2010, Hong Kong, China, May 1-5, 2001*, Vincent Y. Shen, Nobuo Saito, Michael R. Lyu and Mary Ellen Zurko, Eds., ACM, 2001, pages 162–168, ISBN: 1-58113-348-0. DOI: 10.1145/371920.371974 (cited on page 5).

[WS05] Wei Wei and Bart Selman, 'A new approach to model counting', in *Theory and Applications of Satisfiability Testing, 8th International Conference, SAT 2005, St. Andrews, UK, June 19-23, 2005, Proceedings*, Fahiem Bacchus and Toby Walsh, Eds., series Lecture Notes in Computer Science, volume 3569, Springer, 2005, pages 324–339, ISBN: 3-540-26276-8. DOI: 10.1007/11499107_24 (cited on page 88).

[YZNL14] Da Yan, Zhou Zhao, Wilfred Ng and Steven Liu, 'Probabilistic convex hull queries over uncertain data', *IEEE Transactions on Knowledge and Data Engineering*, volume 99, 2014, ISSN: 1041-4347. DOI: 10.1109/TKDE.2014.2340408 (cited on page 3).

[ZB07] Hui Zang and Jean C. Bolot, 'Mining call and mobility data to improve paging efficiency in cellular networks', in *Proceedings of the 13th Annual ACM International Conference on Mobile Computing and Networking*, series MobiCom '07, Montréal, Québec, Canada: ACM, 2007, pages 123–134, ISBN: 978-1-59593-681-3. DOI: 10.1145/1287853.1287868 (cited on page 1).

[ZLZL14] Xianchao Zhang, Han Liu, Xiaotong Zhang and Xinyue Liu, 'Novel density-based clustering algorithms for uncertain data', in *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada.*, Carla E. Brodley and Peter Stone, Eds., AAAI Press, 2014, pages 2191–2197, ISBN: 978-1-57735-661-5 (cited on page 136).

Appendix A

Glossary

Boolean random variables $x_i \in \mathbf{X}$. Throughout this thesis, the notation $x_i \in \mathbf{X}$ is used to refer to the Boolean random variables x_i in the set of random variables $\mathbf{X}$ used to annotate the input data with propositional expressions ($x_i \in \mathbf{X}$).

Certain data (point). A data point or set of data that does not bear any uncertainty. This term does not employ 'certain' as an adjective to refer to a data point (set) *of a specific but unspecified character, quantity, or degree* (Merriam Webster).

Decomposition (Shannon's expansion). A technique for computing the probability of a (possibly intractable) propositional expression containing Boolean random variables. See Section 5.2.

Decision tree. A type of binary decision diagram (BDD) which stores the trace of repeated decomposition. See Section 5.2.

Lineage. Event associated with input data, *e.g.* $\Phi[o_l]$ is the event or *lineage* of input object o_l.

Possible worlds (semantics). Under the *possible worlds semantics*, the input data set $\mathcal{D}$ to an algorithm is a probability distribution over a finite set of of *possible worlds*, whereby every world defines a data set of tuples $D_i \subseteq \mathcal{D}$.

The possible worlds of input objects to an algorithm are defined by the possible valuations $\alpha : \mathbf{X} \rightarrow \{true, false\}$ of Boolean random variables x_i in the set of random variables $\mathbf{X}$ used to annotate the input data with propositional expressions ($x_i \in \mathbf{X}$). See Example 2.5 on page 23 for an example.

Processing of (uncertain) data. Mining, querying, and storing uncertain (probabilistic) and certain data.

Satisfy an expression. A (partial or complete) valuation $\alpha : \mathbf{X} \rightarrow \{true, false\}$ of Boolean random variables $x_i \in \mathbf{X}$ *satisfies* an expression Φ if Φ evaluates to *true* under α.

Top-p most probable worlds. The possible worlds are defined by the possible valuations $\alpha : \mathbf{X} \rightarrow \{true, false\}$ of Boolean random variables $x_i \in \mathbf{X}$. Every world $\mathcal{W}_i$ induced by a valuation α_i has a probability $\Pr[\mathcal{W}_i] = \Pr[\alpha_i]$, which depends on the probabilities of the random variables $x_i \in \mathbf{X}$ and their valuations in α_i. The top-p most probable worlds are the p worlds with the highest probability.

