Harnessing uncertain data structure

Mouhamadou Lamine Ba

ParisTech
INSTITUT DES SCIENCES ET TECHNOLOGIES
PARIS INSTITUTE OF TECHNOLOGY

TELECOM ParisTech

2015-ENST-0013

EDITE - ED 130

Doctorat ParisTech

T H È S E

pour obtenir le grade de docteur délivré par

TELECOM ParisTech

Spécialité “Informatique et Réseaux”

présentée et soutenue publiquement par

Mouhamadou Lamine Ba

le 30 Mars 2015

Exploitation de la structure des données incertaines

Directeur de thèse: **Talel ABDESSALEM**
Co-directeur de thèse: **Pierre SENELLART**

Jury

M. Talel ABDESSALEM, Professeur, Télécom ParisTech	Directeur de thèse
Mme. Laure BERTI-ÈQUILLE, Chercheur senior, QCRI	Rapporteur
M. Dario COLAZZO, Professeur, Université Paris Dauphine	Examinateur
M. Stephane GANÇARSKI, Maître de Conférences HDR, Université Paris VI	Examinateur
M. Pascal MOLLI, Professeur, Université de Nantes	Rapporteur
M. Benjamin NGUYEN, Professeur, INSA Centre-Val de Loire	Examinateur
M. Pierre SENELLART, Professeur, Télécom ParisTech	Co-directeur de thèse

TELECOM ParisTech
école de l’Institut Mines-Télécom - membre de ParisTech
46 rue Barrault 75013 Paris - (+33) 1 45 81 77 77 - www.telecom-paristech.fr

T H È S E

Exploitation de la structure des données incertaines

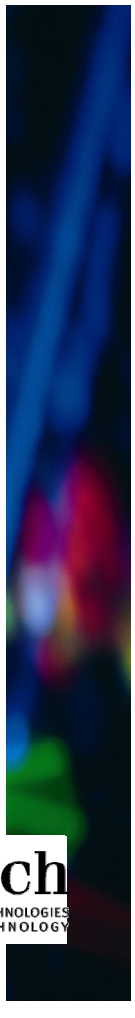
Mouhamadou Lamine Ba

RESUME : Cette thése s'intéresse à certains problémes fondamentaux découlant d'un besoin accru de gestion des incertitudes dans les applications Web multi-sources ayant de la structure, à savoir le contrôle de versions incertaines dans les plates-formes Web à large échelle, l'intégration de sources Web incertaines sous contraintes, et la découverte de la vérité à partir de plusieurs sources Web structurées. Ses contributions majeures sont : la gestion de l'incertitude dans le contrôle de versions de données arborescentes en s'appuyant sur un modéle XML probabiliste ; les étapes initiales vers un système d'intégration XML probabiliste de sources Web incertaines et dépendantes ; l'introduction de mesures de précision pour les données géographiques et ; la conception d'algorithmes d'exploration pour un partitionnement optimal de l'ensemble des attributs dans un processus de recherche de la vérité sur des sources Web conflictuelles.

MOTS-CLEFS : applications Web multi-sources, structure, gestion de l'incertitude, contrôle de versions, intégration de données, dépendances, objets mobiles, XML probabiliste, découverte de la vérité, attributs corrélés.

ABSTRACT: This thesis addresses some fundamental problems inherent to the need of uncertainty handling in multi-source Web applications with structured information, namely uncertain version control in Web-scale collaborative editing platforms, integration of uncertain Web sources under constraints, and truth finding over structured Web sources. Its major contributions are: uncertainty management in version control of tree-structured data using a probabilistic XML model; initial steps towards a probabilistic XML data integration system for uncertain and dependent Web sources; precision measures for location data and; exploration algorithms for an optimal partitioning of the input attribute set during a truth finding process over conflicting Web sources.

KEY-WORDS: multi-source Web systems, structure, uncertainty handling, version control, data integration, dependency, moving objects, probabilistic XML, truth finding, attribute correlations.



TELECOM ParisTech

EDITE

ParisTech
INSTITUT DES SCIENCES ET TECHNOLOGIES
PARIS INSTITUTE OF TECHNOLOGY

Contents

List of Figures

List of Tables

Acknowledgements

This PhD would not be possible without the support of many people. Foremost, I would like to thank my supervisors, Talel Abdessalem and Pierre Senellart, for giving me the possibility to do this PhD research. They allowed me to follow my own research interests without lot of restrictions. They have always trusted in me and in my work and pushed me to broaden my research background through various collaborations and projects on which I have been involved. Their continuous and valuable insights helped me to sharpen this thesis up.

I would also like to thank the member of my PhD committee for accepting to take on that charge, and, particularly, Pascal Molli and Laure Berti-Equille, for reviewing this thesis.

Almost all of the time, I did really enjoy being member of Télécom ParisTech, especially of the DbWeb research group of the department of Computer Science and Networks. Apart of the excellent research conditions, the staff played a major role. I would like to thank Antoine Amarilli, Oana Bălălău, Jean-louis Dessalles, Luis Galárraga, Modou Gueye, Jean-Benoit Griesner, Roxana Horincar, Ziad Ismail, Sébastien Montenez, Mauro Sozio, Danai Symeonidou, and Fabian Suchaneck for their kindness and sociability. Some of my colleagues have contributed to my thesis. Sébastien Montenez, a computer scientist, implemented a Web crawler for information in the maritime domain and set up the maritime traffic application. Roxana Horincar, a postdoctoral fellow, has been actively involved in the truth finding work, in particular by extensively evaluating the precisions of the proposed algorithms against various types of datasets. It was a pleasure and a honor to me to work with them. My office-mate Ziad Ismail allowed me for discussing topics not necessarily related to the domain of computer science. He become a real friend.

My acknowledgements also go to people with which I had the chance to collaborate in other contexts. I have to thank Antoine Amarilli and Ruiming Tang for accepting to exchange and explore some aspects of data management related to their PhD topics. A particular thank to Huayu Wu which offered me the possibility to be enrolled in a postgraduate attachment program within of the Institute for Infocomm Research Institute of Singapore (A*STAR) during three month. He hosted me in his house and has been always present when needed during my stay in Singapore. Many thanks also to Stephane Bressan for his kindness and helpful suggestions during my two visits in Singapore.

At last, I have to be grateful with my family. My mother Seynabou has done its utmost to offer me a good education, and continues to be of a great support for me. My aunt Marie has been prominent throughout my life as a friend and a confidant.

1 Introduction

Providing high-quality content or facilitating a uniform access to content, through effective tools, is undoubtedly one major contribution to the renown of the World Wide Web, as evidenced by an increasing irrational trust in Web data sources for personal and business purposes. The latest advances to the Web are notably represented by the emergence of the *multi-source* paradigm, i.e., Web applications which, rather than relying just on one particular data source, exploit information from various different sources in order to construct valuable data or to ease access to them.

Multi-source Web applications cover nowadays a broad portion of the inestimable amount of data on the Web. These applications can be of various nature with respect to the inherent manner sources are involved for content production. Multi-source Web systems may correspond to *Web-scale collaborative systems* (Wikipedia [1] or Google Drive [2]) where sources, i.e., contributors, are active actors for content creation, or to *domain-specific Web systems* (Google Hotel Finder [3], Yahoo! Finance [4], MarineTraffic [5], etc.) in which data that already reside on multiple heterogeneous sources are unified. The success of both types of Web applications is greatly due to an integration of views from multiple structured sources, the level with which their information – given its structuring – corroborate each other being a qualitative clue about the relevance of the result. As telling examples, the online-encyclopedia Wikipedia is built on a quite large community [6] of active contributors, over structured articles, even though its size has stopped to grow during the last years [Voss, 2005, Halfaker et al., 2013]. Domain-specific Web applications such as travel booking Websites aggregate data from thousands of sources with well-known templates, including hundred Web portals of hotels, flight companies, etc. Relying on multiple sources, thereby a wider spectrum of different data

1. http://www.wikipedia.org/
2. https://www.google.com/drive/
3. https://www.google.com/hotels/
4. http://finance.yahoo.com/
5. https://www.marinetraffic.com/en/
6. The English version of Wikipedia, for instance, counts around $31,000$ active editors besides its vast number of non-registered occasional contributors.

quality level and a higher probability of conflicts, inevitably leads to some expectations in terms of sources' trustworthiness and data relevance levels. This requires efficient uncertainty management and data quality estimation algorithms during data integration. Uncertain data integration is indeed a building block for multi-source Web applications, as few will continue to trust these systems if users cannot find the offered data without uncertainties and relevant to their needs.

Despite considerable progress, various fundamental questions, driven by the need for uncertainty management in multi-source Web applications, are not yet elucidated. Collaborative editing systems in which content relevance and contributors' reliability level, gradually impacting the quality and the integrity of the process, are modeled and assessed throughout the editing process are in their infancy, as existing systems mostly resort to revision control or bots for repairing trivial edit problems [Geiger and Halfaker, 2013] or to trust algorithms for off-line reputation evaluation [Adler and de Alfaro, 2007, De La Calzada and Dekhtyar, 2010] and social link analysis [Maniu et al., 2011a, Maniu et al., 2011b]. Web-scale collaborative systems using revision control constrain the existence of data to follow some derivation rules given by the history of the evolution of the editing process over the shared content. As a more general problem, research on uncertain and heterogeneous Web data integration in real-world applications which capture and evaluate the amount of uncertainty in data under complex constraints – beyond for instance simple mutual exclusion [Van Keulen et al., 2005] – such as data dependency between sources and spatio-temporal restrictions, is still limited. With constraints and more structured information, uncertain data integration is expected to have the ability to do more in modeling and quantifying uncertainties for multi-source Web applications. In particular, it should be able to discover the truth between multiple conflicting Web sources. Truth finding, accounting for structural correlations, in terms of different source quality levels, among subsets of data attributes is not yet fully realized, as existing algorithms estimate a global indicator of the level of accuracy of each source [Li et al., 2012, Waguih and Berti-Equille, 2014].

It becomes apparent that in order to enhance the quality of data provided by multi-source Web applications, uncertainty management is fundamental. There is, particularly, a need for techniques with the ability to (i) *efficiently and dynamically manage uncertainty for shared content*, with the support of data provenance and structure tracking, more semantical information about the trustworthiness levels of sources and the relevance of provided content for enhancing the quality of the collaborative process; (ii) *integrate multiple uncertain Web sources under constraints*, supporting a broader class of constraints in a principled manner that properly models source corroborations and contradictions

for further uncertainty estimation with the representation of real data provenance for provenance queries and efficient view maintenance; (iii) *effectively quantify content relevance and source reliability levels*, for content filtering and recommendation, by accounting for domain-specific knowledge, the nature of handled information, and the structure of data.

In this thesis, we focus on the following important sub-problems: uncertainty management in content built in a collaborative manner, the integration of uncertain Web sources in domains where copying is a real concern, and the effective estimation of content relevance and source accuracy given constraints, with methods which, putting it all together, also form building blocks towards a general integration framework for uncertain and heterogeneous multiple data source management on the Web.

More precisely, we address the aspects below that are fundamental for uncertain data management in multi-source Web applications.

1. We start by studying the problem of modeling and assessing uncertainty in version control within collaborative editing systems. Version control in which modeling and assessing uncertainty are considered as main parts of the process goes further than only finding ways to directly plug confidence estimators [Maniu et al., 2011a, De La Calzada and Dekhtyar, 2010] into the system or checking, with bots or human [Geiger and Halfaker, 2013], the relevance of data versions. This requires a more subtle method in the sense that the amount of uncertainty in each piece of information depends not only on the trustworthiness level of the provider, but also on its relevance and of the consensus around it, which is conditioned by the history of editions over it: the semantics of an edit implicitly indicates the view or opinion (agreement or disagreement) of the responsible contributor during a collaborative editing. As a result, version control with uncertainty is not only checking whether or not the content has been modified. As the implicit opinions of contributors, expressed through version control operations, are crucial for evaluating the amount of uncertainty in data, there is a requirement to properly and efficiently capture, e.g., using the structure of data, the semantics of these operations.

   The publications relevant for this study are [Ba et al., 2013a], [Ba et al., 2013b], and [Ba et al., 2014c] respectively at the ACM International Symposium on Document Engineering (DocEng), the DocEng Workshop on Document Changes, and the French Revue "Ingénerie des Systèmes d'Information".

2. Then, we consider uncertain Web data integration, in its ability to model and assess uncertainties given some constraints. As a natural connection with our first studied

problem, we initially consider uncertain Web data integration where sources are related by copying relationships: a given source copies information from another one, and possibly revises it in order to provide its own knowledge about the real world. In such a scenario, reconciling uncertain Web sources requires techniques stronger than actual data integration systems, e.g., [Van Keulen et al., 2005], which mostly assume independent sources. Indeed, dependencies impact the real provenance of data, data corroborations, and correlations. Therefore, uncertain data integration with dependent Web sources must account for these aspects which undoubtedly affect our belief about the amount of uncertainty in each piece of the merged content; data provenance can be inferred given dependencies and the structure of data. In a second stage, we study the case where constraints are spatio-temporal, which occurs when Web sources provide information about moving objects. This drives the needs for specific techniques, aware of geographical data, that evaluate the precision of spatio-temporal data during the integration process. This evaluation cannot be done as usual, e.g., similarly to textual information.

The publications relevant for this study are [Ba et al., 2014a] and the demonstration [Ba et al., 2014b] respectively at the DASFAA[7] *Workshop on Uncertain and Crowdsourced Data and the International Conference on Advances in Geographic Information Systems.*

3. As one of the main purposes of data integration with uncertainty management, we conclude by the problem of truth finding over structured Web sources, in the perspective of taking advantage of structural correlations among data attributes. It requires techniques, without the independence assumption between the probabilities of correctness of values of distinct data attributes made by actual algorithms [Li et al., 2012, Waguih and Berti-Equille, 2014], which effectively and efficiently discover the truth based on structural correlations among data attributes. The accuracy level of a source for providing true values, instead of being the same regardless the given subset of data attributes, may be variable – such an accuracy is consistent when attributes are structurally correlated; for instance, when the source is highly accurate on some subsets of data attributes while being worse on other ones, when clustering the attributes in a certain manner. This observation cannot be left aside when one wants to provide a truth finding process in which the estimation of the probabilities of correctness of data attribute values is not biased by the use of a global accuracy values for sources.

7. International Conference on Database Systems for Advanced Applications.

*This is an ongoing study with preliminary results not yet submitted for publication. A part of this work is the result of collaborating with the Data Analytics Group of the Institute for Infocomm Research in Singapore (A*STAR) during a postgraduate research attachment.*

The next three sections present an overview of the motivations and our contributions to these problems. We give a detailled description of them in the subsequent chapters which are split into two separate parts.

1.1 Uncertain Multi-Version Tree Data

As highlighted earlier, Web-scale collaborative systems form a significant type of multi-source Web applications with well-known examples such as the free encyclopedia Wikipedia and the Google Drive collaboration tools. Content management, in many collaborative editing systems, especially online platforms, is realized within a version control setting. A typical user, willing to build knowledge around a given subject by involving views from other persons, creates and shares an initial version of the content, possibly consisting of only the title, for further revisions. The version control system then tracks the subsequent versions of the shared content, as well as changes, in order to enable fixing error made in the revision process, querying past versions, and integrating content from different contributors. Much effort related to version control has been carried out both in research and in applications; see surveys [Altmanninger et al., 2009, Koc and Tansel, 2011]. The prime applications were collaborative document authoring process, computer-aided design, and software development systems. Currently, powerful version control tools, such as Subversion [Pilato, 2004] and Git [Chacon, 2009], efficiently manage large source code repositories and shared file-systems.

Unfortunately, existing approaches leave no room for uncertainty handling, for instance, uncertain data resulting from conflicts. Contradictions are usual during a collaborative job – in particular when the setting is open – as appearing whenever concurrent edits try changing the identical content leading to ambiguities in content management. But sources of uncertainties in the version control process are not only due to conflicts. In fact, Web-scale collaborative editing platforms are inherently uncertain: platforms such as Wikipedia or Google Drive enable unbounded interactions between a large number of contributors, without prior knowledge of their level of expertise and reliability. Uncertainty, in such environments, is omnipresent due to the unreliability of the sources, the incompleteness and imprecision of the contributions, the possibility of malicious editing and vandalism acts, etc. This drives the need for an entirely new way to manage

the different versions of the shared content and to access them. In other terms, a version control technique able to properly handle uncertain data may be very useful for this class of applications, as we illustrate next with two applications scenarios.

Uncertainty in Wikipedia Versions Some Web-scale collaborative systems such as Wikipedia have no write-access restrictions over documents. As a result, multi-version documents include data from different users. Wikipedia is built on a quite solid and large community of active contributors, ensuring a continuous increase of the number of the articles and the editions over them [8], even though the growth rate of this community has slowed during these last years [Voss, 2005, Halfaker et al., 2013, Simonite, 2013]. The open and free features lead to contributions with variable reliability and consistency depending both on the contributors' expertise (e.g., novice or expert) and the scope of the debated subjects. At the same time, edit wars, malicious contributions like spams, and vandalism acts can happen at any time during document evolution. Therefore, the integrity and the quality of each article may be strongly altered. Suggested solutions to these critical issues are reviewing access policies for articles discussing hot topics, or quality-driven solutions based on the reputations of authors, statistics on frequency of content change, or the trust a given reader has on the information [Adler and de Alfaro, 2007, De La Calzada and Dekhtyar, 2010, Maniu et al., 2011a]. But restricting editions on Wikipedia articles to a certain group of privileged contributors – resulting in a reticence of an important part of the contributors, e.g., newcomers, to get more involved [Halfaker et al., 2013] – does not suppress the necessity of representing and assessing uncertainties. Indeed, edits may be incomplete, imprecise or uncertain, showing partial views, misinformations or subjective opinions. The reputation of contributors or the confidence level on sources are useful information towards a quantitative evaluation of the quality of versions and even more of each atomic contribution. However, a prior efficient representation of uncertainty across document versions remains a prerequisite.

User Preference at Visualization Time Filtering and visualizing content are also important features in collaborative environments. In Wikipedia, users are not only contributors, but also consumers, interested in searching and reading information on multi-version articles. Current systems constrain the users to visualize either the latest revision of a given article, even though it may not be the most relevant, or the version at a

8. About half of the active editors in Wikipedia spend at least one hour a day editing, and a fifth spend more than three hours [Simonite, 2013].

specific date. The seekers of information, especially in universal knowledge management platforms like Wikipedia, may want to easily access more relevant versions or those of authors whom they trust. Filtering unreliable content is one of the benefits of being able to handle uncertain data. It can be achieved easily by hiding the contributions of the offending source, for instance when a vandalism act is detected, or at query time to fit user preferences and trust in the contributors. Alternatively, to deal with misinformation, it seems useful to provide versions to users with information about their amount of uncertainty and the uncertainty of each part of their content. Last but not least, users at visualization time should be able to search for a document representing the outcome of combining parts (e.g., some of them might be incomplete, imprecise, and even uncertain taken apart) from different versions. We demonstrated in [Abdessalem et al., 2011] an application of these new modes of interaction to Wikipedia revisions: an article is no longer considered as the last valid revision, but as a merge of all possible (uncertain) revisions.

Contribution Version control is primordial in uncertain Web-scale collaborative systems. Representing and evaluating uncertainties throughout data version management becomes thus crucial for enhancing collaboration and for overcoming problems such as conflict resolution and information reliability management.

In this work, we propose an uncertain XML version control model tailored to multi-version tree-structured documents in open collaborative editing contexts. Data, that is, office documents, HTML or XHTML documents, structured Wiki formats, etc., manipulated within the given application scenarios are tree-like or can be easily translated into this form; XML is a natural encoding for tree-structured data.

We propose an uncertain version control framework by starting – as we shall detail in Chapter 2 – with an effective abstraction of uncertain multi-version tree-structured documents. In our model, we handle uncertain data through a probabilistic XML model as a basic component of our version control framework. Each version of a shared document is represented by an XML tree. At the abstract level, we consider a multi-version XML document with uncertain data based on random events, XML edit scripts attached to them, and a directed acyclic graph of these events. For a concrete representation the whole document, with its different versions, is modeled as a probabilistic XML document representing an XML tree whose edges are annotated by propositional formulas over random events. Each propositional formula models both the semantics of uncertain editions (insertion and deletion) performed over a given part of the document and its provenance in the version control process. Uncertainty is evaluated using the probabilistic

model and the reliability measure associated to each source, each contribution, or each editing event, resulting in an uncertainty measure on each version and each part of the document. The directed acyclic graph of random events maintains the history of document evolution by keeping track of its different states and their derivation relationships.

We then study the support of standard version control operations – update and merge operations in particular – with a focus on efficiency and effectiveness of their translation within our model. We formalize the semantics of the standard update operations over an uncertain multi-version XML document, and show that they can be implemented directly as an operation on the probabilistic XML encoding by devising the corresponding algorithm. We demonstrate the efficiency of our update algorithm – and of our version control model in general – with respect to deterministic version control systems like Git and Subversion on real datasets. Chapter 3 revisits and describes, in a deeper manner, updates over uncertain multi-version XML documents and realizes a performance analysis of our model.

Merging is a fundamental operation in revision control systems that enables integrating different changes made to the same documents. This operation is particularly helpful for software configuration management, where the configurations and their components can be built based on a combination of different versions of different elementary parts of the software (see [Estublier, 2000]). In Web-scale collaborative platforms, the merging operation as known in traditional version control systems is not yet supported. Its use in this kind of environment (large-scale, open and collaborative) is of interest – as soon as concurrent editing and alternative revisions are allowed – since it eases the creation of personalized content that fit users' preferences or to automate the tedious and error-prone manual merge of Wikipedia's articles which overlap (articles related to the same topic or sharing a large common part). We propose a merge operation – accounting for the uncertainty associated to merged data – which complements our framework and enables the conciliation of uncertain versions. We devise an efficient algorithm that implements the merge operation and prove its correction (see Chapter 4 for merging in uncertain multi-version XML).

1.2 Web Data Integration under Constraints

Integrating data from multiple heterogeneous sources is one of the most common tasks in classical multi-source Web systems. For example, many platforms on the Web, and in particular domain-specific applications like online hotel reservation services, use

and integrate data from hundreds of different sources [9]. More importantly such platforms, aside their own specific ways of gathering information, often try capitalizing on data already on the Web – nowadays, there is a proliferation of Web systems, e.g., sharing platforms, company Websites, or general Websites, which collect and keep up-to-date as much as possible a vast amount of information about various real-life areas – for their needs in order to meet users' expectations. It is known that Web sources are mostly untrustworthy and their data usually come with uncertainties; for instance, sources can be unreliable, data can be obsolete or extracted using imprecise automatic methods. At the same time, dependent sources – copying between sources is a reality on the Web as described in [Dong et al., 2010] – or geographical information (in the case of moving objects) may be involved in the data integration process, and these aspects should be carefully taken into consideration. When integrating uncertain data, the relationships between sources or the intrinsic nature of the information may indeed influence both the modeling and the assessment of uncertainties.

We consider in this work the problem of uncertain data integration given some constraints. We first study how to reconcile uncertain and dependent Web sources, and then we tackle the problem in the presence of spatio-temporal information.

We present further two application scenarios – fairly self-explanatory – from *monitoring moving objects using uncertain Web data*. The first shows, on the one hand, the fact that dependencies constrain data provenance and how the data are corroborated and contradicted between sources which are key ingredients for uncertainty handling. The second scenario illustrates that the amount of uncertainty in geographical data cannot be modeled and estimated as in a usual manner, e.g., as for static data like the name of a given object. It drives the need for specific precision measures as their veracity is also temporally and spatially restricted; e.g., relying only on the trustworthiness level of their providers may not be sufficient.

Tracking real-world *objects* such as cars, trains, aircrafts, ships, persons (e.g., celebrities), or, more broadly, populations or groups of humans, natural phenomena such as cyclones, epidemics is a real need today. Such moving objects are characterized by timestamped location data and other meta-information such as name, size, maximum reachable speed, acceleration patterns, etc. The analysis and mining of spatio-temporal information about moving objects is common in a variety of applications, e.g., for pattern discovery [Li et al., 2010, Yuan et al., 2014, Dimond et al., 2013, Pianese et al., 2013]

9. Hotel-base.com (http://www.hotel-base.com/) claims to hold the largest online accommodation listing built upon tens of thousands of hotels, bed and breakfast, guest houses, hostels, apartments and self-catering properties in over 200 countries.

or prediction of trajectories and locations [Morris, 2007, De Vries and Van Someren, 2012]. The overall goal may be to better understand certain natural phenomena, to improve city services, to regulate route traffic, etc. Currently used methods for tracking moving objects are often complex, mostly rely on application-specific resources and costly equipment (e.g., satellite or radar tracking of ships and aircrafts).

The Word Wide Web, on the other hand, is a huge source of public information about various real-world moving objects. Timestamped geographical data about the position of moving objects are disseminated on the Web, notably on location-based social networks or media sharing platforms. Social networking sites like Twitter and Facebook have the ability of recording the real-time location of the user posting a message, thanks to data from the GPS system, or mobile or wireless networks. In addition, these messages may also contain location information as free text. Thus, it is theoretically possible to obtain information about the user herself, or any moving object that the user is referring to in her message. Media on sharing platforms like Flickr and Instagram may be annotated with automatically acquired spatio-temporal information, such as the timestamp and location of a picture as added by modern digital cameras.

In addition, the Web also provides in a variety of online databases more general information about moving objects. For instance, data such as the usual residence of a given individual can often be found online, e.g., in Wikipedia or Yellow Pages services. Characteristics of particular flights and ships are available on specialized Web platforms. Data extracted from multiple Web sources can serve to infer the locations of certain moving objects at given times, and to obtain general information about these. This information is uncertain, however, and exhibits many inconsistencies. One of the challenges to overcome is to estimate the inherent reliability of that information.

We claim Web information can be used in a variety of settings where one would like to track moving objects. We illustrate with the applications of celebrity spotting and ship monitoring – we propose and give a detailed description in Section 5.4 of an application whose the sources, data, and scenario are focused on the application of ship monitoring.

Celebrity spotting Journalists, paparazzi, fans, detectives, intelligence services, are routinely interested in gathering data and following the travels of given individuals, usually celebrities. These individuals may be active in social networks such as Twitter and Instagram, where they share geolocated data (or data tagged with location information); they may also be spotted in real life by users of media sharing platforms, who will upload geolocated pictures. Various Web sites, news articles, etc., provide additional meta-information. Exploiting this mass of information would provide a cost-effective and

legal manner to reconstruct a meaningful trajectory.

Ship monitoring Researchers in maritime traffic investigate the routes followed by different kinds of ships to propose traffic optimization methods, to predict pollution levels, or to prevent pirating actions. Though ships do broadcast information about their position using the AIS (Automatic Identification System) [Taka et al., 2013], this information is not made publicly available, and no historical log is kept. Information about the timestamped location of cruise ships, military vessels, tankers, etc., is common on Web sharing platforms such as Flickr and on other specialized Web sources (see also Section 5.4). Integrating ship data from multiple Web sources also helps obtaining more complete and certain information about their characteristics. This raises another important sub-problem which is the handling of possible dependencies among uncertain sources during their integration. As we shall show later, in Section 5.2, there may be some copying relationships between a group of Web sources in the maritime domain; copying relationships force, somehow, the way data will be integrated and uncertainties handled as we describe next in Example 1.2.1:

Example 1.2.1. *To exemplify an integration setting with dependent Web sources, consider the example of three uncertain Web sources S_1, S_2, and S_3 sharing a subset of objects in the maritime domain. Let us focus on values they provide for the draft and the actual port for a given ship named "Costa Serena". The first source S_1 independently reports that the draft of this ship is 8.2 m and it is currently located at the port of "Hamburg". The second source S_2 which relies on S_1 revises the data it copies by updating the draft and the current location to 8.8 m and "Marseille", respectively. Finally, the third source S_3 independently sets the draft to 8.8 m while being not aware about the location, in other words, S_3, taken apart, will provide incomplete information about the considered ship. Now, assume that one issues a query Q requesting the draft and the current port of this ship based on the three sources. The system needs to consider the answer from each source, and then to integrate them in order to provide the result of this query. In a deterministic setting, such a process will either fail or return an arbitrary answer; sources are conflicting about the correct values for the draft and the port name of this ship. In such a scenario, a rather reasonable way to perform this integration should be to present all possible results together with their level of uncertainties and those of their sources – the user afterwards may filter and choose relevant answers with respect to her trusted sources or values with highest probabilities of being correct. Capturing uncertainties in data require to be able to model aspects such as data provenance, contradiction, corrob-*

oration, relevance, etc. The provenance of the data or corroborations and contradictions between sources are given by the dependency relationships between the sources – they must be consistent with the dependencies as these precise how data are shared and have evolved. Information about the tracked ship can be obtained independently from S_1 and S_3 whereas requesting some piece of data from S_2 require to use S_1. On the other hand, the dependency relationships are telling us that S_2 disagrees with S_1 while being corroborated by S_3 for the draft; uncertainty handling is constrained by the dependencies.

To effectively estimate the precision of the location Web data about a given moving object, its characteristics, e.g., maximum speed, should be helpful– we illustrate this with Example 1.2.2.

Example 1.2.2. *Revisiting Example 1.2.1, consider now that an uncertain data integration decides, based on the reliability levels of the sources S_1, S_2, and S_3 that the actual location of the ship is "Marseille". However, at the same time we also have information that the maximal speed of this ship is around 50 km/h and its last reliably registered position, twenty days ago, was at tens thousands kilometers of "Marseille". As a consequence, the maximum speed of the ship, as well as the spatio-temporal information of its last known location, contradict the answer given by the system as the ship cannot be actually in "Marseille" under given constraints.*

Contribution We first formalize and study the problem of uncertain data integration under dependent sources; see Section 5.3. We focus on dependencies introduced by known copying relationships between sources – a copying relationship must be directed, the direction known, and co-copying is disallowed in our case. Intuitively, the set of dependency relationships give an indication about data provenance, its different versions, and the history of their revisions; the copiers first collect data from other sources and then may update (insertion of new content and correction or removal of existing content) them in order to provide their own knowledge of the modeled real-world. The shared data is somehow subject to a revision process whose the history is given by the copying relationships and the versions correspond to the source data – however one main difference with a classical version control setting is that here the data proper to each source are not explicitly known and must be inferred. As a consequence, we approach the integration of the multiple uncertain sources with dependency relationships, by targeting a technique able to reconcile these sources and their amount of uncertainties while being aware of uncertain version control. As a natural connection with our first

studied problem in Section 1.1, we answer such an integration in the special case of tree-structured data by reformulating it as the merge of several uncertain versions of the same shared XML document. We present initial ideas towards a probabilistic tree data integration framework supporting dependent sources by gaining intuitions from our uncertain version control system presented in Chapter 2. We devise a probabilistic XML encoding of the integration that captures data provenance and relevance, as well as contradictions and corroborations, with respect to the dependency relationships between input data sources (see Section 5.3 for details).

Then, we investigate and illustrate, in Section 5.4, how to track moving objects – beyond objects in the maritime domain – by integrating uncertain information from multiple Web sources, with a focus on an effective estimation of the precision of the obtained location data. We first propose an extraction technique of data about moving objects based on keyword search over a set of Web sources – its output is a collection of well-structured objects with information in the form of attribute names and values. We estimate the amount of uncertainty in each location for all kinds of moving object using a process built on two main criteria: outliers and far-fetched trajectories. We introduce another criterion, namely on-land locations, pertaining to the specific maritime traffic scenario for which we present a demonstration application (see Section 5.5). Our system considers and integrates, for each non-geographical information, multiple possible values from different Web sources – for visualization purposes, we show all the possible values of a given data item by highlighting the one with the highest number of providers. Finally, we introduce a method for estimating the trustworthiness level of users sharing location data in platforms such as Flickr based on the precision of their geo-tagged items.

1.3 Truth Finding with Correlated Data Attributes

One of the main purposes of uncertainty management in multi-source Web applications is to come with the ability to identify the most trustworthy sources and the information better describing the real world at the end of the integration process. With more structure – e.g., objects describe by a set of given attributes – and domain-specific knowledge – e.g., every attribute of any object has only one valid value, truth finding algorithms deterministically integrate multiple uncertain Web sources by quantifying the likelihood with which each possible attribute value of an object models the real world; for every object, the attribute values having highest probabilities are returned as the *truth*. Excepting few approaches like *majority voting*, most of the existing truth finding techniques [Li et al., 2012, Waguih and Berti-Equille, 2014] assume that involved sources

come with different levels of trustworthiness (or accuracy). Thereby, these techniques measure and use the levels of accuracy of sources when quantifying the probabilities of correctness of attribute values; this accuracy level of every source is often estimated using the probabilities of correctness [10] of its attribute values for the entire set of shared objects.

In addition, harnessing correlations – in its various forms – during a truth finding process has been proven to be very promising in order to improve the quality of the overall process. Being aware of the high similarity between attribute values provided by different sources for the same objects may increase our belief about the correctness of these values [Yin et al., 2008]. Detecting that there are copying relationships between a subset of involved sources may help to remain unbiased by false values spread out by propagation [Dong et al., 2009a, Dong et al., 2009b]. Furthermore, correlations between sources resulting from sharing the same extraction patterns or algorithms, or maintaining complementary information may be indicative of the direction of the truth [Pochampally et al., 2014].

We consider in this study the problem of truth finding over structured Web sources with structurally correlated data attributes. We focus on a one-truth setting, i.e., each attribute of every object has only one valid value.

To the best of our knowledge, there is no previous study on truth finding which considers structural correlations between data attributes, as existing methods mostly focus on correlations between sources. The level of accuracy of a given source in providing the truth can be variable with respect to distinct subsets of the data attributes. For instance, Web sources can rely on several sensors, extraction patterns or algorithms, etc., which independently collect information about different attributes of the same real-world objects. As a result, the quality of the source on each specific subset of attributes corresponds, in reality, to the accuracy of the specific technique responsible of gathering information about these attributes; we say that such a subset of attributes is structurally correlated in terms of source data quality. Structural correlations between attributes can prevent being biased when using a global indicator of source quality for quantifying the probabilities of correctness of attribute values during a truth finding process. In general these possible structural correlations among subsets of data attributes are unknown since sources only provide an aggregated view of their information. This drives the need for a way to also discover these structural correlations between data attributes during the truth finding process.

10. Sometimes, the probability of correctness associated to a possible attribute value is just obtained as a normalization of a vote count or a confidence measure.

Contribution We formalize and study the problem of truth finding in the presence of structurally correlated data attributes with respect to the levels of accuracy of involved sources. We tackle this issue, in Chapter 6, by searching for an optimal partitioning of the data attributes for any truth finding technique based on an estimation of the level of accuracy of sources. As initial results, we propose two exploration algorithms – an exact and an approximation algorithm – which discover such an optimal partition given a truth finding algorithm. Preliminary experimentations over synthetic datasets provide promising results in terms of precision regarding some existing algorithms; see Section 6.4 for details.

Part I

Uncertain Multi-Version Tree Data

2 Uncertain XML Version Control Model

In this chapter, we detail our approach to the main aspect of the first problem introduced in Chapter 1, the problem of *modeling and assessing uncertainty in version control*. We focus on tree-structured data which describe well the data handled in our application scenario, that is, *Web-scale collaborative editing platforms*. After an overview of the related work in Section 2.1, we start by introducing, in Section 2.2, some preliminary material and by reviewing the probabilistic XML model we use in Section 2.3. After an abstraction of a multi-version XML document through a formal definition of its graph of versions space and its set of versions, we detail, in Section 2.4, our contribution for this work: a probabilistic XML version control model for multi-version uncertain tree-structured documents. This model is built on version control events, a p-document, and a directed acyclic graph of events. At the abstract level, we define an uncertain multi-version XML document using a *possible-world semantics* (Section 2.4.2). To achieve further effectiveness, we devise a compact representation of all the possible versions, together with their probabilities, of an uncertain multi-version XML document through a PrXML$^{\text{fie}}$ p-document encoding. We end by proving the correction of our encoding.

2.1 Related Work

This section overviews the related work on version control, uncertain tree-structured data models, and quality evaluation on collaborative editing systems.

2.1.1 Research on Version Control

Significant research on version control has been done both in research and industrial communities [Cellary and Jomier, 1990, David, 1994, Conradi and Westfechtel, 1998, Al-Khudair et al., 2001, Pilato, 2004, Kögel and Maximilian, 2008, Reichenberger and Kratky, 2009, Rönnau and Borghoff, 2009, Thao and Munson, 2011, Pandey and Munson, 2013, Thao and Munson, 2014]. In general, first systems [Cellary and Jomier, 1990, David,

1994, Conradi and Westfechtel, 1998, Al-Khudair et al., 2001, Kögel and Maximilian, 2008, Reichenberger and Kratky, 2009] are based on object-oriented models, whereas recent research or industrial tools [Pilato, 2004, Chacon, 2009, Rönnau and Borghoff, 2009] are focused on file-oriented models. Abstracting out the differences between both settings, the common problem is to manage changes in documents, e.g., large source codes, flat or structured documents, etc., shared by large teams.

Object-oriented Multi-Version Databases Object-oriented models are used in software engineering environments (i.e., CAD systems [1] and CASE [2]). In these environments, features and behaviors of objects are regarded to be the same as those defined in classical object-oriented approaches in databases or in programming languages. Objects are described as instances of classes with attributes and methods. In the same way, concepts like inheritance and generalization are used to represent some structural constraints between objects. Branching aspects are also taken into account enabling modifications over objects happening in parallel along multiple branches, which is very important in particular in distributed and cooperative development environments. Collaboration and distribution can increase the number of variants of the same object within the system. For storage support, these object-oriented version control systems are implemented on top of existing object-oriented database systems and versions of a same object coexist simultaneously in the same place. As a consequence, most research efforts, e.g., those in [Cellary and Jomier, 1990, Al-Khudair et al., 2001, Reichenberger and Kratky, 2009], have been concentrated on consistency management of versions and configurations, i.e., how to identify the versions that compose a consistent configuration or, more generally, a consistent state of the modeled world: *global version stamps* [Cellary and Jomier, 1990] or *descriptors of consistency* [Al-Khudair et al., 2001] were among popular advocated solutions. In the case of large software engineering environments, version control systems based on file-oriented models (also called *source code management tools*) manage large source codes, documentation, and configuration files.

General-purpose Version Control Tools Many systems intended for version control of documents – covering multiple formats (e.g., text, binary and XML formats) – have been proposed both in research [David, 1994, Rusu et al., 2005, Rönnau and Borghoff, 2009, Pandey and Munson, 2013, Thao and Munson, 2011] and in industry [Pilato, 2004,

1. Computer Aided Design Systems
2. Computer Aided Software Environments

Chacon, 2009]. A large majority among them, seen as *general-purpose*[3] *version control systems*, by default only manipulate text or binary formats. *Subversion, ClearCase, Git, BitKeeper,* and *Bazaar* form a non exhaustive list of them; see [Rönnau and Borghoff, 2009, Koc and Tansel, 2011] for a more detailed survey about practical version control tools. At their basis, general-purpose version control systems are either built on top of line-based differencing algorithms like GNU diff [Myers, 1986], or simply use cryptographic hashing techniques [Chacon, 2009] in order to detect new versions when changes are committed. Both methods are not convenient for structured documents like XML documents, because the semantics of edits cannot be described meaningfully, leading to possible hiding of the collaborative editing aspects. As a consequence, tree differencing algorithms [Cobéna et al., 2002, Lindholm et al., 2006, Cobéna and Abdessalem, 2009, Rönnau and Borghoff, 2012, Thao and Munson, 2014] have been used in order to properly capture structural changes between XML document versions (we defer the discussion about the literature on XML change detection to Section 4.1). Furthermore, version control tools support concurrent access to the same file, which enables collaboration and data sharing over distributed systems. They implement locking mechanisms (generally, shared locking) for concurrency management and include some merging features. Under versioning, a complete history of versions of files is maintained and each version is stored in a compressed form, e.g., using a delta representation. Typically, the version space is represented sequentially or using a graph structure when parallel versions occur. Advanced features such as directory handling under version control are also available within systems like Git [Chacon, 2009]. Observe that the overall functionalities of a file-based version control system can be implemented in general as a virtual file system on top of the file system of an existing operating system as in [Pilato, 2004, Chacon, 2009]. As concurrent editions are frequent in practice, conflicts may appear during the merging phase and their detection is usually application-dependent or manual. Many file-oriented version control systems, being deterministic, require human intervention to resolve conflicts before a final validation of the merge: such a resolution often consists in keeping only one possibility even though it is not the most relevant change. Computing the most relevant changes during a revision process has been studied by [Borgolte et al., 2014] using clustering algorithms but in the context of Web pages. In large-scale collaborative editing platforms – particularly within a distributed setting – which are characterized by a high rate of concurrency, operational transformation algorithms [Weiss et al., 2010, Mehdi et al., 2014] are often proposed for

3. The word general-purpose means that these systems are especially designed for versioning and they offer guarantees for an optimized process in terms of efficiency for change propagation or version retrieval.

ensuring consistency and intentionality; see [Kumawat et al., 2010] for a survey.

Version Control of XML Documents Research on version control of tree-structured documents, in particular XML, has been largely devoted to change detection and merge algorithms, e.g., [Robin, 2002, Cobéna et al., 2002, Lindholm et al., 2006, Rönnau and Borghoff, 2012, Thao and Munson, 2014]; we provide more insights about these two aspects in Section 4.1. However, some efforts have been made on version control models specific to XML documents. The version control models in [Rusu et al., 2005, Thao and Munson, 2011, Pandey and Munson, 2013] define and use special XML tags in order to identify and track the different versions of an XML document. In both models, the versions, as well as the history of their generation, are fully encoded into a unique XML document yielding a *version-aware XML document model* – [Pandey and Munson, 2013] targets LibreOffice documents. Two main disadvantages of these two approaches are content redundancy and costly update operations, the latter being a logical consequence of the former. Rönnau and Borghoff investigate in [Rönnau and Borghoff, 2009] a system for versioning office documents (e.g., OpenDocument and OpenXML documents) shared in a collaborative manner, e.g., via e-mail, within a distributive environment by providing a toolkit for versioning and merging XML documents which is device- and media-independent. However, their model is more alike to an efficient XML change detection and merging tool (see [Rönnau and Borghoff, 2012]) rather than a fully far-fetched XML version control system.

2.1.2 Uncertain Tree-Structured Data Models

Uncertainty handling in tree-structured data was originally associated to the problem of automatic Web data extraction and integration, as we shall show later in Section 5.1. Several efforts have been made on uncertain tree-structured management and some models – based mostly on a possible-world semantics with a compact representation system – have been proposed, especially the work of Van Keulen et al. [Van Keulen et al., 2005, van Keulen and de Keijzer, 2009]. Then a framework, namely probabilistic XML, that generalizes all the existing models was proposed in [Abiteboul et al., 2009] and [Kharlamov et al., 2010]; we refer to [Kimelfeld and Senellart, 2013] for a detailed survey of the probabilistic XML literature. Probabilistic XML models can represent and evaluate various semantics of uncertainties in data, for instance, conflicts, incompleteness, missing or NULL information, and correlations: actual probabilistic XML models can be categorized according to the semantics of uncertainties they capture, e.g., the

classification given in [Abiteboul et al., 2009]. The simple probabilistic tree model of Van Keulen et al. [Van Keulen et al., 2005] captures uncertainties resulting from conflicts on the tree structure or the data values by enumerating the set of possible XML documents together with their probabilities of being the valid instance of the modeled world. The authors introduce two special XML nodes, that is, possibility and probability nodes in order to compactly encode the set of possible XML documents and their probabilities. While the model in [Van Keulen et al., 2005] directly assigns probability values to uncertain nodes – note that a reliable source of these values is not always available – advanced probabilistic XML models such as those of [Abiteboul et al., 2009, Kharlamov et al., 2010] use *random Boolean variables* for modeling and assessing the amount of uncertainties in data which facilitates *logical reasoning* on information or tracking of *data provenance*. These models have the ability to describe a larger class of semantics about uncertainties on data such as *long-distance dependencies*[4] between uncertain data items; we revisit and illustrate probabilistic XML models based on random Boolean variables in Section 2.3.