Unsatisfy an expression. A (partial or complete) valuation $\alpha : \mathbf{X} \rightarrow \{true, false\}$ of Boolean random variables $x_i \in \mathbf{X}$ *unsatisfies* an expression Φ if Φ evaluates to *false* under α.

Appendix B

ENFrame source code and availability

ENFrame was implemented in C++, and is available under the Apache 2.0 License from a Git repository. This open-source license allows for commercial use, but does impose limitations on liability and requirements on redistribution. The full text of the license is available on the website of the Apache Foundation: www.apache.org/licenses/LICENSE-2.0.

A browseable Git repository that contains ENFrame's source code and documentation is located at https://git.traiectum.net/dphil/ENFrame. To clone this repository, point your favourite Git client to https://git.traiectum.net/dphil/enframe.git.

More information about the ongoing research on ENFrame can be found on the project website, www.cs.ox.ac.uk/projects/ENFrame, kindly hosted by the Department of Computer Science of the University of Oxford.

Appendix C

Side-by-side listing of user and event programs

```
(O, n) = loadData()            # list and number of objects
(k, it) = loadParams()         # number of clusters and iterations
M = init()                     # initialise centroids

for t in range(0,it):          # clustering iterations
  InCl = [None] * k            # assignment phase: assign objects to clusters
  for i in range(0,k):         # for every cluster i ...
    InCl[i] = [None] * n
    for l in range(0,n):       # ... and for every object l
      InCl[i][l] = reduce_and( # l is assigned to i if i has closest centroid
        [dist(O[l],M[i]) <= dist(O[l],M[j]) for j in range(0,k)]
      )
  InCl = breakTies2(InCl)      # each object is in exactly one cluster

  M = [None] * k               # update phase: compute new centroids
  for i in range(0,k):         # for every cluster i
    M[i] = scalar_mult(        # the new centroid is at the cluster centre
      invert(reduce_count([1 for l in range(0,n) if InCl[i][l]])),
      reduce2_sum([O[l] for l in range(0,n) if InCl[i][l]])
    )
```

(a) User program

$\vec{o}_0 \ldots \vec{o}_{n-1}$: n object vectors in the feature space

$\Phi[o_0] \ldots \Phi[o_{n-1}]$: n propositional object events

k, it: number of clusters and iterations

$\forall i \; in \; 0..n-1 : O^i \equiv \Phi[o_i] \otimes \vec{o}_i$

$M^0_{-1} \equiv \vec{o}_0; \ldots; M^{k-1}_{-1} \equiv \vec{o}_{(k-1)}$

$\forall t \; in \; 0..it-1 :$

$\forall i \; in \; 0..k-1 :$

$\forall l \; in \; 0..n-1 :$

$$InCl^{i,l}_t \equiv \bigwedge_{j=0}^{k-1} \left[dist\left(O^l, M^i_{t-1}\right) \leq dist\left(O^l, M^j_{t-1}\right) \right]$$

Encoding of `breakTies2` omitted

$\forall i \; in \; 0..k-1 :$

$$M^i_t \equiv \left(\sum_{l=0}^{n-1} InCl^{i,l}_t \otimes 1 \right)^{-1} \cdot \left(\sum_{l=0}^{n-1} InCl^{i,l}_t \otimes O^l \right)$$

(b) Event program

Algorithm C.1: Side-by-side listing of the user and event programs for k-means clustering

```
(O, n) = loadData()             # list and number of objects
(k, it) = loadParams()          # number of clusters and iterations
M = init()                      # initialise medoids

for t in range(0,it):           # clustering iterations
  InCl = [None] * k             # assignment phase: assign objects to clusters
  for i in range(0,k):          # for every cluster i ...
    InCl[i] = [None] * n
    for l in range(0,n):        # ... and for every object l
      InCl[i][l] = reduce_and(  # l is assigned to i if i has closest medoid
        [(dist(O[l],M[i]) <= dist(O[l],M[j])) for j in range(0,k)]
      )
  InCl = breakTies2(InCl)       # each object is in exactly one cluster

  M = [None] * k                # update phase: select new medoids
  for i in range(0,k):          # for every cluster i
    DistSum = [None] * n
    for l in range(0,n):        # compute total distance sum for every object l ...
      DistSum[l] = reduce_sum(  # ... to all other objects in cluster i
        [dist(O[l],O[p]) for p in range(0,n) if InCl[i][p]]
      )

    IsMedoid = [None] * n;
    for l in range(0,n):        # determine whether object is new medoid
      IsMedoid[l] = reduce_and( # object is new medoid if it has min. distance sum
        [DistSum[i][l] <= DistSum[i][p] for p in range(0,n) if InCl[i][p]]
      ) and InCl[i][l]          # and is in the cluster
    IsMedoid = breakTies(IsMedoid)

    M[i] = reduce_sum([O[l] for l in range(0,n) if IsMedoid[l]])
```

(a) User program

$\vec{o_0} \ldots \vec{o_{n-1}}$: n object vectors in the feature space
$\Phi[o_0] \ldots \Phi[o_{n-1}]$: n propositional object events
k, it: number of clusters and iterations

$\forall i \text{ in } 0..n-1 : O^i \equiv \Phi[o_i] \otimes \vec{o_i}$

$M_{-1} \equiv initialiseMedoids()$

$\forall t \text{ in } 0..it-1 :$

$\forall i \text{ in } 0..k-1 :$

$\forall l \text{ in } 0..n-1 :$

$$InCl_t^{i,l} \equiv \bigwedge_{j=0}^{k-1} \left[dist\left(O^l, M_{t-1}^i\right) \leq dist\left(O^l, M_{t-1}^j\right) \right]$$

Encoding of `breakTies2` omitted

$\forall i \text{ in } 0..k-1 :$

$\forall l \text{ in } 0..n-1 :$

$$DistSum_t^{i,l} \equiv \sum_{p=0}^{n-1} \left[InCl_t^{i,p} \otimes dist\left(O^l, O^p\right) \right]$$

$\forall l \text{ in } 0..n-1 :$

$$IsMedoid_t^{i,l} \equiv \bigwedge_{p=0}^{n-1} \left[DistSum_t^{i,l} \leq \left(InCl_t^{i,p} \otimes DistSum_t^{i,p} \right) \right] \wedge InCl_t^{i,l}$$

Encoding of `breakTies` omitted

$$M_t^i \equiv \sum_{l=0}^{n-1} \left[IsMedoid_t^{i,l} \otimes O^l \right]$$

(b) Event program

Algorithm C.2: Side-by-side listing of the user and event programs for k-medoids clustering

```
(n, M) = loadData()          # M: stochastic n*n matrix of edge weights
(r, it) = loadParams()       # r: inflation parameter, it: number of iterations

for t in range(0,it):
  N = [None] * n             # expansion phase
  for i in range(0,n):
    N[i] = [None] * n
    for j in range(0,n):     # compute matrix square result:
      N[i][j] = reduce_sum([M[i][k] * M[k][j] for k in range(0,n)])

  M = [None] * n             # inflation phase
  for i in range(0,n):
    M[i] = [None] * n
    for j in range(0,n):     # inflate and re-normalise values
      M[i][j] = pow(N[i][j],r) *
            invert(reduce_sum([pow(N[i][k],r) for k in range(0,n)]))

InCl = [None] * n            # interpret M to obtain clustering result
for i in range (0,n):
  InCl[i] = [None] * n
  for j in range (0,n):      # determine whether vertex j belongs to cluster i
    InCl[i][j] = M[i][j] > 0
```

(a) User program

$\mathcal{M}$: stochastic $n \times n$ matrix with edge weights
$\Phi[o_0] \ldots \Phi[o_{n-1}]$: n propositional object (vertex) events
r, it: inflation parameter and number of iterations

$$\forall i \text{ in } 0..n-1: \quad \forall j \text{ in } 0..n-1: \quad M_{-2}^{i,j} \equiv \Big(\Phi[o_i] \wedge \Phi[o_j]\Big) \otimes \mathcal{M}^{i,j}$$

$$\forall i \text{ in } 0..n-1: \quad \forall j \text{ in } 0..n-1: \quad M_{-1}^{i,j} \equiv M_{-2}^{i,j} \cdot \left(\sum_{k=0}^{n-1} M_{-2}^{i,k}\right)^{-1}$$

$$\forall t \text{ in } 0..it-1:$$

$$\forall i \text{ in } 0..n-1:$$

$$\forall j \text{ in } 0..n-1: \quad N_t^{i,j} \equiv \sum_{k=0}^{n-1} \left[M_{t-1}^{i,k} \cdot M_{t-1}^{k,j}\right]$$

$$\forall i \text{ in } 0..n-1:$$

$$\forall j \text{ in } 0..n-1: \quad M_t^{i,j} \equiv \left(N_t^{i,j}\right)^r \cdot \left(\sum_{k=0}^{n-1} \left(N_t^{i,k}\right)^r\right)^{-1}$$