2.1.3 Quality in Collaborative Editing Systems

Various ways of measuring and analyzing the quality of the content creation process in Wikipedia have been approached [Adler and de Alfaro, 2007, De La Calzada and Dekhtyar, 2010, Maniu et al., 2011a, Osman, 2013, Geiger and Halfaker, 2013]. [Adler and de Alfaro, 2007] and [Maniu et al., 2011a] have respectively studied the inference of users' reputation scores and users' social connections by performing off-line analysis of the history of versions and edits for a large bunch of Wikipedia's articles: the history of editions made by contributors and the semantics of performed operations are exploited in order to derive a reputation score for each involved user or to construct a signed network between the contributors. Such a signed network should, in some extent, serve as a basis for deriving trusted contributors as those having mostly positive incoming and outgoing links with the others. The impact of conflicts on the quality of Wikipedia articles, as well as how consensus are reached through manual coordinations implying contributors, has been explored and conjectured in [Osman, 2013]. Finally, [Geiger and Halfaker, 2013] studies the role of *bots* in Wikipedia's quality control; they show that the quality of Wikipedia articles does not decline without bots as their automatic correction jobs will be done later by contributors during the revision process.

4. Long-distance dependencies translate the fact that the existence of some nodes, belonging to different sub-trees, are correlated.

Our own previous work in [Abdessalem et al., 2011, Ba et al., 2011] are initial studies towards the design of an uncertain XML version control system: [Abdessalem et al., 2011] is a demonstration system focusing on Wikipedia revisions and showing the benefits of integrating an uncertain XML version control approach in web-scale collaborative platforms; [Ba et al., 2011] gives some early ideas behind modeling XML uncertain version control.

2.2 Preliminaries

In this section, we present some basic version control notions and the semi-structured XML document model underlying our proposal. A *multi-version document* refers to a set of versions of the same document handled within a version control process. Each version of the document represents a given state (instance) of the evolution of this versioned document. A typical version control model is built on the following common notions.

Document Version A version [5] is a conventional term that refers to a document copy in document-oriented version control systems. The different versions of a document are linked by derivation operations. A derivation consists of creating a new version by first copying a previously existing one before performing modifications. Some versions, representing variants, are in a derivation relationship with the same origin. The variants (parallel versions) characterize a non-linear editing history with several distinct branches of the same multi-version document. In this history, a branch is a linear sequence of versions. Instead of storing the complete content of each version, most version control approaches only maintains *diffs* between states, together with meta-information on states. These states (or commits in Git world [Chacon, 2009]) model different sets of changes that are explicitly validated at distinct stages of the version control process. A state also comes with information about the context (e.g., author, date, comment) in which these modifications are done. As a consequence, each version depends on the complete history leading up to a given state. We will follow here the same approach for modeling the different versions of a document within our framework.

Version Space Since the content of each version is not fully saved, there must be a manner to retrieve it when needed. The version space [6] represents the editing history of

5. The term “revision” is also widely used for referring to a version of a document. As a result, we use sometimes the word revision instead of version in the thesis.

6. Observe that the term “version space” is also used, but with a semantics different of ours, in the field of concept learning [Mitchell, 1979].

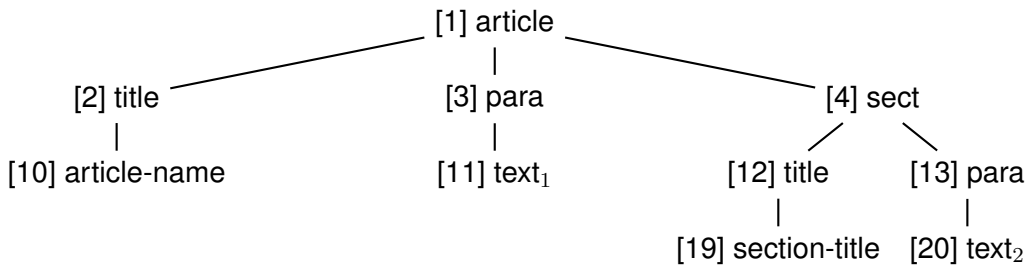


Figure 2.1: Example XML tree $\mathscr{T}$: Wikipedia article

a versioned document (e.g., wiki version history as given in [Sabel, 2007]). It maintains necessary information related to the versions and their derivations. As mentioned above, a derivation relationship implies at least one input version (several incoming versions for merge operations as we will see in Chapter 4) and an output version. Based on this, we model similarly to [Chacon, 2009] a version space of any multi-version document as a *directed acyclic graph*[7].

Unordered XML Tree Documents Our motivating applications handle mostly tree-structured data. As a result, we consider data as unordered XML trees. Note that the proposed model can be extended to ordered trees (this may require restricting the set of valid versions to those complying with a specific order, we leave the details for future work); we choose unordered trees for convenience of exposition given that in many cases order is unimportant. Let us assume a finite set $\mathscr{L}$ of strings (i.e., labels or text data) and a finite set $\mathscr{I}$ of identifiers such that $\mathscr{L} \cap \mathscr{I} = \emptyset$. In addition, let Φ and α be respectively a labeling function and an identifying function. Formally, we define an *XML document* as an *unordered*, *labeled* tree $\mathscr{T}$ over identifiers in $\mathscr{I}$ with α and Φ mapping each node $x \in \mathscr{T}$ respectively to a unique identifier $\alpha(x) \in \mathscr{I}$ and to a string $\Phi(x) \in \mathscr{L}$. The tree is unranked, i.e., the number of children of each node in $\mathscr{T}$ is not assumed to be fixed. Given an XML tree $\mathscr{T}$, we define $\Phi(\mathscr{T})$ and $\alpha(\mathscr{T})$ as respectively the set of its node strings and the set of its node identifiers. For simplicity, we will assume all trees have the same root node (same label, same identifier).

Example 2.2.1. *Figure 2.1 depicts an XML tree $\mathscr{T}$ representing a typical Wikipedia article. The node identifiers are inside square brackets besides node strings. The title of*

7. A directed acyclic graph (or DAG) is a graph consisting of a set of nodes and directed edges, each edge connecting one node to another, such that there is no way to start at some node v and follow a sequence of edges that eventually loops back to v again.

this article is given in node 10*. The content of the document is structured in sections ("sect") with their titles and paragraphs ("para") containing the text data.*

XML Edit Operations We rely on unique node identifiers and consider two basic edit operations over the specified XML document model: node *insertions* and *deletions*.

Node Insertions We denote an insertion by `ins(`*i, x*`)` whose semantics over any XML tree consists of inserting node x (we suppose x is not already in the tree) as a child of a certain node y satisfying $\alpha(y) = i$. If such a node is not found in the tree, the operation does nothing. Note that an insertion can concern a subtree, and in this case we simply refer with x to the root of this subtree.

Node Deletions Similarly, we introduce a deletion as `del(`*i*`)` where i is the identifier of the node to suppress. The delete operation removes the targeted node, if it exists, together with its descendants, from the XML tree.

We conclude by defining an XML edit script, $\Delta =< u_1, u_2, \ldots, u_i >$, as a sequence of a certain number of elementary edit operations u_j (each u_j, with $1 \leq j \leq i$, being either an insertion or a deletion) to carry out one after the other on an XML document for producing a new one. Given a tree $\mathscr{T}$, we denote the outcome of applying an edit script Δ over $\mathscr{T}$ by $[\mathscr{T}]^{\Delta}$. Even though in this work we are dependent on persistent identifiers on tree nodes to define edit operations, the semantics of these operations could be extended to updates expressed by queries, especially useful in distributed collaborative editing environments where identifiers may not be straightforward to share.

2.3 Probabilistic XML

We briefly introduce in this section the probabilistic XML representation system we use as a basis of our uncertain version control system. For more details, see [Abiteboul et al., 2009] for the general framework and [Kharlamov et al., 2010] for the specific PrXML$^{\text{fie}}$ model we used. As outlined earlier, these representation systems are originally intended for XML-based applications such as Web data integration and extraction. For instance, when integrating various semi-structured Web catalogs containing personal data, some problems such as overlapping or contradiction are frequent. Typically, one can find for the same person name two distinct affiliations in different catalogs. A probabilistic XML model can be used to automatically integrate such data sources by enumerating all possibilities: (a) the system considers each incoming source according

to a certain ordering of sources; (b) it maps its data items with the existing items in the probabilistic repository, initially empty, to find matches and; (c) giving that, it represents the matches as a set of possibilities. The resolution of conflicts is thus postponed to query time, where each query will return a set of possibilities together with their probabilities. The main reason to do so is that resolving semantic issues before an effective integration can be tricky in such situations; the resolution of semantic issues is usually a tedious and error-prone resolution process, particularly, when there is no external knowledge about the reliability of the sources and the level of relevance of data .

p-Documents A *probabilistic XML representation system* is a compact way of representing probability distributions over possible XML documents; in the case of interest here, the probability distribution is finite. Formally, a probabilistic XML distribution space, or px-space, $\mathscr{S}$ over a collection of uncertain XML documents is a couple (D, p) where D is a nonempty finite set of documents and $p : D \to (0, 1]$ is a probability function that maps each document d in D to a rational number $p(d) \in (0, 1]$ such that $\Sigma_{d \in D} p(d) = 1$. A *p-document*, or *probabilistic XML document*, usually denoted $\widehat{\mathscr{P}}$, defines a compact encoding of a px-space $\mathscr{S}$.

PrXML$^{\text{fie}}$: Syntax and Semantics We consider for our needs one specific class of p-documents, PrXML$^{\text{fie}}$ [Kharlamov et al., 2010] (where fie stands for *formula of independent events*); restricting to this particular class allows us to give a simplified presentation, see [Abiteboul et al., 2009, Kharlamov et al., 2010] for a more general setting. Assume a set of *independent random Boolean variables*, or *event variables* in short, $b_1, b_2, \ldots, b_m$ and their respective probabilities $\mathsf{P_r}(b_1), \mathsf{P_r}(b_2) \ldots, \mathsf{P_r}(b_m)$ of existence. A PrXML$^{\text{fie}}$ p-document is an unordered, unranked, and labeled tree where every node (except for the root) x may be annotated with an arbitrary propositional formula $fie(x)$ over the event variables $b_1, b_2, \ldots, b_m$. Different formulas can share common events, i.e., there may be some correlation between formulas and the number of event variables in the formulas may vary from one node to another.

A valuation ν of the event variables $b_1 \ldots b_m$ induces over $\widehat{\mathscr{P}}$ one particular XML documents $\nu(\widehat{\mathscr{P}})$: the document where only nodes annotated with formulas valuated to true by ν are kept (nodes whose formulas are valuated to false by ν are deleted from the tree, along with their descendants). Given a p-document $\widehat{\mathscr{P}}$, the *possible worlds* of $\widehat{\mathscr{P}}$, denoted as $pwd(\widehat{\mathscr{P}})$ is the set of all such XML documents. The *probability* of a given possible world d of $\widehat{\mathscr{P}}$ is defined as the sum of the probability of the valuations that yield d. The set of possible worlds, together with their probabilities, defines the *semantics* of

$\widehat{\mathscr{P}}$, the px-space $[\![\widehat{\mathscr{P}}]\!]$ associated to $\widehat{\mathscr{P}}$.

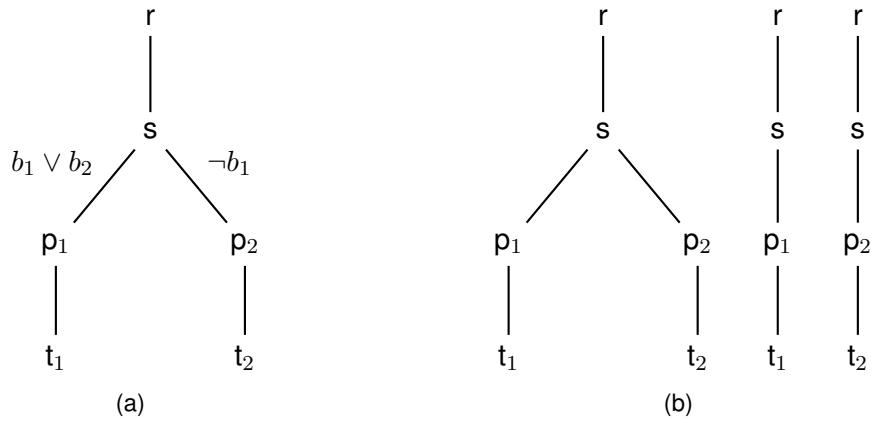


Figure 2.2: (a) PrXML$^{\text{fie}}$ p-document $\widehat{\mathscr{P}}$; (b) Three possible worlds d_1, d_2 and d_3

Example 2.3.1. *Figure 2.2 sketches on the left side a concrete* PrXML$^{\text{fie}}$ *p-document* $\widehat{\mathscr{P}}$ *and on the right side three possible worlds* d_1, d_2 *and* d_3*. Formulas annotating nodes are shown just above them:* $b_1 \vee b_2$ *and* $\neg b_1$ *are bound to nodes* p_1 *and* p_2 *respectively. The three possible worlds* d_1, d_2 *and* d_3 *are obtained by setting the following valuations of* b_1 *and* b_2*: (a)* false *and* true*; (b)* true *and* true *(or* true *and* false*); (c)* false *and* false*. At each execution of the random process, the distributional node chooses exactly the nodes whose formulas are evaluated at* true *given the valuation specified over event variables. Assuming a probability distribution over events, for instance* $\mathsf{P_r}(b_1) = 0.4$ *and* $\mathsf{P_r}(b_2) = 0.5$*, we derive the probability of the possible world* d_1 *as* $\mathsf{P_r}(d_1) = (1 - \mathsf{P_r}(b_1)) \times \mathsf{P_r}(b_2) = (1 - 0.4) \times 0.5 = 0.3$*. We can compute similarly the probabilities of all other possible worlds.*

With respect to other probabilistic XML representation systems [Abiteboul et al., 2009], PrXML$^{\text{fie}}$ is very succinct (since arbitrary propositional formulas can be used, involving arbitrary correlations among events), i.e., exponentially more succinct than the models of [Nierman and Jagadish, 2002, Van Keulen et al., 2005], and offers tractable insertions and deletions [Kharlamov et al., 2010], one key requirement for our uncertain version control model. However, a non-negligible downside is that all non-trivial (tree-pattern) queries over this model are **#P**-hard to evaluate [Kimelfeld et al., 2009]. This is not necessarily an issue, here, since we favor in our application efficient updates and retrieval of given possible worlds, over arbitrary queries.

Data Provenance Uncertain XML management based on the PrXML$^{\text{fie}}$ model also takes advantage of the various possible semantics of event variables in terms of information description. Indeed, besides uncertainty management, the model also provides support for keeping information about *data provenance* (or lineage) based on the event variables. Data provenance is information of traceability such as change semantics, responsible party, timestamp, etc., related to uncertain data. To do so, we only need to use the semantics of event variables as representing information about data provenance. As such, it is sometimes useful to use probabilistic XML representation systems even in the absence of reliable probability sources for individual events, in the sense that one can manipulate them as incomplete data models (i.e., we only care about possible worlds, not about their probabilities). Moreover, tracking information about data provenance within a version control process can be helpful for provenance queries; see [Zhang and Jagadish, 2013] for a concrete example of the use of provenance data within a revision process.

2.4 Uncertain Multi-Version XML Setting

In this section, we elaborate on our uncertain XML version control model for tree-structured documents edited in a collaborative manner. We build our model on three main concepts: version control events, p-document, and directed acyclic graph of events. We start by formalizing a multi-version XML document through a formal definition of its graph of version space and its set of versions. Then we formally introduce the proposed model.

2.4.1 Multi-Version XML Documents

Consider the infinite set $\mathscr{D}$ of all XML documents with a given root label and identifier. Let $\mathscr{V}$ be a set of *version control events* $e_1, \ldots, e_n$. These events represent the different states of a tree. We associate to events contextual information about revisions (authorship, timestamp, etc.). To each event e_i is further associated an *edit script* Δ_i. Based on this, we formalize the graph of version space and the set of versions of any versioned XML document as follows.

Graph of version space The *version space* is a rooted directed acyclic graph (DAG) $\mathscr{G} = (\mathscr{V} \cup \{e_0\}, \mathscr{E})$ where: (i) the initial version control event $e_0 \notin \mathscr{V}$, a special event representing the first state of any versioned XML tree, is the root of $\mathscr{G}$; (ii) $\mathscr{E} \subseteq (\mathscr{V} \cup \{e_0\})^2$, defining the edges of $\mathscr{G}$, consists of a set of ordered couples of version control events.

Each edge implicitly describes a directed derivation relationship between two versions. A *branch* of $\mathscr{G}$ is a directed path that implies a start node e_i and an end node e_j. The latter must be reachable from the former by traversing a set of ordered edges in $\mathscr{E}$. We refer to this branch by B_i^j. A *rooted branch* is a branch that starts at the root of the graph.

XML versions An XML version is the document in $\mathscr{D}$ corresponding to a *set* of version control events, the set of events that made this version happen. In a deterministic version control system, this set always corresponds to a rooted branch in the version space graph. In our uncertain version control system, this set may be arbitrary. Let us consider the set $2^{\mathscr{V}}$ comprising all sub-parts of $\mathscr{V}$. The set of versions of a multi-version XML document is given by a mapping $\Omega : 2^{\mathscr{V}} \rightarrow \mathscr{D}$: to each set of events corresponds a given tree (these trees are typically not all distinct). The function Ω can be computed from edit scripts associated with events as follows:

- $\Omega(\emptyset)$ maps to the root-only XML tree of $\mathscr{D}$.
- For all i, for all $\mathscr{F} \subseteq 2^{\mathscr{V}\setminus\{e_i\}}$ $\Omega(\{e_i\} \cup \mathscr{F}) = [\Omega(\mathscr{F})]^{\Delta_i}$.

A multi-version XML document, $\mathscr{T}_{mv}$, is now defined as a pair $(\mathscr{G}, \Omega)$ where $\mathscr{G}$ is a DAG of version control events, whereas Ω is a mapping function specifying the set of versions of the document. In the following we propose a more efficient way to compute the version corresponding to a set of events, using a p-document for storage.

2.4.2 Uncertain Multi-Version XML Documents

A multi-version document will be *uncertain* if the version control events, staged in a version control process, come with *uncertainty* as in open collaborative contexts. By version control events with uncertainty, we mean random events leading to uncertain versions and content. As a consequence, we will rely on a *probability distribution over* $2^{\mathscr{V}}$, that will, together with the Ω mapping, imply a probability distribution over $\mathscr{D}$.

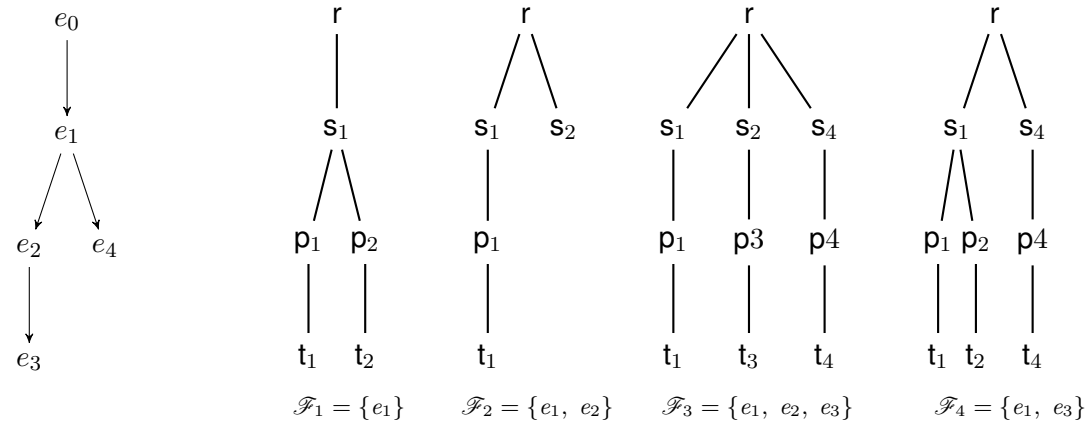
Uncertainty modeling We model uncertainty in events by further defining a version control event e_i in $\mathscr{V}$ as a conjunction of semantically unrelated random Boolean variables $b_1, \ldots, b_m$ with the following assumptions: (i) a Boolean variable models a given source of uncertainty (e.g., the contributor) in the version control environment; (ii) all Boolean variables in each e_i are independent; (iii) a Boolean variable b_j reused across events correlates different version control events; (iv) one particular Boolean *revision* variable $b^{(i)}$, representing more specifically the uncertainty in the contribution, is not shared across other version control events and appears positively in e_i.

Probability Computation We assume given a probability distribution over the Boolean random variables b_j's (this typically comes from a trust estimation in a contributor, or in a contribution), which induces a probability distribution over propositional formulas over the b_j's in the usual manner [Kharlamov et al., 2010]. We now obtain the probability of each (uncertain) version d of as follows:

$$\mathsf{P_r}(d) = \mathsf{P_r}\left(\bigvee_{\substack{\mathscr{F}\subseteq\mathscr{V}\\ \Omega(\mathscr{F})=d}} \phi(\mathscr{F})\right)$$

with the formula $\phi(\mathscr{F})$ associated to $\mathscr{F}$ defined as by:

$$\phi(\mathscr{F}) = \bigwedge_{e_j\in\mathscr{F}} e_j \wedge \bigwedge_{e_k\in\mathscr{V}\setminus\mathscr{F}} \neg e_k. \tag{2.1}$$



(a) Graph $\mathscr{G}$

(b) Possible Versions $\mathscr{T}_1$, $\mathscr{T}_2$, $\mathscr{T}_3$ et $\mathscr{T}_4$

Figure 2.3: (a) Graph of Version Space; (b) Four versions and their corresponding truth-values

Example 2.4.1. *Figure 2.3 sketches an uncertain multi-version XML document $\mathscr{T}_{mv}$ with four staged version control events. On the left side, we have the version space $\mathscr{G}$. The right side shows an example of four possible (uncertain) versions and their associated event set. We suppose that $\mathscr{T}_{mv}$ is initially a root-only document. The first three versions correspond to versions covered by deterministic version control systems, whereas the last one is generated by considering that the changes performed at an intermediate version control event, here e_2, as incorrect. One feature of our model is to provide*

the possibility for viewing and modifying these kinds of uncertain versions representing virtual versions. Only edits performed at the specified version control events are taken into account in the process of producing a version: in $\mathscr{T}_4$, the node r and the subtrees rooted at s_1, s_3 respectively introduced at e_0, e_1 and e_3 are present, while the subtree p_3 added at e_3 does not appear because its parent node s_2 cannot be found. Finally, given probabilities of version control events, we are able to measure the reliability of each uncertain version $\mathscr{T}_i$, for each $1 \leq i \leq 4$, based on its corresponding event set $\mathscr{F}_i$ (and all other event sets that map to the same tree).

We straightforwardly observe, for instance with the simple example in Figure 2.3, that the amount of possible (uncertain) versions of any uncertain multi-version document may grow rapidly (indeed, exponentially in the number of events). As a result, the enumeration and the handling of all the possibilities with the function Ω may become tedious at a certain point. To address this issue, we propose an efficient method for encoding in a compact manner the possible versions together with their truth values. Intuitively, a PrXML$^{\text{fie}}$ p-document compactly models the set of possible versions of an uncertain multi-version XML document. As stressed in Section 2.3, a probabilistic tree based on propositional formulas provides interesting features for our setting. First, it describes well a distribution of truth values over a set of uncertain XML trees while providing a meaningful process to find back a given version and its probability. Second, it provides an update-efficient representation system, which is crucial in dynamic environments such as version-control–based applications.

2.4.3 Probabilistic XML Encoding Model

We introduce a general uncertain XML version control representation framework, denoted by $\widehat{\mathscr{T}_{mv}}$, as a couple $(\mathscr{G}, \widehat{\mathscr{P}})$ where (a) $\mathscr{G}$ is as before a DAG of events, representing the version space; (b) $\widehat{\mathscr{P}}$ is a PrXML$^{\text{fie}}$ p-document with random Boolean variables $b_1 \dots b_m$ representing efficiently all possible (uncertain) XML tree versions and their corresponding truth-values.

We now define the semantics of such an encoding as the uncertain multi-version document $(\mathscr{G}, \Omega)$ where $\mathscr{G}$ is the same and Ω is defined as follows. For all $\mathscr{F} \subseteq \mathscr{V}$, let B^+ be the set of all random variables occurring in one of the events of $\mathscr{F}$ and B^- be the set of all revision variables $b^{(i)}$'s for e_i *not in* $\mathscr{F}$. Let ν be the valuation of $b_1 \dots b_m$ that sets variables of B^+ to true, variables of B^- to false, and other variables to an arbitrary value. We set $\Omega(\mathscr{F}) := \nu(\widehat{\mathscr{P}})$.

The following shows that this semantics is compatible with the px-space semantics of p-documents on the one hand, and the probability distribution defined by uncertain multi-version documents on the other hand.

Proposition 2.4.1. *Let $(\mathscr{G}, \widehat{\mathscr{P}})$ be an uncertain version control representation framework and $(\mathscr{G}, \Omega)$ its semantics as just defined. We further assume that all formulas occurring in $\widehat{\mathscr{P}}$ can be expressed as formulas over the events of $\mathscr{V}$ (i.e., we do not make use of the b_j's independently of version control events). Then the px-space $[\![\widehat{\mathscr{P}}]\!]$ defines the same probability distribution over $\mathscr{D}$ as Ω.*

The proof is straightforward and relies on Equation (2.1).

2.5 Conclusion

We considered in the chapter the problem of modeling and assessing uncertainty in version control, proposing an uncertain XML version control approach that have the ability to effectively capture uncertainties in tree-structured data collaboratively edited by a large community of users with different levels of reliability and providing contributions whose degree of relevance is also variable. To the best of our knowledge, our solution, targeting Web-scale collaborative editing applications, is the first that accounts for uncertain data within a version control process. We formalized a multi-version XML document through its graph of version space and its set of versions. After that, we formally introduced an uncertain multi-version XML document as a set of possible versions with their probabilities of being valid (based on random Boolean variables which model and assess uncertain data) and a graph of version space. For more effectiveness, we devised and proposed a general probabilistic XML version control model which properly encodes in a compact manner the set of all possible versions of an uncertain multi-version XML document.

We continue in the two next chapters by detailing the support of the standards version control operations, in particular update and merge operations, and by studying the performance of our model.

3 Updates in Uncertain XML Version Control

In this chapter, we consider the following problems: the formalization of the semantics of standard update operations within our uncertain XML version control framework, and the evaluation of their overall performance.

Updates are crucial in version control because they constitute the principal means for generating new versions based on old ones. In our application scenarios, an update operation consists of uncertain elementary edit operations leading to several possible versions. We initially translate and formalize the semantics of an uncertain update operation over a given uncertain multi-version XML document through the possible-world intepretation of its set of possible versions and the corresponding truth values (Section 3.1.1). In Section 3.1.2, we show that such uncertain update can be directly performed as an operation over the probabilistic XML encoding of the uncertain multi-version XML document by devising the corresponding update algorithm. We empirically demonstrate the correction of our update algorithm, the observation that our proposed semantics for update is compatible with the classical update operation of version control systems, and finally the full scalability of our algorithm in terms of running time and the growth of the size of the output probabilistic tree.

For performance analysis purposes, we demonstrate, in Section 3.2, the efficiency of our model with respect to deterministic version control systems through measures on real-world datasets. Then, we revisit in Section 3.2.2 some of the content filtering capabilities of our approach. In brief, in comparison with robust version control tools like Git and Subversion, our experiments show that our system achieves update propagation and version retrieval in comparable, or even better in some cases, running times.

3.1 Updating Uncertain Multi-Version XML

We implement the semantics of standard update operations on top of our probabilistic XML representation system introduced in the previous chapter. Our system considers and supports edit scripts consisting of two basic elementary edit operations, namely node

insertions and deletions, over unordered XML documents (cf. Section 2.2). Recall that the definition of these edit operations rely on unique node identifiers, and we consider the introduction of update operations under these assumptions.

3.1.1 Uncertain Update Operation

An update over an uncertain multi-version document $\mathscr{T}_{mv} = (\mathscr{G}, \Omega)$ corresponds to the evaluation of some uncertain edits on a given (uncertain) version. With the help of a triple (Δ, e, e'), we refer to an update operation as $\texttt{updOP}_{\Delta,\ e,\ e'}$ where Δ is an edit script, e is an existing version control event pointing to the edited version and e' is an incoming version control event evaluating the amount of uncertainty in this update. We formalize $\texttt{updOP}_{\Delta,\ e,\ e'}$ over $\mathscr{T}_{mv}$ as follows.

$$\texttt{updOP}_{\Delta,\ e,\ e'}(\mathscr{T}_{mv}) := (\mathscr{G} \cup (\{e'\}, \{(e, e')\}),\ \Omega').$$

An update operation thus results in the insertion of a new node and a new edge in $\mathscr{G}$, and an extension of Ω with Ω' that we define now. For any subset $\mathscr{F} \subseteq \mathscr{V}'$ ($\mathscr{V}'$ is the set of nodes in $\mathscr{G}$ after the update), we have:

- if $e' \notin \mathscr{F}$: $\Omega'(\mathscr{F}) = \Omega(\mathscr{F})$;
- otherwise: $\Omega'(\mathscr{F}) = [\Omega(\mathscr{F} \backslash \{e'\})]^{\Delta}$.

Example 3.1.1. *Revisiting Figure 2.1 with the tree $\mathscr{T}$ being the version generated from the initial root-only version of a shared document. Let us now consider that an uncertain update, whose edit script $\Delta = \{\texttt{ins}(1, \textit{sect}), \texttt{del}(11), \texttt{ins}(3, \textit{text}_3)\}$ inserts a new section to the article, deletes $\textit{text}_1$ and replaces it by $\textit{text}_3$, is issued on $\mathscr{T}$. This update, according to the semantics of updates over uncertain multi-version XML documents, will result in two new possible versions: one version corresponding exactly to the result of Δ over $\mathscr{T}$ when all the context is reliable and another version consisting of applying Δ on the initial root-only version when the update that has produced $\mathscr{T}$ is unreliable. Concerning the graph of version space, it will be updated with the new event e_2 and the edge (e_1, e_2) where e_1 represents the event associated to the update having lead to $\mathscr{T}$.*

What precedes gives a semantics to updates on uncertain multi-version documents; however, the semantics is not practical as it requires considering every subset $\mathscr{F} \subseteq \mathscr{V}'$.

3.1.2 Uncertain Update over Probabilistic XML Encoding

For a more usable solution, we perform updates directly on the p-document representation $(\mathscr{G}, \widehat{\mathscr{P}})$ of the uncertain multi-version document $(\mathscr{G}, \Omega)$. Algorithm 3.1

Algorithm 3.1: updPrXML <$(\mathscr{G}, \Omega)$, updOP $_{\Delta,e,e'}$>

Input: $(\mathscr{G}, \widehat{\mathscr{P}})$, $\texttt{updOP}_{\Delta,e,e'}$
Output: result of applying $\texttt{updOP}_{\Delta,e,e'}$ over $(\mathscr{G}, \widehat{\mathscr{P}})$

```
1  G := G ∪ ({e'}, {(e, e')});
2  foreach (u in Δ) do
3      if u = ins(i, x) then
4          y := findNodeById (P̂, i) ;
5          if matchIsFound(T_y, x) then
6              fie_o(x) := getFieOfNode(x) ;
7              setFieOfNode (x, fie_o(x) ∨ e');
8          else
9              updContent(P̂, ins(i, x));
10             setFieOfNode(x, e');
11         end
12     end
13     else if u = del(i) then
14         x := findNodeById (P̂, i) ;
15         fie_o(x) := getFieOfNode(x) ;
16         setFieOfNode(x, fie_o(x) ∧ ¬e');
17     end
18 end
19 return (G, P̂);
```

(`updPrXML()`) describes how such an update operation $\texttt{updOP}_{\Delta,e,e'}$ is performed on top of an uncertain representation $(\mathscr{G}, \widehat{\mathscr{P}})$. First, the graph is updated as given above. Then, for each operation u in Δ, the algorithm retrieves the targeted node in $\widehat{\mathscr{P}}$ using `findNodeById()` (this is a constant-time operation). According to the type of operation, there are two possibilities.

1. If u is an insertion of a node x, the algorithm checks if x does not already occur in $\widehat{\mathscr{P}}$, for instance by looking for a node with the same label (the function `matchIsFound()` searches a matching for x in the subtree $\mathscr{T}_y$ rooted at y). If such a matching exists, `getFieOfNode()` returns its current formula $\mathit{fie}_o(x)$ and the algorithm updates it to $\mathit{fie}_n(x) := \mathit{fie}_o(x) \vee e'$, specifying that x appears when this update is valid. Otherwise, `updContent()` and `setFieOfNode()` respectively inserts the node x in $\widehat{\mathscr{P}}$ and sets its associated formula as $\mathit{fie}_n(x) = e'$.
2. If u is a deletion of a node x, the algorithm gets its current formula $\mathit{fie}_o(x)$ and

sets it to $\mathit{fie}_n(x) := \mathit{fie}_o(x) \wedge \neg e'$, specifying that x must be removed from possible worlds where this update is valid.

Example 3.1.2. *Continuing with Example 3.1.1, the content of the p-document – consisting of nodes in $\mathscr{T}$ having each the formula e_1 – after its update according Algorithm 3.1 and by considering the script $\Delta = \{\texttt{ins}(\textit{1, sect}), \texttt{del}(\textit{11}), \texttt{ins}(\textit{3, text}_3)\}$ and the new event e_2 will contain the new nodes sect and text_3 with associated formulas e_2. The node text_2 is still in the p-document but its formula will be updated to $e_1 \wedge \neg e_2$ to represent its removal (in other terms, its no relevance) when e_2 is true.*

The rest of this section shows the correctness and the efficiency of our approach: First, we establish that Algorithm 3.1 respects the semantics of updates. Second, we show that the behavior of deterministic version control systems can be simulated by considering only a specific kind of event set. Third, we characterize the complexity of the algorithm.

Proposition 3.1.1. *Algorithm 1, when ran on a probabilistic XML encoding $\widehat{\mathscr{T}_{mv}} = (\mathscr{G}, \widehat{\mathscr{P}})$ of a multi-version document $\mathscr{T}_{mv} = (\mathscr{G}, \Omega)$, together with an update operation $\texttt{updOP}_{\Delta,e,e'}$, computes a representation $\texttt{updOP}_{\Delta,e,e'}(\widehat{\mathscr{T}_{mv}})$ of the multi-version document $\texttt{updOP}_{\Delta,e,e'}(\mathscr{T}_{mv})$.*

Proof. Let: $\begin{cases} \texttt{updOP}_{\Delta,e,e'}(\widehat{\mathscr{T}_{mv}}) = (\mathscr{G}', \widehat{\mathscr{P}'}) \\ \texttt{updOP}_{\Delta,e,e'}(\mathscr{T}_{mv}) = (\mathscr{G}', \Omega') \end{cases}$

It is clear that the version space DAG is the same in both cases. We need to show that Ω' corresponds to the semantics of $\widehat{\mathscr{P}'}$; that is, if we denote the semantics of $(\mathscr{G}', \widehat{\mathscr{P}'})$ as $(\mathscr{G}', \Omega'')$, we need to show that $\Omega' = \Omega''$. By definition, for $\mathscr{F} \subseteq \mathscr{V}'$, $\Omega'(\mathscr{F}) = \Omega(\mathscr{F})$ if $e' \notin \mathscr{F}$, and $\Omega'(\mathscr{F}) = [\Omega(\mathscr{F} \backslash \{e'\})]^{\Delta}$ otherwise. Let us distinguish these two cases.

In the first scenario implying subsets $\mathscr{F}$ which do not contain e', we have $\Omega'(\mathscr{F}) = \Omega(\mathscr{F})$. Since $\mathscr{T}_{mv}$ is the semantics of $\widehat{\mathscr{T}_{mv}}$, we know that $\Omega(\mathscr{F}) = \nu(\mathscr{F})$ for a valuation ν that sets the special revision variable b' corresponding to e' to false. Now, let us look at the document $\nu(\widehat{\mathscr{P}'})$. By construction the update algorithm does not delete any node from $\widehat{\mathscr{P}}$ but just inserts new nodes and modifies some formulas. Suppose that there exists a node $x \in \nu(\widehat{\mathscr{P}})$ such that $x \notin \nu(\widehat{\mathscr{P}'})$. Since $x \in \nu(\widehat{\mathscr{P}})$, x cannot be a new node in $\widehat{\mathscr{P}'}$. Thereby, its new formula $\mathit{fie}_n(x)$ after the update is either $\mathit{fie}_o(x) \vee e'$ or $\mathit{fie}_o(x) \wedge \neg e'$. In both cases, $\mathit{fie}_n(x)$ satisfies ν, because $\mathit{fie}_o(x)$ satisfies ν and ν sets b' (and therefore e') to false. This leads to a contradiction and we can conclude that for all node $x \in \nu(\widehat{\mathscr{P}})$, we have $x \in \nu(\widehat{\mathscr{P}'})$. Similarly, if a node x is in $\mathscr{F}(\widehat{\mathscr{P}'})$, because ν sets e' to false, x will also be in $\nu(\widehat{\mathscr{P}})$. Combining the two, $\Omega''(\mathscr{F}) = \nu(\widehat{\mathscr{P}'}) = \nu(\widehat{\mathscr{P}}) = \Omega(\mathscr{F})$.

The second scenario concerns subsets $\mathscr{F}'$ in which e' appears. We obtain a version $\Omega'(\mathscr{F}')$ by updating $\Omega(\mathscr{F}'\backslash\{e'\})$ with Δ. Let us set $\mathscr{F} = \mathscr{F}'\backslash\{e'\}$. There exists a valuation ν such that $\nu(\widehat{\mathscr{P}}) = \Omega$ (and thus, $\Omega'(\mathscr{F}') = [\nu(\widehat{\mathscr{P}})]^{\Delta}$) with ν setting all variables of events in $\mathscr{F}$ to true, and making sure that all other events are set to false. Let ν' be the extension of ν where all variables of e' are set to true. It suffices to prove that $[\nu(\widehat{\mathscr{P}})]^{\Delta} = \nu'(\widehat{\mathscr{P}'})$. First, it is clear that the nodes in $\nu(\widehat{\mathscr{P}})$ which are not modified by Δ are also in $\nu'(\widehat{\mathscr{P}'})$. Indeed, their associated formulas do not change in $\widehat{\mathscr{P}'}$, and hence the fact these satisfy ν are sufficient for selecting them in $\widehat{\mathscr{P}'}$ with the valuation ν'. Suppose now an operation u in Δ involving a node x: u either adds x as a child of a certain node y or deletes x. In the former case, if y exists in $\nu(\widehat{\mathscr{P}})$, then its formula satisfies ν and x is added in the document when it does not already exist. With Algorithm 3.1, u is interpreted in $\widehat{\mathscr{P}'}$ by the existence of x under y with an attached formula being either $\mathit{fie}_n(x) = e'$ (newly added) or $\mathit{fie}_n(x) = \mathit{fie}_o(x) \vee e'$ (reverted node). As a consequence, $\nu'(\widehat{\mathscr{P}'})$ selects x as in both possible expressions of $\mathit{fie}_n(x)$. Let us analyze the case where u is a deletion of x. If x is not present in $\nu(\widehat{\mathscr{P}})$, i.e., u changes nothing in this document. Through Algorithm 3.1, u results in a new associated formula set to $\mathit{fie}_n(x) = \mathit{fie}_o(x) \wedge \neg e$ for the node x in $\widehat{\mathscr{P}'}$. Obviously, we can see that x will not be in $\nu'(\widehat{\mathscr{P}'})$ because the satisfiability of $\mathit{fie}_n(x)$ requires the falseness of e' whose condition does not hold in $\mathscr{F}$. Now, if x is found in $\nu(\widehat{\mathscr{P}})$, u deletes the node, as well as its children, from the document. As a result, the outcome does not contain x, which is conform to the fact that $x \notin \nu'(\widehat{\mathscr{P}'})$. We have proved that for all node x in $[\nu(\widehat{\mathscr{P}})]^{\Delta}$, x is also in $\nu'(\widehat{\mathscr{P}'})$. By similar arguments, we can show that the converse is verified, i.e., for all node x in $\nu'(\widehat{\mathscr{P}'})$, x belongs to $[\nu(\widehat{\mathscr{P}})]^{\Delta}$. □

The semantics of update is therefore the same, whether stated on uncertain multi-version documents, or implemented as in Algorithm 3.1. We now show that this semantics is compatible with the classical update operation of version control systems.

Proposition 3.1.2. *The formal definition of updating in uncertain multi-version documents implements the semantics of the standard update operation in deterministic version control systems when sets of events are restricted to rooted branches.*

Proof. (Sketch) The update in our model changes the version space $\mathscr{G}$ similarly to a deterministic version control setting. As for its evaluation over the set of versions, we only need to show that the operation also produces a new version by updating the version mapping B_0^i (with e the ith version control event in $\mathscr{G}$) with Δ as in a deterministic formalism. For building the resulting version set, the operation as given above is defined

such that for all subset $\mathscr{F} \subseteq \mathscr{V}$ with $e \in \mathscr{F}$, we carry out Δ on $\Omega(\mathscr{F})$ for producing a new version $\Omega'(\mathscr{F} \cup \{e'\})$. Amongst all the subsets satisfying this condition, obviously there is at least one which maps to B_0^i. □

We conclude by showing that our algorithm is fully scalable:

Proposition 3.1.3. *Algorithm 3.1 performs the update process over the representation of any uncertain multi-version XML document with a constant time complexity with respect to the size of the input document. The size of the output probabilistic tree grows linearly in the size of the update script.*

Proof. The first part of the algorithm consists in updating $\mathscr{G}$. This is clearly a constant-time operation, which results in a single new node and a single new edge in $\mathscr{G}$ for every edit script. As for the second part of the algorithm, i.e., the evaluation of the update script over the probabilistic tree, let $|\widehat{\mathscr{P}}|$ and $|\Delta|$ be respectively the size of the input probabilistic document $\widehat{\mathscr{P}}$ and the length of Δ. By implementing $\widehat{\mathscr{P}}$ as an amortized hash table, we execute a lookup of nodes in $\widehat{\mathscr{P}}$ based on `findNodeById()` or `matchIsFound()` in constant time. (`matchIsFound()` requires storing hashes of all subtrees of the tree, but this data structure can be maintained efficiently – we omit the details here.) The upper bound of Algorithm 3.1 occurs when Δ consists only of insertions. Since the functions `getFieOfNode()`, `updContent()` and `setFieOfNode()` also have constant execution costs, we can state that the overall running time of Algorithm 3.1 is only a function of the number of operations in Δ. As a result, we can conclude that the update algorithm performs in $O(1)$ with respect to the number of nodes in $\widehat{\mathscr{P}}$ and $\mathscr{G}$.

At each execution, Algorithm 3.1 will increase the input probabilistic tree by a size bounded by a constant for each update operation, together with the size of all inserts. To sum up, the size increase is linear in the size of the original edit script. □

3.2 Evaluation of the Uncertain XML Version Control Model

This section describes the experimental evaluation of the complexity of the update operations – and the performance of our uncertain XML version control model in general – based on real-world applications. We present a comparative study of our model with two popular version control systems Git and Subversion, in order to prove its efficiency.

All times shown are CPU time, obtained by running in-memory tests, avoiding disk I/O costs by putting all accessed file systems in a RAM disk. Measures have been carried out using the same settings for all three systems.

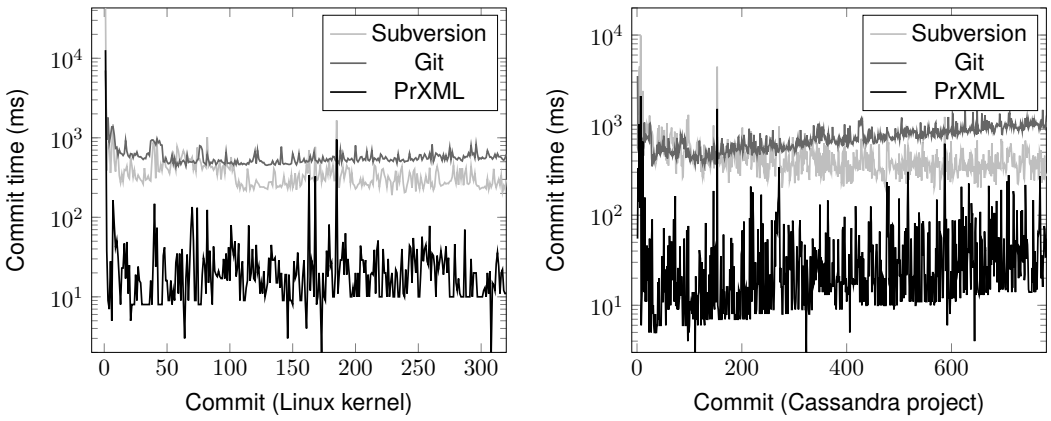


Figure 3.1: Measures of commit time over real-world datasets (logarithmic y-axis)

3.2.1 Performance analysis

We measured the time needed for the execution of two main operations: the *commit* and *checkout* of a version. The tests were conducted on Git, Subversion, and the implementation of our model (PrXML). The goal is to show the feasibility of our model rather than to prove that it is more efficient than the mentioned version control systems. We stress that, though for comparison purposes our system was tested in a deterministic setting, its main interest relies in the fact that it is able to represent uncertain multi-version documents, as we illustrate further in Section 3.2.2.