$$\forall i \text{ in } 0..n-1:$$

$$\forall j \text{ in } 0..n-1: \quad InCl^{i,j} \equiv M_{it-1}^{i,j} > 0$$

(b) Event program

Algorithm C.3: Side-by-side listing of the user and event programs for Markov clustering

```
# U = unlabelled points, nu = number of unlabelled points in U
# L = labelled points, nl = number of labelled points in L
# C = class of each labelled point, nc = number of classes
# SLN[u] = sorted list of labelled neighbours for object u in U
(U, nu, L, nl, C, nc, SLN) = loadData()
(k) = loadParams()                  # k: no. of nearest neighbours to consider

Votes = [None] * nl;                # prepare votes of labelled objects
for l in range (0, nl):             # iterate over labelled objects
  Votes[l] = [None] * nc            # votes from object l
  for i in range (0, nc):           # iterate over possible classes
    Votes[l][i] = 1 + (1 if C[l] == i else 0)  # 1/2-vote for class i

ClassAssign = [None] * nu;          # stores class assignments
for u in range (0, nu):             # iterate over unlabelled points
  VoteSum = [None] * nc             # sum of votes by nearest neighbours
  for i in range (0, nc):           # iterate over possible classes
    NearestVotes = [None] * nc;     # sort votes: nearest neighbours first
    for m in range (0, nl):
      NearestVotes[m] = Votes[SLN[u][m]][i]   # vote from mth nearest neighbour

    VoteSum[i] = reduce_sum(select_first(k, NearestVotes))  # sum first k votes

  ClassAssign[u] = [None] * k       # class assignments for point u
  for i in range (0, nc):           # iterate over classes
    ClassAssign[u][i] = reduce_and(  # assignment to class i
      VoteSum[i] >= VoteSum[p] for p in range (0, nc)
    )

  ClassAssign[u] = breakTies(ClassAssign[u])   # break ties
```

```
# n_l    = number of labelled data points
# n_c    = number of classes
# C^l    = class of labelled data point l  (0 ≤ C^l < n_c)
# N_u^m  = m^th closest neighbour of unlabelled point o_u

∀l in 0...(n_l − 1) :

    ∀i in 0...(n_c − 1) :
        Votes^{l,i} ≡ Φ[o_l] ⊗ (1 + ((C^l = i) ⊗ 1))

∀u in 0...(n_u − 1) :

    ∀i in 0...(n_c − 1) :

        ∀m in 0...(n_l − 1) :
            NearestVotes_u^m ≡ Votes^{N_u^m, i}

        VoteSum_u^i ≡ Σ F_k(NearestVotes_u)

    ∀i in 0...(n_c − 1) :
        ClassAssign_u^i ≡ ⋀_{p=0}^{n_c−1} [VoteSum_u^i ≥ VoteSum_u^p]

# definition of breakTies omitted
```

Algorithm C.4: Side-by-side listing of the user and event programs for k-nearest neighbour classification.

Appendix D

Detailed rewritings of expressions

Event expressions in Example 4.3

The expanded medoid expressions in Example 4.3 are:

$$\Phi\Big[M^0 = o_0\Big] \equiv \Phi[o_0] \;\equiv\; x_0 \vee x_2$$

$$\Phi\Big[M^0 = o_1\Big] \equiv \Phi[o_1] \wedge \neg\Phi\Big[M^0 = o_0\Big] \;\equiv\; \Phi[o_1] \wedge \neg\Phi[o_0] \;\equiv\; x_1 \wedge \neg(x_0 \vee x_2)$$

$$\Phi\Big[M^0 = o_2\Big] \equiv \Phi[o_2] \wedge \neg\Phi\Big[M^0 = o_0\Big] \wedge \neg\Phi\Big[M^0 = o_1\Big] \;\equiv\; x_3 \wedge \neg(x_0 \vee x_2) \wedge \neg(x_1 \wedge (\neg x_0 \vee x_2)) \;\equiv$$

$$\equiv x_3 \wedge \neg(x_0 \vee x_2) \wedge \neg x_1 \quad (\equiv\; \Phi[o_2] \wedge \neg\Phi[o_0] \wedge \neg\Phi[o_1])$$