Datasets and Implementation For datasets, we used the history of the master branches of the *Linux kernel development*[1] and the *Apache Cassandra project*[2] for the tests. These data represent two large file systems and constitute two examples of tree-structured data shared in an open and collaborative environment. The Linux kernel development natively uses Git. We obtained a local copy of its history by cloning the master development branch. We maintained up-to-date our local copy by pulling every day the latest changes from the original source. We followed a similar process with the Cassandra dataset (a Subversion repository).

In total, each local branch has more than ten thousand commits (or revisions). Each commit materializes a set of changes, to the content of files or to their hierarchy (the file system tree). In our experiments, we focused on the commits applied to the file system tree and ignored content change. We determined the commits and the derivation

1. https://www.kernel.org/
2. http://cassandra.apache.org/

relationships from Git and Subversion logs. We represented the file system in an XML document and we transposed the atomic changes to the file system into edit operations on the XML tree. To each insertion, respectively deletion, of a file or a directory in the file system corresponds an insertion, respectively a deletion, of a node in the XML tree.

We implemented our version control model (PrXML) in Java. We used the Java APIs SVNKit [3] and JGit [4] to set up the standard operations of Subversion and Git. The purpose was to perform all the evaluations in the same conditions. Subversion uses a set of log files to track the changes applied to the file system at the different commits. Each log file contains a set of paths and the change operations associated to each path. As for Git, it handles several versions of a file system as a set of related Git tree objects represented by the hashes of their content. A Git tree object represents a snapshot of the file system at a given commit.

Cost Analysis Figures 3.1 and 3.3 compare the cost of the *commit* and the *checkout* operations in Subversion, Git, and PrXML.

Commit Time The commit time indicates the time needed by the system to create a version (commit), whereas the checkout time corresponds to the time necessary to compute and retrieve the sought version. The obtained results show clearly that PrXML have good performance with respect to Git and Subversion systems. The experiments were done using the datasets obtained from the Linux Kernel and Cassandra projects, as indicated above. For both datasets, we observe in Figure 3.1 that our model has in general a low commit cost [5] (note that the y-axes are logarithmic on Figure 4).

An in-depth analysis of the results show that the commit costs depend in our model on the number of edit operations associated to the commits (see Figure 3.2), as implied by Proposition 3.1.3. However, PrXML remains efficient compared to the other systems, except for some few commits characterized by a large number of edits (at least one hundred edit operations). This can be explained by the fact that our model performs the edit operations over XML trees, whereas Git stores the hashes of the files indexed by the directory names, and Subversion logs the changes together with the targeted paths in flat files. An insertion of a subtree (a hierarchy of files and directories) in the file system can be treated as a simple operation in Git and Subversion, whereas it requires a series of node insertions in our model.

3. http://svnkit.com/
4. http://www.eclipse.org/jgit/
5. Our measures of the commit time in PrXML include the computation cost of the edit scripts Δ.

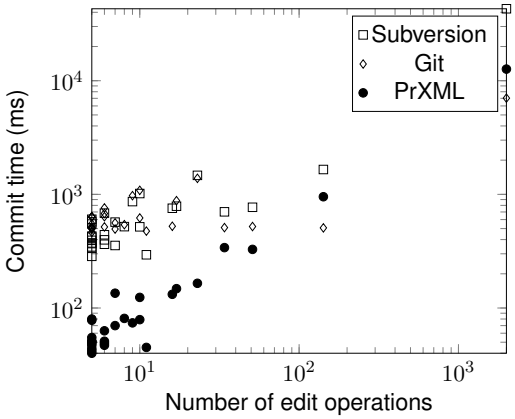


Figure 3.2: Commit time vs number of edit operations (for edit scripts of length ≥ 5)

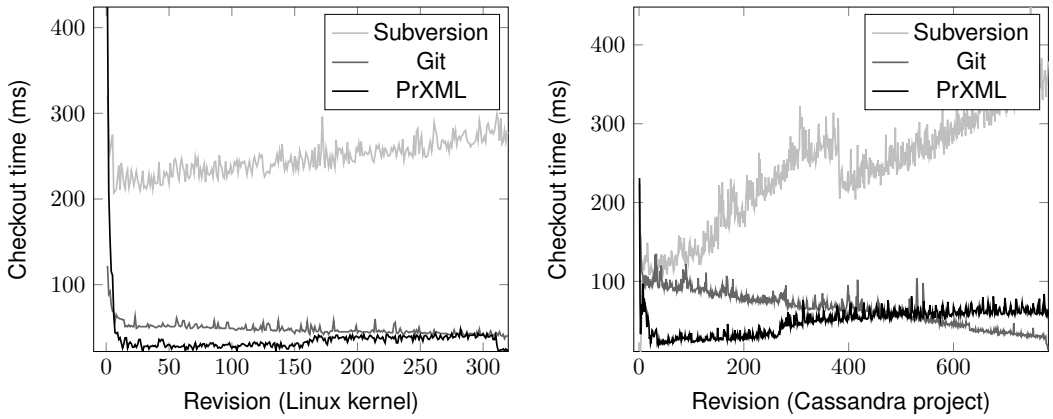


Figure 3.3: Measures of checkout time over real-world datasets (linear axes)

Checkout Time Our model is able to generate linear versions (corresponding to event sets that are rooted branches) as well as arbitrary ones. However, traditional version control systems are only able to produce linear versions. As a consequence, in this paper we focused our experiments on retrieving linear versions for comparison purposes. Figure 3.3 shows the measures obtained for the checkout of successive versions in PrXML, Git and Subversion. The x-axis represents version numbers. Retrieving a version number n requires the reconstruction of all previous versions (1 to $n-1$). The results obtained show that our model is significantly more efficient than Subversion for both datasets (Linux Kernel and Cassandra projects). Compared to Git, PrXML has a lower checkout cost for initial versions, while it becomes less efficient in retrieving recent versions for the Cassandra dataset. Note that, traditional version control models mostly

use reversible diffs [Rusu et al., 2005] in order to speed up the process of reconstructing the recent versions in a linear history.

3.2.2 Filtering Capabilities

Efficient evaluation of the uncertainty and automatic filtering of unreliable contents are two key issues for large scale collaborative editing systems. Evaluation of uncertainty is needed because a shared document can result from contributions of different persons, who may have different levels of reliability. This reliability can be estimated in various ways, such as an indicator of the overall reputation of an author (possibly automatically derived from the content of contributions, cf. [Adler and de Alfaro, 2007]) or the subjective trust a given reader has in the contributor. For popular collaborative platforms, like Wikipedia, an automatic management of conflicts is also necessary because the number of contributors is often very large. This is especially true for documents related to hot topics, where the number of conflicts and vandalism acts can evolve rapidly and compromise document integrity.

In our model, filtering unreliable contents can be done easily by setting to false the Boolean variables modeling the corresponding sources. This can be done automatically, for instance when a vandalism act is detected, or at query time to fit user preferences and opinion about the contributors. A shared document can also be regarded as the merge of all possible worlds modeled by the generated revisions. We demonstrated in [Abdessalem et al., 2011] an application of these new filtering and interaction capabilities to Wikipedia revisions: an article is no longer considered as the last valid revision, but as a merge of all possible (uncertain) revisions. The overall uncertainty on a given part of the article is derived from the uncertainty of the revisions having affected it. Moreover, the user can view the state of a document at a given revision, removing the effect of a given revision or a given contributor, or focusing only on the effect of some chosen revisions or some reliable contributors.

We also tested the possibility for the users to handle more advanced operations over critical versions of articles such as vandalized versions. We chose the most vandalized Wikipedia articles [6], and we used our model to study the impact of considering as reliable some versions affected by vandalism. We succeeded in reconstructing the chosen articles as if the vandalism had never been removed; obtaining this special version of the article is very efficient, since it consists in applying a given valuation to the probabilistic document, which is a checkout operation whose timing is comparable to what is shown

6. http://en.wikipedia.org/wiki/Wikipedia:Most_vandalized_pages

in Figure 3.3. Note that, in the current version of Wikipedia, the content of vandalized versions is systematically removed from the presented version of an article, even if some users may want to visualize them for various reasons. Our experiments have shown that we can detect the vandalism as well as Wikipedia robots do (see [Geiger and Halfaker, 2013]), and automatically manage it in PrXML, keeping all uncertain versions available for checkout.

3.3 Conclusion

We considered in this chapter the interrelated problems of the support of the standard update operations within our uncertain XML version control model, its performance evaluation, and some of its filtering capabilities. We translated and formalized the semantics of the standard XML update as corresponding to an uncertain operation which yields a set of possible versions when performing over any uncertain multi-version XML document. We empirically investigated and proved the well-definedness of the corresponding update algorithm over our probabilistic XML encoding model. We also showed that such an update algorithm has a constant time complexity with respect to the size of the input document. The size of the resulting probabilistic tree grows linearly according to the size of the update script. The comparison of our model with the most popular version control systems, done on real-world data, shows its efficiency. Moreover, our model offers new filtering and interaction capabilities which are crucial in open collaborative environments, where the data sources, the contributors and the shared content are inherently uncertain.

4 Merging in Uncertain XML Version Control

Merging is a fundamental operation in revision control systems that enables integrating different changes made to the same documents. In open platforms, such as Wikipedia, uncertainty is ubiquitous, essentially due to a lack of knowledge about the reliability of contributors and the relevance of contributions. In the perspective of modeling and assessing uncertain data in Web-scale editing collaborative platforms, we have studied and proposed, in Chapter 2, an effective and efficient version control framework designed for uncertain multi-version tree-structured documents, based on a probabilistic XML model, which already supports the standard update operation (Chapter 3). In this chapter, we investigate and define a merge operation that complements our framework and enables the conciliation of uncertain versions, answering the last point of the first problem introduced in Chapter 1. Such a merge is uncertain as it involves uncertain versions which have been produced with uncertain edits (Section 4.3.1). A general sub-problem that usually arises when merging documents is *conflict management*. At the abstract level, we detect conflicts based on a *three-way* differencing procedure. Their resolution, within our system, is transparent since conflicts correspond to a kind of uncertainties that our model naturally captures. We devise an efficient algorithm that implements the merge operation and prove its correction in Section 4.3.2; we show how conflicts can be effectively approached and managed in the case of the probabilistic XML encoding of an uncertain multi-version XML document by leveraging the *provenance formulas* associated to its uncertain nodes. We start our study with an overview of the related work about merging XML documents.

4.1 Related Work

The increasing use of XML-based systems, in particular those with a built-in version control engine, has lead to the adoption of new XML merge techniques, e.g., [Robin, 2002, Suzuki, 2002, Lindholm et al., 2006, Ma et al., 2010, Rönnau and Borghoff, 2012, Thao and Munson, 2014]. These algorithms, aware of the tree-like structure of

XML documents, have arisen as a reliable alternative to classical methods within XML settings. Indeed, traditional methods for merging text or binary files, offered by default in general-purpose version control tools (see Section 2.1), cannot detect meaningfully the semantics of changes over trees. Most current XML merge algorithms share as a baseline the diff step (edit detection) always preceding the generation of the merged document [1]. Some main differences can be stated as follows: (i) two-way versus three-way approaches, that is, the use or not of the common base document from which merged ones are derived; (ii) the set of handled edit operations; (iii) the compliance to ordered XML elements or unordered ones; and (iv) the conflict management strategy. In the following, we briefly survey a few of these algorithms for merging XML documents. We refer to [Peters, 2005, Cobéna and Abdessalem, 2009, Thao and Munson, 2014] for a more exhaustive overview of XML merge and edit detection techniques.

Deterministic Merge Merging in [Suzuki, 2002] and [Lindholm et al., 2006] has tackled ordered XML trees, more suitable in some human-edited contexts such as structured reports, rich text formats, etc. In [Suzuki, 2002], the motivation was the synchronization of versions of documents edited by different users. The author has explored a structural two-way merge via a polynomial-time algorithm which directly computes two isomorphic trees representing the merge output from the two input XML documents. The trees are progressively built in a bottom-up fashion with nodes (having unique identifiers) from the two documents, while ensuring their isomorphism during this construction by applying a series of node insertion, deletion and update when a difference is detected. As a result, the process of generating isomorphic trees, thereby the merge result, slightly involves a detection of the differences between merged XML documents. Therefore, there is an implicit processing of edit changes. However, no details are given by the author about the processing of conflicts. As for [Lindholm et al., 2006], the focus was on the reintegration [2] of changes to a document in cases when multiple independently modified copies of the document have been made. The paper has proposed a three-way XML merging algorithm with clear merge rules (e.g., node sameness, node context) and a categorization of conflicts based on real-world use cases. In contrast to [Suzuki, 2002], the algorithm of Lindholm [Lindholm et al., 2006] uses a tree matching process detecting move operations

1. A *state-based merge* relies often on external tools able to compare and merge versions whereas an *operation-based merge* assumes that the set of edits to be fused are already recorded within the editing systems, which can drive the need for application-depending merge algorithm as the encoding of edits can be specific to each individual application.

2. Merging changes that led to two distinct documents and applying the merge result into a third document.

in addition to insertions, deletions, and updates of nodes. In its merge step, *core* and *optional* conflicts are defined: a core conflict (e.g., update/update of a node) will cause a failure of the merge, whereas an optional conflict (e.g., delete/update of a sub-tree) may be tolerated. The system does not claim to resolve all conflicts, but it always reports unresolved scenarios. [Robin, 2002] has focused on the best XML matching strategy regarding node insertions and deletions. An intermediate (optimal) XML diff file encoding the matches is used to ease the merge process with the help of an XML transformation language such as XSLT [3]. This algorithm was designed to run both in a two-way setting and a three-way one regardless of the considered XML document model. [Rönnau and Borghoff, 2012] have explored and presented a merging toolkit, built on top of a context-oriented delta model, that allows for comparing and merging XML documents. The diff model uses the *context fingerprints* accounting for surrounding nodes in order to reliably identify the correct position of edit operations – this technique tries to overcome the non-persistence of node identifiers, particularly in distributed settings, which makes harder the matching of nodes. The authors claim that by doing so they can achieve an efficient merging with detection of possible conflicts. More recently, [Thao and Munson, 2014] have introduced a three-way tree merge algorithm using a specialized tree data structure that supports node identity. Their goal is to enhance the performances of existing three-way approaches by the use of the robust GNU *diff3* method, at the level of node children, in its diff detection procedure. They provide a prototype of their algorithm with a tool enabling to visualize and manually resolve conflicts. Note that the aforementioned XML merge algorithms are all deterministic, i.e., they assume trustworthy editors and edits without uncertainty.

Merge with uncertainty [Ma et al., 2010] and ourselves [Abdessalem et al., 2011] have proposed two-way merging algorithms that are intended for uncertain XML documents. The process followed consists of the same steps as in deterministic settings. The main distinction with [Ma et al., 2010] is that its merge outcome is an XML document where nodes come with some elements modeling their amount of uncertainty (the synchronizer [Ma et al., 2010] is based on *Dempster–Shafer theory* [4] to deal with uncertainty, in the form of probability values, degrees of beliefs, or necessity measures, associated to data) that does not retain enough information for retrieving individual merged versions. Our previous work, in [Abdessalem et al., 2011], is the closest study in spirit to this

3. eXtensible Stylesheet Language Transformations

4. According to Wikipedia, the Dempster–Shafer theory is a well defined setting for reasoning with uncertainty, with understood connections to other frameworks such as probability, possibility, and imprecise probability theories.

current study since both rely on the same general framework for managing uncertain XML in a typical versioning process; merging is not formally considered in [Abdessalem et al., 2011].

4.2 A Typical Three-Way Merge Process

A merge operation considers a set of versions and integrates their content in a single new one. In our proposition, detailed in the next section, we view this outcome as obtained via a three-way merge [5], that is, an integration of the changes from the inputs with respect to their common base version. We focus throughout our study on merging two versions which is the most common case in real applications. However, an extension to $n > 2$ versions is straightforward. In addition, we also assume that all merged versions only originate from updates over the base versions, i.e., we do not consider merging of versions with a different merge history – this is again for the sake of clarity of the exposition.

The merge process usually implies two steps:

1. an extraction of the different sets of edit scripts that have led to the input versions; and

2. a generation of the merge result by evaluating a unified set of the extracted edit scripts over the initial data. This last step must deal with possible conflicting edits (for the definition of conflicts, see next) due to concurrent changes (i.e., when two editors independently changes the same piece of data). The resolution of conflicts may yield several different content items for the merge. As a result, each possible merge outcome is obtained by making a choice between several possible edits. This naturally fits, as we will show in our translation, in the system with uncertainty handling because in such a setting there is no longer an only one truth but several different possibilities, each with a certain probability of validity.

We present, in this section, a typical three-way merge procedure by detailing the process of computing the edit scripts to use for the merge, as well as common merge scenarios. Assume an unordered XML document under version control. Let us give two arbitrary versions $\mathscr{T}_1$ and $\mathscr{T}_2$, along with their common lowest ancestor $\mathscr{T}_a$, of this.

5. A three-way merge enables a better matching of nodes and detection of conflicts as the structural matching of nodes is more accurate.

4.2.1 Edit Detection

We do not assume here given any explicit edit script, i.e., we concentrate on a *state-based merge* setting for the sake of more generality. As a consequence, we include edit detection as an integral part of the merge process. We define the edit script specifying the merge of versions $\mathscr{T}_1$ and $\mathscr{T}_2$ through the three-way diff algorithm $\mathit{diff3}(\mathscr{T}_1, \mathscr{T}_2, \mathscr{T}_a)$ on unordered trees with unique identifiers for nodes. The algorithm will return a script with only node inserts and deletes as allowed edit operations, an assumption which is reasonable and consistent with our general setting in Section 2.2. Similarly to [Khanna et al., 2007], we set up our $\mathit{diff3}$ based on the two-way diffs $\mathit{diff2}(\mathscr{T}_a, \mathscr{T}_1)$ and $\mathit{diff2}(\mathscr{T}_a, \mathscr{T}_2)$ as subroutines. These two-way functions separately compute two intermediate edit scripts using the same process.

- $\mathit{diff2}(\mathscr{T}_a, \mathscr{T}_1)$ initially matches the nodes in trees $\mathscr{T}_a$ and $\mathscr{T}_1$ in order to find out the shared, deleted, and inserted nodes. Then, the algorithm encodes the matches in terms of a set of node insertions and deletions which evaluated on $\mathscr{T}_a$ give $\mathscr{T}_1$. A node $x \in \mathscr{T}_a$ with no match in $\mathscr{T}_1$ is deleted, whereas a node $y \in \mathscr{T}_1$ with no match in $\mathscr{T}_a$ is added. Let us denote this edit script by Δ_1.
- $\mathit{diff2}(\mathscr{T}_a, \mathscr{T}_2)$ follows the same process and provides the script Δ_2 leading to $\mathscr{T}_2$ from $\mathscr{T}_a$.

A more global edit script, referred as Δ_3, models the final value of the $\mathit{diff3}$; Δ_3 is obtained by mixing Δ_1 and Δ_2. We describe this combination with three types of edits as follows.

Equivalent Edits An *equivalence* occurs between all edits in Δ_1 and Δ_2 with the same semantics and the same arguments (same identifiers and same labels). Specifically, two insertions $u_2 \in \Delta_1$ and $u_4 \in \Delta_2$ are equivalent if they specify the same node identifier and the same subtree to be added. As for deletions in Δ_1 and Δ_2, there is an equivalence between two if these target the same node. Given two equivalent edits, only one of the two operations is kept in Δ_3.

Conflicting Edits Any two given operations $u_2 \in \Delta_1$, $u_4 \in \Delta_2$ are *conflicting edits* when they come with different semantics, i.e, if u_2 is an insertion, then u_4 is a deletion (and conversely), and the insertion has added some new nodes as descendants of the node that is removed with the delete operation. We introduce conflicted edits in Δ_3 to be those satisfying the properties given above. Given that, we refer to the set of all conflicting edits in Δ_3 with $\Delta^{\mathscr{C}}$. We say that a node handled by conflicted edits is a

conflicted node.

Independent Edits The edits in Δ_1 and Δ_2 that do not belong to the first two classes are independent edits. The set of equivalent and independent edits form the *non-conflicted* edits of a given diff algorithm. A node impacted by a non-conflicted edit is a *non-conflicted node* for a given merge operation. (Note that conflicted and non-conflicted nodes together form the set of all nodes impacted by edit scripts in Δ_3).

Now, let us briefly present the merging scenarios (cf. usual merge options, especially *mine-conflict* and *theirs-conflict*, in tools like SubVersion [Pilato, 2004]) using *diffs* and *input versions*.

4.2.2 Common Merge Cases

A large majority of current versioning models provide three common merge scenarios that consider the resolution of possible conflicts. Recall that in most cases, this resolution is manual, that is, it requires user involvement. Let $\mathscr{T}_m$ be the outcome of the merge of $\mathscr{T}_1$ and $\mathscr{T}_2$. We formalize the three common merge scenarios as follows.

1. First, one would like to perform the merge based on $\mathscr{T}_1$ and by updating this with the non-conflicted edits from Δ_2.

$$\mathscr{T}_m = [\mathscr{T}_1]^{\Delta_2 - \Delta^{\mathscr{C}}}$$

2. The second scenario is symmetric to the first one: it considers as a base version $\mathscr{T}_2$ and fetches the non-conflicted edits from Δ_1.

$$\mathscr{T}_m = [\mathscr{T}_2]^{\Delta_1 - \Delta^{\mathscr{C}}}$$

3. Finally, the last case maps to the update of the common version $\mathscr{T}_a$ with the non-conflicted edits in Δ_3, that is, one would like to reject all the conflicting edits in the merge outcome.

$$\mathscr{T}_m = [\mathscr{T}_a]^{\Delta_3 - \Delta^{\mathscr{C}}}$$

Straightforwardly, we can show that when $\Delta^{\mathscr{C}} = \emptyset$, then we obtain the same content for the three merge scenarios. This observation is inherent to the computation of the edit scripts and the definition of the merge outcome in each scenario. Observe that we do not deal with the (intuitive and naive) merge case where the user corrects the conflicting parts with new inputs. However, this case can be simply treated by first choosing one the three outcome above and then by performing updates over this.

4.3 Merging uncertain Multi-Version XML

We detail in this section the translation of the usual three-way XML merge operation within our uncertain versioning model. Surely, an uncertain context induces an inherent uncertain merge; involved versions and *diffs* come with uncertainties. We rely on the definition of our model in Chapter 2 and consider $\mathscr{T}_{mv} = (\mathscr{G}, \Omega)$ to be an uncertain multi-version XML document having n staged version control events. We also assume its probabilistic XML encoding $\widehat{\mathscr{T}_{mv}} = (\mathscr{G},\ \widehat{\mathscr{P}})$. Remember again that each version of $\mathscr{T}_{mv}$ is identified bys a particular event in $\mathscr{G}$, the one representing the tail of the branch of $\mathscr{G}$ leading to this version. We reason on events instead of full versions since these are here uncertain and can be defined in an arbitrary manner using events.

We first abstract the uncertain merge operation under the possible-world interpretation of the set of possible versions of an uncertain multi-version XML and then we present an efficient merge algorithm over its probabilistic XML encoding.

4.3.1 Uncertain Merge Operation

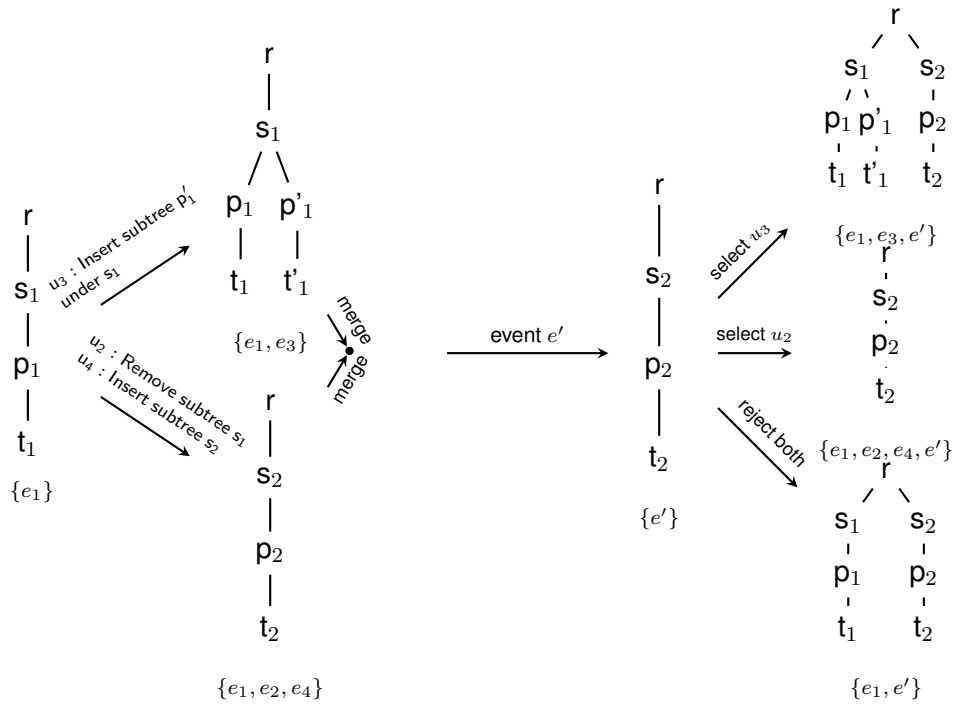
We now introduce our abstraction of an uncertain merge operation (covering at least the set up of the three common merge scenarios given in Section 4.2.2) within the uncertain multi-version XML document model.

With the help of the triple $(e_1,\ e_2,\ e')$, we refer in our setting with uncertainty to a merge operation as $\texttt{mergeOP}_{e_1,\ e_2,\ e'}$ where e_1 and e_2 point to the two versions to be merged and e' is a new event assessing the amount of uncertainty in the merge operation. We evaluate the semantics of such a merge operation over $\mathscr{T}_{mv}$ with uncertainty as follows.

$$\texttt{mergeOP}_{e_1,\ e_2,\ e'}(\mathscr{T}_{mv}) := (\mathscr{G} \cup (\{e'\}, \{(e_1,\ e'), (e_2,\ e')\}),\ \Omega').$$

On the one hand, this evaluation inserts a new event and two edges in the version space $\mathscr{G}$. On the other hand, it generates a new distribution Ω' which represents an extension of Ω with new possible versions and event sets. Let $\mathscr{A}_{e_1}$ and $\mathscr{A}_{e_2}$ be the set of all strict *ancestor events* in $\mathscr{G}$ of e_1 and e_2 respectively. We denote the common set by $\mathscr{A}_s = \mathscr{A}_{e_1} \cap \mathscr{A}_{e_2}$. For all subset $\mathscr{F} \in 2^{\mathscr{V} \cup \{e'\}}$, formally we set:

- if $e' \notin \mathscr{F}$: $\Omega'(\mathscr{F}) := \Omega(\mathscr{F})$;
- if $\{e_1, e_2, e'\} \subseteq \mathscr{F}$: $\Omega'(\mathscr{F}) := \Omega(\mathscr{F} \setminus \{e'\})$;
- if $\{e_1, e'\} \subseteq \mathscr{F}$ and $e_2 \notin \mathscr{F}$: $\Omega'(\mathscr{F}) := [\Omega((\mathscr{F} \setminus \{e'\}) \setminus (\mathscr{A}_{e_2} \setminus \mathscr{A}_s))]^{\Delta_2 - \Delta^{\mathscr{C}}}$;
- if $\{e_2, e'\} \subseteq \mathscr{F}$ and $e_1 \notin \mathscr{F}$: $\Omega'(\mathscr{F}) := [\Omega((\mathscr{F} \setminus \{e'\}) \setminus (\mathscr{A}_{e_1} \setminus \mathscr{A}_s))]^{\Delta_1 - \Delta^{\mathscr{C}}}$;



(a) base version $\mathscr{T}_a$, variants $\mathscr{T}_1$ and $\mathscr{T}_2$; scripts $\langle u_2, u_4 \rangle$ and $\langle u_3 \rangle$

(b) Merge generation: first, validate u_4 then resolve conflits between u_2 and u_3

Figure 4.1: Merge approach: (a) uncertain versions and (b) merge result

– if $\{e_1, e_2\} \cap \mathscr{F} = \emptyset$ and $e' \in \mathscr{F}$: $\Omega'(\mathscr{F}) := [\Omega((\mathscr{F} \setminus \{e'\}) \setminus ((\mathscr{A}_{e_1} \setminus \mathscr{A}_s) \cup (\mathscr{A}_{e_2} \setminus \mathscr{A}_s)))]^{\Delta_3 - \Delta^{\mathscr{C}}}$;

The aforementioned edit scripts are all deemed to be obtained using the *diff3* process that is described in Section 4.2.1. In every required case, the *diff3* works on the folowoing inputs:

– uncertain arbitrary versions $\mathscr{T}_1 = \Omega((\mathscr{F} \setminus \{e'\} \cap \mathscr{A}_{e_1}) \cup \{e_1\})$ and $\mathscr{T}_2 = \Omega((\mathscr{F} \setminus \{e'\} \cap \mathscr{A}_{e_2}) \cup \{e_2\})$ and;
– its common lowest ancestor uncertain version $\mathscr{T}_a = \Omega(\mathscr{F} \setminus \{e'\} \cap \mathscr{A}_s)$ where $\mathscr{F}$ is the subset of valid events in $\mathscr{V} \cup \{e'\}$.

Example 4.3.1. *Figure 4.1 describes the process of merging two possible versions, denoted by $\mathscr{T}_1$ and $\mathscr{T}_2$, from Figure 2.3 given their common base $\mathscr{T}_a$. In our proposal, this operation is simply encompassed with the merge specified over events e_3 and e_4 which point to the two input versions. On the left-hand side of the example, we provide the versions $\mathscr{T}_1$, $\mathscr{T}_2$ and $\mathscr{T}_a$ together with edit scripts $\{u_2, u_4\}$ and $\{u_3\}$ that led to them*

from the base $\mathscr{T}_a$. Typically, we view these scripts as given by diff functions outlined in Section 4.2.1 based on full versions. The right-hand side in Figure 4.1 explains the process of merging $\mathscr{T}_1$ and $\mathscr{T}_2$ (with the merge event e' evaluating the uncertainty in the merge) as follows: (i) First, all the edits in the scripts above coming with no conflicts, i.e., here only u_4 are validated for building the part of the merge (seen as an intermediate outcome) that is certain with the existence of e'; (ii) Then, generating the set of possible merge items by enumerating the different possibilities with the conflicting edits u_2 and u_3. The two initial possible results are obtained by propagating respectively u_2 and u_3 given the intermediate outcome. Such a propagation will give in the first case a merged version that only contains the sub-tree s_2, and in the second case a merged version with the sub-tree s_1 (including nodes p_1 and p'_1) in addition. Concretely, our merge approach will compute the same merged documents by first considering the input versions $\mathscr{T}_1$ and $\mathscr{T}_2$, and then by updating these with the edits without conflicts respectively from $\{u_3\}$ and $\{u_2, u_4\}$. Finally, the last possible content for the merge is obtained by discarding all conflicting edits and by combining the concurrent nodes in the base version with the intermediate result.

The uncertain merging operation as formalized above remains however intractable since it requires to evaluate every possible version for computing the overall merge result. Below, we propose a more convenient way to do this merge.

4.3.2 Uncertain Merging over Probabilistic XML Encoding

We properly present the semantics of an uncertain merge operation over $(\mathscr{G}, \widehat{\mathscr{P}})$ as Algorithm 4.1. This devised algorithm is both effective and efficient as we shall show shortly. Prior to a deeper description of the proposed algorithm, we start by introducing the notion of *conflicted nodes* in the PrXML$^{\text{fie}}$ probabilistic encoding given an uncertain merge with input events e_1 and e_2.

The history of edits over any specific node in $\widehat{\mathscr{P}}$ is encoded with its attached formula. We rely on this for detecting the conflicted nodes. Let us set the following valuations of events in $\mathscr{G}$: (i) ν_s setting the events in $\mathscr{A}_s$ to true and the revision variables of all other events to false; (ii) ν_1 assigning a true value to the events in $\mathscr{A}_{e_1} \cup \{e_1\}$ and a false value to the revision variables of the other events and finally; (iii) ν_2 setting the events in $\mathscr{A}_{e_2} \cup \{e_2\}$ to true and all the revision variables in the remaining events to false.

We first introduce the lineage (or provenance) of an uncertain node in the PrXML$^{\text{fie}}$ p-document.

Definition 4.3.1. *(Node lineage) The lineage formula of a given node $x \in \widehat{\mathscr{P}}$, denoted by $\mathit{fie}^{\uparrow}(x)$, is the propositional formula resulting from the conjunction of the formula of this node x with the formulas attached to all its ancestor nodes in $\widehat{\mathscr{P}}$.*

Instead of its formula [6], the lineage of a given node in the p-document encodes the entire history of edits, starting from the initial event, over the path leading to this node. Given that, we can approach the conflicted nodes in the p-document using their lineage formulas as follows.

Definition 4.3.2. *(Conflicted node) Under the general syntax $\widehat{\mathscr{T}_{mv}}$, we say that a given x in $\widehat{\mathscr{P}}$ is a conflicted node with respect to the merge implying the events e_1 and e_2 when its lineage satisfies the following conditions:*

1. $\mathit{fie}^{\uparrow}(x) \models \nu_s$;
2. $\mathit{fie}^{\uparrow}(x) \not\models \nu_1$ *(or* $\mathit{fie}^{\uparrow}(x) \not\models \nu_2$*); and*
3. $\exists y \in \widehat{\mathscr{P}}$, `desc(`*x, y*`)`*:* $\mathit{fie}^{\uparrow}(y) \not\models \nu_s$ *and* $\mathit{fie}^{\uparrow}(y) \models \nu_2$ *(or* $\mathit{fie}^{\uparrow}(y) \models \nu_1$*) where* `desc(`*x, y*`)` *means that y is a descendant of the node x.*

Proposition 4.3.1. *Definition 4.3.2 is consistent with the definition of conflicted nodes given in Section 4.2.1.*

Proof. (Sketch) Let x in $\widehat{\mathscr{P}}$ be a conflicted node such that 1) $\mathit{fie}^{\uparrow}(x) \models \nu_s$; 2) $\mathit{fie}^{\uparrow}(x) \not\models \nu_1$; 3) $\mathit{fie}^{\uparrow}(y) \not\models \nu_s$ and $\mathit{fie}^{\uparrow}(y) \models \nu_2$ with `desc(`*x, y*`)` true. The relation 1) yields $x \in \nu_s(\widehat{\mathscr{P}})$ which is a document corresponding to the common lowest ancestor of the versions $\nu_1(\widehat{\mathscr{P}})$ and $\nu_2(\widehat{\mathscr{P}})$. The relations 2) means that $x \notin \nu_1(\widehat{\mathscr{P}})$, i.e., in the history of edits that gave $\nu_1(\widehat{\mathscr{P}})$ from $\nu_s(\widehat{\mathscr{P}})$ there was at least a deletion u_2 over the node x. This is implied by the way `updPrXML()` proceeds. Besides that, 3) enables us to write on the one hand $x \in \nu_2(\widehat{\mathscr{P}})$ since $y \in \nu_2(\widehat{\mathscr{P}})$ and on the other hand $y \notin \nu_s(\widehat{\mathscr{P}})$. As a result, in the history of edits that led to $\nu_2(\widehat{\mathscr{P}})$ from $\nu_s(\widehat{\mathscr{P}})$ there was an insertion u_4 which added y as a child of x. In other words, u_2 and u_4 define two conflicted edits performed on the same node x. □

A conflicted node in $\widehat{\mathscr{P}}$ results in conflicting descendants. We refer to the conflicted set of nodes in $\widehat{\mathscr{P}}$ according to the merge of events e_1 and e_2 as the restriction $\widehat{\mathscr{P}}_{|\mathscr{C}_{\{e_1,\ e_2\}}}$. Under this, we infer below the non-conflicted set of nodes.

6. The formula just describes the semantics of edits from the event where the node was inserted for the fist time.

Definition 4.3.3. *(Non-conflicted node) For the merge of events e_1 and e_2, we define a non-conflicted node x as a node in $\widehat{\mathscr{P}} \setminus \widehat{\mathscr{P}}_{|\mathscr{C}_{\{e_1,\ e_2\}}}$ having a formula $fie(x)$ satisfying one of the following conditions.*

1. $fie(x) \models \nu_s$, $fie(x) \not\models \nu_1$ *and* $fie(x) \not\models \nu_2$
2. $fie(x) \not\models \nu_s$, $fie(x) \models \nu_1$ *and* $fie(x) \models \nu_2$
3. $fie(x) \models \nu_s$, $fie(x) \models \nu_1$ *and* $fie(x) \not\models \nu_2$
4. $fie(x) \models \nu_s$, $fie(x) \not\models \nu_1$ *and* $fie(x) \models \nu_2$
5. $fie(x) \not\models \nu_s$, $fie(x) \models \nu_1$ *and* $fie(x) \not\models \nu_2$
6. $fie(x) \not\models \nu_s$, $fie(x) \not\models \nu_1$ *and* $fie(x) \models \nu_2$

Proposition 4.3.2. *Definition 4.3.3 is consistent with the definition of non-conflicted nodes given in Section 4.2.1.*

The proof is straightforward.
To be exhaustive about non-conflicted nodes, we infer the following lemma.

Lemma 4.3.1. *Let us assume the merge over events e_1 and e_2. Given the sets $\mathscr{F}_s \subseteq \mathscr{A}_s$, $\mathscr{F}_1 \subseteq (\mathscr{A}_{e_1} \cup \{e_1\}) \setminus \mathscr{A}_s$ and $\mathscr{F}_2 \subseteq (\mathscr{A}_{e_2} \cup \{e_2\}) \setminus \mathscr{A}_s$, the expression of $fie(x)$ for any non-conflicted node $x \in \widehat{\mathscr{P}} \setminus \widehat{\mathscr{P}}_{|\mathscr{C}_{\{e_1,\ e_2\}}}$ is consistent with one of the following formulas.*

1. $\left(\bigwedge_{e_i \in \mathscr{F}_s} (e_i)\right) \wedge \neg \left(\bigwedge_{e_j \in (\mathscr{F}_1 \cup \mathscr{F}_2)} (e_j)\right)$
2. $\left(\bigwedge_{e_i \in \mathscr{F}_1} (e_i)\right) \vee \left(\bigwedge_{e_j \in \mathscr{F}_2} (e_j)\right)$
3. $\left(\left(\bigwedge_{e_i \in \mathscr{F}_s} (e_i)\right) \vee \left(\bigwedge_{e_j \in \mathscr{F}_1} (e_j)\right)\right) \wedge \neg \left(\bigwedge_{e_k \in \mathscr{F}_2} (e_k)\right)$
4. $\left(\left(\bigwedge_{e_i \in \mathscr{F}_s} (e_i)\right) \wedge \neg \left(\bigwedge_{e_j \in \mathscr{F}_1} (e_j)\right)\right) \vee \left(\bigwedge_{e_k \in \mathscr{F}_2} (e_k)\right)$
5. $\left(\bigwedge_{e_i \in \mathscr{F}_1} (e_i)\right)$
6. $\left(\bigwedge_{e_i \in \mathscr{F}_2} (e_i)\right)$

Proof. The proof relies on Definition 4.3.3. □

We continue this section by first describing Algorithm 4.1, then by demonstrating its correctness with respect to the abstraction of the merge operation in Section 4.3.1.

Algorithm 4.1: `mergePrXML` <$(\mathscr{G}, \widehat{\mathscr{P}})$, `mergeOP` $_{e_1,\ e_2,\ e'}$>

Input: $(\mathscr{G}, \widehat{\mathscr{P}})$, `mergeOP`$_{e_1,\ e_2,\ e'}$

Output: result of applying `mergeOP`$_{e_1,\ e_2,\ e'}$ over $(\mathscr{G}, \widehat{\mathscr{P}})$

1 $\mathscr{G} := \mathscr{G} \cup (\{e'\}, \{(e_1, e'), (e_2, e')\})$;

2 **foreach** *non-conflicted node* x *in* $\widehat{\mathscr{P}} \setminus \widehat{\mathscr{P}}_{|\mathscr{C}_{\{e_1,\ e_2\}}}$ **do**

3 `replace`$(fie(x),\ e_1,\ (e_1 \vee e'))$;

4 `replace`$(fie(x),\ e_2,\ (e_2 \vee e'))$;

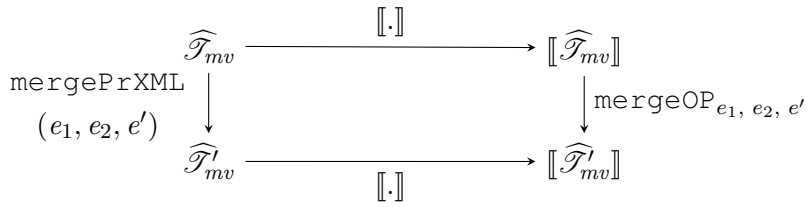
5 **end**

6 **return** $(\mathscr{G}, \widehat{\mathscr{P}})$

Description of our Merge Algorithm Algorithm 4.1 considers as inputs the probabilistic encoding $(\mathscr{G}, \widehat{\mathscr{P}})$ of an uncertain multi-version XML document $\mathscr{T}_{mv}$ and an uncertain merge operation `mergeOP`$_{e_1,\ e_2,\ e'}$; events e_1 and e_2 already existing in $\mathscr{G}$, and the fresh version control event e' modeling both the merge content items and the amount of uncertainty in these. Given that, Algorithm 4.1 first updates $\mathscr{G}$ as specified in Section 4.3.1. Then, the merge in $\widehat{\mathscr{P}}$ will result in a slight change in formulas attached to certain non-conflicting nodes in $\widehat{\mathscr{P}} \setminus \widehat{\mathscr{P}}_{|\mathscr{C}_{\{e_1,\ e_2\}}}$. The function `replace()` modifies such formulas by substituting all occurrences of e_1 and e_2 by $(e_1 \vee e')$ and $(e_2 \vee e')$ respectively. The idea is that each possible merge outcome, which occurs when e' is valuated to `true` regardless of the valuation of the other events, must come with at least the non-conflicted nodes from $\widehat{\mathscr{P}}$ seen as valid with e_1 and e_2. The remaining non-conflicted nodes, whose existence are independent of e_1 and e_2, will depend uniquely on the valuation of their ancestor events in each given valid event set including e'. At least, the validity of a conflicting node in a merge result relies on the *probability* of e_1 and e_2 when the event is e' *certain*. If e', together with e_1, are only valuated to `true`, we say that e_1 is more *probable* than e_2 for the merge; in this case, only conflicted nodes valid with $\mathscr{A}_{e_1} \cup \{e_1\}$ are chosen. The converse works in the same manner. Any conflicted node will be rejected with a valuation setting e' to `true` and the revision variables in both e_1 and e_2 to `false`.

Assume an uncertain multi-version XML document $\mathscr{T}_{mv} = (\mathscr{G}, \Omega)$ and the corresponding probabilistic XML encoding $\widehat{\mathscr{T}}_{mv} = (\mathscr{G}, \widehat{\mathscr{P}})$. In addition, let us define $[\![.]\!]$ as the semantics operator which, applied on $\widehat{\mathscr{T}}_{mv}$, yields its correct semantics $[\![\widehat{\mathscr{T}}_{mv}]\!] = (\mathscr{G}, [\![\widehat{\mathscr{P}}]\!])$ such that $\mathscr{G}$ is the same as in $\mathscr{T}_{mv}$ and $[\![\widehat{\mathscr{P}}]\!]$ defines the same probability distribution over a subset of documents in $\mathscr{D}$ than Ω. Given a merge operation `mergeOP`$_{e_1,e_2,e'}$, we now show the main result of this paper:

Proposition 4.3.3. *The definition of Algorithm 4.1 is correct with respect to the semantics of the merge operation over the uncertain multi-version XML document. In other words, the following diagram commutes:*



Proof. Assume: $\begin{cases} \texttt{mergeOP}_{e_1,e_2,e'}([\![\widehat{\mathscr{T}}_{mv}]\!]) = (\mathscr{G}', \Omega') \\ \widehat{\mathscr{T}}'_{mv} = (\mathscr{G}', \widehat{\mathscr{P}'}) \text{ and } [\![\widehat{\mathscr{T}}'_{mv}]\!] = (\mathscr{G}', \Omega'') \end{cases}$

Seeing that we reach the same version space using the procedure `mergePrXML()` is trivial. Now, we have to show that to Ω' will correspond $[\![\widehat{\mathscr{P}'}]\!]$; that is, $\Omega' = \Omega''$. Given each set $\mathscr{F} \subseteq \mathscr{V}'$, five scenarios must be checked for this equality.

1. For each subset $\mathscr{F}$ such that $e' \notin \mathscr{F}$, we have $\Omega'(\mathscr{F}) = \Omega(\mathscr{F})$. By definition, $\Omega(\mathscr{F}) = \nu(\widehat{\mathscr{P}})$ where ν is a valuation setting the special revision variable in e' to false and the other events to an arbitrary value. Abstracting out the formulas, we can claim that $\widehat{\mathscr{P}} \sim \widehat{\mathscr{P}'}$ regarding the procedure `mergePrXML()`. Since $e' \not\models \nu$, the result of the evaluation of ν over $(e_1 \vee e')$ and $(e_2 \vee e')$ (or their negation) only depends on the truth values of e_1 and e_2 respectively. Thus by replacing in formulas of $\widehat{\mathscr{P}'}$ all occurrences of $(e_1 \vee e')$ and $(e_2 \vee e')$ by e_1 and e_2 respectively, we are sure to build a p-document $\widehat{\mathscr{P}''}$ with $\nu(\widehat{\mathscr{P}'}) = \nu(\widehat{\mathscr{P}''})$. But by the definition of the procedure `mergePrXML()` $\widehat{\mathscr{P}''}$ is exactly $\widehat{\mathscr{P}}$. As a result, we obtain $\nu(\widehat{\mathscr{P}}) = \nu(\widehat{\mathscr{P}'})$. Knowing beforehand that $\Omega''(\mathscr{F}) = \nu(\widehat{\mathscr{P}'})$, we can state that $\Omega'(\mathscr{F}) = \Omega''(\mathscr{F})$ for any $\mathscr{F} \subseteq \mathscr{V}' \setminus \{e'\}$.
2. For each subset $\mathscr{F}$ such that $\{e_1, e_2, e'\} \cap \mathscr{F} \neq \emptyset$, we have $\Omega'(\mathscr{F}) = \Omega(\mathscr{F} \setminus \{e'\})$. Let ν be a valuation setting all the events in $\mathscr{F} \setminus \{e'\}$ to true, the revision variable in e' to an arbitrary value and the revision variables in the remaining events to false. Since e' does not occur in formulas in $\widehat{\mathscr{P}}$, we can write $\Omega(\mathscr{F} \setminus \{e'\}) = \nu(\widehat{\mathscr{P}})$ for sure. At this step, we resort to the logical consequences $(e_1 \models \nu) \Rightarrow ((e_1 \vee e') \models \nu)$ and $(e_2 \models \nu) \Rightarrow ((e_2 \vee e') \models \nu)$ regardless of the truth-value of the event e'. In the same way, $(\neg e_1 \not\models \nu) \Rightarrow (\neg(e_1 \vee e') \not\models \nu)$ and $(\neg e_2 \not\models \nu) \Rightarrow (\neg(e_2 \vee e') \not\models \nu)$. Therefore, by substituting in formulas of $\widehat{\mathscr{P}'}$ all occurrences of $(e_1 \vee e')$ and $(e_2 \vee e')$ by e_1 and e_2 respectively, we obtain the old p-document $\widehat{\mathscr{P}}$ with $\nu(\widehat{\mathscr{P}'}) = \nu(\widehat{\mathscr{P}})$ given

the semantics of `mergePrXML()`. Moreover, $\Omega'(\mathscr{F}) = \nu(\widehat{\mathscr{P}})$ because $\Omega'(\mathscr{F}) = \Omega(\mathscr{F} \setminus \{e'\})$ and $\Omega(\mathscr{F} \setminus \{e'\}) = \nu(\widehat{\mathscr{P}})$. So by inference, we can demonstrate that $\Omega'(\mathscr{F}) = \Omega''(\mathscr{F})$ using the relations $\Omega'(\mathscr{F}) = \nu(\widehat{\mathscr{P}})$, $\nu(\widehat{\mathscr{P}}) = \nu(\widehat{\mathscr{P}'})$ and $\nu(\widehat{\mathscr{P}'}) = \Omega''(\mathscr{F})$[7].

3. For each subset $\mathscr{F}$ such that $\{e_1, e'\} \cap \mathscr{F} \neq \emptyset$ and $e_2 \notin \mathscr{F}$, we have $\Omega'(\mathscr{F}) = [\Omega((\mathscr{F} \setminus \{e'\}) \setminus (\mathscr{A}_{e_2} \setminus \mathscr{A}_s))]^{\Delta_2 - \Delta^{\mathscr{C}}}$. Let $\mathscr{F}_1^+ = ((\mathscr{F} \setminus \{e'\}) \setminus (\mathscr{A}_{e_2} \setminus \mathscr{A}_s))$ be the set of valid events excepting the valid ancestor events of e_2 in $\mathscr{F} \cap (\mathscr{A}_{e_2} \setminus \mathscr{A}_s)$. For the valuation ν setting all the events in $\mathscr{F}_1^+$ to true and the revision variables in the remaining events to false, Scenarios 1 and 2 enable us to write $\nu(\widehat{\mathscr{P}})$ = $\Omega(\mathscr{F}_1^+)$ and $\nu(\widehat{\mathscr{P}}) = \nu(\widehat{\mathscr{P}'})$. Now, we just need to show that $[\nu(\widehat{\mathscr{P}'})]^{\Delta_2 - \Delta^{\mathscr{C}}}$ = $\nu'(\widehat{\mathscr{P}'})$ (ν' being the valuation that sets all the events in $\mathscr{F}$ to true and the revision variables in all others to false) to obtain the expected proof, that is, $\Omega'(\mathscr{F}) = \Omega''(\mathscr{F})$. Let us set Δ = $\Delta_2 - \Delta^{\mathscr{C}}$. We distinguish the two following cases.

   (a) For the class of nodes x in $\nu(\widehat{\mathscr{P}'})$ unmodified by Δ. That is, $x \in \nu(\widehat{\mathscr{P}'})$ and $x \in [\nu(\widehat{\mathscr{P}'})]^{\Delta}$. For each node x in this class, $\mathit{fie}^{\uparrow}(x)$ in $\widehat{\mathscr{P}'}$ requires the trueness of events in $\mathscr{F}_1^+$ since $x \in \nu(\widehat{\mathscr{P}'})$. Moreover, by the definition of Δ, x may be either a conflicted node or its existence is independent of the values of events in $(\mathscr{F} \cap (\mathscr{A}_{e_2} \setminus \mathscr{A}_s)) \cup \{e_2\})$. If x is a conflicted node, it is intuitive to see that $\mathit{fie}^{\uparrow}(x)$ in $\widehat{\mathscr{P}'}$ is satisfied if events in $\mathscr{F}_1^+$ are all set to true and not if only those in $(\mathscr{F} \cap \mathscr{A}_{e_2})$ are set to true (cf. Definition 4.3.1 and `updPrXML`). In the another case, $\mathit{fie}^{\uparrow}(x)$ is only function of the values of events in $\mathscr{F}_1^+$ for the set $\mathscr{F}$ (typically when $\mathit{fie}^{\uparrow}(x)$ is Expression 5 in Lemma 4.3.1 with $\mathscr{F}_1 = \mathscr{F}_1^+ \cap ((\mathscr{A}_{e_1} \setminus \mathscr{A}_s) \cup \{e_1\})$). In both cases, we can state that $\mathit{fie}^{\uparrow}(x) \models \nu'$ since ν' similarly to ν sets all the events in $\mathscr{F}_1^+$ to true. As a result, we have $x \in \nu'(\widehat{\mathscr{P}'})$ when x belongs to this class.

   (b) For the class of nodes $x \in \nu(\widehat{\mathscr{P}'})$ handled with Δ. Let $x \notin [\nu(\widehat{\mathscr{P}'})]^{\Delta}$, that is, there is an operation u in Δ that removes x from $\nu(\widehat{\mathscr{P}'})$. By the construction of Δ, it is easy to show that $\mathit{fie}(x)$ in $\widehat{\mathscr{P}}$ maps to Expression 3 in Lemma 4.3.1 with $\mathscr{F}_s = \mathscr{F}_1^+ \cap \mathscr{A}_s$, $\mathscr{F}_1 = \mathscr{F}_1^+ \cap ((\mathscr{A}_{e_1} \setminus \mathscr{A}_s) \cup \{e_1\})$ and $\mathscr{F}_2 = (\mathscr{F} \cap (\mathscr{A}_{e_2} \setminus \mathscr{A}_s)) \cup \{e_2\}$. In $\widehat{\mathscr{P}'}$, this formula $\mathit{fie}(x)$ is just updated across Algorithm 4.1 by replacing the events e_1 (the disjunction) in the first member and e_2 in the second one respectively by $(e_1 \vee e')$ and $(e_2 \vee e')$. Since ν' sets all the events in $\mathscr{F}$ to true, therefore the first member of $\mathit{fie}(x)$ will be valuated to true while the second member will be valuated to false. As a consequence, clearly we can state that

7. The last relation is due to the fact that the events e_1 and e_2 are both valuated to true.

$fie(x) \not\models \nu'$, i.e., $x \notin \nu'(\widehat{\mathscr{P}'})$. In summary, we proven that for each node x in $\nu(\widehat{\mathscr{P}'})$ deleted with Δ, x is also not chosen by ν' in $\nu'(\widehat{\mathscr{P}'})$. Note that the case where x does not occur in $\nu(\widehat{\mathscr{P}})$ is trivial and it corresponds to a scenario in which $fie(x)$ maps to Expression 1 in Lemma 4.3.1 with $\mathscr{F}_s = \mathscr{F}_1^+ \cap \mathscr{A}_s$, $\mathscr{F}_1 = \mathscr{F}_1^+ \cap ((\mathscr{A}_{e_1} \setminus \mathscr{A}_s) \cup \{e_1\})$ and $\mathscr{F}_2 = (\mathscr{F} \cap (\mathscr{A}_{e_2} \setminus \mathscr{A}_s)) \cup \{e_2\}$. Now, let $x \notin \nu(\widehat{\mathscr{P}'})$ and $x \in [\nu(\widehat{\mathscr{P}'})]^{\Delta}$. That is, there is an insertion u in Δ that adds the node x as a child of a node y in $\nu(\widehat{\mathscr{P}'})$. Let $\mathscr{F}_2^+ = (\mathscr{F} \setminus \{e'\} \cap \mathscr{A}_{e_2})$ be the set of all valid ancestor events of e_2 in $\mathscr{F}$. By the definition of Δ, we can state that the formula $fie(x)$ of x in $\widehat{\mathscr{P}}$ maps to Expression 4 or 6 in Lemma 4.3.1 with $\mathscr{F}_s = \mathscr{F}_1^+ \cap \mathscr{A}_s$, $\mathscr{F}_1 = \mathscr{F}_1^+ \cap ((\mathscr{A}_{e_1} \setminus \mathscr{A}_s) \cup \{e_1\})$ and $\mathscr{F}_2 = (\mathscr{F} \cap (\mathscr{A}_{e_2} \setminus \mathscr{A}_s)) \cup \{e_2\}$. If $fie(x)$ is Expression 4, this formula in $\widehat{\mathscr{P}'}$ is updated by Algorithm 4.1 which replaces e_1 in the first member (the conjunction) and e_2 in the second member by $(e_1 \vee e')$ and $(e_2 \vee e')$ respectively. Since ν' sets all the events in $\mathscr{F}$ to true, the first member of $fie(x)$ will be valuated to false. As for the second member, it will be valuated to true under ν' because all the events in $\mathscr{F}_2$ are set to true and $(e_2 \vee e') \models \nu'$. As a result, ν' will select x in $\nu'(\widehat{\mathscr{P}'})$ since $fie(x)$ is such that $fie(x) \models \nu'$. Otherwise, if $fie(x)$ is Expression 6, it is updated in $\widehat{\mathscr{P}'}$ by `mergePrXML()` which substitutes e_2 by $(e_2 \vee e')$. Given that it is straightforward to prove that $fie(x) \models \nu'$ since all the events in $\mathscr{F}_2$ are set to true and $(e_2 \vee e') \models \nu'$.

Scenarios (a) and (b) demonstrate that when $x \in [\nu(\widehat{\mathscr{P}'})]^{\Delta}$, then $x \in \nu'(\widehat{\mathscr{P}'})$. Similarly, the converse can be reached. We conclude $[\nu(\widehat{\mathscr{P}'})]^{\Delta} = \nu'(\widehat{\mathscr{P}'})$ which joint with $\nu(\widehat{\mathscr{P}'}) = \Omega(\mathscr{F}_1^+)$ and $\nu'(\widehat{\mathscr{P}'}) = \Omega''(\mathscr{F})$ yield $[\Omega(\mathscr{F}_1^+)]^{\Delta} = \Omega''(\mathscr{F})$. At last, the result $\Omega'(\mathscr{F}) = \Omega''(\mathscr{F})$ relies on $\Omega(\mathscr{F}) = [\Omega(\mathscr{F}_1^+)]^{\Delta}$.

4. For each subset $\mathscr{F}$ such that $\{e_2, e'\} \cap \mathscr{F} \neq \emptyset$ and $e_1 \notin \mathscr{F}$, we have $\Omega'(\mathscr{F}) = [\Omega((\mathscr{F} \setminus \{e'\}) \cap (\mathscr{A}_{e_2} \cup \{e_2\}))]^{\Delta_1 - \Delta^{\mathscr{C}}}$. This scenario is entirely symmetric to Scenario 3.

5. For each subset $\mathscr{F}$ such that $\{e_1, e_2\} \cap \mathscr{F} = \emptyset$ and $e' \in \mathscr{F}$, we have $\Omega'(\mathscr{F}) = [\Omega((\mathscr{F} \setminus \{e'\}) \setminus ((\mathscr{A}_{e_1} \setminus \mathscr{A}_s) \cup (\mathscr{A}_{e_2} \setminus \mathscr{A}_s)))]^{\Delta_3 - \Delta^{\mathscr{C}}}$. Let set $\mathscr{F}' = (\mathscr{F} \setminus \{e'\}) \setminus ((\mathscr{A}_{e_1} \setminus \mathscr{A}_s) \cup (\mathscr{A}_{e_2} \setminus \mathscr{A}_s))$. Given a valuation ν setting all the events in $\mathscr{F}'$ to true and the revision variables in all other events to false, we can write $\Omega(\mathscr{F}') = \nu(\widehat{\mathscr{P}})$ and $\nu(\widehat{\mathscr{P}}) = \nu(\widehat{\mathscr{P}'})$ according to Scenarios 1 and 2. Similarly to Scenario 3, we have now to demonstrate that $[\nu(\widehat{\mathscr{P}'})]^{\Delta_3 - \Delta^{\mathscr{C}}} = \nu'(\widehat{\mathscr{P}'})$ (where ν' is the valuation setting all the events in $\mathscr{F}$ to true and the revision variables of all the remaining events to false) to obtain that $[\Omega(\mathscr{F}')]^{\Delta_3 - \Delta^{\mathscr{C}}} = \nu'(\widehat{\mathscr{P}'})$. This result will be sufficient

for stating that $\Omega'(\mathscr{F}) = \Omega''(\mathscr{F})$ since by definition $\Omega'(\mathscr{F}) = [\Omega(\mathscr{F}')]^{\Delta_3 - \Delta^{\mathscr{C}}}$ and $\nu'(\widehat{\mathscr{P}'}) = \Omega''(\mathscr{F})$. Let us consider $\Delta = \Delta_3 - \Delta^{\mathscr{C}}$. For the proof, the intuition here is to see that by implementation Δ can be rewritten as $(\Delta_1 - \Delta^{\mathscr{C}}) \cup (\Delta_2 - \Delta^{\mathscr{C}})$ on one side. So, if we arrive to show that for all the nodes x, y such that $x \in [\nu(\widehat{\mathscr{P}'})]^{\Delta_1 - \Delta^{\mathscr{C}}}$ and $y \in [\nu(\widehat{\mathscr{P}'})]^{\Delta_2 - \Delta^{\mathscr{C}}}$, then $x \in \nu'(\widehat{\mathscr{P}'})$ and $y \in \nu'(\widehat{\mathscr{P}'})$, we can trivially deduct that for each node $x \in [\nu(\widehat{\mathscr{P}})]^{\Delta}$, then $x \in \nu'(\widehat{\mathscr{P}'})$. These relations can be proven in the same spirit as in Scenario 3 by just noting that:

(a) For the set of unmodified nodes x in $\nu(\widehat{\mathscr{P}'})$ with Δ, $fie(x)$ in $\widehat{\mathscr{P}}$ is compatible to Expression 1 in Lemma 4.3.1 with $\mathscr{F}_1 = \mathscr{F}' \cap \mathscr{A}_s$, $\mathscr{F}_1 = (\mathscr{F} \cap (\mathscr{A}_{e_1} \setminus \mathscr{A}_s)) \cup \{e_1\}$, and $\mathscr{F}_1 = (\mathscr{F} \cap (\mathscr{A}_{e_2} \setminus \mathscr{A}_s)) \cup \{e_2\}$.

(b) For the set of nodes x in $\nu(\widehat{\mathscr{P}'})$ handled with Δ. If the operation comes from $(\Delta_1 - \Delta^{\mathscr{C}})$, $fie(x)$ is compatible to Expression 3 or 5 in case of insertions and this formula maps to Expression 4 for deletions. Concerning operations in $(\Delta_2 - \Delta^{\mathscr{C}})$, $fie(x)$ is compatible to Expression 4 or 6 for insertions and to Expression 3 for deletions.

Furthermore, the inclusion in the opposite direction, that is for each node $x \in \nu'(\widehat{\mathscr{P}'})$, then $x \in [\nu(\widehat{\mathscr{P}'})]^{\Delta}$, is obvious for nodes unchanged by Δ. For the other nodes, we only need to verify that they are correctly handled by Δ in $\nu(\widehat{\mathscr{P}'})$ depending whether it corresponds to an addition or a deletion in this document. Following that, we can state that $\Omega'(\mathscr{F}) = \Omega''(\mathscr{F})$ holds for each subset $\mathscr{F}$ of events in $\mathscr{V}$ that contains e' but not e_1 and e_2.

□

We conclude by showing the efficiency of Algorithm 4.1.

Proposition 4.3.4. *Algorithm 4.1 performs the merge over the encoding of any uncertain multi-version XML document in time proportional to the size of the formulas of nodes impacted by the updates in merged branches.*

Proof. (Sketch) The intuition behind the time complexity is that Algorithm 4.1 results in a constant-time update of DAG $\mathscr{G}$, firstly. Secondly, by viewing $\widehat{\mathscr{P}}$ as an amortized hash table, the algorithm retrieves any node impacted by a (non-conflicting) update in constant time. Finally, the `replace()` method only depends on the lengths of formulas. □

4.4 Conclusion

We have presented in this chapter our translation of the semantics of the standard merge algorithm within our uncertain XML version control model, by considering a state-based approach (which offers more generality) and a three-way procedure (which enables more reliability in the detection of edit operations and thereby conflicts). We have proposed a formalism for such a merge, which is deemed as uncertain in our case, on the abstract definition of an uncertain multi-version XML document and then we have devised a well-defined corresponding algorithm over the probabilistic XML encoding model. We have shown that conflicts can be effectively and transparently handled at both levels. We have translated and defined the notion of conflicting and non-conflicting edits on the probabilistic XML encoding of an uncertain multi-version XML document by relying on the provenance formulas of the uncertain nodes. Regarding efficiency, we have demonstrated that our merge algorithm has a time complexity which only depends on the size of the formulas of the nodes impacted by the updates in merged branches.

Part II

Structured Web Data Integration

5 Web Data Integration under Constraints

We present in this chapter our study of the problem of uncertain Web data integration under constraints – namely, *dependent sources* and *spatio-temporal restrictions* – targeting application scenarios such as the monitoring of moving objects in the maritime domain based on multiple Web sources which require accounting for both kinds of aforementioned constraints.

As we shall show shortly, in Section 5.2, Web sources in the maritime domain are various, uncertain, and dependent. The main reasons of these uncertainties come from imprecise information extractors or imperfect knowledge from human beings. Dependency relationships, on the other hand, result from the recurrent copying relationships among sources. Combining data from these uncertain sources – for instance, in order to reply to users' queries – drives the need to take into consideration the dependency relationships – leaving aside the common independence assumption – as they can constrain the provenance of data and their amount of uncertainties, as discussed in Example 1.2.1. We first approach and formalize in Section 5.3 the challenge of integrating several Web data sources under uncertainty and directed dependency relationships. We tackle this integration by rephrasing it as the fusion of several uncertain versions – represented by data from sources – of the same shared content. In the special case of tree-structured data, we put forward a probabilistic tree data integration model inspired from the first contribution of this thesis, that is, our uncertain XML version control model, detailed in Chapter 2; we give main prerequisites and detail how the outcome of the integration can be materialized while properly managing uncertainties with respect to the dependency relationships.

More generally, a number of applications – besides the maritime domain scenario – deal with monitoring moving objects: cars, aircrafts, persons (e.g., celebrities), or, more broadly, populations or groups of humans, natural phenomena such as cyclones, epidemics. Traditionally, this requires capturing data from sensor networks, image or video analysis, or using other application-specific resources. We further show in Section 5.4 how structured Web content can be exploited instead to gather information

(trajectories, metadata) about moving objects. As this content is marred with uncertainty and inconsistency, we develop a methodology for estimating uncertainty and filtering the resulting data; we mainly focus on uncertain location data management – constrained spatially and temporally – and devise some precision criteria for this goal.

Finally, we present in Section 5.5 as an application a demonstration of a system that constructs trajectories of ships from social networking data, presenting to a user inferred trajectories, meta-information, as well as uncertainty levels on extracted information and trustworthiness of data providers.

5.1 Related Work

Uncertain Data Integration Data integration with uncertainty is a recent and highly active research area – [Magnani and Montesi, 2010, Ayat et al., 2012] provide, together, a detailed insight over the literature – with mostly the use of a probabilistic approach to take into consideration uncertainties during the process. Probabilistic schema mappings, in [Dong et al., 2007, Das Sarma et al., 2008], capture uncertainty at both the level of schema mappings, data in sources, and user queries within a relational setting in which tuples and semantic mappings have associated probability values. The resulting probabilistic schema mappings describes a probability distribution over a set of possible schema mappings between a source schema and a target mediated schema: two semantics for inferring a reasonable single mapping, with respect to an entire table or subsets of specific tuples, are presented for query answering [1]. Query views in [Agrawal et al., 2010] through containment constraints are used to define the mappings between a set of uncertain sources and a probabilistic mediated database having a fixed schema in a *local-as-view* perspective [Abiteboul et al., 2012]. [Ayat et al., 2012] revisits a probabilistic mediated schema and constrain its definition with correlations between attributes implied by functional dependency rules; an occurrence of a set of correlated attributes which cannot be consistently described using an only one mediated schema yields to several possible mediated schema.

On the other hand, much effort has been made on uncertain tree data integration systems, exacerbated by the advent of the Web. Particularly, [Van Keulen et al., 2005, de Keijzer and van Keulen, 2008, van Keulen and de Keijzer, 2009] introduce a probabilistic XML framework for data integration that reconciles multiples uncertain XML

1. Observe that one of the main purposes of the setting up of data integration systems is query answering over multiple sources using a unique entry point. We defer the problem of query answering in data integration systems for future work (see Chapter 7)

documents by enumerating the set of all possible deterministic trees resulting mainly from conflicts on data values or tree structure. They present a probabilistic XML tree, that is, an ordinary XML tree equipped with *possibility* and *probability* nodes, that compactly represents this set of possible deterministic trees with probability values attached to uncertain nodes in order to describe their closeness with the reality. In light of these studies, many theoretical probabilistic XML approaches – targeting also tree data integration with uncertainty – have been further investigated with models supporting a wider semantics of probability distribution over possible XML documents. The concept of probabilistic documents (abbr. p-documents) [Abiteboul et al., 2009, Kharlamov et al., 2010] – already addressed in Sections 2.1.2 and 2.3 – generalizes all previously proposed uncertain XML models.

Moving Objects and the Web [Pianese et al., 2013] discovers and predicts user routines, i.e., regularly-occurring user activities, by integrating and filtering users' content having geographical data extracted from Twitter and Foursquare [2]. [Liu et al., 2014] demonstrates a system which extracts representative messages from Twitter posted when anomalies such as traffic jam occur in order to detect traffic events. Similar visited places by different persons have been studied and inferred by [Kanza et al., 2014] based on a clustering of identical location data – from Twitter and Instagram [3] – about the mobility of these people.

5.2 Motivating Application

In this section, we detail Web sources in the maritime domain by showing their various nature, uncertainties, and dependency relationships. We claim that in such a scenario a model able to effectively integrate multiple sources with uncertainty and dependencies should be useful. As we illustrate later with our demonstration system in Section 5.5, users or experts in the maritime domain may gain more correct insights about objects like ships when they have the ability to query and navigate through all these sources with information about real data provenance and their amount of uncertainties.

5.2.1 Multiple Web Sources

We start our study of the maritime domain by some statistics about the number of Web sources covering objects such as ships. Our first observation is that such Web

2. https://fr.foursquare.com/
3. http://instagram.com/

sources, having different nature, are numerous. Figure 5.1 shows a sample list of those sources including some popular Web platforms. The sources in Figure 5.1(a) correspond to the *social networking sites* Flickr and Twitter, and the *Web-scale collaborative platform* Wikipedia. In Flickr and Twitter, users share items like photos about ships with associated meta-information such as their names and their current locations inserted inside a tweet [4], a description, a list of tags, ect. In Wikipedia, editors collaboratively build information about the general description of some classes of ships, e.g., cruise ships. In contrast, the sources in Figure 5.1(b) – i.e., MarineTraffic [5], GrossTonnage [6], and ShippingExplorer [7] – are specifically dedicated to the real-time monitoring of objects in the maritime domain by relying, for instance, on data sent by AIS systems [8] on ships. Observe that while being heterogeneous at schema and value levels, the example sources mostly use the same unique identifiers, that is, IMO [9], for objects like ships, which makes easier the mapping of objects from distinct sources.

5.2.2 Uncertain Web Data Sources

We continue with an analysis of the reliability of the Web sources, as well as their data, based on the precision of the data acquisition process and the level of consistency of information among sources.

In the maritime domain, Web sources are mostly uncertain due to imprecise and imperfect data extraction technique used: these uncertainties should not be ignored. The AIS systems are inherently imprecise (e.g., incomplete information or noises can be sent by sensors) whereas items shared by human may be irrelevant. Besides incompleteness, another important indication of untrustworthiness of sources are contradictions between information provided about the same objects. Consider Figures 5.2(a) and 5.2(b) which respectively depict the dimensions (length, width, and draft) and the company (owner and manager) the same ship from ShippingExplorer, Wikipedia, and ShipSpotting. Obviously, we can observe that ShippingExplorer and Wikipedia agree on the owner of this ship – Wikipedia seems to be more accurate – whereas ShipSpotting indicates a different owner. In contrast, all three sources agree on the manager for which the information from ShipSpotting is more complete. As for the dimensions of the ship, some

4. A tweet denotes a short 140-character message.
5. http://www.marinetraffic.com/fr/ais/home/
6. http://grosstonnage.com/
7. http://www.shippingexplorer.net/en
8. The Automatic Identification Systems (AIS) is an automatic tracking system used on ships and vessel traffic services for identifying and locating vessels by electronically exchanging data with other nearby ships, AIS base stations, and sattelites.
9. International Maritime Organization numbers

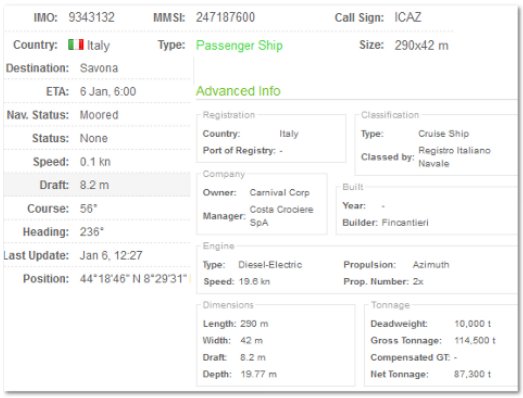
Flickr

Twitter

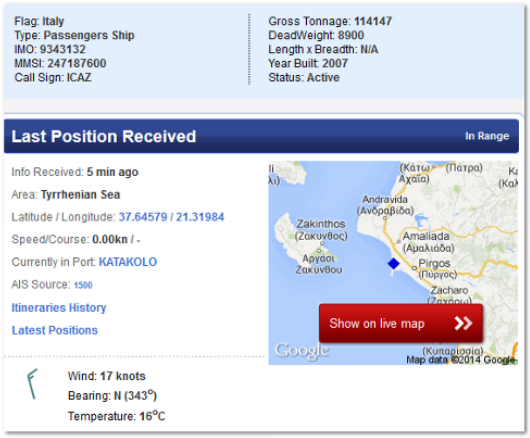
Career	
Name:	*Costa Serena*
Owner:	Carnival Corporation & plc
Operator:	Costa Crociere
Port of registry:	Italy, Genoa
Ordered:	1 October 2004
Yard number:	6130
Laid down:	1 February 2005
Launched:	4 August 2006
Completed:	9 March 2007
In service:	2007
Identification:	Call sign: ICAZ IMO number: 9343132 MMSI number: 247187600 247187600
Status:	In service
Notes:	[1][2]
General characteristics	
Class & type:	*Concordia*-class cruise ship
Tonnage:	114,147 GT 87,196 NT 10,000 DWT
Length:	289.59 m (950.1 ft)
Beam:	35.5 m (116 ft)
Draught:	8.30 m (27.2 ft)
Depth:	14.18 m (46.5 ft)

Wikipedia

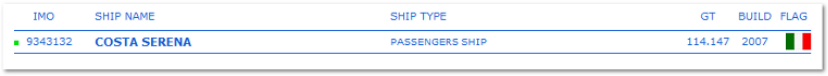
(a) Social Networks & Collaborative Platforms



ShippingExplorer



MarineTraffic



IMO	SHIP NAME	SHIP TYPE	GT	BUILD	FLAG
9343132	COSTA SERENA	PASSENGERS SHIP	114.147	2007	

GrossTonnage

(b) Sources Monitoring in Real-Time Maritime Activities

Figure 5.1: Collection of Web sources in the maritime domain

contradictions also appear between the values given by the sources. As an example, ShippingExplorer and ShipSpotting provide 42 m for the width while Wikipedia indicates 35.5 m. On the other hand, Wikipedia and ShipSpotting track 8.3 m for the draft whereas ShippingExplorer provides 8.2 m. One may consider that the difference between given distinct values for the draft attribute is not very significant, and just choose one among the two possible values. However, deciding about the correct values for both the owner and the width is less evident. We sketch in the next section a solution that keeps all conflicting values with their amount of uncertainties, enabling various forms of reasoning about the correctness of the data.

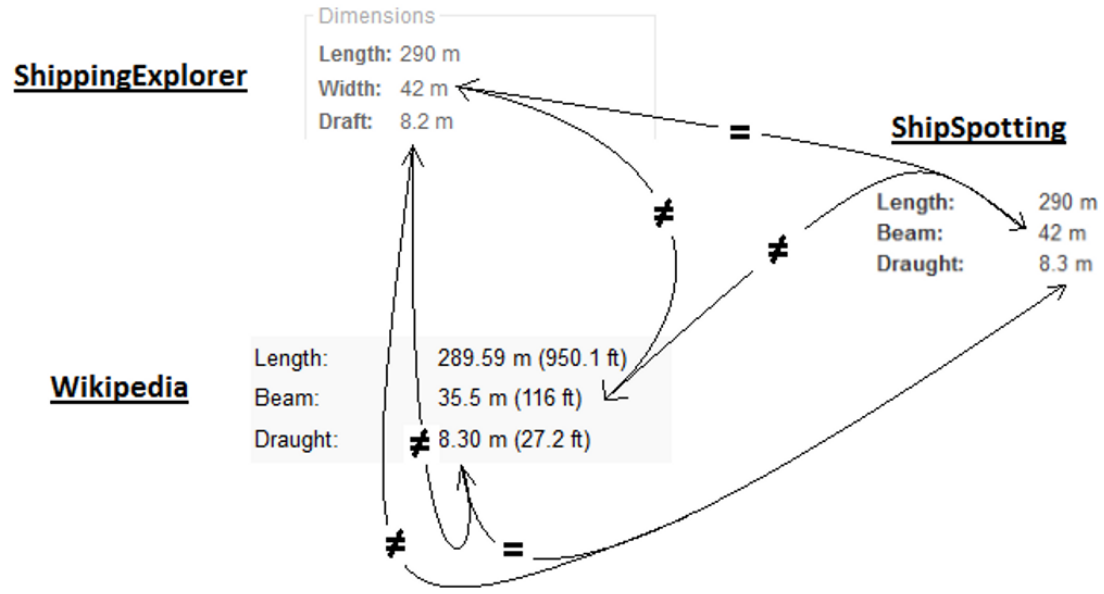
5.2.3 Copying Relationships between Sources

We finish the description of the application case by answering the following question: *Are Web sources in the maritime domain independent?*

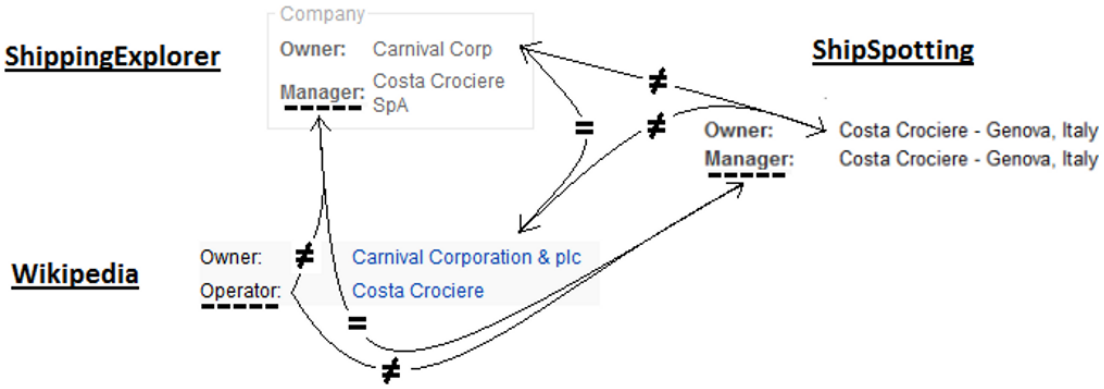
Straightforwardly, we conclude with the example in Figure 5.3 that Web sources in the maritime domain are not all independent. The main reason is that some sources collect information from other ones by copying them. In some situations, these copying relationships are explicitly mentioned by the sources, e.g., Shipspotting obtains some data from GrossTonnage as shown by Figure 5.3. In practice, however, dependent sources do not always cite their reference sources, which drives the need for algorithms to discover the dependency relationships among a set of sources in the same domain. The detection of copying relationships is beyond the scope of this work; see [Dong et al., 2009a, Dong et al., 2010] for details about detecting copying relationships on the Web. More importantly, the copiers may revise their collected data in order to provide their own knowledge about shared real-word objects. As a consequence, knowing the set of dependency relationships is crucial for finding the real provenance of each data item and for detecting in an effective way contradictions and corroborations during uncertainty management.

5.3 Web Data Integration under Dependent Sources

We approach and formalize, in this section, the integration of uncertain Web sources with dependency relationships, targeting an effective technique able to represent and assess the amount of uncertainty in data with respect to the dependencies. We first present some challenges towards such an integration technique. In the particular case of tree-structured data, we extend ourselves and show that our uncertain XML version



(a)



(b)

Figure 5.2: Uncertain Web sources: discrepancies and incompleteness

system – introduced in Chapter 2 – can be used as a basis of a probabilistic tree data integration for uncertain Web sources under dependencies. We then put forward a model and explain how it can be used for materializing this integration in the scenario where the dependencies between sources are deterministic.

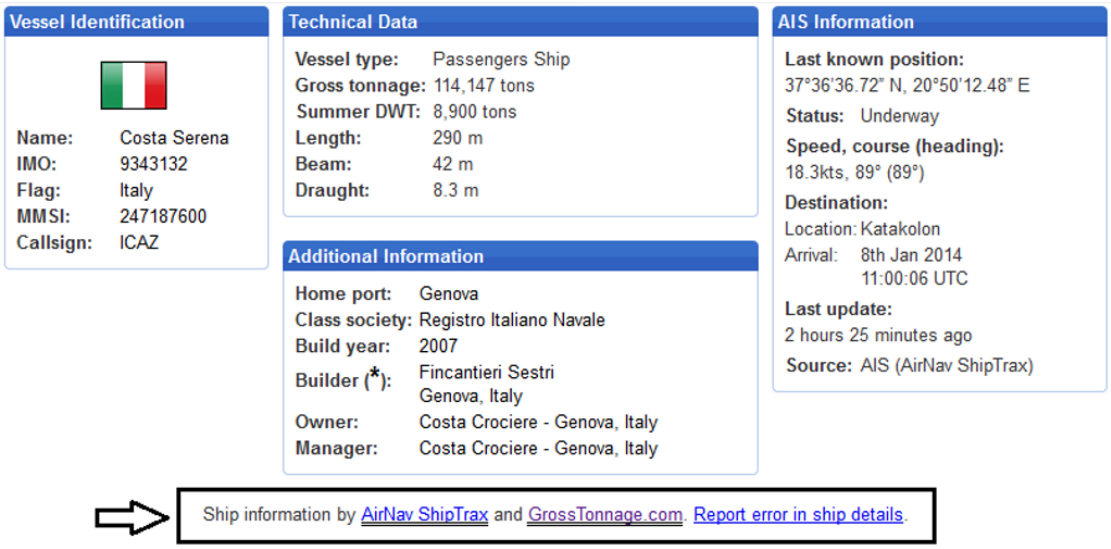


Figure 5.3: Shipspotting depends on GrossTonnage

As a general setting, we consider a set $\mathscr{S}$ of Web sources $S_1, \ldots, S_n$ – that share a set of real-world objects in the same domain – under uncertainty and dependencies. Every object is uniquely tagged with an identifier shared by all sources. Each particular source maintains and provides information about a subset of objects which have been possibly obtained from another source before being revised (in Section 5.3.2 we will suppose that these objects, as well as their description, are encoded in a sole XML tree describing content provided by the source) . On the other hand, we assume that, first, a dependency relationship involving two sources, if it occurs, is directed and there is no cycle; second, each dependency relationship is deterministic, that is, it is known with certainty and; third, a given source cannot be implied in more than one dependency relationship, that is, the dependency on multiple sources is disregarded.

5.3.1 Main Prerequisites

We highlight and claim that a data integration system intended to uncertain Web data sources with dependency relationships must account for the three following requirements.

1. First, the presence of untrustworthy sources and uncertain data drives the needs for representing and evaluating these uncertainties during the integration: this is a a common requirement in uncertain data integration systems, e.g., [Dong et al., 2007, Agrawal et al., 2010]. The dependency relationships, in addition, introduce the problem of the real provenance of data items, an issue not disconnected to the

management of uncertainties. Hence, a trustworthy source has tends to provide correct information and conversely for an untrustworthy source. An effective integration system should have the ability to (i) correlate the amount of uncertainty in sources and their content; (ii) capture the provenance of each data item: for instance, a given source may be trusted but its data are not relevant, requiring a model that allows explanation and understanding of possible integration results and their levels of relevance.

2. Second, a formal abstraction of the dependency relationships between the sources is needed. An effective and efficient algorithm – when the dependencies are not given all – in order to discover the dependency relationships between the sources is necessary.

3. Third, the integration approach – which formalizes the result of the integration and its semantics mapping with the data sources – should be defined by focusing especially on the uncertain nature of our setting and the dependency relationships: a given uncertain source may first copy others, and then revise this information with its knowledge of the modeled world. As a consequence, the intended system should have the capability to capture provenance, contradictions, and corroborations in the integration outcome with respect to the dependency relationships.

5.3.2 Probabilistic Tree Data Integration System

We detail here first steps towards a probabilistic tree data integration approach for uncertain Web sources with dependency relationships. We go further on each requirement given above by using our XML version control model with uncertainty, presented in 2, as a basis of our data integration setting. We start by formalizing the problem.

We focus on tree-structured data by considering that the content provided by every Web source is in this form. As in Chapter 2, we also adopt the class of unordered XML documents in order to describe and represent those tree-structured data.

Problem Formulation Independently of the integration problem, a setting involving a set of uncertain Web sources $S_1, \ldots, S_n$ related by dependency relationships corresponds, somehow, to an uncertain version control setting $(\mathscr{G}, \Omega)$ where Ω is a mapping over $\{S_1, \ldots, S_n\}$ giving a view over the possible unordered XML trees of their integration with respect to the dependency graph $\mathscr{G}$ under some considerations that we will clarify shortly. From such an intuition, we can reformulate the problem of integrating uncertain

Web sources with dependency relationships as the reconciliation of all the possible unordered trees defined by Ω. In other terms, this drives the need for a clear definition of the mapping between $\{S_1, \ldots, S_n\}$ and each possible unordered tree together with its probability through Ω; Lenzerini in [Lenzerini, 2002] states that the mapping is one of the key components of a data integration system. Then, a reconciliation process for the overall integration result must be clarified. As we shall see next, to reconcile this integration we use a PrXML$^{\text{fie}}$ p-document $\widehat{\mathscr{P}}$ along with the dependency graph $\mathscr{G}$. According to the given formulation of the problem, we approach our probabilistic tree data integration model as follows.

Modeling Uncertainties To be consistent with our uncertain XML version control framework, in Chapter 2, we rely on random event variables in order to deal with uncertainties in a similar manner. We repeat and show how this uncertainty management is transposed to the integration context.

Consider again B as a set of independent random Boolean variables $b_1 \ldots b_m$ and their probability values $\mathsf{P}_{\mathsf{r}}(b_1) \ldots \mathsf{P}_{\mathsf{r}}(b_m)$ of being $\mathtt{true}$ as well. We restrict B to consist of two types of disjoint sets of variables which we denote B_{r} and B_{s}. We use variables in B_{r} to manage the uncertainty in the content really provided by each source: the data not copied from the other sources, whose content, as we show shortly, is not explicitly known and should be inferred. Variables in B_{s} are used to model and to assess the reliability level of the sources. Given a source S_i, we refer to the uncertainty on its content and its reliability level with $b_{\mathsf{r},\,\mathsf{i}}$ and $b_{\mathsf{s},\,\mathsf{i}}$ respectively. We consider now the set of events $e_1, \ldots, e_n$ and manage the overall amount of uncertainty in each source S_i from $\mathscr{S}$ with an event $e_i = b_{\mathsf{r},\,\mathsf{i}} \wedge b_{\mathsf{s},\,\mathsf{i}}$ (where $b_{\mathsf{r},\,\mathsf{i}} \in \mathrm{B}_{\mathrm{r}}$ and $b_{\mathsf{s},\,\mathsf{i}} \in \mathrm{B}_{\mathrm{s}}$) associated to it. Intuitively, e_i is $\mathtt{true}$ when it produces a correct content on a reliable source.

Modeling Dependencies We consider and describe the dependency relationships between the sources with a directed acyclic graph (DAG) $\mathscr{G}$: a DAG structure is also adopted in [Dong et al., 2009a, Dong et al., 2010] for identical needs. Consistently with the uncertain XML version control, we consider that the DAG consists of events (representing implicitly sources and their data) as nodes. Given the set of events $e_1, \ldots, e_n$ associated to the sources $S_1, \ldots, S_n$, $\mathscr{G}$ is built over the set $\{e_0\} \cup \{e_1, \ldots, e_n\}$ of nodes and their edges translating the dependency relationships between the corresponding sources. The event e_0, corresponding to the root of the graph, is introduced for technical purposes as the parent event for those events associated to independent sources.

Probabilistic XML Tree Data Integration We propose a probabilistic XML tree data integration system which reconciles uncertain tree-structured Web data sources $S_1, \ldots, S_n$ with dependency graph $\mathscr{G}$. Through a probabilistic model the semantics of integrating several uncertain tree-structured sources – as shown in [Van Keulen et al., 2005] – is a set of possible integration results.

We first introduce the notion of a probabilistic XML global view ($\mathrm{PrGView}$) $\mathbb{M}$ that abstracts the set of possible integration trees $m_1, \ldots, m_k$, together with their probabilities $\mathsf{P_r}(m_1), \ldots, \mathsf{P_r}(m_k)$ of being valid, over $\{S_1, \ldots, S_n\}$. Given the infinite set of all unordered XML trees $\mathscr{D}$, an integration tree is an unordered XML document in $\mathscr{D}$ resulting from the integration of versions of the shared content from subsets of sources in $S_1, \ldots, S_n$. Such an integration tree may not be unique, especially in the presence of uncertainty, but several integration trees may be possible to describe, first, distinct views about the trust one may have on involved sources and their own information; second, the different ways to deal with contradictions and incompleteness of the data. We show later that $\mathrm{PrGView}$ is exactly introduced in order to capture all such possible results of this integration.