$$\Phi\Big[M^0 = o_3\Big] \equiv \Phi[o_3] \wedge \neg\Phi\Big[M^0 = o_0\Big] \wedge \neg\Phi\Big[M^0 = o_1\Big] \wedge \neg\Phi\Big[M^0 = o_2\Big] \;\equiv$$

$$\equiv (\neg x_1 \wedge x_3) \wedge \neg(x_0 \vee x_2) \wedge \neg(x_1 \wedge (\neg x_0 \vee x_2)) \wedge \neg(x_3 \wedge \neg(x_0 \vee x_2) \wedge \neg x_1) \;\equiv$$

$$\equiv \neg x_1 \wedge x_3 \wedge \neg x_0 \wedge \neg x_2 \wedge \neg x_3 \;\equiv\; \bot$$

$$\Phi\Big[M^1 = o_0\Big] \equiv \Phi[o_0] \wedge \neg\Phi\Big[M^0 = o_0\Big] \;\equiv\; \Phi[o_0] \wedge \neg\Phi[o_0] \;\equiv\; \bot$$

$$\Phi\Big[M^1 = o_1\Big] \equiv \Phi[o_1] \wedge \neg\Phi\Big[M^1 = o_0\Big] \wedge \neg\Phi\Big[M^0 = o_1\Big] \;\equiv\; x_1 \wedge \neg\bot \wedge \neg(x_1 \wedge \neg(x_0 \vee x_2)) \;\equiv$$

$$\equiv x_1 \wedge (x_0 \vee x_2)$$

$$\Phi\Big[M^1 = o_2\Big] \equiv \Phi[o_2] \wedge \neg\Phi\Big[M^1 = o_0\Big] \wedge \neg\Phi\Big[M^1 = o_1\Big] \wedge \neg\Phi\Big[M^0 = o_2\Big] \;\equiv$$

$$\equiv x_3 \wedge \neg\bot \wedge \neg(x_1 \wedge (x_0 \vee x_2)) \wedge \neg(x_3 \wedge \neg(x_0 \vee x_2) \wedge \neg x_1) \;\equiv$$

$$\equiv x_3 \wedge \neg(x_1 \wedge (x_0 \vee x_2)) \wedge \neg(\neg(x_0 \vee x_2) \wedge \neg x_1)$$

$$\equiv x_3 \wedge \neg(x_1 \wedge (x_0 \vee x_2)) \wedge (x_0 \vee x_1 \vee x_2)$$

$$\Phi\Big[M^1 = o_3\Big] \equiv \Phi[o_3] \wedge \neg\Phi\Big[M^1 = o_0\Big] \wedge \neg\Phi\Big[M^1 = o_1\Big] \wedge \neg\Phi\Big[M^1 = o_2\Big] \wedge \neg\Phi\Big[M^0 = o_3\Big] \;\equiv$$

$$\equiv (\neg x_1 \wedge x_3) \wedge \neg\bot \wedge \neg(x_1 \wedge (x_0 \vee x_2)) \wedge \neg(x_3 \wedge \neg(x_1 \wedge (x_0 \vee x_2)) \wedge (x_0 \vee x_1 \vee x_2)) \wedge \neg\bot$$

$$\equiv (\neg x_1 \wedge x_3) \wedge \neg(x_1 \wedge (x_0 \vee x_2)) \wedge \neg(x_3 \wedge \neg(x_1 \wedge (x_0 \vee x_2)) \wedge (x_0 \vee x_1 \vee x_2)) \;\equiv$$

$$\equiv (\neg x_1 \wedge x_3) \wedge \neg(x_3 \wedge \neg(x_1 \wedge (x_0 \vee x_2)) \wedge (x_0 \vee x_1 \vee x_2)) \;\equiv$$

$$\equiv (\neg x_1 \wedge x_3) \wedge \neg(x_3 \wedge (x_0 \vee x_2)) \;\equiv\; (\neg x_1 \wedge x_3) \wedge \neg(x_0 \vee x_2) \;\equiv$$

$$\equiv (\neg x_1 \wedge x_3) \wedge (\neg x_0 \wedge \neg x_2) \;\equiv\; \neg x_0 \wedge \neg x_1 \wedge \neg x_2 \wedge x_3$$

For:

$$\Phi[o_0] = x_0 \vee x_2 \qquad \Phi[o_1] = x_1$$

$$\Phi[o_2] = x_3 \qquad \Phi[o_3] = \neg x_1 \wedge x_3$$