Definition 5.3.1. *Given a set $S_1, \ldots, S_n$ of uncertain sources with a dependency graph $\mathscr{G}$, a probabilistic XML global view $\mathbb{M}$ over their integration is a set $\{(m_i, \mathsf{P_r}(m_i)) \mid 1 \leq i \leq k\}$ of possible integration trees where (i) every m_i is constructed with respect to the dependency graph $\mathscr{G}$ and; (ii) $0 < \mathsf{P_r}(m_i) \leq 1$ with $\sum_{i=1}^{k} \mathsf{P_r}(m_i) = 1$.*

We now detail how we obtain $\mathbb{M}$ based on the set of input sources and the mapping Ω. We will focus more on the representation of the set of possible worlds than on their probabilities. We start by defining the *contribution* of a given source, i.e., its real knowledge about the real-world, within our integration setting.

Definition 5.3.2. *Let $S_1, \ldots, S_n$ be a set of uncertain sources with a dependency graph $\mathscr{G}$. The contribution of any given source S_i with respect to $\mathscr{G}$ corresponds to the real content provided by this source. We describe the contribution of the source S_i as a sequence of edit operations δ_i over an initial content.*

For every source S_i, $1 \leq i \leq n$, let $\mathscr{G}_{|\rightarrow e_i}$ be the event of the source S_l on which S_i directly depends according to the graph of dependencies $\mathscr{G}$; for each $1 \leq l \leq n$ with $i \neq l$, $e_l = \mathscr{G}_{|\rightarrow e_i}$ when the relation (e_l, e_i) holds in $\mathscr{G}$. Otherwise, $\mathscr{G}_{|\rightarrow e_i} = e_0$ in which case S_i has been generated from the root-only tree S_0. Algorithm 5.1, namely `contribute()`, computes the contribution of S_i giving its dependent data source S_l [10].

10. We also refer with S_l to the content provided by the source S_l.

The procedure `diff()` is a differencing function (see Section 4.1) that computes the difference between two unordered XML tree versions. Its output is a sequence of edit operations (node insertions and node deletions) over XML nodes.

Algorithm 5.1: `contribute` <S_l, S_i>

Input: S_l, S_i
Output: δ_i
1 $\delta_i := \texttt{diff}(S_l, S_i)$;
2 **return** (δ_i);

The events $e_1 \dots e_n$ associated to the sources $S_1, \dots S_n$ also track and manage contributions $\delta_1, \dots, \delta_n$, as well as their amount of uncertainty, of these sources. As a consequence, the events are enough to fully describe the sources because they contain necessary information about both their amount of uncertainty and their contributions. We now use Ω and define the mapping between possible integration trees in $\mathbb{M}$ and the sources $S_1, \dots, S_n$ by following a construction similar in our uncertain XML version control system in Chapter 2.

Fix $\mathscr{D}' \subseteq \mathscr{D}$ such that $\mathscr{D}'$ consists exactly to the trees $\{m_1, \dots, m_k\}$ plus the root-only tree document S_0 in $\mathscr{D}$. The mapping $\Omega : 2^{\{e_1,\dots,e_n\}} \to \mathscr{D}'$ maps every possible view over $\{S_1, \dots, S_n\}$ to a possible integration tree in $\mathbb{M}$ as follows: for each $\mathscr{F} \subseteq 2^{\{e_1,\dots,e_n\}\setminus\{e_i\}}$, $\Omega(\mathscr{F} \cup \{e_i\}) = [\Omega(\mathscr{F})]^{\delta_i}$, that is, integration of content from sources whose associated events are in $\mathscr{F} \cup \{e_i\}$ with $\Omega(\{\})$ corresponding to the root-only tree in $\mathscr{D}'$. Let us assume that $m_k = \Omega(\mathscr{F})$ for a fixed $1 \leq k \leq m$. The probability of m_k is estimated as follows.

$$\mathsf{P_r}(m_k) = \sum_{\substack{\mathscr{F} \subseteq \{e_1,\dots,e_n\} \\ \Omega(\mathscr{F}) = m_k}} \prod_{\substack{1 \leq i \leq n \\ e_i \in \mathscr{F}}} \mathsf{P_r}(e_i) \times \prod_{\substack{1 \leq i \leq n \\ e_i \notin \mathscr{F}}} (1 - \mathsf{P_r}(e_i)) \,. \tag{5.1}$$

The Ω mapping, as it is given, provides a way to construct the possible integration trees in $\mathbb{M}$ with respect to the dependency graph $\mathscr{G}$ while taking into consideration data provenance and the amount of uncertainty in data; Ω is a valid mapping between data in sources and trees in $\mathbb{M}$. The set of integration trees specified by Ω, regarding the dependency graph $\mathscr{G}$, can be reconciled with a PrXML$^{\text{fie}}$ p-document $\widehat{\mathscr{P}}$ by following the same encoding [11] of node formulas and construction of the p-document as in `updPrXML()` algorithm (see Chapter 3) according to the probabilistic XML encoding of our uncertain XML version model in Chapter 2. The edit scripts, i.e., contributions, coming with each

11. One difference between the version control setting and the integration one is that in the latter the graph of dependencies is fully complete while in the former this graph is populated when uncertain updates are issued.

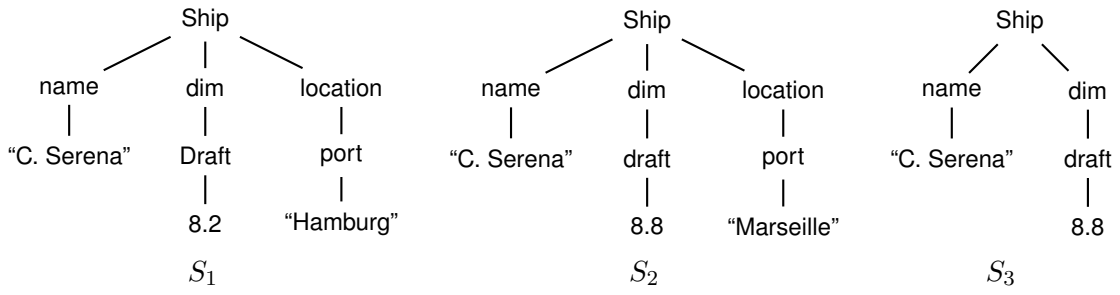
source event are previously determined with the procedure `contribute()`. Concretely speaking giving the set of uncertain sources $S_1, \ldots, S_n$ along with its dependency graph $\mathscr{G}$, we traverse $\mathscr{G}$, in a *top-down* manner, from the root to the leaves and perform the following operations: initially, at the root of $\mathscr{G}$, we set $\widehat{\mathscr{P}}$ as a root only document and then; at each other node e_i in $\mathscr{G}$, we consider the corresponding sources S_i and its direct reference source S_l with the event e_l such that (e_l, e_i) occurs in $\mathscr{G}$ and respectively we compute $\delta_i =$`contribute(`S_l, S_i`)` and use `updPrXML()` (Lines 2 to 17) for updating $\widehat{\mathscr{P}}$ with δ_i, e_i, and e_l. We repeat the stage 2 until we finish to process all the events in $\mathscr{G}$.

Definition 5.3.3. *Given a set of uncertain sources $S_1, \ldots, S_n$ with the dependency graph $\mathscr{G}$, we introduce a probabilistic tree data model which reconciles, with respect to the graph $\mathscr{G}$, these sources as a p-document $\widehat{\mathscr{P}}$ by following a similar encoding of node formulas and construction of the p-document as in* `updPrXML()` *– Lines 2 to 17 of the algorithm in Chapter 3 – and by using the procedure* `contribute()` *in order to compute the contribution of each source.*

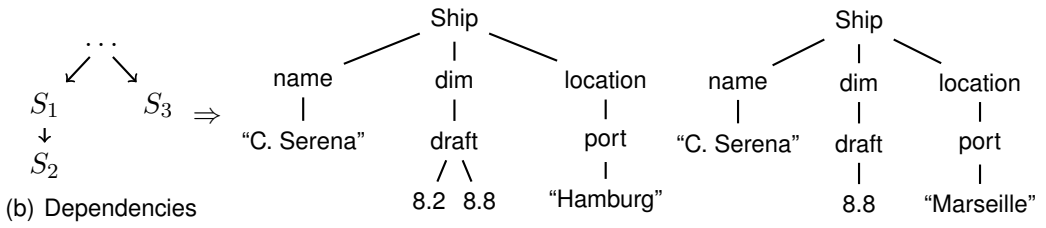
We conclude this section by first summarizing our probabilistic tree data integration model, and then by showing an example output.

A probabilistic tree data integration model over a set of sources $S_1, \ldots, S_n$ under uncertainty and dependencies is defined with the help of a triple $(\mathscr{G}, \mathbb{M}, \Omega)$ where (i) $\mathscr{G}$ is a DAG of events $\{e_0\} \cup \{e_1, \ldots, e_n\}$ where e_i, for every $1 \leq i \leq n$, is associated to a source S_i in order to manage its overall amount of uncertainty and its contribution; (ii) $\mathbb{M}$ is an abstraction of the set of possible integration trees over $\{S_1, \ldots, S_n\}$ and; (iii) Ω a mapping of every possible integration tree in $\mathbb{M}$ to data sources $S_1, \ldots, S_n$ according to the valuation of their associated events $e_1, \ldots, e_n$. This probabilistic tree data integration model reconciles these sources as a PrXML$^{\text{fie}}$ p-document $\widehat{\mathscr{P}}$.

Example 5.3.1. *Revisiting Example 1.2.1 in Chapter 1, Figure 5.4(a) shows trees representing information provided by the three sources S_1, S_2, and S_3. The dependency relationships are given by Figure 5.4(b) showing that S_1 and S_3 are independent sources while S_2 depends on S_1. These dependencies enable to conclude without ambiguity that S_1 and S_2 are conflicting on the values of the draft and the port of the ship while S_3 independently corroborate S_2 for the former. Figure 5.4(c) is an example of two possible integration trees over $\{S_1, S_2, S_3\}$: the first tree is obtained by considering that S_2 is not reliable whereas S_1 and S_3 do; the second tree assumes that all the sources, as well as their knowledge, are reliable. The p-document that reconciles all the three source trees while caring about uncertainties is given by Figure 5.4(d). This tree is constructed based on the procedure sketched earlier.*

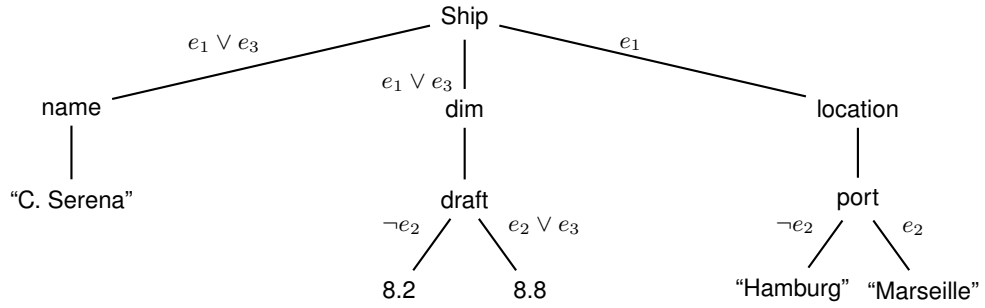


(a) Trees representing Data from S_1, S_2, and S_3



(b) Dependencies

(c) Two Possible Integration Trees



(d) PrXML$^{\text{fie}}$ p-document reconciliation of trees S_1, S_2, and S_3

Figure 5.4: Probabilistic tree data integration model

5.4 Uncertain Web Information on Moving Objects

In this section, we go further about studying the needs, in terms of quality data, driven by application scenarios handling moving objects with the Web perspective. We show how more structured Web content can be gathered and exploited in order to infer useful knowledge (trajectories, metadata) about moving objects. We present in Section 5.4.1 our Web extraction approach. Section 5.4.2 describes a method for evaluating the precision of obtained locations, trustworthiness of users and for integrating uncertain

attribute values.

5.4.1 Data Extraction

We distinguish two types of Web information about a moving object: *location data* and *general information*. We extract object information from Web sources through keyword search. That is, we suppose given the name of the moving object (a key phrase) and crawl data we obtain from a set of Web sources (see Section 5.5 for the specific sources used for our maritime application). We focus on location-based platforms and social networks, providing geolocated data items, for object locations.

Gathering General Information We collect general information about moving objects based on a *supervised* extraction over a fixed set of Web sources. The main intuition is that for many moving objects, e.g., ships, general information provided by a number of Web sources is structured into Web templates. This is particularly true for domain-specific resources. Inside this template, each particular characteristic has a meaningful label, with a value associated to it. We implement, based on such observation, an extraction process over these Web sources by using source-specific functionality for keyword search, and then crawling and parsing obtained HTML pages. Through hand-written schema mapping rules, we return data items in a global schema as a collection of attributes and corresponding values. There may be some conflicting attribute values from different sources as shown in Section 5.2 for the maritime domain.

Location Extraction We extract locations of moving objects by searching Web data items such as pictures, posts, and tweets that have geographical information attached to them (either directly as semantic geolocation information, or as can be extracted by a gazetteer [12] on tags and free text). This type of Web data can be found on the majority of popular location-based networks and social Web platforms like Flickr, Instagram, Facebook, etc. A geolocated Web data item comes with geographical data (latitude and longitude), a date, and additional meta-information such as title, description, set of tags, user name, etc. As an example, picture geolocation is sometimes available as *Exif* tags, automatically recorded by a digital camera at the time the picture was captured. The extraction proceeds as follows. Given a key phrase and a set of social Web sources, we first look for relevant geolocated data items regarding the input keyword. Then, for each data item we extract geographical data, dates, and meta-information as sketched above.

12. A gazetteer is a geographical dictionary or directory used in conjunction with a map or atlas.

5.4.2 Uncertainty Estimation

We evaluate in this section the amount of uncertainty – beyond just a simple modeling – in moving object data extracted from the Web. We first estimate the precision of spatio-temporal data according to three criteria and compute from that a trust score for each item provider. Then we sketch our integration of general information from different sources.

5.4.2.1 Precision of Location Data

As already mentioned, we harness geolocated Web data items for computing the different locations, thereby hypothetical trajectories, of moving objects. Geographical information associated to their geolocated data items come with imprecisions, however. First, for various reasons a keyword extraction approach is imprecise. As a result, the search may return wrong or irrelevant results regarding the moving object of interest. For instance, when searching geolocated Web data items about a moving object O, one can get from a given Web source results related to another type of real-world objet, e.g., a street with a similar name. Second, even if the results obtained really describe the object O, either the timestamp or the spatial information may be wrong (because of poorly configured software, purposedly introduced errors, ambiguous location names for gazetteers, etc.). We need an automated manner to detect these potential errors, and estimate the uncertainty of the data.

As a general framework, we estimate the precision of locations related to any O against two criteria. First, we detect outliers, that is, isolated locations, which represent locations with high probabilities to be impossible compared to other ones, that form a more consistent set. Second, we evaluate the amount of imprecision in geographical data by analyzing whether two successive locations in a chronological sense form a realizable trajectory of O with respect to its maximum speed. For the purpose of our demonstration application (see Section 5.5), dealing with ships, we also consider a third criterion which determines whether a location is in (or near) a water area. We next explain how we measure precision in each case.

Let $I_1, \ldots, I_j$ be a chronological sequence of distinct geolocated Web data items about the specific moving object O. We use the simple point-location model of [Wolfson, 2002] and represent formally a specific location I_j of the moving object O as a couple $(gd(I_j), dat(I_j))$ where $gd(I_j)$ is geographical coordinates (latitude and longitude) and $dat(I_j)$ is a date. Fix two locations $(gd(I_i), dat(I_i))$ and $(gd(I_j), dat(I_j))$. Necessarily, $i \leq j$ if and only if $dat(I_i) \leq dat(I_j)$. Given the roundness of the Earth, the distance d_{ij}

between these two locations of O is computed via the Haversine formula [Sinnott, 1984].

Detecting Possible Outliers An outlier is a location far away from a set of other locations, that are all within a given time interval and a maximum distance. Fix a given time window (say, one week for the maritime traffic application). We say that a point is an outlier if it falls in the middle of an interval where it is at distance Kd of the centroid of all other points of the interval, where d is the maximum distance between these points and $K > 1$ is an application-specific constant, e.g., $K = 5$.

Far-fetched Trajectories w.r.t. Maximum Speed A possible itinerary of the moving object O is a connected set of chronologically ordered locations. A trajectory may be far-fetched, i.e., unreasonable, if reaching one location from a previous one is impossible when we consider the reference speed of O. Let V be the reference maximum speed of O induced from gathered general information. Given two consecutive locations $(gd(I_i), dat(I_i))$ and $(gd(I_j), dat(I_j))$ with $j = i + 1$, we verify whether the following inequality holds:

$$d_{ij} \leq V \times (dat(I_j) - dat(I_i)).$$

If not, we are in the presence of an impossible trajectory. At least one of these two locations is wrong. As we do not know in advance which one, both are marked as potentially uncertain.

On-land Locations This measure pertains to our maritime traffic application (see Section 5.5) in which we are mostly interested in locations falling in water areas. Other applications have similar application-specific ways of detecting impossible points.

A location on land is defined as a point that is out of water areas. All the water regions on Earth can be found on the Web platform Natural Earth [13] in the form of *multi-polygons* for lakes, seas and oceans, and *polylines* for rivers. Based on these shapes and the *ray-casting algorithm*, we check whether a given location $(gd(I_j),\ dat(I_j))$ of the moving object O (here typically a ship) falls within one of the considered polygons or polylines. We estimate all on-land locations for O in this manner. Observe that some of these locations on land could be relevant for our application. In particular, locations on land, e.g., ports, that are close to water areas. To account for those kinds of interesting locations on land, we introduce a *tolerance factor* by considering the disc with radius of x and centered on a location. In the demonstration application, in Section 5.5, we set

13. http://www.naturalearthdata.com/

x to 0.1 degree of latitude/longitude. Given that, we find on-land locations whose disc, w.r.t. the tolerance factor, intersects one of the polygons or polylines of a water area. The overall set of on-land locations are finally those that do not satisfy this condition.

5.4.2.2 Computing User Trust Score

Uncertainty about moving objects in social Web can be inherent to the users instead of used imperfect sensors or an error-prone extraction process. In fact, some users can be *out-of scope* while others may intentionally spread false information in order to pollute social platforms. Consequently, one would want to have an indication about the trustworthiness of users sharing items: a trustworthy provider is more likely to share relevant items than an untrusted one. Given a snapshot of Web items about a particular moving object, we estimate hereafter the trustworthiness of their associated users based on the precision level of those items.

Consider again the set $I_1, \ldots, I_j$ of distinct geo-located items about O. We also suppose known the users which shared these items. Each user can provide more than one item about the same moving object. Suppose a particular user U posting a subset of items $O(U)$ of O. We put items in $O(U)$ into two groups w.r.t. the precision level of their locations, as previously determined: *less precise items* (outliers, participating in a far-fetched trajectory, on-land location in our application) and *more precise items* (items whose locations do not exhibit anomalies w.r.t. precision measures).

Even tough the more precise items seem to be good candidate to reliably describe the mobility of the considered object, we cannot be certain of their correctness. Instead, we assign more precise items with a probability value of α. In the same spirit, less precise items may still be correct. Their level of imprecision varies, however, in function of their specific types and can be application-driven. For instance, it seems reasonable to give less chance of being correct to on-land items than outliers on water areas for our maritime application. We define our belief about the level of relevancy of on-land items, outlier items, and those implied in far-fetched trajectories as β, γ, and θ respectively. We constrain β to be lower than γ and θ. In addition, the probability of relevance of any less precise item cannot be greater than that of any more precise item.

Let us fix the set of more precise items $MP(I)$ and the set of less precise items $LP(I)$ in $O(U)$ with $MP(I) \cap LP(I) = \emptyset$. Consider the set of on-land items $Land(I)$, outliers $Out(I)$, and far-fetched items $Far(I)$ in $LP(I)$ with pair-wise intersections not necessary empty. We define the level of trustworthiness $Trust(U)$ of user U as the average of the

probabilities of relevance of its provided items. That is, we set:

$$Trust(U) = \frac{\alpha \times |MP(I)| + \beta \times |Land(I)| + \gamma \times |Out(I)| + \theta \times |Far(I)|}{|MP(I)| + |Land(I)| + |Out(I)| + |Far(I)|}. \quad (5.2)$$

The trust score of each user can be recomputed when external feedbacks or knowledge are available. This can help lower or increase the probability of relevance of some items.

5.4.2.3 Integrating Uncertain Attribute Values

We do not only extract from the Web geographical information. We also collect general information about the moving object O in the form of attributes and corresponding values. Attributes are distinguished by meaningful labels specific to the individual sources. In general, Web sources have different level of completeness in terms of the data they provide. In addition, some of them can provide conflicting information, i.e., there can be multiple possible values for a given attribute, coming from different Web sources.

As general information comes from multiple sources, we need to integrate them in order to provide to the user a unique global view. In this integration process, we have to deal with the uncertainty that is inherent to the Web, but also that results from contradictions. The probabilistic tree data presented in Section 5.3 can be used in order to integrate these uncertain sources. However since general information is well-structured and for the purposes of our demonstration system we simplify the task – even with a possible loss of precision – by adopting the following process. We integrate general information about O from multiple Web sources by first matching values of the same attribute provided by distinct sources, using a manually constructed schema mapping across sources, and then by merging identical values. When a conflict occurs, we consider the value provided by the majority of sources as the most reliable one, but we keep all different values, as will be clear in the demonstration. This process for choosing the most probable values among conflicting ones corresponds to a voting approach. We study in details the problem of finding the correct attribute values of objects based on more elaborate voting strategies, such as those given in [Dong et al., 2009a], in the next chapter.

5.5 Maritime Traffic Application

Putting all results presented so far together yields our final contribution of this chapter: a demonstration of a system monitoring ships in the maritime domain as detailed in this section.

5.5.1 Use Case

The use case of our demonstration is the monitoring of ships. We rely on Flickr for collecting a large amount of geographical information about the different locations of a given ship. Flickr provides an easy-to-use API [14], with a set of predefined functions, e.g., `flickr.photos.search`, for extracting all pictures (each with a unique identifier), together with necessary meta-data including geographical coordinates and dates, whose title, description, or tags contains a certain keyword given in input; Figure 5.5 is an example of the encoding of a geolocated picture from Flickr. The extraction process on top of the Flickr platform is automated using API Blender [Gouriten and Senellart, 2012], an open-source library facilitating interactions with the Flickr API. As for the general information on ships, in particular details about their specifications, we integrated information from GrossTonnage, Marinetraffic, ShippingExplorer, ShipSpotting [15], and Wikipedia. These sources contain general information about various types of ships. The purpose of the first three is to gather data about objects in the maritime domain, especially vessels. However, excepting Marinetraffic that provides partial information under an API, these Web sources do not provide a way to extract specific information from their platforms, and need to be crawled.

```
%[["8442802776", {"username": "Michael Bentley", "userID": "354
   56872@N00"}, "2013-01-28 19:53:50", [18, {"source": "http://
   farm9.staticflickr.com/8231/8442802776_6bebdf9ff1.jpg"}], "
   tagged", [25.865209, -80.031677] ]...]
%
```

Figure 5.5: Geolocated picture from Flickr

5.5.2 System Implementation

Our system is a Web application with a map displaying ship locations. We implemented the full system using HTML, CSS, and JavaScript on the client-side, and Python on the server-side. The projection of raw geographical data onto a map uses the popular Google Maps JavaScript API [16]. Finally, features such as filtering options are performed using the jQuery JavaScript library [17].

14. https://www.flickr.com/services/api/
15. http://shipspotting.com/
16. https://developers.google.com/maps/documentation/javascript/
17. http://jquery.com/

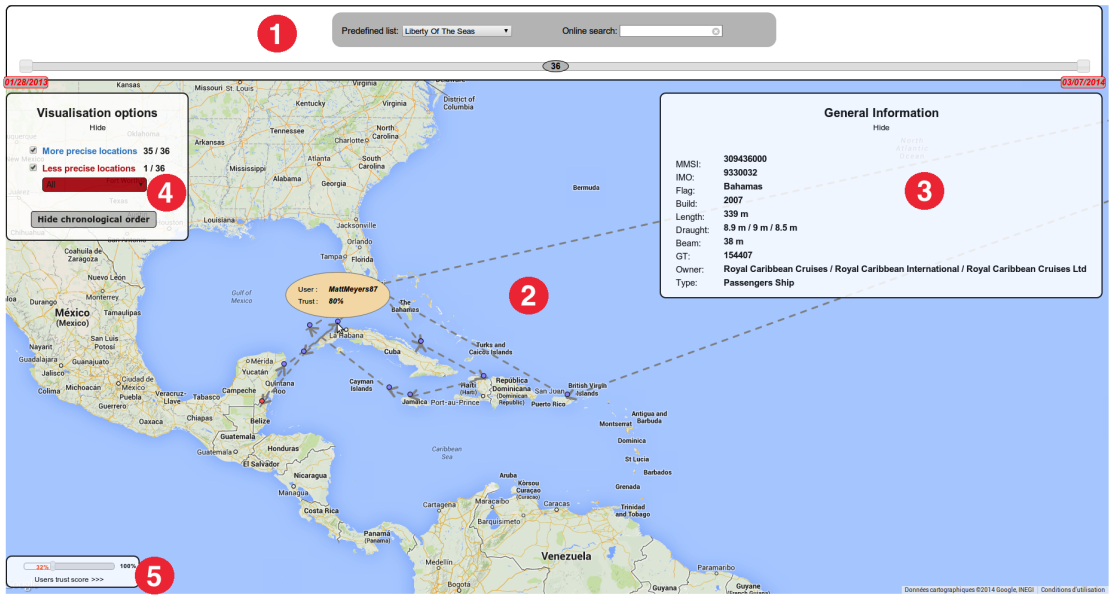


Figure 5.6: Main interface of the maritime traffic application

5.5.3 Demonstration scenario

A video accompanying this demonstration scenario is available at `http://dbweb.enst.fr/ships.mpg`.

Interface To interact with the system, a given user can either choose a ship name in a predefined list with locally saved data, or trigger an on-line search over the considered Web sources by providing a keyword (see Region 1 in Figure 5.6). For live Web search, the user can restrict the proposed set of sources for the extraction of the general information about the requested ship. The system will integrate obtained information when multiple sources are involved. Once information is obtained in local or from the Web, the different locations are displayed on the map and the general information is shown (Regions 2 and 3).

Ship positions are divided by default into two categories with different colors. Blue points on the map correspond to more precise locations, red points to less precise. As for the general information, we only show the most probable value for each attribute. The user has, however, the possibility to see details about possible other values by hovering the mouse over each attribute label. The user can restrict the visualization to ship positions in a given time interval with the slider at the top of the interface. Over

this slider, we have the total number of mapping locations. More advanced visualization options are available, as shown in Regions 4 and 5 in Figure 5.6: The user can filter locations with high or low precision. For low precision locations, she can focus either on those on the land, outliers, or locations leading to impossible trajectories. Finally, the user can visualize hypothetical itineraries, filter users according to given trust scores, or restrict to specific users.

User Experience An expert in the maritime domain would like to acquire new ships with specific characteristics and history for business purposes. A company sells vessels which may correspond to her needs. Two particular passenger ships “Liberty of the seas”and “Costa Serena”are of interest. Thus, she decides to verify from various Web sources whether the details given by the seller are correct before making a definitive choice. To do so, she uses our maritime traffic application which already holds information about “Liberty of the seas”and “Costa Serena”in local. The user selects the first one and obtains the map view of locations. She primarily overviews the general information about the ship, and observes conflicting values for its draught and its owner. The user thus checks the values of these two attributes, as given by her most trusted Web sources. These values seem to be consistent with the seller’s data after verification. The user remembers that she is very interested in positions of “Liberty of the seas”at some periods of the year (January to March and August to November). She filters positions corresponding to these date intervals with the slider. Surprisingly, the user remarks that the ship was near the Caribbean Sea and the Mediterranean Sea in these times. This information contradicts the seller who had stated that the ship has never left Europe. To obtain more insight about the journeys, the user triggers the view of hypothetical trajectories. She examines the choice list of less precise locations to understand why some points are incorrect. She concludes that all among them are on-land and indeed invalid. Finally, she removes these kinds of locations from the map, which confirms that ship routes mostly cover two main regions.

The user pursues explorations by considering “Costa Serena”now. She only focuses on its positions and past destinations. She notes that less precise locations make the visualization cumbersome with no clear overview on routes. Therefore, the user filters one by one each type of less precise locations for explanations. For instance, she picks on-land locations and notices that all of them are located on *Corsica*, an island which contains a region named “Costa Serena”– she learns this information by clicking on an on-land location and reading the corresponding Flickr page. Observing that providers of less precise locations have trust scores below 100%, the user sets the minimum trust to

80%. Finally, she refines remaining locations w.r.t. given intervals of dates, comparing with data from the selling company, and can therefore make an informed purchase decision.

5.6 Conclusions

We have considered in this chapter the problem of uncertain Web data integration with additional constraints implied by source dependencies, on the one hand, and spatio-temporal restrictions on the other hand. We have motivated the problem by giving concrete application scenarios, e.g., monitoring objects in the maritime domain, in which the dependency relationships between uncertain sources or uncertain location data cannot be ignored for a rigorous and high quality integration process.

We first approached the general issue of uncertain data integration under dependent sources. We have given its main prerequisites and then we have provided initial steps towards a probabilistic tree data integration model that represents and evaluates the amount of uncertainties in data and sources according to the dependencies in the particular case of tree-structured data. We have formalized the integration of uncertain tree-structured data sources under dependencies as the merge of several possible versions of a shared tree. Based on our uncertain XML version control model, presented in Chapter 2, we have shown how the mapping between the set of possible integration results and data sources can be abstracted. Finally, we have presented an algorithm that reconciles, consistently with the dependency relationships, this set of possible integration results and their probabilities using a $\text{PrXML}^{\text{fie}}$ p-document.

In a second phase, we investigated how structured Web content could be used in order to extract and exhibit useful knowledge about moving objects. For such a purpose, we have presented a Web extraction method that collects well-structured general and location data about a given moving object from Web platforms like social networking sites and general Websites. To evaluate the amount of uncertainties in the extracted information – in particular location data – we introduced precision measures based on some intuitive criteria for these location data, integrated attribute values from different sources for general information, and estimated the reliability level of users sharing geolocated items on the Web; for attribute values, we followed a straighforward truth discovery process, i.e., *majority voting*, by choosing the value having the highest number of providers as the truth whenever conflicts occur.

Finally, we have implemented and proposed an application for monitoring ships in the maritime domain by leveraging the results presented above. Our proposed system

has the ability to provide past itineraries, as well as general information, about a given ship with their level of precision.

The majority voting approach belongs to a particular class of data integration algorithms that deterministically tell the truth given the fusion of multiple Web sources in the same domain. However one of the main drawbacks of majority voting is that it assumes equally reliable sources and no data correlations, two facts that do not hold in practice on the Web. We continue in the next chapter with this problem of truth finding over structured Web sources – considering algorithms more sophisticated than simple majority voting – in a setting where data attributes are structurally correlated according to different levels of quality of data sources.

6 Truth Finding over Structured Web Sources

We present in this chapter our ongoing study on telling the truth when integrating multiple heterogeneous and structured sources, focusing on effectiveness in the precision of the truth finding output by leveraging on positive and negative structural correlations between data attributes with respect to the quality of fused sources.

With more structure and domain-specific knowledge, intuitions from multiple source data and their quality level are exploited in order to devise the truthfulness of information. In some practical application cases, however, the quality level – also known as the accuracy level – with which a particular source provides true information is not necessarily uniform on the entire set of data attributes, but variable with respect to distinct subsets leading to unknown local accuracies rather than a global one. One possible research avenue to account for this aspect in the truth discovery process – that we will refer to as the *AccuPartition* problem – is to investigate a given partition of the attribute set that maximizes the precision of its result.

After discussing the related work in Section 6.1, the initial part of this chapter, in Section 6.2, revisits main concepts related to the general problem of truth finding and introduces the *AccuPartition* challenge. We start by presenting our general setting – common to many truth finding algorithms – and by showing how a typical advanced technique tells, based on an iterative estimation of source accuracy and value confidence scores, the truth through the description of the core source accuracy aware model in [Dong et al., 2009a] that we use as a baseline throughout this study. We then formulate and introduce the *AccuPartition* problem.

Using a weight function that evaluates the optimality of a given partition based on the accuracy of the sources over its blocks, we solve the *AccuPartition* problem by providing exploration algorithms in Section 6.3. Our algorithms search for an optimal partitioning of the attributes with higher quality with respect to the truth finding output. As a reference, we start with an exact exploration method that considers the overall search space and returns the best partition based on the weight function. Such an exact exploration technique is intractable in practice because it presents a double exponential exploration

time complexity with respect to the number of attributes. To mitigate this problem, we investigate and devise an approximate exploration algorithm, based on some heuristics, which finds a near optimal partition while reducing the exploration space.

Finally, we present preliminary results about the effectiveness of our approaches via experiments over synthetic datasets in Section 6.4.

6.1 Related Work

There has been significant research on finding the truth about the instances of a set of objects shared by multiple conflicting sources under various distinct names such as truth discovery [Yin et al., 2008, Dong et al., 2009b, Zhao et al., 2012, Zhao et al., 2014], fact finding, data fusion [Bleiholder et al., 2007, Dong and Naumann, 2009, Pochampally et al., 2014, Li et al., 2014], etc.; see [Dong and Naumann, 2009, Li et al., 2012] for early surveys of the state-of-the-art and [Waguih and Berti-Equille, 2014] for a more recent deeper comparative evaluation. The simplest truth finding approach is majority voting which trusts the data instances provided by the majority – or at least a certain percentage – of equally reliable sources. This naive technique, however, become rapidly ineffective since sources, in particular on the Web, come with different reliability level in terms of data quality. As a consequence, the majority of truth finding techniques have largely accepted a weighted voting process with the source quality, as it is unknown in general, estimated based on the level of truthfulness of the object instances. Some domain-specific characteristics, especially those leading to correlations between sources or object instances, have driven the need to implement the weighted voting model differently by each existing algorithm for further precision.

[Yin et al., 2008] accounts for the similarity between data values from different sources in its weighted voting model by leveraging the intuition that the level of truthfulness of two similar values should influence each other in a certain sense. [Dong et al., 2009a, Dong et al., 2009b] explores and detects, based on a Bayesian analysis [Dong et al., 2010], positive correlations between sources caused by copying relationships. They devise a truth finding approach that does not significantly increase our belief about the level of correctness of information impacted by copying relationships as false data can be spread out by copy. Instead of that, it gives more credits to data from independent providers. [Pochampally et al., 2014] proposes the support of a more broader class of correlations between sources including positive correlations (e.g., similar extraction patterns, implementing similar algorithms) beyond source copying and negative correlations (e.g., the fact that two sources cover complementary information). The authors

model those correlations between sources using conditional probability theory in order to use them for the computation of the truthfulness scores of data instances through a Bayesian analysis within a multi-truth setting; in their technique, positive correlations are translated similarity as in [Dong et al., 2009a] while negative correlations between sources, in contrast, are handled in such a way that they do not significantly decrease our belief about the correctness level of information.

Other important aspects, besides correlations, related to the truth finding challenge have been also investigated. [Galland et al., 2010] presents probabilistic approaches which take into consideration the level of hardness of telling the truth about some particular objects. A truth discovery framework which models sources with multiple heterogeneous data types is introduced in [Li et al., 2014]. The long-tail phenomenon in truth finding, i.e., that most sources only cover a few objects and only a few sources provide lot of object instances, is approached by [Li et al., 2015]. The authors propose a truth finding algorithm that uses a confidence interval for source accuracy value in order to decrease the impact of sources having low data coverage. As earlier in [Dong et al., 2009b] where the temporal dimension of data source evolution is captured, [Zhao et al., 2014] study the scenarios where dynamic data sources, in particular stream data, are involved in the truth finding process.

6.2 Preliminaries and Problem Definition

This section first introduces some concepts pertaining to the general problem of truth finding and then formally states the particular problem studied in this chapter, namely the *AccuPartition problem*.

6.2.1 Preliminary Definitions

We revisit here the definitions of some basic concepts related to the truth finding problem.

As a common usage, we consider that information in presence is provided by sources in the forms of a set of objects (describing data in the same domain), each of them being characterized by a unique identifier and a set of shared attributes. Let $\mathscr{L}$ be a countable set of labels and $\mathscr{V}$ a countable set of values. Formally:

Definition 6.2.1. *An object, representing a real-word entity, is a pair* $(\iota_O, \mathscr{A}_O)$ *where* $\iota_O \in \mathscr{L}$ *is a unique identifier for this object, and* $\mathscr{A}_O \subset \mathscr{L}$ *is a set of attribute names. A set of objects* $\mathscr{O}$ *never has two distinct objects with the same identifier.*

When a set of objects $\mathscr{O}$ is given and the attribute set $\mathscr{A}_O$ is the same for all objects $O \in \mathscr{O}$ we simply denote, instead of $\mathscr{A}_0$, $\mathscr{A}$ to refer to the attributes of any object O in $\mathscr{O}$. Given a set of objects in a certain domain, an instance of every object is a partial mapping of its attributes to some domain of values (say, $\mathscr{V}$). Usually, object instances are provided by data sources for a given domain of interest. Given that, we introduce a source as follows.

Definition 6.2.2. *Given a set of objects $\mathscr{O}$, we formally define a source S over $\mathscr{O}$ as a pair $(\iota_S, \mathscr{I}_\mathscr{O})$ where $\iota_S \in \mathscr{L}$ identifies the source and $\mathscr{I}_\mathscr{O}$ is a set of instances of a subset of $\mathscr{O}$. That is, every element of $\mathscr{I}_\mathscr{O}$ is a pair (ι_O, ν) where ι_O is the identifier of some object in $\mathscr{O}$ and ν is a partial mapping from $\mathscr{A}_O$ to $\mathscr{L}$. We require no two instances to share the same ι_O.*

For simplicity, we will use in the remaining the word *objects* to also denote *object instances* when the context is clear. One very important observation on source data is that it does not necessary contain instances for every object in $\mathscr{O}$. In the same way, a source can only focus on a subset of attributes in $\mathscr{A}_O$ for any object in $O \in \mathscr{O}$ the source instantiates. Determining the proportion of the set of objects or attributes covered by a particular source comes down to estimating its data coverage, a notion that can be formally defined as follows.

Definition 6.2.3. *Consider a particular source S in $\mathscr{S}$ over the set of objects $\mathscr{O}$ sharing the set of attributes $\mathscr{A}$. The object coverage of S, denoted $Cov(S, \mathscr{O}')$, on any subset $\mathscr{O}'$ of $\mathscr{O}$ can be defined as the proportion of objects in $\mathscr{O}'$ for which S provides instances. Similarly, the attribute coverage of S, denoted by $Cov(S, \mathscr{A}')$, on any subset $\mathscr{A}'$ of $\mathscr{A}$ can be defined as the proportion of attributes in $\mathscr{A}'$ such that there is some object instance in S that map the attribute to a value.*

Observe that distinct sources can present different profiles in terms of data coverage given the same sets of objects and attributes as one can see with Figure 6.1.

Example 6.2.1. *Figure 6.1 shows an example of three sources (hungrygowhere, yelp, and openrice) providing information about a set of restaurants. The first and second column of each row in the table in Figure 6.1 correspond to the source and the identifier of the restaurant (its name here). The remaining columns represent the different attributes (Address, CuisineType, PlaceType, and Branch) of a restaurant object. To derive the truth about each restaurant, we perform a manual check from trustworthy sources; the true information for each attribute of every restaurant is rounded by a rectangular shape.*

		Address	CuisineType	PlaceType	Branch
hungrygowhere	KFC (Hougang Mall)	90 Hougang Avenue 10	Halal, Western	Fast Food	City Mall
hungrygowhere	Kee Tin Kopi Roti	111 Killinery Road	Asian, Local	Food Stall	Kopi Roti branch
hungrygowhere	Kopitiam Square	10 Sengkang Square	Asian, Local	Restaurant	Kopitiam
yelp	KFC (Hougang Mall)	90 Hougang Avenue 10	Universal Food	Fast Food	Hougang Mall
yelp	Kee Tin Kopi Roti	11 Killinery Avenue	Japanese Food	Baker	Tampines
yelp	Kopitiam Square	10 Sengkang	Asian, Local	Food Stall & Kiosques	City Square
openrice	KFC (Hougang Mall)	90 Hougang Avenue 10	–	–	Mustafi Center
openrice	Kee Tin Kopi Roti	Marsiling Road	–	–	Kopi Roti branch

Figure 6.1: Information about restaurants from three different sources

We now fix a finite set of sources $\mathscr{S}$ (such that no two sources have the same source identifier) and a finite set of objects $\mathscr{O}$. We assume all objects share the same set of attributes $\mathscr{A}$.

Note that we require object identifiers and attribute labels to act as real identifiers of the corresponding objects and attributes: two attributes with distinct labels are considered to refer to different real-world attributes, and two objects with distinct identifiers are considered to be distinct. This means neither entity resolution nor schema mappings is a concern in our study.

On the other hand, although heterogeneity at object and attribute levels cannot happen between any two sources, these can disagree about attribute values for any object as we point out throughout this thesis and in particular in this chapter. Reconsidering our example sources in Figure 6.1, one can easily observe that hungrygowhere states that the CuisineType for the restaurant "Kee Tin Kopi Roti" is "Asian, Local" while yelp claims "Japanese Food". Given any shared attribute a_i of a common object O in $\mathscr{O}$, we say that S and S' agree on $\nu(a_i)$ if their provided values coincide. If they give different values, that means the sources disagree about this attribute value.

Deriving the truth about a set of objects based on instances from multiple conflicting sources can rapidly become an arduous task. A typical truth finding process is an automated integration process which tells the truth knowing a set of sources in the same domain with some possible conflicting views about the actual values of some attributes for some objects. Existing algorithms, reviewed in Section 6.1, can be distinguished based on some specific assumptions, e.g., as to the domain of correct attribute values for objects and the trustworthiness of involved sources. We restrict ourselves in this work

to the most common framework where any attribute of every object has only *one correct value* and *many wrong values*, and integrated sources have different unknown level of trustworthiness. Specifically, we consider a truth finding process in which the probability of correctness of an attribute value is computed based on *source accuracy values*, and conversely the accuracy of a given source is estimated using the probabilities of its provided attribute values. Such a truth finding method aims at giving higher vote counts to attribute values coming from trustworthy sources. Without any loss of generality we consider as a baseline the `Accu` model (without dependency detection, value popularity computation, and value similarity consideration) proposed by Luna et al. in [Dong et al., 2009a] which shares common characteristics with most of the advanced truth finding algorithms. We briefly revisit `Accu` in the next and defer to [Dong et al., 2009a, Dong et al., 2010] for details.

6.2.2 `Accu` Truth Finding Algorithm

Assume a set of sources $\mathscr{S}$, each providing subsets of objects in the object set $\mathscr{O}$ where every object is described by the set of attributes $\mathscr{A}$. When integrating sources in $\mathscr{S}$, `Accu` determines the correct value for each attribute of every object by computing a *confidence score* for each distinct value provided for this attribute: the correct value is the one with the highest confidence value. These confidence values are computed based on the *accuracy scores* of sources which in turn are derived via source accuracy valuesfootnote`Accu` always starts with a priori accuracy value ϵ for all the sources.. We revisit below the definitions of these basic notions underlying `Accu`; see [Dong et al., 2009a, Dong et al., 2010] for details.

Definition 6.2.4. *The confidence score of an attribute value, representing its vote count, amounts to its likelihood to be the actual information for this attribute. Let $\mathscr{V}_{a_i}$ be the domain of possible values of an attribute a_i of a given object O from $\mathscr{O}$. For every possible value $\nu(a_i)$ in $\mathscr{V}_{a_i}$ provided by a subset of sources in $\mathscr{S}$, we denote the confidence score of $\nu(a_i)$ being the true value of the attribute a_i for the particular object O by $C(O, a_i, \nu(a_i))$.*

Correct attribute values are more likely to come from trustworthy sources than untrusted ones. In the same spirit, trustworthy sources have a trend to maintain attribute values with high confidence scores.

Definition 6.2.5. *The accuracy value of a given source gives a global indication about the correctness of attribute values provided by this source. We denote the accuracy of any source S in $\mathscr{S}$ over $\mathscr{A}$ by $\varepsilon(S, \mathscr{A})$.*

Observe that $\varepsilon(S, \mathscr{A})$ is initially sets to a fixed value in `Accu`, and will be updated in an iterative manner. Given the number n of false values for each given attribute for every object, the *accuracy score*[1] of a particular source S from $\mathscr{S}$ is computed from its accuracy with the following formula.

$$\alpha(S, \mathscr{A}) \coloneqq \ln\left(\frac{n \times \varepsilon(S, \mathscr{A})}{1 - \varepsilon(S, \mathscr{A})}\right). \tag{6.1}$$

The confidence score (or vote count) of any attribute value is estimated by summing up the confidence scores of the subset of sources which exactly provide this value. Given the subset $\mathscr{S}_{\nu(a_i)}$ of sources in $\mathscr{S}$ that agree on the value $\nu(a_i)$ of the attribute a_i of a certain object O in $\mathscr{O}$, we pose:

$$C(O, a_i, \nu(a_i)) \coloneqq \sum_{S \in \mathscr{S}_{\nu(a_i)}} \alpha(S, \mathscr{A}). \tag{6.2}$$

The accuracy value of a particular source S over $\mathscr{A}$ and given its subset of covered $\mathscr{O}_S$ is evaluated as being:

$$\varepsilon(S, \mathscr{A}) := \frac{1}{\mid \mathscr{O}_S \mid \times \mid \mathscr{A} \mid} \sum_{\substack{I \in \mathscr{I}_{\mathscr{O}_S} \\ \nu(a_i) \in I}} \frac{\mathrm{e}^{C(O, a_i, \nu(a_i))}}{\sum_{\nu(a_i) \in \mathscr{V}_{a_i}} \mathrm{e}^{C(O, a_i, \nu(a_i))}}. \tag{6.3}$$

Observe that this accuracy value is globally inferred by considering the entire collection of attributes. At the end of the process[2], `Accu` chooses, for every object O in $\mathscr{O}$, for each attribute a_i in $\mathscr{A}$, the correct value as having the highest confidence score.

$$\mathrm{TrueValue}(O, a_i) = \operatorname*{Argmax}_{\nu(a_i) \in \mathscr{V}_{a_i}} C(O, a_i, \nu(a_i)) \tag{6.4}$$

Let F refers to the `Accu` algorithm – and in general to any truth finding technique. We denote by $\mathrm{F}(\mathscr{S}, \mathscr{A})$ the result of F over sources in $\mathscr{S}$ giving the set $\mathscr{A}$ of attributes characterizing objects in $\mathscr{O}$. The precision of a truth finding algorithm is given by the percentage of correctly returned attribute values with respect to the actual reality. We refer

1. The accuracy score (or confidence score) of a source corresponds to the ratio between its accuracy value and its error rate.
2. The `Accu` process converges in a certain sense when the estimated accuracy values for sources do not change from one iteration to another one.

to this precision by $\varepsilon(\mathrm{F}(\mathscr{S}, \mathscr{A}))$ for an algorithm F, a set of sources $\mathscr{S}$, and an overall set of attributes $\mathscr{A}$. In practice, computing the exact precision of an algorithm is hard because we have no access to the ground truth, which leads to approximation analysis or manually constructed gold standard; [Dong et al., 2012] present an approximation of such a precision through a probabilistic analysis.

Example 6.2.2. *Reconsider the example sources in Figure 6.1 and apply* Accu *by starting with a priori accuracy value* $\epsilon = 0.8$ *and the number of false values for every attribute of all object as being* $n = 100$*. The algorithm converges after two iterations by returning* 0.97*,* 0.29*, and* 0.53 *as accuracy values of hungrygowhere, yelp, and openrice respectively. Concerning true instances of the restaurants,* Accu *concludes that the true instance of, e.g., "KFC (Hougang Mall)", is* {*"90 Hougang Avenue 10", "Halal, Western", "Fast Food", "City Mall"*}*. The algorithm derives this truth by computing the confidence score of each possible value for every attribute; the confidence scores of the two possible values "City Hall" and "Hougang Mall" about the Branch attribute of the restaurant "KFC (Hougang Mall)" are respectively* 7.6 *and* 3.6 *yielding the choice of the first value as the correct one.*

6.2.3 Problem Definition

Algorithms similar to Accu consider global source accuracy values when computing the confidence scores of attribute values. By doing so, however, the confidence scores of certain specific attribute values could be biased, which, in turn, should drastically impact the precision of the truth finding process. According to Example 6.2.2 hungrygowhere has an accuracy of 0.97 on the overall attribute set and thereby the Accu algorithm makes mistakes by selecting values from this source. Indeed, we know from Example 6.1 that hungrygowhere is only very accurate on {Address, CuisineType} while being inaccurate on {PlaceType, Branch}. More importantly, we have that whenever hungrygowhere provides a correct value for Address, it does for CuisineType. Meanwhile whenever hungrygowhere gives a wrong value for Address, it mostly does for CuisineType. As a consequence, these two groups of attributes are structurally correlated, in a positive sense, by the accuracy of this source while being negatively correlated between them; considering two independent local accuracy values for this source seems to be reasonnable.

We just show below that in some cases the correctness of possible values of a given attribute is only tied to that of some other attributes. This is particularly true when there

Table 6.1: Summary of notation used throughout this study

Notations	Meaning
$\mathscr{A}$	Collection of attributes
a_i	The i-th attribute in $\mathscr{A}$
$\nu(a_i)$	Value of the attribute a_i
$\mathscr{V}_{a_i}$	Domain of possible value of the attribute a_i
$C(O, a_i, \nu(a_i))$	Confidence score of a value of a_i in O
$\mathscr{O}$	Collection of objects
O	Specific object in the set $\mathscr{O}$
ι_O	Identifier of the object O
$\mathscr{A}_0$	Set of attributes of the object 0
$\mathscr{S}$	Collection of sources providing objects
S	A specific data source in $\mathscr{S}$
$\mathscr{I}$	A set of instances for some objects in $\mathscr{O}$
$Cov(S, \mathscr{O}')$	Object coverage of S on $\mathscr{O}'$
$Cov(S, \mathscr{A}')$	Attribute coverage of S on $\mathscr{A}'$
$\varepsilon(S, \mathscr{A}')$	Accuracy of a given source S on $\mathscr{A}'$
$\alpha(S, \mathscr{A}')$	Confidence score of a given source on $\mathscr{A}'$
F	A given truth finding algorithm
$\mathrm{F}(\mathscr{S}, \mathscr{A}')$	Result of applying F on $\mathscr{S}$ and $\mathscr{A}'$
$\varepsilon(\mathrm{F}(\mathscr{S}, \mathscr{A}'))$	Accuracy of the algorithm F on $\mathscr{S}$ and $\mathscr{A}'$

exist structural relationships such as *positive correlations* between some attributes in terms of source accuracy. These correlations can reveal different independent behaviors of a given source in terms of data quality on distinct subsets of attributes. For instance a source can be very accurate on a particular group of attributes while being fair on another one. We consider such a type of structural relationships between attributes. In this direction, we revisit the definition of the accuracy of a source by introducing this concept of *locality* in accuracy.

Definition 6.2.6. *(Local and global source accuracy) For any source S from $\mathscr{S}$, we say that the accuracy of S is* global *when it is estimated by considering the entire attribute set $\mathscr{A}$. In contrast, when the computation is only performed over a subset of attributes in $\mathscr{A}$ we say the obtained accuracy value is* local *to that subset.*

Analogously to the source accuracy, we also redefine the confidence score of a source, the result of a truth finding process, and its precision on a given subset of attributes. Given a subset $\mathscr{A}' \subseteq \mathscr{A}$, we will write accordingly to these definitions

$\varepsilon(S, \mathscr{A}'), \alpha(S, \mathscr{A}'), \mathrm{F}(\mathscr{S}, \mathscr{A}')$, and $\varepsilon(\mathrm{F}(\mathscr{S}, \mathscr{A}'))$. As we shall show shortly, rather than using a principle way, e.g., a probability analysis, to translate the fact that any two pairs of attributes are structurally correlated we express this relationship by discovering partitions with consistent blocks.

We claim that data source quality – with structural relationships – is better modeled by an estimation of the local accuracy values as we discussed in Example 6.2.3. Moreover, the computation of the probability of correctness of a given possible value of any attribute should be only affected by local source accuracy values corresponding to the correlated subset to which this attribute belongs. We believe that in such a context any truth finding algorithm using source accuracy should take benefit of the use of local accuracy values of sources instead of global accuracy values for improving the precision of its outcome. The idea is to perform the process on each subset, containing correlated attributes, and then to aggregate the results obtained from each subset of the input attribute set.

Example 6.2.3. *Let us consider again our running example in Figure 6.1. Clearly, we observe that hungrygowhere is very accurate in providing information about the address and the CuisineType of each restaurant while being worse for PlaceType and Branch. At the same time, yelp is accurate for PlaceType and Branch while being bad for Address and CuisineType. A truth finding process that can independently capture these two distinct behaviors should obtain a better precision; typically knowing that by splitting the attribute set into two subsets* $\{$*Address, CuisineType*$\}$ *and* $\{$*PlaceType, Branc*$\}$ *we can mainly choose correct information from hungrygowhere for the first class of attributes and yelp for the other class.*

Structural correlations between attributes ensure that local accuracy values of sources are computed on the right subsets (these nonempty subsets typically form a partition of the input attribute set). These structural relationships between attributes are not given beforehand in general, driving the need for an algorithm to discover these correlations for the truth finding process. In this study, we will consider the discovering of structural relationship among attributes as an integral part of the truth finding process. We refer to the process of performing truth finding for attribute values with exploration of subsets of correlated attributes as the *AccuPartition problem* which we introduce in the next.

Definition 6.2.7. *A partition of the set* $\mathscr{A}$*, denoted by* $\mathrm{P}^{\mathscr{A}}$*, is a set of nonempty subsets (or blocks) such that every element* a_i *in* $\mathscr{A}$ *is in exactly one of these subsets. In other words, we have:*

– $\bigcup_{\mathscr{A}' \in \mathrm{P}^{\mathscr{A}}} \mathscr{A}' = \mathscr{A}$ *and;*
– $\mathscr{A}_j \cap \mathscr{A}_k = \emptyset$ *for all* $\mathscr{A}_j \in \mathrm{P}^{\mathscr{A}}$ *and* $\mathscr{A}_k \in \mathrm{P}^{\mathscr{A}}$ *with* $j \neq k$.

We explore correlation relationships between attributes in $\mathscr{A}$ which yield a partition of $\mathscr{A}$. There may be many such partitions of $\mathscr{A}$. We are specially interested ins partitions whose subsets of correlated attributes mostly exhibit very high accuracy values for F over $\mathscr{S}$. Given a truth finding algorithm F, *AccuPartition* consists of finding a particular partition of $\mathscr{A}$, called *optimal partition*, for which F achieves highest precision on the majority of found subsets of correlated attributes. In other words, such an optimal partition should maximize the precision of F on the entire input attribute set.

Problem 6.2.1. *(AccuPartition problem) Assume a truth finding process* F *on sources in* $\mathscr{S}$ *providing subsets of objects in* $\mathscr{O}$*, each described by the set of attributes in* $\mathscr{A}$*. The AccuPartition problem aims at finding an optimal partition* $\mathrm{P}^{\mathscr{A}}$ *of* $\mathscr{A}$ *which maximizes the precision of* F *on the overall set* $\mathscr{A}$*.*

Intuitively, accuracy values of sources estimated by a given truth finding algorithm indicate in some extent the direction of its resulting true attribute values. Given that, we propose a truth finding process with partitions where an optimal partition is discovered by accumulating evidence from local source accuracy values on partition blocks.

Again, this study assumes independence among involved sources and identical false value distribution for all the attributes for every object. In addition, it assumes that the attributes are distinguishable in terms of source accuracy.

6.3 Partition-Aware Truth Finding Process

This sections details our proposed model for performing truth finding with an optimal partitioning of the input attribute set. We start by defining the weight of a given partition, a notion that represents the building block of our proposed model. Then as a reference we introduce a general (non-efficient) algorithm for the *AccuPartition problem*. Finally, we derive and propose an approximation algorithm which reduces the search space for discovering an optimal partition for any truth finding process.

Consider $\mathscr{S}$, $\mathscr{O}$, and $\mathscr{A}$ a set of independent sources, a set of objects, and a set of attribute respectively. Let F be a truth finding algorithm over $\mathscr{S}$.

6.3.1 Weight Function for Partitions

We propose a *weight function* ϖ over partitions of $\mathscr{A}$ in order to evaluate the optimality or not of a given partition when performing F over $\mathscr{S}$ given the entire attribute set. The weight function ϖ estimates a partition's weight by accumulating evidence from local source accuracy values on the different subsets of attributes (or blocks) in this partition. We use local source accuracy values computed by F in order to derive a score, in terms of quality, of each block. Let $\mathrm{P}^{\mathscr{A}}$ and $\mathscr{A}'$ be a partition of $\mathscr{A}$ and a block in $\mathrm{P}^{\mathscr{A}}$, respectively. Accordingly, we introduce the following definitions.

Definition 6.3.1. *(block score) We define the score of the block $\mathscr{A}'$ in the partition $\mathrm{P}^{\mathscr{A}}$, denoted by $\tau(\mathscr{A}')$, as a function of local source accuracy values returned by F when performed over sources in $\mathscr{S}$ by only considering attributes in $\mathscr{A}'$.*

Since local source accuracy values involved are estimated (in general we have no external knowledge about the real accuracy values, especially in real applications), selecting a prior correct score function for partition blocks is not trivial. To overcome this problem, in our evaluation in Section 6.4 we will compare results obtained by using different aggregate functions, as well as an *oracle*[3], as score functions. Given the scores of its blocks, we define the weight of a given partition as follows.

Definition 6.3.2. *(partition weight) We define the weight of any partition $\mathrm{P}^{\mathscr{A}}$ of $\mathscr{A}$, denoted by $\varpi(\mathrm{P}^{\mathscr{A}})$, as the average of the scores of its different blocks when F performs on sources in $\mathscr{S}$ given the entire input attribute set.*

$$\varpi(\mathrm{P}^{\mathscr{A}}) = \frac{1}{\mid \mathrm{P}^{\mathscr{A}} \mid} \times \sum_{\mathscr{A}' \in \mathrm{P}^{\mathscr{A}}} \tau(\mathscr{A}') \tag{6.5}$$

where $\mid \mathrm{P}^{\mathscr{A}} \mid$ represents the size of the partition $\mathrm{P}^{\mathscr{A}}$.

We can now introduce formally an optimal partition for F.

Definition 6.3.3. *(optimal partition) When performing F over sources $\mathscr{S}$ providing subsets of objects in $\mathscr{O}$ sharing the set of attributes in $\mathscr{A}$, an optimal partition $\varpi(\mathrm{P}^{\mathscr{A}})$ is defined as being a partition with the highest weight value.*

The *AccuPartition* problem (see Problem 6.2.1) is thus resolved by performing the truth finding against an optimal partition as given in Definition 6.3.3. As a consequence we devise the following proposition.

3. We refer to an oracle any comparison function that has the ability to compare with the real data, i.e., the truth.

Proposition 6.3.1. *An optimal partition of $\mathscr{A}$ is a solution of the AccuPartition problem for any given truth finding process F over $\mathscr{S}$. Such a solution always exists.*

Proof. Consider that we have access to a score function for partition blocks that estimates the score of a given block similarity to an oracle that has the ability to compare with the real data. As a result, an optimal partition of $\mathscr{A}$, whose weight is maximal, will clearly maximize the accuracy of F over the entire set of attributes. □

The next proposition is a direct definition of the AccuPartition problem.

Proposition 6.3.2. *When the entire input set of attributes is the only optimal set, AccuPartition comes down to performing the classical truth finding technique F with a global source accuracy measure.*

In the following, as a reference we present a general exponential algorithm for the *AccuPartition* problem which relies on an exploration complete of the partition space. Then, we devise an approximation algorithm for the *AccuPartition* problem.

6.3.2 Exact Exploration Algorithm

An exact exploration algorithm for the *AccuPartitiom* problem corresponds to an exploration of the entire set of all possible partitions of the input attribute set. The algorithm will compute the weight of each possible partition, and derive an optimal partition as one having the highest weight. This section details such an algorithm for *AccuPartition*, namely `GenAccuPartition`, as the reference for computing an exact optimal partition.

`GenAccuPartition` (see Algorithm 6.1) considers as input a set $\mathscr{S}$ of independent sources, a set of objects $\mathscr{O}$, and a set of attributes $\mathscr{A}$. The algorithm finds an optimal partition $\mathrm{P}^{\mathscr{A}}_{opt}$ of $\mathscr{A}$ for $\mathrm{F}(\mathscr{S}, \mathscr{A})$ by proceeding as follows.

Generation of the set of partitions `GenAccuPartition()` first generates the set of all possible partitions of the input attribute set $\mathscr{A}$, namely $\mathscr{W}(\mathrm{P}^{\mathscr{A}})$. Each generated partition is a potential optimal one for the truth finding process.

Computation of partition weight `GenAccuPartition()` computes the weight of every partition $\mathrm{P}^{\mathscr{A}}$ in the set of partitions $\mathscr{W}(\mathrm{P}^{\mathscr{A}})$. Given $\mathrm{P}^{\mathscr{A}}$, it first estimates the score of each of its block $\mathscr{A}_j$ based on local source accuracy values obtained by running $\mathrm{F}(\mathscr{S}, \mathscr{A}')$. Then it determines the weight of the partition $\mathrm{P}^{\mathscr{A}}$ as given by Expression 6.5.

Algorithm 6.1: `GenAccuPartition` <$\mathscr{S}$, $\mathscr{O}$, $\mathscr{A}$, F>

Input: $\mathscr{S}$: set of independent sources;
$\mathscr{O}$: set of objects;
$\mathscr{A}$: set of attributes;
F: truth finding algorithm
Output: An optimal partition $\mathrm{P}^{\mathscr{A}}_{opt}$ of $\mathscr{A}$ for F over $\mathscr{S}$

1 $\mathscr{W}(\mathrm{P}^{\mathscr{A}}) \leftarrow$ generate the set of possible partitions of $\mathscr{A}$;
2 $\mathrm{P}^{\mathscr{A}}_{opt} \leftarrow \emptyset$;
3 $\varpi(\mathrm{P}^{\mathscr{A}}_{opt}) \leftarrow 0$;
4 **foreach** ***partition*** $\mathrm{P}^{\mathscr{A}}$ *in* $\mathscr{W}(\mathrm{P}^{\mathscr{A}})$ **do**
5 $\quad \varpi(\mathrm{P}^{\mathscr{A}}) \leftarrow 0$;
6 $\quad$ **for** $1 \leq j \leq |\ \mathrm{P}^{\mathscr{A}}\ |$ **do**
7 $\quad\quad$ Get block $\mathscr{A}_j$ of the partition $\mathrm{P}^{\mathscr{A}}$;
8 $\quad\quad$ Execute truth finding process $\mathrm{F}(\mathscr{S}, \mathscr{A}')$;
9 $\quad\quad$ Compute $\tau(\mathscr{A}_j)$ of $\mathscr{A}_j$ using local source accuracy values;
10 $\quad$ **end**
11 $\quad \varpi(\mathrm{P}^{\mathscr{A}}) \leftarrow \frac{1}{|\mathrm{P}^{\mathscr{A}}|} \times \sum_{\mathscr{A}_j \in \mathrm{P}^{\mathscr{A}}} \tau(\mathscr{A}_j)$;
12 $\quad$ **if** $\varpi(\mathrm{P}^{\mathscr{A}}) \geq \varpi(\mathrm{P}^{\mathscr{A}}_{opt})$ **then**
13 $\quad\quad \mathrm{P}^{\mathscr{A}}_{opt} \leftarrow \varpi(\mathrm{P}^{\mathscr{A}})$;
14 $\quad\quad \varpi(\mathrm{P}^{\mathscr{A}}_{opt}) \leftarrow \varpi(\mathrm{P}^{\mathscr{A}})$;
15 $\quad$ **end**
16 **end**
17 **return** $(\mathrm{P}^{\mathscr{A}}_{opt})$;

Deriving an optimal partition `GenAccuPartition()` then discovers an optimal partition by comparing partition weights. Starting from an empty optimal partition with a weight equals to 0, the algorithm process each partition by first computing its weight and compares it to that of the current elected optimal partition. Two cases can occurs. If the weight of the newly processed partition is greater than the weight of the currently chosen optimal partition, the former is elected as the optimal partition and its weight is set now as the highest current one. Otherwise, the currently chosen optimal partition does not change. At the end of this process, the selected partition is returned by `GenAccuPartition()`s as the optimal one.

`GenAccuPartition()` ensures computing a correct optimal partition, but the algorithm does not scale. Indeed, the number of partitions grows *exponentially* when the size of the set increases. Fortunately, the number of attributes characterizing objects in many real applications is not very large. This observation leads in some extent to a

hope for an applicability of the general exploration algorithm in many real word scenarios where integrating sources with data quality is critical.

Proposition 6.3.3. `GenAccuPartition()` *has a search time complexity exponential in the size of the input attribute set.*

Proof. The proof follows by asymptotic approximations of Bell numbers [Graham et al., 1994]. □

In the following we investigate an approximation algorithm enabling to reduce the search spaces while returning a near optimal partition in practice.

6.3.3 Approximative Exploration

We propose `BottomUpAccuPartition()` which is an approximation procedure that minimizes the number of considered candidate partitions during the process of finding an optimal partition. We leverage the next two intuitions for defining `BottomUpAccuPartition()`: (i) first, there is a high probability that when same sources are high accurate on some individual attributes, they are on their union and; (ii) second, individual attributes sharing the same *top-one* list of more accurate sources are likely to be structurally and positively correlated.

Let us now describe how `BottomUpAccuPartition` (Algorithm 6.2) proceeds. The algorithm start with the trivial partition with blocks consisting each of a singleton attribute and further clusters the attributes according to their top-one lists of more accurate sources. Then separately `BottomUpAccuPartition()` explores the set of partitions of each derived cluster in order to determine the corresponding optimal sub-partition. Finally, the procedure merges the set of discovered optimal sub-partitions in order to return a global optimal partition. We detail below each step of the process.

Clustering of attributes `BottomUpAccuPartition()` starts by running the truth finding process over each block of the trivial partition $\mathrm{P}^{\mathscr{A}}_{sing}$ of the input attribute set $\mathscr{A}$. The goal is to determine the top-one list of most accurate sources for each individual attribute in $\mathscr{A}$. Given that, the algorithm clusters, in the same set, individual attributes sharing the same list of sources. The result of this phase is a set of disjoint clusters of attributes.

Local optimal partitions In this step, `BottomUpAccuPartition()` considers each cluster discovered in the previous phase. For a given cluster, it generates its set of

Algorithm 6.2: `BottomUpAccuPartition` <$\mathscr{S}$, $\mathscr{O}$, $\mathscr{A}$, F>

Input: $\mathscr{S}$: set of independent sources;
$\mathscr{O}$: set of objects;
$\mathscr{A}$: set of attributes;
F: truth finding algorithm
Output: An optimal partition $\mathrm{P}^{\mathscr{A}}_{opt}$ of $\mathscr{A}$ for F over $\mathscr{S}$

$\mathrm{P}^{\mathscr{A}}_{opt} \leftarrow \emptyset$;
$\mathrm{CL} \leftarrow \emptyset$;
$\mathrm{P}^{\mathscr{A}}_{sing} \leftarrow \{\{a_i\} \mid a_i \in \mathscr{A}\}$;
for *j-th* ***singleton subset*** $\mathscr{A}_i$ *in* $\mathrm{P}^{\mathscr{A}}_{sing}, 1 \leq j \leq \mid \mathrm{P}^{\mathscr{A}}_{sing} \mid$, **do**
  Execute truth finding process $\mathrm{F}(\mathscr{S}, \mathscr{A}_j)$;
  Set $\mathscr{S}_{\mathscr{A}_j} \leftarrow$ set of sources whose local accuracy values are maximal;
end
For all the singleton subsets $\mathscr{A}_j$'s such that the subsets $\mathscr{S}_{\mathscr{A}_j}$'s are same, add the set formed by $\bigcup a_i$, $a_i \in \mathscr{A}_j$, in CL;
foreach ***candidate list*** cl *in* CL **do**
  Set $\mathrm{P}^{cl}_{opt} \leftarrow$ `GenAccuPartition(`$\mathscr{S}, \mathscr{O}, \mathrm{cl}, \mathrm{F}$`)` ;
end
$\mathrm{P}^{\mathscr{A}}_{opt} \leftarrow \bigcup \mathscr{A}'$ such that $\mathscr{A}' \in \mathrm{P}^{cl}_{opt}$ for every P^{cl}_{opt};
return $(\mathrm{P}^{\mathscr{A}}_{opt})$;

all possible partitions and then determines the optimal one by comparing the partition weights after running the truth finding process on each partition. The computation of the local optimal partition for a given cluster is independent to that of any other cluster. At the end of this step, `BottomUpAccuPartition()` will return a set of optimal sub-partitions.

Generation of a global optimal partition `BottomUpAccuPartition()` ends with the generation of a global optimal partition which will represents the approximated optimal partition for the truth finding process over $\mathscr{A}$. The global optimal partition is generated by merging optimal sub-partitions computed at the previous phase. The merge simply consists of putting together the blocks of all the optimal sub-partition in the same set $\mathrm{P}^{\mathscr{A}}_{opt}$.

`BottomUpAccuPartition()` can be bounded in terms of search space. Its *lower bound* occurs when its first phase return several clusters with only one attribute for each of them without the same top-one list of most accurate sources. Its *upper bound* corresponds to the case where the first step returns only one cluster which will contain all the attributes. In this latter case, `BottomUpAccuPartition()` will have the same

search space than `GenAccuPartition()`. We provide experimental insights about the precision of the result of `BottomUpAccuPartition()`, as well its time complexity, in the next section.

6.4 Experimental Evaluation

We report in this section initial results obtained with tests conducted on synthetic data. We studied the precisions of our algorithms against `Accu` algorithm and `MajorityVoting` when sources exhibit distinct accuracies on different subsets of data attributes. We implemented all these algorithms [4] in Java by considering the `Accu` model as the basic truth finding procedure for `GenAccuPartition` and `BottomUpAccuPartition`. We started by comparing the precisions of `GenAccuPartition`, `Accu`, and `MajorityVoting` by considering an *oracle* and two aggregate functions (*maxAccu* and *avgAccu*) for estimating the score of blocks for partitions. The oracle computes the score of a block as the exact precision – knowing the real data – of a given truth finding output over this block. The aggregate functions *maxAccu* and *avgAccu* correspond respectively to the choice of the maximum local accuracy value and the average local accuracy value as the score of a block. We then focused on a comparison of the precisions, as well as running times, of `BottomUpAccuPartition` and `GenAccuPartition`.

Initially, we present our synthetic data generation and the different configurations set up for experimentations.

Synthetic Data Generation The purpose of our experimentation with synthetic datasets is to have the ability to compare the outputs of the tested algorithms against a ground truth in various configurations. Observe that one main challenging problem for evaluation within the settings of real datasets is to find the corresponding gold standard.

We implemented a tool in Java for generating synthetic data for our needs. The generation process requires five parameters: the expected number of attributes (na), the number of objects (no), the number of sources (ns), and two uniform distributions $\mathrm{U1}$ and $\mathrm{U2}$. The uniform distributions are used in order to randomly choose high source accuracy values and low source accuracy values for given subsets of attributes, respectively. Our generator proceeds with this input as follows. First, it generates the empty sets $\mathscr{A}$, $\mathscr{O}$, and $\mathscr{S}$ of respectively expected number of attributes, objects (each object having na attributes), and sources where each source instantiates all the objects (we assume here

4. The code source of the implementation of `Accu` not being provided by the authors, we did our best to properly implement the algorithm. In contrast, the implementation of `MajorityVoting` is trivial.

that every source presents a full coverage for both attributes and objects). After this stage, the generator produces the collection of all partitions of the entire attribute set and randomly selects one partition $\mathrm{P}^{\mathscr{A}}$. For every block in $\mathrm{P}^{\mathscr{A}}$, it randomly picks a source from $\mathscr{S}$ which is deemed to be highly accurate on this block (when it is possible, we search for a random partition so that to a source can be associated only one block). Fix the subset $\mathscr{S}'$ of chosen sources for blocks in $\mathrm{P}^{\mathscr{A}}$ and the subset $\mathscr{S}''$ of no selected sources. The generation of attribute values for every object for each source is now performed as follows. For every block $\mathscr{A}'$ in $\mathrm{P}^{\mathscr{A}}$ together with the associated source S in $\mathscr{S}'$, we set:

(i) $\varepsilon(S, \mathscr{A}')$ with a high local accuracy value using $\mathrm{U1}$; and

(ii) $\varepsilon(S, \mathscr{A}'')$ with a low accuracy value using $\mathrm{U2}$ for every $\mathscr{A}''$ in $\mathrm{P}^{\mathscr{A}}$ such that $\mathscr{A}'' \neq \mathscr{A}'$.

In addition, we consider no selected sources in $\mathscr{S}''$ and randomly select some of them, say $\frac{|\mathscr{S}''|}{2}$, which also provide correct information on attributes in $\mathscr{A}'$ for at least a certain percentage of objects depending on the accuracy value of S on $\mathscr{A}'$: we reduce this proportion whenever needed in order to obtain the desired configuration. We subtly choose these sources such that the same ones are not selected for several different blocks for avoiding the fact to obtain at the end of the process sources globally accurate on the entire set of attributes. Since any two distinct attributes can have different weights in the accuracy value of a source given its object instances, we compute a finer source accuracy value at the level of an attribute when knowing its local accuracy on a block and its covered number of objects. This computation is done so that the average of the accuracy values of a source on a subset of attributes is exactly equal to its local accuracy on this subset; this accuracy value at each attribute gives an indication about the proportion of objects on which the source really provides a correct value for each particular attribute. Finally, the generator – based on these accuracy values on attributes – defines the attributes of each object in which a source tells or not the truth. In particular, for every object, mostly correct and wrong values[5] are respectively set for attributes in $\mathscr{A}'$ and in each $\mathscr{A}''$ for the source S leading, accordingly, to the construction of the object instances for this source. The object instances of the other sources are set in the same manner. The domain of false values of each attribute for every object is set so that the probability that sources in $\mathscr{S}'$ and $\mathscr{S}''$ provide the identical wrong values is very low; this allows to prevent many corroborations between sources in $\mathscr{S}'$ and $\mathscr{S}''$ on subsets of attributes in which they badly say the truth. Observe that the object instances

5. Recall again that there is only one correct value and several possible false values for each attribute of each object. The choice of the false value to affect to an attribute is done randomly by the generator.

of the sources in $\mathscr{S}''$ are defined in such that these sources have neither significant local accuracy values nor significant global accuracy values. Last, our generator also produces the corresponding ground truth of the synthetic dataset.

Experimental Setup We generated four synthetic datasets by using different configurations. We considered, however, the same following number of sources, objects, and attributes for all these datasets: $\mathrm{na} = 6$, $\mathrm{no} = 100$, and $\mathrm{ns} = 10$. Each dataset is thus composed of 600 attribute values per source, i.e., a total of 6000 attribute values. Recall that the focus of our initial tests were made on the evaluation of the precision of the algorithms and we defer the study of their scalability for further work. Concerning the parametrization, i.e., ϵ and n, of the `Accu` algorithm we have experimented with $\epsilon = 0.8$ and $n = 20$. The configurations, in Table 6.2, consider different settings of $\mathrm{U1}$ and $\mathrm{U2}$ which define the average high and low accuracy values of sources. The first configuration, for instance, considers 1.0 and 0 as means for $\mathrm{U1}$ and $\mathrm{U2}$ respectively. In other terms, the corresponding dataset contains sources from $\mathscr{S}'$ having high local accuracy values mostly equal to 1.0 (i.e., they are fully accurate on some subsets of attributes) and low local accuracy values equal to 0.0 (i.e., they are fully inaccurate on some other subsets of attributes). While the two first configurations are extreme cases, the last two relax the mean value of $\mathrm{U2}$.

	Mean $\mathrm{U1}$	**Mean** $\mathrm{U2}$
configuration 1	1.0	0
configuration 2	0.8	0
configuration 3	0.8	0.2
configuration 4	0.6	0.4

Table 6.2: Configurations for synthetic datasets

Analysis of the Results Table 6.3 reports the precisions of `MajorityVoting`, `Accu` algorithm, and `GenAccuPartition` over the four synthetic datasets. For each configuration, the highest precision value is given in bold. The first results show that `GenAccuPartition` mostly presents a better precision compared to `MajorityVoting` and `Accu`. In the first two extreme configurations with `GenAccuPartition` the precision of the truth finding process is improved of at least 10% regarding the precisions of `Accu` and `MajorityVoting` and independently of the considered score function for the blocks of the partitions. In the third configuration, even if the improvement is less significant, it is of at least 6% when considering the *oracle* or the *maxAccu* for

the score function. In addition, one can also observe that Accu is on average better than MajorityVoting in the third initial dataset. As a consequence, first we can conclude that when sources are highly accurate on distinct subsets of data attributes Accu improves the precision over MajorityVoting; accounting for the accuracy of sources during the process is thus crucial. On the other hand, exploring an optimal partition, i.e., using GenAccuPartition, is even better. This last result strengthens our observation that in such a scenario performing truth finding should take advantage of local source accuracy values rather than a global accuracy value as in Accu since GenAccuPartition largely outperforms Accu. In the last configuration Accu and GenAccuPartition– both slightly worse than MajorityVoting– are comparable in terms of precision. One plausible cause could be the fact that the local accuracy values of sources are not very significant regarding the global accuracy values of sources in this configuration.

In the second stage we focus on the *maxAccu* score function whose results are more consistent with the ones returned by the *oracle* regarding Table 6.3. Table 6.4 presents the precisions of BottomUpAccuPartition over the four synthetic datasets. As a general observation, BottomUpAccuPartition does not decrease a lot the precision of the truth finding process with partitions compared to GenAccuPartition in the considered synthetic datasets. At the same time, we compare the average time needed by GenAccuPartition and BottomUpAccuPartition to complete their process over the entire set of synthetic datasets. We observe that BottomUpAccuPartition is four times faster than GenAccuPartition.

	MajorityVoting	Accu	GenAccuPartition (oracle)	GenAccuPartition (maxAccu)	GenAccuPartition (avgAccu)
configuration 1	0.795	0.85	**0.999**	**0.991**	**0.991**
configuration 2	0.705	0.701	**0.806**	**0.805**	**0.803**
configuration 3	0.743	0.763	**0.826**	**0.825**	**0.775**
configuration 4	**0.668**	0.605	0.615	0.603	0.615

Table 6.3: Precisions of MajorityVoting, Accu, and GenAccuPartition

	BottomUpAccuPartition (maxAccu)
configuration 1	0.991
configuration 2	0.800
configuration 3	0.780
configuration 4	0.591

Table 6.4: Precision of BottomUpAccuPartition on synthetic datasets

6.5 Conclusion

We have presented in this chapter our approach to discover the truth about instances of a set of objects shared by multiple conflicting sources, accounting for the possible independent local accuracy values of sources given by positive and negative correlations between data attributes. We have observed that, in some scenarios, sources can present different (unknown) profiles, in terms of data quality, over distinct subsets of data attributes – concretely a source may be very accurate on a subset while being worse on another one. This is a plausible characteristic of source qualities that the majority of truth finding methods, which estimate the accuracy level of sources in order to evaluate the truthfulness of data values, should model to avoid being biased by the use of a global indicator of the quality of sources.

To solve this problem one possible research avenue is to search for an optimal partitioning of the set of data attributes for the truth finding process. In this spirit, we have introduced the *AccuPartitiom* problem for which we have presented two exploration algorithms: an exact exploration algorithm and an approximation algorithm. These algorithms are built on the definition of a weight function that estimates the optimality of a given partition based on the accuracy values of sources on its blocks of attributes. Furthermore, we have shown that the exact exploration algorithm, while being unreasonable in practice, finds an optimal partition. In the contrast, the approximation algorithm may drastically reduce the search space but produces a near optimal partition.

Finally, we have considered first steps towards the experimental validation of the effectiveness of our approach against existing algorithms, drawing that it reaches – through both two exploration algorithms – better precisions with tests conducted on several synthetic datasets with different configurations and independently of the way the weight function is implemented. We also established that the approximation algorithm is four times faster than the exact exploration algorithm while ensuring comparable precision results.

7 Research Perspectives

In this chapter, we discuss the most promising research directions raised by the study carried out during this thesis for the following three main challenges: uncertain multi-version tree data, integration of Web sources under constraints, truth finding over structured Web sources with correlated data attributes.

7.1 Uncertain Multi-Version Tree Data

In addition to the investigation of other potential application scenarios for uncertain version control, a deeper evaluation of the qualitative aspects of our model, and its interplay with reputation or trust algorithms, we consider the following relevant problems for further developments.

Supporting More Complex Editions One of the main next steps is the support of more complex edit operations like moves and copy of intermediate tree nodes, or updates of node content. Our current model systematically handles any type of edit using a sequence of insertions and deletions, hiding some informative aspects of the collaborative editing. The probabilistic model used is, fortunately, enough flexible to easily incorporate any other type of edit operations. Complex edit operations have their own semantics (see [Lindholm et al., 2006, Thao and Munson, 2014]), which, taken into consideration, may enable to retain more semantical information about the collaborative work and thereby improve modeling and assessing of uncertainty during the process.

Querying Uncertain Multi-Version Tree Data Another interesting problem is a query language for multi-version tree data. Retrieving a given version of any uncertain multi-version tree data must account for uncertainties, thereby possibility and probability. One possible direction for dealing with this problem is to harness current research on querying probabilistic XML [Liu et al., 2013, Kimelfeld and Senellart, 2013] in order to enrich usual

version retrieval functions with additional predicates aware of the sources of uncertainties by enabling to reason on their levels of reliability.

7.2 Web Data Integration under Constraints

Besides a study of the effect of uncertain dependencies in the representation of the set of possible worlds and a more detailed comparison of our probabilistic tree data approach against the literature (especially, the model in [Van Keulen et al., 2005]), a further study could tackle the following problems.

Integration via Query Views It could be also of interest to investigate the definition of the probabilistic tree integration in terms of query views, that is, defining the integration result with respect to a specific class of queries over data. Such an approach is more suitable in the case of virtual data integration where data reside in distributed sources.

Accounting for New Knowledge Knowledge rules and user feedback can help to resolve some uncertainties in the integration by refining the set of possible worlds (cf. [van Keulen and de Keijzer, 2009]). Relying on domain experts, e.g., opinions from maritime experts regarding our ship monitoring application in the maritime domain, is a reliable way to obtain such types of knowledge.

7.3 Truth Finding with Correlated Data Attributes

Scoring Functions for Blocks in Partitions The scoring function, which aims at measuring the precision of a given truth finding process in every given subset of attributes in each partition, is fundamental when searching for an optimal partition in the presence of structurally correlated data attributes. Until now, we rely on aggregate functions such as maximum and average – with no guarantees of their correctness – over local accuracy values on subsets of attributes for estimating these scores. As a consequence, we are interested in devising a better scoring function for the blocks of each partition in order to improve the process of discovering an optimal partition within our setting. One possible direction could be an adaptation of the precision algorithm in [Dong et al., 2012] which approximatively measures the quality of the truth finding outcome given a set of data attributes and a subset of sources. This accuracy function uses a Bayesian analysis and achieves an estimation near to an oracle.

Approximation Search Algorithms for Optimal Partitions We are also exploring more efficient approximation search algorithms for discovering an optimal partition when performing truth finding over Web structured data with correlated data attributes. As mentioned earlier, an exact exploration become unreasonable when the number of attributes is significant. One possible solution, that we are actually studying, is a sampling technique which can drastically reduce the exploration space while ensuring a near-optimal partition for any existing partition-aware truth finding algorithm based on an estimation of the levels of accuracy of involved sources.

A Other Collaborations

Uncertainties and Ordered Domains [Amarilli et al., 2013, Amarilli et al., 2014] In some applications, uncertainty stems from *order-incomplete* information, driving the need for a framework that can represent uncertainty about the possible orderings. For example, when combining data from multiple sources correspnding to lists of properties (such as hotels and restaurants) ranked by an unknown function reflecting relevance or customer ratings, and when merging different contributions to documents edited concurrently with uncertainty on the order of contributions. First, we investigate data transformations – in a relational setting – which involve order-incomplete data by studying their complexity and the possibility to support meta-data tracking with a semiring-based provenance framework. In a second study, we extend the positive relational algebra to ordered and order-incomplete data by introducing a set of axioms to guide the design of a bag semantics for the language. We introduce two simple such semantics, one of which is shown to be the most general for our set of axioms. We next design a strong representation system for them, based on partial orders interpreted through a possible worlds semantics. We study the expressiveness of our query language, introduce a top-k operator, and investigate the complexity of query evaluation in terms of certain and possible answers. We last introduce a duplicate elimination operator to return to set semantics.

This work has been done in the context of the Ph.D. of Antoine Amarrili at Télécom ParisTech with the collaboration of Daniel Deutch, Assistant Professor at Tel Aviv University, and Pierre Senellart, Professor at Télécom ParisTech.

Conditioning Probabilistic Databases [Tang et al., 2014a, Tang et al., 2014b] A probabilistic database represents uncertain data as describing a set of possible worlds, each of which has a probability. Direct observations and general knowledge, in the form of constraints, help refining the probabilities of the possible worlds, possibly ruling out some of them by *conditioning* the probabilistic database. The conditioning consists to find a new probabilistic database that denotes the valid possible worlds with their new

probabilities. We consider the conditioning problem for probabilistic XML with a language of formulae of independent events to express the probabilistic dependencies among the nodes of the XML tree. For reference, we present an exponential algorithm. We then focus on the specific case of independent events and mutually exclusive constraints. This case goes beyond local mutually exclusive constraints as considered so far in the literature. We devise and present polynomial-time algorithms for conditioning probabilistic XML data in tractable cases. We also explore the conditioning problem for probabilistic relational databases with referential constraints.

*This work has been done in the context of the Ph.D. of Ruiming Tang at National University of Singapore with the collaboration of Dongxu Shao, Scientist at A*STAR, Stephane Bressan, Associate Professor at National University of Singapore, Pierre Senellart, Professor at Télécom ParisTech, and Wu Huayu, Research Scientist at A*STAR.*

B Résumé en Français

Dans cette thèse, nous étudions certains problèmes fondamentaux découlant d'un besoin accru de gestion des incertitudes dans plusieurs applications Web multi-sources avec de la structure, à savoir le contrôle de versions incertaines dans les plates-formes Web à large échelle, l'intégration de sources Web incertaines sous contraintes, et la découverte de la vérité à partir de plusieurs sources Web structurées. Nous nous basons sur la théorie existante sur la gestion de données incertaines, et proposons des méthodes effectives et efficaces qui modélisent et évaluent les incertitudes dans les plates-formes d'édition collaborative à large échelle ou dans un cadre d'intégration de sources Web incertaines. En particulier, l'une des contributions majeures de ce travail est une thèse : *la gestion des documents multi-versions partagés dans les plates-formes d'édition collaborative à large échelle devrait grandement bénéficier d'un système de contrôle de versions incertaines*. Nous résumons ci-après notre proposition d'un tel cadre pour les applications telles que Wikipedia où les données manipulées ont une structure arborescente qui favorise une prise en charge plus intelligente des incertitudes.

Dans de nombreuses plates-formes d'édition collaborative, au sein desquelles les contributions peuvent émaner de différents utilisateurs, la gestion du contenu est basée sur le contrôle de versions. Un système de contrôle de versions trace les versions aussi bien du contenu que des modifications. Un tel système facilite la résolution d'erreurs survenues durant le processus de révision, l'interrogation de versions antérieures, et la fusion de contenu provenant de contributeurs différents. Tels que résumés dans [Koc and Tansel, 2011, Altmanninger et al., 2009], des efforts considérables liés à la gestion de versions des données ont été entrepris à la fois dans la recherche et dans les applications. Les premières applications furent le processus collaboratif de rédaction de documents, la conception assistée par ordinateurs et les systèmes de développement logiciels. Présentement, des outils de contrôle de versions puissants tels que Subversion [Pilato, 2004] et Git [Chacon, 2009] gèrent efficacement de très grands dépôts de code source et des systèmes de fichiers partagés.

Toutefois, les approches actuelles ne supportent pas la gestion de l'incertitude

dans les données. C'est le cas de l'incertitude qui résulte de conflits. Les conflits sont fréquents dans les processus d'édition collaborative, en particulier lorsque le cadre est ouvert. Ils apparaissent dès lors que des éditions concurrentes tentent de modifier le même contenu. Les conflits conduisent, donc, à de l'ambiguïté dans la gestion des mises à jour sur les données. Les sources d'incertitudes dans un processus de contrôle de versions ne se limitent pas uniquement aux conflits. En effet, il existe des applications utilisant le contrôle de versions qui sont à la base incertaines. Parmi elles, on peut citer les plates-formes collaboratives Web à large échelle telles que Wikipedia [1] ou Google Drive [2] qui permettent des interactions sans restrictions entre un très grand nombre de contributeurs. Celles-ci se font sans une connaissance préalable du niveau d'expertise et de fiabilité des participants. Dans ces systèmes, le contrôle de versions est utilisé pour tracer l'évolution de contenus partagés et la provenance de chaque contribution. Au sein de tels environnements, l'incertitude est omniprésente à cause des sources non fiables, des contributions incomplètes et imprécises, des éditions malveillantes et des actes de vandalisme possible, etc. Par conséquent, une technique de contrôle de versions capable de manipuler efficacement l'incertitude dans les données pourrait être d'une grande aide pour ce type d'applications. Nous détaillons ci-après les cas d'utilisation possibles.

Les systèmes d'édition collaborative Web à large échelle comme Wikipedia ne définissent aucune restriction pour l'accès en écriture aux données. Il en résulte que les documents multi-versions contiennent des informations qui proviennent de différents utilisateurs. Comme esquissé dans [Voss, 2005], Wikipedia a connu un accroissement exponentiel du nombre de contributeurs et d'éditions par articles. Les caractères libre et ouvert conduisent à des contributions avec différents niveaux de fiabilité et de cohérence en fonction à la fois de l'expertise du contributeur (p. ex., novice ou expert) et de la portée du sujet traité. En même temps, les guerres d'édition, les contributions malveillantes telles que les spams et les actes de vandalisme peuvent survenir à tout instant au cours de l'évolution d'un document partagé. Par conséquent, l'intégrité et la qualité de chaque article peuvent être fortement altérées. Les solutions préconisées contre ces problèmes critiques sont la révision des politiques d'accès pour les articles portant sur des sujets très sensibles, ou des solutions évaluant la qualité des contributions sur la base de la réputation des auteurs, de statistiques sur la fréquence de mise à jour des contenus, ou la confiance qu'un lecteur donné a sur l'information [Maniu et al., 2011a, De La Calzada and Dekhtyar, 2010, Adler and de Alfaro, 2007]. Cependant,

1. http ://www.wikipedia.org/
2. https://drive.google.com/

restreindre les éditions sur les articles Wikipedia à un groupe de contributeurs privilégiés ne résout pas le besoin de représenter et d'évaluer les incertitudes. En effet, ces articles peuvent demeurer incomplets, imprécis ou incertains, faire référence à des vues partielles, des informations fausses ou des opinions subjectives. La réputation des contributeurs ou le niveau de confiance sur les sources constituent des informations utiles pour une estimation quantitative de la qualité des versions, voire de chaque contribution. Cependant, une représentation efficace et préalable des incertitudes dans les versions de documents reste un pré-requis.

Par ailleurs, le filtrage et la visualisation de données constituent également des défis très importants dans les environnements collaboratifs. Les utilisateurs de Wikipedia, par exemple, ne sont pas uniquement des contributeurs, mais également des internautes intéressés par la recherche et la lecture d'articles sous contrôle de versions. Les systèmes actuels forcent les utilisateurs à visualiser soit la dernière version d'un article donné, même si cette dernière pourrait ne pas être la plus pertinente, soit une version à une date bien précise. Les utilisateurs, particulièrement dans les plates-formes de gestion de connaissance universelle comme Wikipedia, voudront sans doute accéder aisément aux versions les plus pertinentes ou celles des auteurs de confiance. Le filtrage de contenu est un des avantages de notre approche. Il peut être effectué de façon commode en cachant les contributions de sources non fiables, par exemple lorsqu'un acte de vandalisme est détecté, ou au moment de l'interrogation en vue de prendre en compte les préférences des utilisateurs et leurs degrés de confiance sur les contributeurs. Alternativement, pour lutter contre la désinformation, il semble utile de présenter aux utilisateurs les versions accompagnées d'informations sur leurs niveaux d'incertitude et celui de chaque partie de leurs contenus. Enfin, les utilisateurs devraient être capables, au moment de la visualisation, de demander un document qui correspond à la fusion de parties prises de différentes versions (p. ex., certaines d'entre elles peuvent être incomplètes, imprécises, et même incertaines prises à part). Nous proposons dans [Abdessalem et al., 2011] une application mettant en évidence de nouvelles possibilités d'interaction et de sélection sur le contenu des pages Wikipedia. En particulier, le contenu d'une page n'est plus réduit à la derniére révision valide, mais correspond à la fusion de plusieurs révisions incertaines.

Vu que le contrôle de versions est incontournable dans les systèmes d'édition collaborative Web incertains à large échelle, la représentation et l'estimation des incertitudes au travers de la gestion de versions de données deviennent cruciales. Ceci dans l'optique d'améliorer la collaboration et de surmonter les problèmes comme la résolution des conflits et la gestion de la fiabilité de l'information. Dans cet thèse, nous proposons

un modèle de contrôle de versions XML incertaines pour les documents arborescents multi-versions dans les contextes d'édition collaborative ouverte. Les données que sont les documents bureautiques, les documents HTML ou XHTML, les formats wiki structurés, etc., manipulés au sein des scénarios d'application cités ont une structure arborescente ou peuvent être transcrits sous ce format ; XML est un encodage naturel pour les données arborescentes.

Les travaux liés à la gestion de versions de documents XML ont surtout porté sur la détection de changements [Rönnau and Borghoff, 2012, Lindholm et al., 2006, Wang et al., 2003, Khan et al., 2002, Cobéna et al., 2002]. Seul certains, par exemple [Thao and Munson, 2011, Rönnau and Borghoff, 2009, Rusu et al., 2005], ont proposé un modèle de données semi-structurées complet qui supporte le contrôle de versions. La gestion de l'incertitude a reçu une grande attention au sein de la communauté des bases de données probabilistes [Kimelfeld and Senellart, 2013, Suciu et al., 2011], spécialement pour XML à des fins d'intégration de données. En effet, un ensemble de modèles de données XML probabilistes [Van Keulen et al., 2005, Nierman and Jagadish, 2002, Abiteboul et al., 2009, Kharlamov et al., 2010] élaborés avec des sémantiques variées de distribution de probabilité sur les éléments de données, a été proposé. Ces modèles couvrent des techniques simplistes, par exemple [Van Keulen et al., 2005], ne modélisant que des contraintes de dépendances locales (p. ex., l'exclusion mutuelle) entre nœuds et d'autres plus complexes qui prennent en compte une plage plus étendue de contraintes ainsi que des corrélations globales possibles. Un cadre général XML probabiliste, qui généralise tous les modèles XML incertains proposés dans la littérature, est défini par [Abiteboul et al., 2009] et [Kharlamov et al., 2010]. Ce cadre introduit la notion de documents probabilistes (ou, plus court, p-documents).

Nous prenons en compte, dans notre cadre, l'incertitude dans les données au travers d'un modèle XML probabiliste comme élément de base de notre système de contrôle de versions. Chaque version d'un document partagé est représentée par un arbre XML. À un niveau abstrait, nous décrivons un document XML multi-version avec des données incertaines en utilisant des événements aléatoires, des scripts d'édition XML qui leur sont associés et un graphe acyclique orienté de ces événements. Pour une représentation concrète, le document tout entier, y compris ses différentes versions, est modélisé comme un document XML probabiliste correspondant à un arbre XML dont les branches sont annotées par des formules propositionnelles sur les événements aléatoires. Chaque formule propositionnelle décrit à la fois la sémantique des éditions incertaines (insertion et suppression) effectuées sur une partie donnée du document et sa provenance dans le processus de contrôle de versions. L'incertitude est évaluée à l'aide du modèle

probabiliste et de la valeur de fiabilité associée à chaque source, chaque contributeur, ou chaque événement d'édition. Ceci résulte en une mesure d'incertitude sur chaque version et chaque partie du document. Le graphe acyclique orienté d'événements aléatoires maintient l'historique de l'évolution du document en traçant ses différents états et leurs liens de dérivation. Comme dernière contribution majeure, nous montrons que les opérations standard de contrôle de versions, en particulier les opérations de mise à jour et de fusion, peuvent être implantées directement comme opérations sur le modèle XML probabiliste ; en comparaison aux systèmes déterministes de contrôle de versions tels que Git et Subversion est démontrée sur des données réelles.

Après quelques préliminaires dans la section B.1, nous revisitons le modèle XML probabiliste que nous utilisons dans la section B.2. Nous détaillons le modèle de contrôle de versions XML probabiliste proposé et les principales propriétés sous-jacentes dans la section B.3. Les sections B.4 et B.5 présentent la traduction des opérations de contrôle de versions usuelles, à savoir la mise à jour et la fusion respectivement, dans notre modèle. Dans la section B.6, nous prouvons l'efficacité de notre modèle en le comparant à des systèmes de contrôle de versions déterministes au travers d'expériences sur données réelles. Nous esquissons par la suite (voir section B.6.2) quelques-unes des capacités de filtrage de contenu inhérentes à l'approche probabiliste adoptée.

Les idées initiales qui ont conduit à ce travail ont été présentées comme un article de séminaire de doctorants dans [Ba et al., 2011] ; le présent chapitre est une extension (et traduction) de [Ba et al., 2013a, Ba et al., 2013b] auquel le lecteur peut se reporter pour les preuves de certains des résultats énoncés dans cet article.

B.1 Préliminaires

Nous présentons dans cette section les notions de base du contrôle de versions et la classe de documents XML semi-structurés qui sous-tendent notre proposition.

Un document multi-version fait référence à un ensemble de versions d'un même document dans un processus de contrôle de versions. Chaque version du document représente un état (instance) donné de l'évolution du document versionné. Un modèle de contrôle de versions classique est construit sur les notions fondamentales suivantes.

B.1.1 Concepts de base du contrôle de versions : versions et espace des versions

Par convention, une version désigne une copie d'un document sous contrôle de versions. Des opérations de dérivation lient les différentes versions d'un document. Une dérivation consiste à créer une nouvelle version en copiant d'abord une version existante avant de la modifier. Certaines versions, représentant des variantes, dérivent de la même origine. Les variantes (versions parallèles) caractérisent un historique d'édition non-linéaire avec plusieurs branches distinctes. Dans cet historique, une branche est une séquence linéaire de versions. Au lieu de stocker le contenu complet de chaque version, la plupart des techniques de contrôle de versions maintiennent uniquement les *diffs* entre les états, avec des méta-informations. Ces états (ou *commits* dans le monde Git [Chacon, 2009]) modélisent les différents scripts de mises à jour qui ont été explicitement validés à des instants distincts du processus de contrôle de versions. Un état vient aussi avec des informations sur le contexte (p. ex., auteur, date, commentaire) dans lequel ces modifications ont été faites. Il en résulte que chaque version est fonction de l'historique complet menant à un état donné. Nous adopterons ici la même approche pour modéliser les différentes versions d'un document.

Puisque le contenu de chaque version n'est pas complètement stocké, il doit y avoir un moyen de le retrouver en cas de besoin. L'espace des versions [3] représente l'histoire des éditions sur le document versionné (p. ex., l'historique des versions wiki comme décrit dans [Sabel, 2007]). Il contient toutes les information nécessaires liées aux versions et leurs dérivations. Comme indiqué ci-dessus, une dérivation implique au moins une version en entrée (plusieurs versions en cas de fusion) et une version en sortie. Nous décrivons sur cette base, et similairement à [Chacon, 2009], l'espace des versions de tout document multi-version comme un *graphe acyclique orienté*.

B.1.2 Modèle de documents XML non ordonnés : définition et mise à jour

Les applications qui motivent ce travail manipulent des données arborescentes. Nous considérons les données en présence comme des arbres XML non ordonnés, en conséquence. L'extension du modèle proposé aux arbres ordonnés est possible. Cela pourrait par exemple consister à restreindre l'ensemble des versions valides à celles se conformant à un ordre spécifique. Ce probléme est hors du champ de cet article et nous

3. Noter que la notion d'espace des versions utilisée ici a une sémantique differente de celle du même nom définie dans le domaine de l'apprentissage des concepts [Mitchell, 1979].

nous focalisons ici sur le cas des arbres non ordonnés. Considérons un ensemble fini $\mathscr{L}$ de chaînes de caractères (c'est-à-dire, étiquettes ou données textes) et un ensemble fini $\mathscr{I}$ d'identifiants tels que $\mathscr{L} \cap \mathscr{I} = \emptyset$. De plus, soient Φ et α respectivement une fonction d'étiquetage et une fonction d'identification. D'une manière formelle, nous définissons un document XML comme étant un arbre $\mathscr{T}$ non ordonné et étiqueté : les fonctions α et Φ associent chaque $x \in \mathscr{T}$ respectivement à un identifiant unique $\alpha(x) \in \mathscr{I}$ et à une chaîne de caractères $\Phi(x) \in \mathscr{L}$. L'arbre est non-borné, c'est-à-dire, le nombre de fils de chaque nœud dans $\mathscr{T}$ n'est pas fixé à l'avance. Par soucis de simplicité, nous supposerons que tous les arbres ont la même racine (même étiquette, même identifiant).

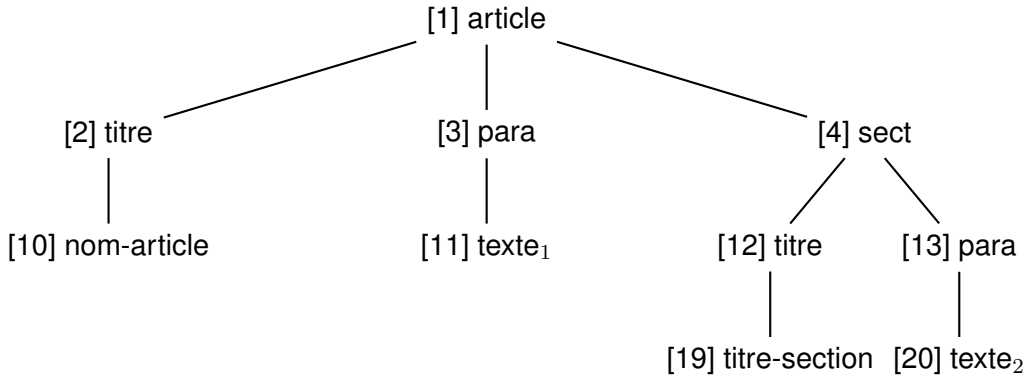


FIGURE B.1 – arbre XML $\mathscr{T}$: article Wikipedia

Example B.1.1. *La figure B.1 montre un arbre XML $\mathscr{T}$ qui représente un article Wikipedia classique. Les identifiants de nœuds sont à l'intérieur des crochets précédant les chaînes de caractères de ses derniers. Le titre de cet article est donné par le nœud 10. Le contenu du document est structuré en sections (« sect ») avec leurs titres et paragraphes (« para ») renfermant les données textes.*

À l'aide des identifiants uniques, nous considérons deux types d'opérations d'édition sur le modèle de documents XML spécifié : *insertions* et *suppressions* de nœuds. Nous désignons une insertion par `ins(i, x)` dont la sémantique sur tout arbre XML consiste à ajouter le nœud x (nous supposons que x n'est pas déjà dans l'arbre) comme fils d'un certain nœud y tel que $\alpha(y) = i$. Si un tel nœud n'est pas trouvé dans l'arbre, l'opération ne fait rien. Observer qu'une insertion peut concerner un sous-arbre, et dans ce cas nous nous référons simplement par x à la racine de ce sous-arbre. De façon analogue, nous introduisons une suppression comme étant `del(i)` où i est l'identifiant du nœud à supprimer. L'opération de suppression élimine le nœud cible, s'il existe, ainsi

que ses descendants, de l'arbre XML. Nous concluons en définissant un script d'édition XML, $\Delta = \langle u_1, u_2, \ldots, u_i \rangle$, comme une séquence d'opérations d'édition élémentaires u_j (chaque u_j, avec $1 \leq j \leq i$, étant soit une insertion soit une suppression) à évaluer l'une après l'autre sur un document XML pour en créer un nouveau. Pour un arbre $\mathscr{T}$, nous notons le résultat de l'évaluation d'un script d'édition Δ sur $\mathscr{T}$ par $[\mathscr{T}]^{\Delta}$. Même si dans ce travail nous nous basons sur des identifiants persistants de nœuds pour définir nos opérations d'édition, la sémantique de ces opérations pourrait être étendue aux mises à jour exprimées par des requêtes, particulièrement utile dans les environnements d'édition collaborative distribués où les identifiants peuvent ne pas être facilement partageables.

B.2 XML probabiliste

Nous présentons brièvement dans cette section le système de représentation XML probabiliste que nous utilisons comme élément de base de notre système de contrôle de versions incertaines. Pour plus de détails, voir [Abiteboul et al., 2009] pour le cadre général et [Kharlamov et al., 2010] pour le modèle spécifique PrXML$^{\text{fie}}$ que nous employons. Ces modèles de représentation d'incertitude sont conçus à l'origine pour la gestion de données dans les domaines de l'intégration de données du Web et de l'extraction d'informations.

B.2.1 Les p-documents PrXML$^{\text{fie}}$: syntaxe et sémantique

Un *système de représentation XML probabiliste* est une manière compacte d'encoder des distributions de probabilité sur des documents XML ; dans le cas d'intérêt ici, la distribution de probabilité est finie. Formellement, un espace de distribution XML probabiliste, ou px-espace, $\mathscr{S}$ sur une collection de documents XML est un couple (D, p) où D est un ensemble fini non vide de documents et $p : D \to]0, 1]$ une mesure de probabilités qui affecte à chaque document d de D un nombre rationnel $p(d) \in]0, 1]$ tel que $\Sigma_{d \in D} p(d) = 1$. Un *p-document*, ou *document XML probabiliste*, habituellement noté $\widehat{\mathscr{P}}$, définit un encodage compact d'un px-espace $\mathscr{S}$.

Nous considérons dans cet article une classe spécifique de p-documents, PrXML$^{\text{fie}}$ [Kharlamov et al., 2010] (où fie est l'abréviation de *formula of independent events* ou *formule d'événements indépendants*) ; se restreindre à cette classe en particulier nous permet de donner une présentation simplifiée, se référer à [Abiteboul et al., 2009, Kharlamov et al., 2010] pour un cadre plus général. Supposons donné un ensemble de *variables aléatoires booléennes indépendantes*, ou *variables d'événements*, $b_1, b_2, \ldots, b_m$ et leurs

probabilités d'existence respectives $\mathsf{P_r}(b_1), \mathsf{P_r}(b_2), \ldots, \mathsf{P_r}(b_m)$. Un document PrXML$^{\mathsf{fie}}$ est un arbre non ordonné, sans limite sur le nombre de fils par nœud, et étiqueté, où chaque nœud (à l'exception de la racine) x peut être annoté avec une formule propositionnelle arbitraire $fie(x)$ sur les variables d'événements $b_1, b_2, \ldots, b_m$. Des formules différentes peuvent partager des événements communs, ce qui veut dire qu'il peut y avoir des corrélations entre formules, et le nombre de variable d'événements peut changer d'un nœud à l'autre.

Une affectation de valeurs de vérités ν des variables d'événements $b_1 \ldots b_m$ induit sur $\widehat{\mathscr{P}}$ un document XML particulier $\nu(\widehat{\mathscr{P}})$: le document où seuls les nœuds annotés avec des formules qui s'évaluent à true par ν sont conservés (les nœuds dont les formules s'évaluent à false par ν sont supprimés de l'arbre, avec leurs descendants). Étant donné un p-document $\widehat{\mathscr{P}}$, les *mondes possibles* de $\widehat{\mathscr{P}}$, notés $pwd(\widehat{\mathscr{P}})$, sont l'ensemble des tels documents XML. La *probabilité* d'un monde possible donné d de $\widehat{\mathscr{P}}$ est définie comme la somme des probabilités des affectations de valeurs de vérité qui donnent d. L'ensemble des mondes possibles, avec leurs probabilités, définit la *sémantique* de $\widehat{\mathscr{P}}$, le px-espace $[\![\widehat{\mathscr{P}}]\!]$ associé à $\widehat{\mathscr{P}}$.

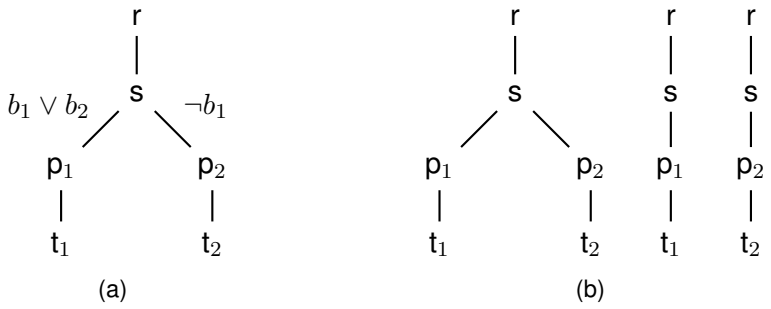


FIGURE B.2 – (a) un p-document PrXML$^{\mathsf{fie}}$; (b) trois mondes possibles d_1, d_2 et d_3

Example B.2.1. *La figure B.2 montre (à gauche) un p-document* PrXML$^{\mathsf{fie}}$ *concret* $\widehat{\mathscr{P}}$ *et (à droite) trois mondes possibles* d_1, d_2 *et* d_3. *Les formules annotant les nœuds sont indiquées au-dessus d'eux :* $b_1 \vee b_2$ *et* $\neg b_1$ *sont liés aux nœuds* p_1 *et* p_2, *respectivement. Les trois mondes possibles* d_1, d_2 *et* d_3 *sont obtenus en choisissant les valeurs de vérités de* b_1 *et* b_2 *suivantes : (i)* false *et* true *; (ii)* true *et* true *(ou* true *et* false*) ; (iii)* false *et* false. *Pour chaque affectation de valeurs de vérités, on garde exactement les nœuds dont les formules s'évaluent à* true. *Si on fixe une distribution de probabilités sur les événements (grâce par exemple à un algorithme évaluant la réputation de la source de*

chaque événement), par exemple $\mathsf{P_r}(b_1) = 0.4$ *et* $\mathsf{P_r}(b_2) = 0.5$*, on dérive la probabilité du monde possible* d_1 *comme* $\mathsf{P_r}(d_1) = (1 - \mathsf{P_r}(b_1)) \times \mathsf{P_r}(b_2) = 0.6 \times 0.5 = 0.3$*. Il est possible de calculer la probabilité des autres mondes possibles de manière similaire.*

Par rapport aux autres systèmes de représentation XML probabiliste [Abiteboul et al., 2009], PrXML$^{\text{fie}}$ est très succinct (car des formules propositionnelles arbitraires peuvent être utilisées, avec des corrélations arbitraires entre événements), c'est-à-dire, exponentiellement plus succinct que les modèles de [Van Keulen et al., 2005, Nierman and Jagadish, 2002] ; il offre par ailleurs des insertions et suppressions en temps polynomial [Kharlamov et al., 2010], un besoin clef pour notre modèle de contrôle de versions incertain. Cependant, un inconvénient non négligeable est que toutes les requêtes (à motif d'arbre) non triviales sur ce modèle sont **#P**-difficile à évaluer [Kimelfeld et al., 2009]. Ce n'est pas nécessairement un problème ici, puisque nous privilégions dans notre cas des mises à jour efficaces et le fait de pouvoir extraire des mondes possibles, plutôt que l'exécution de requêtes arbitraires.

B.2.2 Provenance des données

La gestion de données XML incertaines basée sur le modèle PrXML$^{\text{fie}}$ bénéficie également des sémantiques multiples des variables d'événements en termes de description de l'information. En effet, en plus de la gestion de l'incertitude, ce modèle peut également supporter la conservation de l'information sur la *provenance des données* (ou *lignée*)[4] à l'aide des variables d'événements. La provenance des données consiste en des informations de traçabilité telles que la sémantique des changements, le responsable de ces changements, un estampillage temporel, etc., pour des données incertaines. Pour ce faire, il suffit d'utiliser la sémantique des variables d'événements comme représentant de l'information sur la provenance des données. Il est ainsi parfois utile d'utiliser des systèmes de représentation XML probabiliste même en l'absence de sources de probabilités fiables pour les événements individuels, au sens où ces événements peuvent être manipulés comme faisant partie d'un modèle de données incomplet (c'est-à-dire, on ne prend en compte que les mondes possibles et non leurs probabilités).

4. Comme étudié dans [Zhang and Jagadish, 2013], garder des informations de provenance dans un processus de contrôle de versions pourrait être utile dans le cas de requêtes sur la provenance des données.

B.3 Modèle XML multi-version incertain

Dans cette partie, nous développons notre modèle de contrôle de versions XML incertain pour des documents arborescents édités de manière collaborative. Nous basons notre modèle sur trois concepts principaux : les événements de contrôle de versions, un p-document et un graphe orienté acyclique d'événements. Nous débutons par une formalisation des documents XML multi-version à travers une définition formelle de leur graphe d'espace de versions et de leur ensemble de versions. Le modèle proposé est ensuite introduit.

B.3.1 Documents XML multi-version

Considérons l'ensemble infini $\mathscr{D}$ de tous les documents XML ayant une racine avec étiquette et identifiant fixés. Soit $\mathscr{V}$ l'ensemble des *événements de contrôle de versions* $e_1 \dots e_n$. Ces événements représentent les différents états d'un arbre. Nous associons aux événements des informations contextuelles à propos des révisions (auteur, estampille temporelle, etc.). À chaque événement e_i est également associé un *script d'édition* Δ_i. En partant de ces bases, nous formalisons le graphe de l'espace de versions et l'ensemble des versions d'un document XML versionné comme suit.

L'espace des versions est un graphe orienté acyclique (*directed acyclic graph* ou DAG) $\mathscr{G} = (\mathscr{V} \cup \{e_0\}, \mathscr{E})$ où : (i) l'événement de contrôle de versions initial $e_0 \notin \mathscr{V}$, un événement spécial représentant le premier état de chaque arbre XML versionné, est la racine de $\mathscr{G}$; (ii) $\mathscr{E} \subseteq (\mathscr{V} \cup \{e_0\})^2$, définissant les arêtes de $\mathscr{G}$, consiste en un ensemble de couples ordonnés d'événements de contrôle de versions. Chaque arête décrit implicitement une relation de dérivation orientée entre deux versions. Une *branche* de $\mathscr{G}$ est un chemin orienté impliquant un nœud initial e_i et un nœud final e_j. Ce dernier doit être atteignable depuis e_i en traversant un ensemble d'arêtes orientées de $\mathscr{E}$. Nous notons cette branche B_i^j. Une *branche enracinée* est une branche qui démarre à la racine du graphe.

Une version XML est le document de $\mathscr{D}$ qui correspond à un *ensemble* d'événements de contrôle de versions, l'ensemble des événements qui a donné lieu à cette version. Dans un système de contrôle de versions déterministe, cet ensemble correspond toujours à une branche enracinée du graphe de l'espace de versions. Dans notre système de contrôle de versions incertain, cet ensemble peut être arbitraire. Considérons l'ensemble $2^{\mathscr{V}}$ formé de toutes les sous-parties de $\mathscr{V}$. Cet ensemble de versions d'un document XML multi-version est donné par une fonction $\Omega : 2^{\mathscr{V}} \to \mathscr{D}$: à chaque ensemble d'événements

correspond un arbre donné (ces arbres ne sont en général pas tous distincts). La fonction Ω peut être calculée à partir des scripts d'édition associés aux événements de la manière suivante :

- $\Omega(\emptyset)$ est l'arbre XML, formé d'une racine seule, de $\mathscr{D}$.
- Pour tout i, pour tout $\mathscr{F} \subseteq 2^{\mathscr{V}\setminus\{e_i\}}$ $\Omega(\{e_i\} \cup \mathscr{F}) = [\Omega(\mathscr{F})]^{\Delta_i}$.

On définit maintenant un document XML multi-version, $\mathscr{T}_{mv}$, comme une paire $(\mathscr{G}, \Omega)$ où $\mathscr{G}$ est un DAG d'événements de contrôle de versions et Ω est une fonction définissant l'ensemble des versions du document. Dans ce qui suit nous proposons une manière plus efficace de calculer la version correspondant à un ensemble d'événements, en utilisant un p-document comme modèle.

B.3.2 Document XML multi-version incertain : gestion des données incertaines

Un document multi-version est dit *incertain* si les événements de contrôle de versions, utilisés dans un processus de contrôle de versions, sont munis d'*incertitude*, comme dans des contextes collaboratifs ouverts. Quand nous parlons d'incertitude des événements de contrôle de versions, nous voulons dire que ce sont des événements aléatoires conduisant à des versions, et du contenu, incertains. Par conséquent, nous nous appuierons sur une *distribution de probabilité sur* $2^{\mathscr{V}}$ qui induira, en conjonction avec la fonction Ω, une distribution de probabilités sur $\mathscr{D}$.

Nous modélisons l'incertitude des événements en définissant maintenant un événement de contrôle de versions e_i de $\mathscr{V}$ comme une conjonction de variables booléennes aléatoires indépendantes $b_1 \dots b_m$ avec les hypothèses suivantes : (i) une variable booléenne modélise une source donnée d'incertitude (p. ex., le contributeur) dans l'environnement de contrôle de versions ; (ii) les variables booléennes de chaque e_i sont deux à deux indépendantes ; (iii) une variable booléenne b_j réutilisée d'un événement à l'autre corrèle des événements de contrôle de version différents ; (iv) une variable booléenne particulière $b^{(i)}$, dite de *révision*, représentant plus spécifiquement l'incertitude dans la contribution, n'est pas partagée avec les autres événements de contrôle de versions et apparaît uniquement dans e_i.

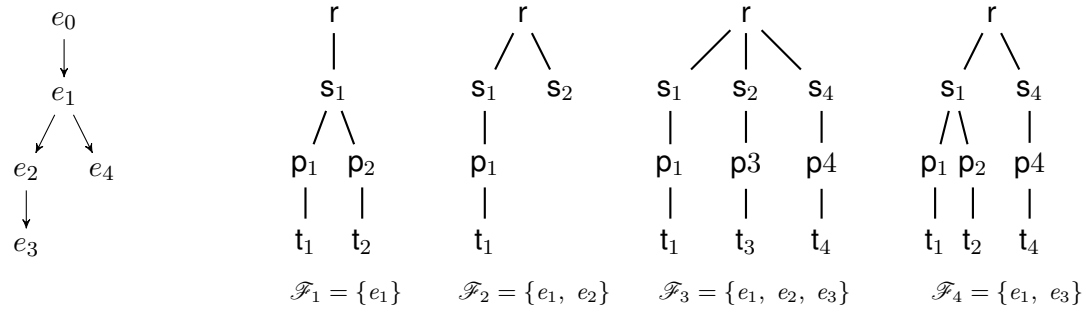
Supposons donnée une distribution de probabilité sur les variables booléennes aléatoires b_j (cela vient typiquement d'une estimation de la confiance en un contributeur, ou en une contribution), qui induit une distribution de probabilité sur les formules propositionnelles sur les b_j de la manière usuelle [Kharlamov et al., 2010]. Nous obtenons

maintenant la probabilité de chaque version (incertaine) d ainsi :

$$\mathsf{P_r}(d) = \mathsf{P_r}\left(\bigvee_{\substack{\mathscr{F} \subseteq \mathscr{V} \\ \Omega(\mathscr{F}) = d}} \phi(\mathscr{F}) \right)$$

avec la formula $\phi(\mathscr{F})$ définie par :

$$\phi(\mathscr{F}) = \bigwedge_{e_j \in \mathscr{F}} e_j \wedge \bigwedge_{e_k \in \mathscr{V} \setminus \mathscr{F}} \neg e_k. \tag{B.1}$$



(a) Graphe $\mathscr{G}$

(b) Versions possibles $\mathscr{T}_1$, $\mathscr{T}_2$, $\mathscr{T}_3$ et $\mathscr{T}_4$

FIGURE B.3 – a) graphe d'espace de versions ; (b) 4 versions et leurs valeurs de vérité

Example B.3.1. *La figure B.3 montre un document XML multi-version $\mathscr{T}_{mv}$ sujet à quatre événements de contrôle de versions. À gauche, nous avons l'espace des versions $\mathscr{G}$. La partie droite montre un exemple de quatre versions (incertaines) et leurs ensembles d'événements associés. On suppose que $\mathscr{T}_{mv}$ est initialement un document formé uniquement de la racine. Les trois premières versions correspondent aux versions couvertes par les systèmes déterministes de contrôle de versions, tandis que la dernière est engendrée en considérant que les changements accomplis par un événement de contrôle de versions intermédiaire, ici e_2, ne sont pas corrects. L'une des particularités de notre modèle est de fournir la possibilité de voir et de modifier ces types de versions incertaines qui représentent des versions virtuelles. Seules les éditions faites par les événements de contrôle de versions indiqués sont prises en compte dans le processus de production d'une version : dans $\mathscr{T}_4$, le nœud r et les sous-arbres enracinés en s_1, s_3 introduits respectivement par e_0, e_1 et e_3 sont présents, tandis que le sous-arbre p_3 ajouté par e_3 n'apparaît pas parce que son nœud père s_2 n'existe pas. Enfin, étant*

données des probabilités pour les événements de contrôle de versions, il est possible de mesurer la fiabilité de chaque version incertaine $\mathscr{T}_i$, pour $1 \leq i \leq 4$, en fonction de son ensemble d'événements correspondant $\mathscr{F}_i$ (et de tous les autres ensembles d'événements qui conduisent au même arbre).

On observe directement, en particulier dans l'exemple de la figure B.3, que le nombre de versions (incertaines) possibles d'un document multi-version incertain croît rapidement (en fait, exponentiellement en le nombre d'événements). Le résultat en est que l'énumération et la gestion de toutes les possibilités avec la fonction Ω peut devenir difficile à partir d'un certain point. Pour résoudre ce problème, nous proposons une méthode efficace d'encodage de manière compacte des versions possibles, avec leurs valeurs de vérité. Intuitivement, un p-document PrXML$^{\text{fie}}$ modèle de manière compacte l'ensemble des versions possibles d'un document XML multi-version. Comme nous en avons discuté dans la section B.2, un arbre probabiliste basé sur des formules propositionnelles fournit des caractéristiques utiles à notre contexte. Premièrement, il décrit de manière appropriée une distribution de valeurs de vérité sur un ensemble d'arbres XML incertains tout en fournissant un processus sensé pour retrouver une version donnée et sa probabilité. Deuxièmement, il fournit un système de représentation efficace pour les mises à jour, ce qui est crucial dans les environnements dynamiques tels que les applications basées sur le contrôle de versions.

B.3.3 Encodage XML probabiliste

Nous définissons un formalisme général de représentation de contrôle de versions XML incertain, habituellement noté $\widehat{\mathscr{T}_{mv}}$, comme un couple $(\mathscr{G}, \widehat{\mathscr{P}})$ où (a) $\mathscr{G}$ est, comme auparavant, un DAG d'événements, représentant l'espace de versions ; (b) $\widehat{\mathscr{P}}$ est un p-document PrXML$^{\text{fie}}$ avec des variables aléatoires booléennes $b_1 \ldots b_m$ représentant efficacement l'ensemble des versions XML (incertaines) possibles et leurs valeurs de vérités correspondantes.

Nous définissons maintenant la sémantique d'un tel encodage comme un document multi-version incertain $(\mathscr{G}, \Omega)$ où $\mathscr{G}$ est le même et Ω est défini comme suit. Pour $\mathscr{F} \subseteq \mathscr{V}$, soit B^+ l'ensemble de toutes les variables aléatoires apparaissant dans l'un des événements de $\mathscr{F}$ et B^- l'ensemble de toutes les variables de révision $b^{(i)}$ pour e_i *non membre de* $\mathscr{F}$. Soit ν l'affectation de valeur de vérité de $b_1 \ldots b_m$ qui associe aux variables de B^+ la valeur true, aux variables de B^- la valeur false et aux autres variables une valeur arbitraire. Nous posons : $\Omega(\mathscr{F}) := \nu(\widehat{\mathscr{P}})$.

Le résultat suivant montre que cette sémantique est compatible avec la sémantique de px-espace des p-documents d'une part, et avec la distribution de probabilité définie par les documents multi-version incertains d'autre part.

Theorem B.3.1. *Soit $(\mathscr{G}, \widehat{\mathscr{P}})$ un système de représentation de contrôle de versions incertain et $(\mathscr{G}, \Omega)$ sa sémantique. Nous supposons que toutes les formules de $\widehat{\mathscr{P}}$ peuvent s'exprimer comme des formules sur les événements de $\mathscr{V}$ (nous n'utilisons donc pas les b_j indépendamment des événements de contrôle de versions). Alors le px-espace $[\![\widehat{\mathscr{P}}]\!]$ définit la même distribution de probabilité sur $\mathscr{D}$ que Ω.*

La preuve est directe et s'appuie sur l'équation (B.1).

B.4 Mise à jour de XML multi-version incertain

Nous implantons la sémantique des opérations standards de mise à jour au-dessus de notre système de représentation XML probabiliste. Une mise à jour sur un document multi-version incertain correspond à l'évaluation d'éditions incertaines sur une version (incertaine) donnée. Étant donné un triplet (Δ, e, e'), nous parlons de l'opération de mise à jour $\texttt{updOP}_{\Delta,\, e,\, e'}$ où Δ est un script d'édition, e est un événement de contrôle de versions existant pointant vers la version éditée et e' est un nouvel événement de contrôle de versions évaluant l'incertitude dans cette mise à jour. Nous formalisons $\texttt{updOP}_{\Delta,\, e,\, e'}$ sur $\mathscr{T}_{mv}$ ainsi :

$$\texttt{updOP}_{\Delta,\, e,\, e'}(\mathscr{T}_{mv}) := (\mathscr{G} \cup (\{e'\}, \{(e, e')\}),\ \Omega').$$

Une opération de mise à jour résulte donc en l'insertion d'un nouveau nœud et d'une nouvelle arête dans $\mathscr{G}$ et en l'extension de Ω en un Ω' que nous définissons maintenant. Pour chaque sous-ensemble $\mathscr{F} \subseteq \mathscr{V}'$ ($\mathscr{V}'$ est l'ensemble des nœuds de $\mathscr{G}$ après la mise à jour), on pose :

- si $e' \notin \mathscr{F}$: $\Omega'(\mathscr{F}) = \Omega(\mathscr{F})$;
- sinon : $\Omega'(\mathscr{F}) = [\Omega(\mathscr{F} \backslash \{e'\})]^{\Delta}$.

Ce qui précède donne une sémantique aux mises à jours sur des documents multi-version incertains ; cependant, la sémantique n'est pas utilisable en pratique car elle demande de considérer chaque sous-ensemble $\mathscr{F} \subseteq \mathscr{V}'$. Pour une solution plus commode, nous appliquons directement les mises à jour sur la représentation compact p-document du document multi-version. L'algorithme B.1 décrit comment une telle opération de mise à jour $\texttt{updOP}_{\Delta,\, e,\, e'}$ est réalisée sur une représentation incertaine $(\mathscr{G}, \widehat{\mathscr{P}})$. Tout d'abord, le graphe est mis à jour comme auparavant. Ensuite, pour chaque opération

Algorithme B.1: Algorithme de mise à jour (`updPrXML`)

Entrées: $(\mathscr{G}, \widehat{\mathscr{P}})$, $\texttt{updOP}_{\Delta,\, e,\, e'}$
Sorties: mise à jour de $(\mathscr{G}, \widehat{\mathscr{P}})$ par $\texttt{updOP}_{\Delta,\, e,\, e'}$

$\mathscr{G} := \mathscr{G} \cup (\{e'\}, \{(e, e')\})$;
pour chaque *(**opération d'édition** u **dans** Δ)* **faire**
  si u ***est une insertion*** `ins(`*i, x*`)` **alors**
    y := `findNodeById` $(\widehat{\mathscr{P}}, i)$;
    si `matchIsFound` $(\mathscr{T}_y, x)$ **alors**
      $\mathit{fie}_o(x)$:= `getFieOfNode` (x) ;
      `setFieOfNode` $(x, \mathit{fie}_o(x) \vee e')$;
    sinon
      `updContent` $(\widehat{\mathscr{P}}$, `ins(`*i, x*`))` ;
      `setFieOfNode` (x, e') ;
    fin
  fin
  sinon si u ***est une suppression*** `del(`*i*`)` **alors**
    x := `findNodeById` $(\widehat{\mathscr{P}}, i)$;
    $\mathit{fie}_o(x)$:= `getFieOfNode` (x) ;
    `setFieOfNode` $(x, \mathit{fie}_o(x) \wedge \neg e')$;
  fin
fin
retourner $(\mathscr{G}, \widehat{\mathscr{P}})$;

u dans Δ, l'algorithme récupère le nœud cible de $\widehat{\mathscr{P}}$ avec `findNodeById` (typiquement une opération en temps constant). Selon le type d'opération, il y a deux possibilités :

1. Si u est une insertion d'un nœud x, l'algorithme vérifie si x n'est pas déjà dans $\widehat{\mathscr{P}}$, par exemple en cherchant un nœud avec la même étiquette (la fonction `matchIsFound` recherche une correspondance pour x dans le sous-arbre $\mathscr{T}_y$ enraciné en y). Si une telle correspondance existe, `getFieOfNode` retourne sa formule courante $\mathit{fie}_o(x)$ et l'algorithme la met à jour en $\mathit{fie}_n(x) := \mathit{fie}_o(x) \vee e'$, spécifiant que x apparaît quand la mise à jour est valide. Sinon, `updContent` et `setFieOfNode` respectivement insère le nœud x dans $\widehat{\mathscr{P}}$ et met sa formule associée à $\mathit{fie}_n(x) = e'$.

2. Si u est une suppression d'un nœud x, l'algorithme obtient sa formule courante $\mathit{fie}_o(x)$ et la change en $\mathit{fie}_n(x) := \mathit{fie}_o(x) \wedge \neg e'$, indiquant que x doit être supprimé des mondes possibles quand cette mise à jour est valide.

Le reste de cette partie montre la correction et l'efficacité de notre approche. Tout d'abord, nous établissons que l'algorithme B.1 respecte la sémantique des mises à jour. Ensuite, nous montrons que le comportement des systèmes de contrôle de versions

déterministes peut être simulé en considérant uniquement un type spécifique d'ensemble d'événements. Finalement, nous caractérisons la complexité de l'algorithme.

Theorem B.4.1. *L'algorithme B.1, quand lancé sur un encodage XML probabiliste* $\widehat{\mathscr{T}_{mv}} = (\mathscr{G}, \widehat{\mathscr{P}})$ *d'un document multi-version* $\mathscr{T}_{mv} = (\mathscr{G}, \Omega)$*, avec une opération de mise à jour* $\mathtt{updOP}_{\Delta,e,e'}$*, calcule une représentation* $\mathtt{updOP}_{\Delta,e,e'}(\widehat{\mathscr{T}_{mv}})$ *du document multi-version* $\mathtt{updOP}_{\Delta,e,e'}(\mathscr{T}_{mv})$*.*

La preuve de ce théoréme est détaillée dans [Ba et al., 2013a].

La sémantique des mises à jour est donc la même, qu'elle soit énoncée sur les documents multi-version ou implantée par l'algorithme B.1. Nous montrons maintenant que cette sémantique est compatible avec les opérations classiques de mise à jour des systèmes de contrôle de versions déterministes.

Theorem B.4.2. *La définition formelle de mise à jour dans les documents multi-version incertains implante la sémantique des opérations standards de mise à jour dans les systèmes de contrôle de versions déterministes quand les ensembles d'événements sont restreints à des branches enracinéees.*

Nous terminons cette partie en énonçant le passage à l'échelle de notre algorithme :

Theorem B.4.3. *L'algorithme B.1 accomplit le processus de mise à jour sur la représentation d'un document XML multi-version en temps constant en la taille du document d'entrée. La taille de l'arbre probabiliste de sortie croît linéairement en la taille du script d'édition.*

B.5 Fusion de XML multi-version incertain

Nous détaillons dans cette section la transposition de l'opération de fusion XML usuelle dans notre modèle de contrôle de versions incertain. Nous présentons en premier lieu le processus de calcul des scripts d'édition à utiliser pour la fusion, et les scénarios de fusion les plus fréquents. Ensuite nous introduisons la sémantique de la fusion sur des versions incertaines d'un même document XML, ainsi qu'un algorithme efficace sur l'encodage XML probabiliste.

B.5.1 Stratégie de fusion : calcul de scripts d'édition et scénarios de fusion usuels

Une opération de fusion prend un ensemble de versions et fusionne leurs contenus dans une seule et nouvelle version. Nous nous concentrons ici sur la fusion de deux versions (le cas le plus usité dans les applications de tous les jours) à l'aide d'une approche *tripartite* (ou *three-way*), à savoir, une intégration de mises à jour provenant des entrées suivant leur version de base commune. L'intérêt principale de la fusion tripartite par rapport à celle bipartite (ou *two-way*) vient du fait qu'elle assure une meilleure correspondance de nœuds et détection de conflits. Nous supposons aussi que toutes les versions à fusionner sont uniquement générées à partir de mises à jour sur les versions de bases – cela, une fois de plus, à des fins de clarté dans la présentation. Le processus de fusion se divise en général en deux étapes : (i) une extraction des différents scripts d'édition qui ont mené aux versions en entrée et ; (ii) une génération du résultat de la fusion, en évaluant d'abord l'ensemble unifié des scripts extraits sur des données initiales, puis en résolvant les conflits éventuels. Supposons donné un document XML non ordonné sous contrôle de versions. Soient $\mathscr{T}_1$ et $\mathscr{T}_2$ deux versions arbitraires de ce document, ainsi que leur plus proche ancêtre commun $\mathscr{T}_a$. Nous détaillons ci-après les deux étapes du processus de fusion sur les versions $\mathscr{T}_1$ et $\mathscr{T}_2$.

Tout d'abord, nous déterminons le script d'édition spécifiant la fusion des versions $\mathscr{T}_1$ et $\mathscr{T}_2$ à l'aide de la fonction $\mathit{diff3}(\mathscr{T}_1, \mathscr{T}_2, \mathscr{T}_a)$. Cette fonction s'exécute sur des arbres non ordonnés dans lesquels les nœuds ont des identifiants uniques. Elle retournera un script avec seulement des insertions et des suppressions de nœuds comme opérations autorisées. De manière similaire à [Khanna et al., 2007], notre $\mathit{diff3}$ se base sur les fonctions $\mathit{diff2}(\mathscr{T}_a, \mathscr{T}_1)$ et $\mathit{diff2}(\mathscr{T}_a, \mathscr{T}_2)$, appelées algorithmes de diff *bipartite* (*two-way*). Les algorithmes bipartites ainsi définis apparient tout d'abord les nœuds de leurs entrées, et ensuite expriment les correspondances sous la forme de scripts d'édition Δ_1 et Δ_2, respectivement. Le résultat du $\mathit{diff3}$, que nous notons Δ_3, est égal à $\Delta_1 \cup \Delta_2$. Cette union peut résulter en trois types d'édition : éditions *équivalentes*, *conflictuelles* et *indépendantes*. Deux opérations $u_1 \in \Delta_1$ et $u_2 \in \Delta_2$ sont *équivalentes* si elles ont la même sémantique et les mêmes arguments. Seule une des deux est retenue dans Δ_3. À l'inverse, u_1 et u_2 sont *conflictuelles* lorsque u_1 est une insertion et u_2 est une suppression (ou inversement), et u_1 a ajouté des nœuds comme descendants de nœuds supprimés par u_2. Nous référons à l'ensemble de toutes les éditions conflictuelles dans Δ_3 par $\Delta^{\mathscr{C}}$. Nous disons qu'un nœud modifié par des éditions conflictuelles est un *nœud*

conflictuel. Enfin, les éditions indépendantes sont celles de Δ_1 et Δ_2 qui n'appartiennent pas aux deux premières classes. L'ensemble des éditions équivalentes et indépendantes représentent les éditions *non conflictuelles* d'un algorithme de diff donné. Un nœud impacté par une édition non conflictuelle est un nœud *non conflictuel* pour une fusion.

La plupart des modèles de gestion de versions actuels propose trois scénarios de fusion qui prennent en compte la résolution de conflits éventuels (voir options de fusion, en particulier *mine-conflict* et *theirs-conflict*, dans Subversion [Pilato, 2004]). Rappelons que dans la grande majorité des cas, cette résolution est manuelle. Soit $\mathscr{T}_f$ la version résultante de la fusion de $\mathscr{T}_1$ et $\mathscr{T}_2$. Nous abstrayons ces cas de fusion possibles à l'aide des scripts et des versions en entrée comme suit :

1. $\mathscr{T}_f = [\mathscr{T}_1]^{\Delta_2 - \Delta^{\mathscr{C}}}$, c'est-à-dire, une fusion en considérant $\mathscr{T}_1$ et les éditions non conflictuelles provenant de Δ_1 ;
2. $\mathscr{T}_f = [\mathscr{T}_2]^{\Delta_1 - \Delta^{\mathscr{C}}}$ (ce cas est symétrique au précédent) ;
3. $\mathscr{T}_f = [\mathscr{T}_a]^{\Delta_3 - \Delta^{\mathscr{C}}}$, c'est-à-dire, une fusion sur la base de la version $\mathscr{T}_a$ et des éditions non conflictuelles provenant de Δ_3.

On peut noter (a) qu'il est facile de montrer que lorsque $\Delta^{\mathscr{C}} = \emptyset$, on obtient le même résultat pour les trois cas et ; (b) que la cas intuitif et naïf de fusion où l'utilisateur corrige les conflits peut être traité en choisissant en prime abord l'un des trois résultats ci-dessus avant d'effectuer ensuite les modifications souhaitées.

B.5.2 Fusion de versions XML incertaines

Nous considérons à présent notre abstraction de la fusion (couvrant au moins les scénarios abstraits ci-dessus) sur tout document XML multi-version incertain et l'algorithme correspondant sur son encodage XML probabiliste. Un contexte incertain induit une fusion à la base incertaine ; les versions en entrée et les scripts sont fournis avec de l'incertitude. Considérons un document XML multi-version incertain $\mathscr{T}_{mv} = (\mathscr{G}, \Omega)$ sujet à n événements de contrôle de versions et $\widehat{\mathscr{T}_{mv}} = (\mathscr{G},\ \widehat{\mathscr{P}})$ son encodage XML probabiliste.

Étant donné un triplet $(e_1,\ e_2,\ e')$, nous parlons de l'opération de fusion avec de l'incertitude $\texttt{mergeOP}_{e_1,\ e_2,\ e'}$ où e_1 et e_2 pointent vers les deux versions à être fusionné et e' est un nouvel événement évaluant l'incertitude dans cette fusion. Nous formalisons la sémantique de $\texttt{mergeOP}_{e_1,\ e_2,\ e'}$ sur $\mathscr{T}_{mv}$ comme suit :

$$\texttt{mergeOP}_{e_1,\ e_2,\ e'}(\mathscr{T}_{mv}) := (\mathscr{G} \cup (\{e'\}, \{(e_1,\ e'), (e_2,\ e')\}),\ \Omega').$$

D'une part, cette évaluation ajoute un nouvel événement et deux arêtes dans l'espace des versions $\mathscr{G}$. D'autre part, elle génère une nouvelle distribution Ω' laquelle étend

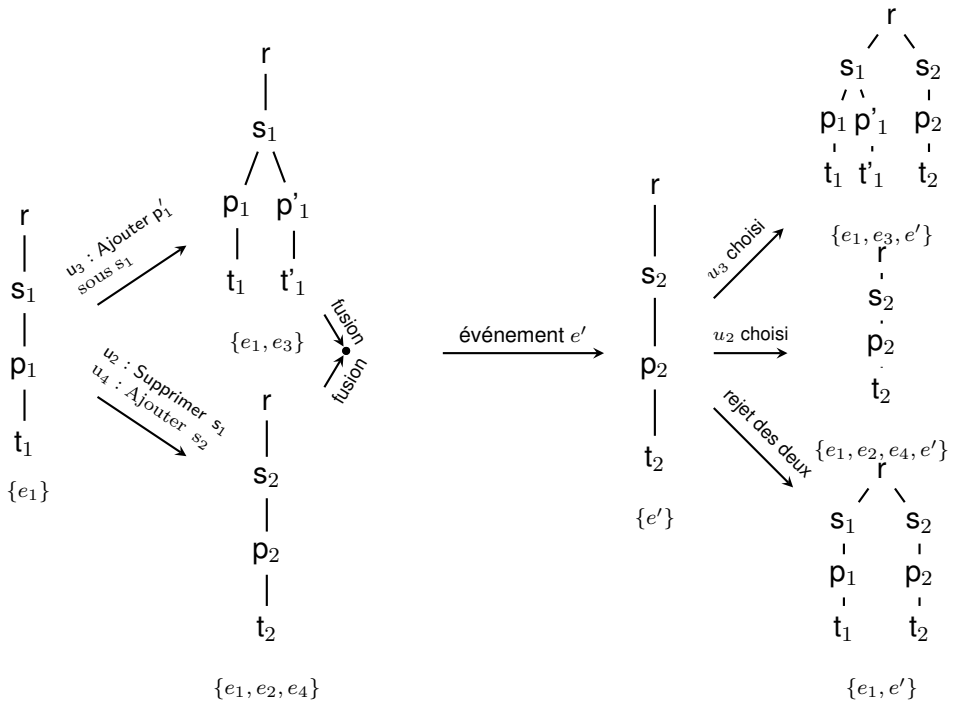
Ω avec de nouvelles versions possibles et ensembles d'événements. Soit $\mathscr{A}_{e_1}$ et $\mathscr{A}_{e_2}$ l'ensemble de toutes les ancêtres strictes dans $\mathscr{G}$ de e_1 et e_2 respectivement. Nous notons l'intersection par $\mathscr{A}_s = \mathscr{A}_{e_1} \cap \mathscr{A}_{e_2}$. Pour tout $\mathscr{F} \in 2^{\mathscr{V} \cup \{e'\}}$, formellement nous établissons :

- Si $e' \notin \mathscr{F}$: $\Omega'(\mathscr{F}) := \Omega(\mathscr{F})$;
- Si $\{e_1, e_2, e'\} \subseteq \mathscr{F}$: $\Omega'(\mathscr{F}) := \Omega(\mathscr{F} \setminus \{e'\})$;
- Si $\{e_1, e'\} \subseteq \mathscr{F}$ et $e_2 \notin \mathscr{F}$: $\Omega'(\mathscr{F}) := [\Omega((\mathscr{F} \setminus \{e'\}) \setminus (\mathscr{A}_{e_2} \setminus \mathscr{A}_s))]^{\Delta_2 - \Delta^{\mathscr{C}}}$;
- Si $\{e_2, e'\} \subseteq \mathscr{F}$ et $e_1 \notin \mathscr{F}$: $\Omega'(\mathscr{F}) := [\Omega((\mathscr{F} \setminus \{e'\}) \setminus (\mathscr{A}_{e_1} \setminus \mathscr{A}_s))]^{\Delta_1 - \Delta^{\mathscr{C}}}$;
- Si $\{e_1, e_2\} \cap \mathscr{F} = \emptyset$ et $e' \in \mathscr{F}$: $\Omega'(\mathscr{F}) := [\Omega((\mathscr{F} \setminus \{e'\}) \setminus ((\mathscr{A}_{e_1} \setminus \mathscr{A}_s) \cup (\mathscr{A}_{e_2} \setminus \mathscr{A}_s)))]^{\Delta_3 - \Delta^{\mathscr{C}}}$.

Les scripts ci-dessus sont tous calculés à l'aide de la fonction tripartite décrite dans la section B.5.1. Cette fonction prend en entrée dans chacun des cas requis les versions arbitraires $\mathscr{T}_1 = \Omega((\mathscr{F} \setminus \{e'\} \cap \mathscr{A}_{e_1}) \cup \{e_1\})$, $\mathscr{T}_2 = \Omega((\mathscr{F} \setminus \{e'\} \cap \mathscr{A}_{e_2}) \cup \{e_2\})$, et $\mathscr{T}_a = \Omega(\mathscr{F} \setminus \{e'\} \cap \mathscr{A}_s)$ où $\mathscr{F}$ est un sous-ensemble d'événements dans $\mathscr{V} \cup \{e'\}$ considérés comme valides.

Example B.5.1. *La figure B.4 décrit le processus de fusion de deux versions possibles $\mathscr{T}_1$ and $\mathscr{T}_2$ de l'exemple de la figure B.3 connaissant leur version de base $\mathscr{T}_a$. Dans notre modèle, cette opération est simplement prise en compte avec la fusion des événements e_3 et e_4 lesquels pointent sur les versions en entrée. À gauche de la figure, nous avons les versions $\mathscr{T}_1$, $\mathscr{T}_2$ et $\mathscr{T}_a$ ainsi que les scripts $\langle u_2, u_4 \rangle$ et $\langle u_3 \rangle$ qui ont conduit aux variantes. Ces scripts sont extraits avec les fonctions diff de la section B.5.1. À droite, nous explicitons le processus de fusion (avec l'événement de fusion e' estimant l'incertitude dans l'opération) comme suit : (i) Avant tout, toutes les éditions non conflictuelles dans les scripts, c'est-à-dire, ici uniquement u_4, sont validées pour obtenir la partie de la fusion (ou résultat intermédiaire) qui est certaine avec la validité de e' ; (ii) Ensuite, l'ensemble des fusions possibles est engendré en énumérant les différentes possibilités de traitement des éditions conflictuelles u_2 et u_3. Les deux premières versions possibles sont obtenues en propageant respectivement u_2 et u_3 sur le résultat intermédiaire. Concrètement, notre stratégie de fusion retournera les mêmes documents fusionnés en prenant $\mathscr{T}_1$ et $\mathscr{T}_2$ et en les mettant à jour avec les éditions non conflictuelles provenant respectivement de $\langle u_3 \rangle$ et $\langle u_2, u_4 \rangle$. Enfin, la dernière version possible est produite en annulant toutes les éditions conflictuelles, à savoir, en ajoutant les nœuds conflictuels de la version de base dans le résultat intermédiaire.*

L'opération de fusion incertaine ainsi formalisée ci-dessus ne passe cependant pas à l'échelle puisqu'elle exige d'évaluer toute version possible pour calculer le résultat global



(a) version de base $\mathscr{T}_a$, variantes $\mathscr{T}_1$ et $\mathscr{T}_2$; scripts $\langle u_2, u_4 \rangle$ et $\langle u_3 \rangle$

(b) génération de la fusion : d'abord, validation de u_4 puis résolution du conflit entre u_2 et u_3

FIGURE B.4 – stratégie de fusion : (a) versions incertaines et (b) résultat de la fusion

de la fusion. Dans ce qui suit, nous proposons une manière plus efficace de réaliser cette fusion.

B.5.3 Fusion sur l'encodage XML probabiliste

Nous introduisons Algorithme B.2 (`mergePrXML`) comme étant une méthode plus efficiente sur l'encodage XML probabiliste $\widehat{\mathscr{T}_{mv}} = (\mathscr{G},\ \widehat{\mathscr{P}})$ de réaliser la fusion de XML multi-version incertain. La définition de cette algorithme requiert toutefois la connaissance des nœuds conflictuels et non conflictuels pour la fusion. Ainsi nous définissons d'abord la notion de *nœuds conflictuels* étant donné une fusion $\texttt{mergeOP}_{e_1,\ e_2,\ e'}$ sur $\widehat{\mathscr{T}_{mv}}$, puis nous détaillons l'algorithme proprement dit. Nous nous appuierons sur les formules associées aux nœuds pour détecter ceux qui sont conflictuels.

Considérons les affectations de valeurs de vérité suivantes sur les événements dans $\mathscr{G}$: (i) ν_s initialisant les événements dans $\mathscr{A}_s$ à `true` et les variables de révision de toutes les autres événements à `false` ; (ii) ν_1 affectant `true` à toutes les événements

dans $\mathscr{A}_{e_1} \cup \{e_1\}$ et false aux variables de révision des autres événements ; et enfin (iii) ν_2 initialisant les événements dans $\mathscr{A}_{e_2} \cup \{e_2\}$ à true et les variables de révision des autres événements à false. Nous introduisons en premier lieu la lignée (ou provenance) d'un nœud incertain dans le p-document $\widehat{\mathscr{P}}$.

Definition B.5.1. *(Lignée d'un nœud) La lignée d'un nœud donné $x \in \widehat{\mathscr{P}}$, notée $fie^{\uparrow}$(x), est la formule propositionnelle résultante de la conjonction de la formule de ce nœud x avec celles associées à tous ses nœuds ancêtres dans $\widehat{\mathscr{P}}$.*

Au contraire de sa formule[5], la lignée d'un nœud dans le p-document encode l'histoire toute entière des éditions, à partir de l'événement initial, sur le chemin menant à ce nœud. En conséquence, nous pouvons définir les nœuds conflictuels dans le p-document en utilisant leurs lignées comme suit.

Definition B.5.2. *(Nœud conflictuel) Sous l'encodage XML probabiliste $\widehat{\mathscr{T}_{mv}}$, nous disons qu'un x donné dans $\widehat{\mathscr{P}}$ est un nœud conflictuel au regard de la fusion des événements e_1 et e_2 lorsque sa lignée satisfait les conditions suivantes. a) $fie^{\uparrow}(x) \models \nu_s$; b) $fie^{\uparrow}(x) \not\models \nu_1$ (ou $fie^{\uparrow}(x) \not\models \nu_2$) et ; c) $\exists y \in \widehat{\mathscr{P}}$, `desc(x, y)` : $fie^{\uparrow}(y) \not\models \nu_s$ et $fie^{\uparrow}(y) \models \nu_2$ (ou $fie^{\uparrow}(y) \models \nu_1$) où `desc(x, y)` veut dire que y est un descendant de x.*

Theorem B.5.1. *La définition B.5.2 est cohérente avec celle des nœuds conflictuels donnée dans la section B.5.1.*

La preuve du théorème B.5.1 est une conséquence directe de la façon dont `updPrXML` procède. Un nœud conflictuel dans $\widehat{\mathscr{P}}$ produit des descendants conflictuels. Nous nous référons aux nœuds conflictuels dans $\widehat{\mathscr{P}}$ suivant la fusion des événements e_1 et e_2 avec la restriction $\widehat{\mathscr{P}}_{|\mathscr{C}_{\{e_1,\ e_2\}}}$. C'est sur cette base que nous déduisons ci-après l'ensemble des nœuds non conflictuels.

Definition B.5.3. *(Nœud non conflictuel) Pour la fusion des événements e_1 et e_2, nous définissons un nœud non conflictuel x comme un nœud dans $\widehat{\mathscr{P}} \setminus \widehat{\mathscr{P}}_{|\mathscr{C}_{\{e_1,\ e_2\}}}$ dont la formule $fie(x)$ satisfait à une des conditions suivantes :*

- $fie(x) \models \nu_s$, $fie(x) \not\models \nu_1$ *et* $fie(x) \not\models \nu_2$;
- $fie(x) \not\models \nu_s$, $fie(x) \models \nu_1$ *et* $fie(x) \models \nu_2$;
- $fie(x) \models \nu_s$, $fie(x) \models \nu_1$ *et* $fie(x) \not\models \nu_2$;
- $fie(x) \models \nu_s$, $fie(x) \not\models \nu_1$ *et* $fie(x) \models \nu_2$;
- $fie(x) \not\models \nu_s$, $fie(x) \models \nu_1$ *et* $fie(x) \not\models \nu_2$;

5. La formule d'un nœud décrit juste la sémantique des éditions à partir de l'événement où il a été inséré pour la première fois.

Algorithme B.2: Algorithme de fusion (`mergePrXML`)

Entrées: $(\mathscr{G}, \widehat{\mathscr{P}})$, e_1, e_2, e'
Sorties: fusion dans $(\mathscr{G}, \widehat{\mathscr{P}})$ avec $\texttt{mergeOP}_{e_1,\, e_2,\, e'}$

1 $\mathscr{G} := \mathscr{G} \cup (\{e'\}, \{(e_1, e'), (e_2, e')\})$;
2 **pour chaque *nœud non conflictuel*** x *dans* $\widehat{\mathscr{P}} \setminus \widehat{\mathscr{P}}_{|\mathscr{C}_{\{e_1,\, e_2\}}}$ **faire**
3 | `replace` $(fie(x), e_1, (e_1 \vee e'))$;
4 | `replace` $(fie(x), e_2, (e_2 \vee e'))$;
5 **fin**
6 **retourner** $(\mathscr{G}, \widehat{\mathscr{P}})$

– $fie(x) \not\models \nu_s$, $fie(x) \not\models \nu_1$ *et* $fie(x) \models \nu_2$.

Theorem B.5.2. *La définition B.5.3 est cohérente avec celle des nœuds non conflictuels donnée dans la section B.5.1.*

La preuve du théorème B.5.2 est triviale.

Nous poursuivons cette section en détaillant d'abord l'algorithme de fusion, puis en énonçant sa correction.

L'algorithme B.2 prend en entrée l'encodage XML probabiliste $(\mathscr{G}, \widehat{\mathscr{P}})$ de $\mathscr{T}_{mv}$, les événements e_1 et e_2 dans $\mathscr{G}$, et le nouvel événement e' modélisant à la fois les éléments de la fusion et leur degré d'incertitude. L'algorithme met d'abord à jour $\mathscr{G}$ comme indiqué dans la section B.2. Ensuite, la fusion dans $\widehat{\mathscr{P}}$ résulte en une modification mineure des formules associées aux nœuds non conflictuels dans $\widehat{\mathscr{P}} \setminus \widehat{\mathscr{P}}_{|\mathscr{C}_{\{e_1,\, e_2\}}}$ au travers la fonction `replace`. Cette fonction substitue à ces formules toutes les occurrences de e_1 et e_2 par $(e_1 \vee e')$ et $(e_2 \vee e')$ respectivement. L'intuition est que chaque fusion possible, laquelle est valide avec l'existence de e' indépendamment de celle des autres événements, doit contenir au moins les nœuds non conflictuels dans $\widehat{\mathscr{P}}$ supposés certains avec e_1 et e_2. Les nœuds non conflictuels dont l'existence est indépendante de e_1 et e_2, dépendront uniquement des valeurs de vérité de leurs événements ancêtres au sein de chaque sous-ensemble d'événements incluant l'événement de fusion. Enfin, l'existence des nœuds conflictuels dans le résultat de la fusion repose sur une idée subjective de priorité implicite accordée à e_1 ou e_2 lorsque e' est true. Dans le scénario où e' et e_1 sont vrais et e_2 est false, nous supposons que e_1 est plus probable que e_2 pour la fusion ; dans ce cas seuls les nœuds conflictuels valides avec $\mathscr{A}_{e_1} \cup \{e_1\}$ sont sélectionnés. La situation inverse fonctionne de la même manière. Par contre tout nœud conflictuel sera supprimé lorsqu'aucune priorité n'est définie, c'est-à-dire, pour une affectation mettant e' à true et les variables de révision de e_1 et e_2 à false.

Soient $\mathscr{T}_{mv} = (\mathscr{G}, \Omega)$ et $\widehat{\mathscr{T}_{mv}} = (\mathscr{G}, \widehat{\mathscr{P}})$ respectivement un document XML multi-version incertain et son encodage XML probabiliste. Nous supposons, en plus, définie la fonction sémantique $[\![.]\!]$ telle que $[\![\widehat{\mathscr{T}_{mv}}]\!] = (\mathscr{G}, [\![\widehat{\mathscr{P}}]\!])$ avec $[\![\widehat{\mathscr{P}}]\!]$ renvoyant la même distribution de probabilité que Ω. Étant donné une fusion $\texttt{mergeOP}_{e_1,\, e_2,\, e'}$, nous énonçons à présent que `mergePrXML` respecte la sémantique de l'opération de fusion définie dans la section B.5.2.

Theorem B.5.3. *La définition de l'algorithme B.2 est correcte au regard de la sémantique de l'opération de fusion sur un document XML multi-version incertain. En d'autres mots, le diagramme suivant commute :*

$$\begin{array}{ccc} \widehat{\mathscr{T}_{mv}} & \xrightarrow{\quad [\![.]\!] \quad} & [\![\widehat{\mathscr{T}_{mv}}]\!] \\ \Big\downarrow \texttt{mergePrXML}\ (e_1, e_2, e') & & \Big\downarrow \texttt{mergeOP}_{e_1,\, e_2,\, e'} \\ \widehat{\mathscr{T}'_{mv}} & \xrightarrow[\quad [\![.]\!] \quad]{} & [\![\widehat{\mathscr{T}'_{mv}}]\!] \end{array}$$

Nous concluons cette section en énonçant le passage à l'échelle de `mergePrXML`.

Theorem B.5.4. *L'algorithme B.2 exécute la fusion sur l'encodage XML probabiliste de tout document XML multi-version incertain en temps proportionnel à la taille des formules des nœuds impactés par les mises à jour dans les branches fusionnées.*

Pour la preuve des théorèmes B.5.3 et B.5.4, nous référons à [Ba et al., 2013b].

B.6 Expérimentation et évaluation

Nous présentons, dans cette section, une évaluation expérimentale de notre modèle. Nous nous intéressons en premier aux opérations de création et de lecture de versions, et comparons l'efficacité de notre modèle sur ces deux opérations par rapport à celle de deux systèmes de gestion de versions de référence que sont Git et Subversion. Ensuite, nous présentons les avancées en termes de possibilité de filtrage de contenus qu'offre notre modèle. Toutes les mesures de temps montrées dans cette section correspondent à des temps de calcul en mémoire centrale (temps CPU). Les données sont préalablement chargées en mémoire centrale pour éviter le surcoût des accès disque. Les tests ont été réalisés dans les mêmes conditions matérielles pour les trois systèmes comparés.

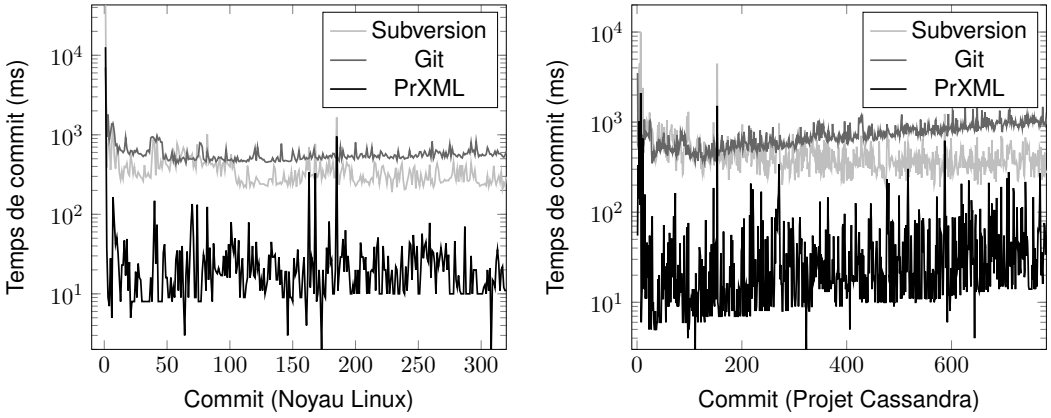


FIGURE B.5 – évaluation de la durée du commit (échelle logarithmique pour l'axe des ordonnées)

B.6.1 Analyse de performances : données tests, implantation et analyse de coûts

Nous avons mesuré le temps d'exécution des opération de création (ou *commit*) et de restitution (ou *checkout*) de versions, sur Git, Subversion et l'implantation de notre modèle (PrXML). L'objectif est de montrer la faisabilité de notre proposition. Vu que nos systèmes de référence sont tous déterministes, nous précisons que les tests pour l'analyse de performances ont été réalisés sur des données déterministes. Le principal intérêt de notre modèle demeurant sa capacité à gérer des versions de données incertaines, nous abordons et évaluons cet aspect dans la section B.6.2.

B.6.1.1 Jeux de données et implantation des tests

Les jeux de données utilisés pour les tests ont été constitués à partir des branches principales des projets de développement du noyau Linux [6], et de Apache Cassandra [7]. Les données représentent l'organisation hiérarchique de deux systèmes de fichiers assez larges, et constituent deux exemples de données arborescentes partagées dans un environnement collaboratif. Le projet de développement du noyau Linux utilise Git comme système de gestion de versions. Nous avons obtenu une copie de l'historique du système de fichiers en clonant la branche principale de développement du projet, et nous l'avons maintenue à jour en propageant régulièrement sur notre copie locale les dernières mises à jour effectuées sur la branche originale. Nous avons suivi la même

6. https://www.kernel.org/
7. http://cassandra.apache.org/

démarche pour les données du projet Cassandra, qui utilise Subversion comme système de gestion de versions. Au total, nous avons dans chaque branche locale plus de dix mille versions (ou commits), chacune matérialisant un ensemble de mises à jour. Dans nos expérimentations, nous nous sommes focalisé sur l'évolution de l'arborescence des systèmes de fichiers des projets considérés, en ignorant les changements réalisés sur le contenu plat des fichiers. Nous avons tracé les commits et les opérations de dérivation de versions à partir des logs de Git et de Subversion. Nous avons représenté la hiérarchie de chacun des deux systèmes de fichiers sous forme d'arbre XML sur lequel nous avons appliqué les changements successifs observés. À chaque insertion, respectivement suppression, d'un fichier ou d'un répertoire dans le système de fichiers est appliquée une insertion, respectivement suppression, d'un nœud dans l'arbre XML correspondant.

L'implantation de notre modèle de gestion de versions (PrXML) a été réalisée en Java. Nous nous sommes aussi basés sur les interfaces de programmation SVNKit [8] et JGit [9] de Java pour mettre en place les opérations standards de manipulation de versions dans Subversion et Git respectivement. Le but poursuivi est de réaliser toutes nos expérimentations dans les mêmes conditions. Le système Subversion mémorise dans ses fichiers de traces (logs) les changements effectués sur l'arborescence du système de fichiers à chaque commit. Chaque fichier log contient une liste d'expressions de chemins (*paths*), des identifiants de fichiers ou de répertoires dans le système de fichiers, et les mises à jours qui leurs sont associées. Dans le cas de Git, l'évolution du système de fichiers est représentée sous forme d'objets Git représentant les différents états de l'arborescence du système. Chaque objet correspond à un état du système de fichiers obtenu à la suite d'un commit.

B.6.1.2 Analyse des coûts

Les figures B.5 et B.7 comparent le coût, en termes de temps d'exécution, des opérations de commit et de checkout de versions dans Git, Subversion et notre système PrXML. Les résultats montrent que le temps d'exécution de ces opérations est souvent plus rapide sur notre système et, au pire des cas, comparable à celui obtenu sur Git ou Subversion. En particulier, la figure B.5 montre clairement que l'opération de commit est plus rapide sur notre système. Les mesures effectuées sur PrXML tiennent, bien évidemment, compte du temps de calcul additionnel des scripts de mises à jour ($deltas$) dans Git et Subversion.

8. http://svnkit.com/
9. http://www.eclipse.org/jgit/

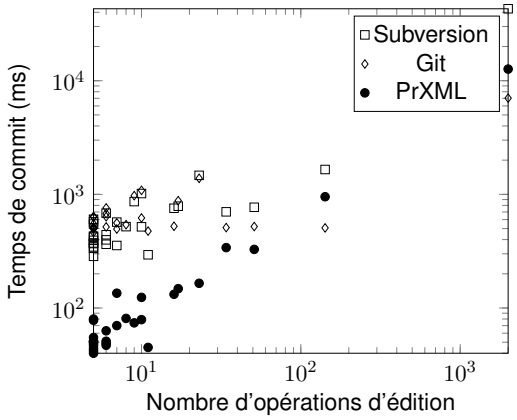


FIGURE B.6 – coût du commit en fonction du nombre d'opérations de mise à jour (scripts $\geq$ 5 opérations)

Une analyse en profondeur des résultats montre que le temps de commit est fonction de la taille du script d'édition associé à chaque version (voir figure B.6), ce qui confirme la proposition B.4.3. Cependant, PrXML reste efficace, à l'exception de quelques commits qui correspondent à des versions obtenues à la suite d'un grand nombre de mises à jour (plus d'une centaine d'opérations de mise à jour). Dans notre modèle, la création de versions se traduit par la mise à jour de l'arbre XML probabiliste représentant le document multi-version. Le script d'édition associé à chaque nouvelle version est transposé sur l'arbre XML probabiliste, ce qui rend le coût du commit dépendant du nombre d'opérations de mises à jour associées à chaque version. En revanche, Git hache les fichiers et stocke la valeur de hachage (SHA-1) obtenue, indexée par le nom du répertoire ou du fichier. Le système Subversion trace dans un fichier log les mises à jour associées aux chemins concernés dans l'arborescence du système de fichiers. Ainsi, l'insertion d'un sous-arbre (ensemble de fichiers organisés en répertoires et sous-répertoires) dans le système de fichiers peut se traduire par une simple opération dans Git et Subversion, alors qu'elle peut nécessiter une série d'insertions de nœuds dans notre modèle.

La figure B.7 montre les résultats obtenus pour la mesure du temps d'exécution de l'opération de checkout de versions sur Git, Subversion et PrXML. Nous nous intéressons en particulier aux versions dérivées de façon séquentielle les unes des autres. Dans ce cas, le calcul de la version numéro n se traduit souvent par une cascade de calculs intermédiaires sur les versions précédentes, ce qui peut alourdir le temps d'exécution du checkout. Dans la figure B.7, le numéro de version est précisé sur l'axe des abscisses

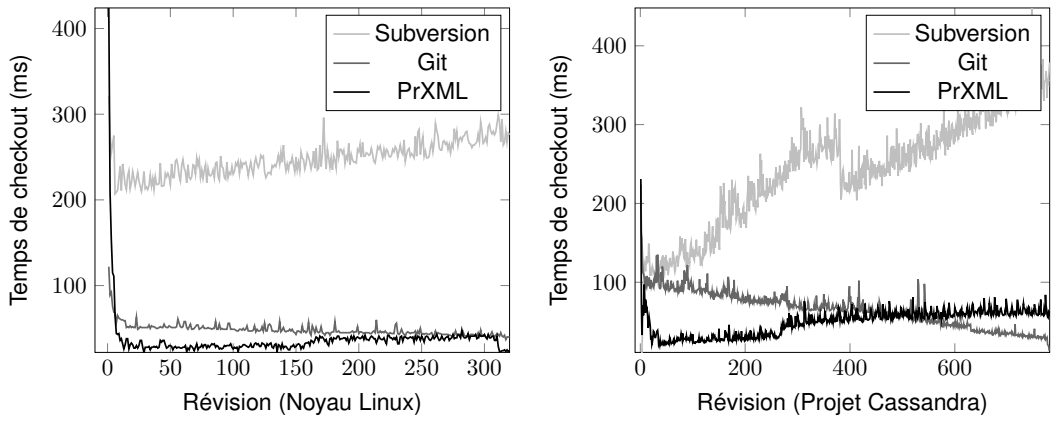


FIGURE B.7 – coût du checkout (restitution de version)

et le coût (en ms) du checkout sur l'axe des ordonnées. Les résultats montrent que le checkout est plus rapide sur notre système que sur Subversion, pour les deux jeux de données considérés (noyau Linux et Cassandra). Par rapport à Git, notre système a de meilleures performances pour les premières versions créées (celles précédées par un faible nombre de versions dans la hiérarchie de dérivation), alors qu'il devient moins performant sur les dernières versions qui découlent d'une série importante de versions successives. Ceci vaut pour les tests effectués sur les deux jeux de données. Il faut préciser que certains systèmes de gestion de versions utilisent des diffs réversibles [Rusu et al., 2005] pour accélérer le calcul des versions, lorsque celles-ci sont obtenues à la suite d'une longe série de dérivations linéaires.

B.6.2 Évaluation de la gestion des incertitudes : capacités de filtrage des données

L'évaluation de l'incertitude et le filtrage automatique des données non fiables sont deux défis importants pour les systèmes d'édition collaborative à large échelle. L'évaluation de l'incertitude est utile car les documents partagés sont produits par différents contributeurs, avec différents niveaux d'expertises et de fiabilité. Cette fiabilité peut être estimée de différentes manières, en s'appuyant par exemple sur les indicateurs de confiance ou la réputation de chaque contributeur (voir [Adler and de Alfaro, 2007, Maniu et al., 2011b, Maniu et al., 2011a]).

Dans le cas de plates-formes comme Wikipedia, notre approche peut permettre une gestion automatique des conflits entre contributeurs. Étant donné le nombre élevé de ces contributeurs et la fréquence des conflits, sur certains sujets sensibles, une

gestion automatique des conflits est très utile. Notre approche permet de filtrer de façon transparente, et sans blocage des pages conflictuelles, les spams et les contributions d'utilisateurs peu fiables ou malveillants. Dans PrXML, des variables booléennes sont associées aux sources des données (les contributeurs dans le cas d'une plate-forme comme Wikipédia). Le fait d'affecter la valeur false à une variable permet de filtrer les données produites par la source correspondante. Ceci peut être fait de façon automatique, lorsqu'un acte de vandalisme est détecté, ou lors de l'accès aux données par un utilisateur. Dans ce derniers cas, le système peut adapter le contenu en fonction des préférences de l'utilisateur et de son opinion (ou confiance) vis à vis des différents contributeurs. Nous avons montré dans [Abdessalem et al., 2011] une application de ces possibilités d'interaction et de filtrage sur des contenus de la plate-forme Wikipedia. Une page Wikipedia est vue comme une fusion d'un ensemble de contributions incertaines et n'est plus limitée à la dernière version valide. Le degré d'incertitude associé à chaque partie d'une page (une section par exemple) est déduit de l'incertitude associée à chacune des révisions qui ont affecté cette partie. L'utilisateur peut ainsi isoler le contenu correspondant à une révision donnée, éliminer les parties introduites par une révision ou un contributeur, ou limiter le contenu de la page aux seules parties introduites par quelques révisions ou quelques contributeurs de confiance.

Nous avons aussi testé la capacité de notre approche à gérer les actes de vandalisme, en prenant comme exemple les pages les plus vandalisées dans Wikipedia (voir most vandalized Wikipedia pages [10]). Nous avons réussi à reconstituer le contenu des pages dans l'état où il aurait du se trouver si les actes de vandalisme n'avaient pas été enlevés. Nous avons vu qu'il est facile avec notre modèle d'avoir la vision souhaitée (avec ou sans vandalisme) d'une page, en jouant sur les valeurs de vérité attribuées aux variables booléennes. Notons que Wikipedia élimine systématiquement les versions vandalisées de ses pages, alors qu'il serait utile, pour différentes raisons, de les laisser accessibles aux utilisateurs qui le souhaitent. Nous avons pu détecter les actes de vandalisme sur les pages choisies aussi bien que peuvent le faire les robots de Wikipedia, sans élimination des versions vandalisées et en laissant le choix aux utilisateurs de tenir en compte ou pas ces versions.

B.7 Conclusion

Nous avons présenté dans ce résumé l'une des principales contribution de cette thèse, à savoir, un modèle de gestion de versions de documents XML spécialement

10. http://en.wikipedia.org/wiki/Wikipedia:Most_vandalized_pages

adapté aux environnements d'édition collaborative à large échelle. Ce travail fait partie des premières applications concrètes des travaux théoriques sur le XML probabiliste [Nierman and Jagadish, 2002, Van Keulen et al., 2005, Abiteboul et al., 2009, Kimelfeld et al., 2009, Kimelfeld and Sagiv, 2008, Kharlamov et al., 2010, Kimelfeld and Senellart, 2013]. Notre modèle a été implanté et évalué sur des jeux de données réels. Nous présentons dans ce résumé les résultats obtenus qui montrent l'efficacité de notre approche. Nous avons également rappelé les avantages de notre modèle et les possibilités inédites de filtrage sur les données incertaines qu'il permet de réaliser. Toutefois, une utilisation de notre modèle dans un environnement réel, combiné avec un algorithme évaluant la réputation des participants et la possibilité de connaître les préférences des lecteurs, devrait permettre d'avoir un aperçu plus détaillé des avantages introduits pour les aspects probabilistes de notre système.

Self References

[Abdessalem et al., 2011] Talel Abdessalem, M. Lamine Ba, and Pierre Senellart. Probabilistic XML Merging Tool. *In Proc. EDBT*, 2011. (Demonstration).

[Ba et al., 2011] M. Lamine Ba, Talel Abdessalem, and Pierre Senellart. Towards a Version Control Model with Uncertain Data. *In Proc. PIKM*, 2011.

[Ba et al., 2013a] M. Lamine Ba, Talel Abdessalem, and Pierre Senellart. Uncertain Version Control in Open Collaborative Editing of Tree-Structured Documents. *In Proc. DocEng*, 2013. Similar version presented at BDA 2013.

[Ba et al., 2013b] M. Lamine Ba, Talel Abdessalem, and Pierre Senellart. Merging Uncertain Multi-Version XML Documents. *In Proc. DChanges*, 2013.

[Ba et al., 2014a] M. Lamine Ba, Sebastien Montenez, Ruiming Tang, and Talel Abdessalem. Integration of Web sources under uncertainty and dependencies using probabilistic XML. *In Proc. Uncrowd*, 2014.

[Ba et al., 2014b] M. Lamine Ba, Sebastien Montenez, Talel Abdessalem, and Pierre Senellart. Monitoring moving objects using uncertain Web data. *In Proc. SIGSPATIAL*, 2014. Preliminary version presented at BDA 2014.

[Ba et al., 2014c] M. Lamine Ba, Talel Abdessalem, and Pierre Senellart. Gestion de versions de documents XML incertains. *Ingénerie des Systèmes d'Information*, vol. 19, n°4, 2014.

[Amarilli et al., 2013] Antoine Amarilli, M. Lamine Ba, Daniel Deutch, Pierre Senellart. Provenance for Nondeterministic Order-Aware Queries. *Preliminary Version*, 2013.

[Amarilli et al., 2014] Antoine Amarilli, M. Lamine Ba, Daniel Deutch, Pierre Senellart. Querying Order-Incomplete Data. *Submitted for publication to ACM PODS*, 2014.

[Tang et al., 2014a] Ruiming Tang, Dongxu Shao, M. Lamine Ba, Huayu Wu. Conditioning Probabilistic Relational Data with Referential Constraints. *In Proc. Uncrowd*, 2014.

[Tang et al., 2014b] Ruiming Tang, Dongxu Shao, M. Lamine Ba, Pierre Senellart, Stéphane Bressan. A Framework for Conditioning Probabilistic XML Data. *Submitted for publication to ACM TOIT*, 2014.

External References

[Abiteboul et al., 2009] Abiteboul, S., Kimelfeld, B., Sagiv, Y., and Senellart, P. (2009). On the expressiveness of probabilistic XML models. *VLDB Journal*.

[Abiteboul et al., 2012] Abiteboul, S., Manolescu, I., Rousset, M.-C., Rigaux, P., and Senellart, P. (2012). *Web Data Management*. Cambridge University Press.

[Adler and de Alfaro, 2007] Adler, B. T. and de Alfaro, L. (2007). A content-driven reputation system for the Wikipedia. In *Proc. WWW*, Banff, Alberta, Canada.

[Agrawal et al., 2010] Agrawal, P., Sarma, A. D., Ullman, J., and Widom, J. (2010). Foundations of uncertain-data integration. *VLDB Endow.*

[Al-Khudair et al., 2001] Al-Khudair, A., Gray, W. A., and Miles, J. C. (2001). Dynamic evolution and consistency of collaborative configurations in object-oriented databases. In *Proc. TOOLS*, Santa Barbara, CA, USA.

[Altmanninger et al., 2009] Altmanninger, K., Seidl, M., and Wimmer, M. (2009). A survey on model versioning approaches. *IJWIS*.

[Ayat et al., 2012] Ayat, N., Afsarmanesh, H., Akbarinia, R., and Valduriez, P. (2012). An uncertain data integration system. In *On the Move to Meaningful Internet Systems*. Springer Berlin Heidelberg.

[Bleiholder et al., 2007] Bleiholder, J., Draba, K., and Naumann, F. (2007). FuSem: exploring different semantics of data fusion. In *Proc. VLDB*, Vienna, Austria.

[Borgolte et al., 2014] Borgolte, K., Kruegel, C., and Vigna, G. (2014). Relevant change detection: A framework for the precise extraction of modified and novel web-based content as a filtering technique for analysis engines. In *Proc. WWW*, Seoul, Korea.

[Cellary and Jomier, 1990] Cellary, W. and Jomier, G. (1990). Consistency of versions in object-oriented databases. In *Proc. VLDB*, Brisbane, Queensland, Australia.

[Chacon, 2009] Chacon, S. (2009). *Pro Git*. Apress.

[Cobéna and Abdessalem, 2009] Cobéna, G. and Abdessalem, T. (2009). A comparative study of XML change detection algorithms. In *Services and Business Computing Solutions with XML: Applications for Quality Management and Best Processes*. IGI Global.

[Cobéna et al., 2002] Cobéna, G., Abiteboul, S., and Marian, A. (2002). Detecting Changes in XML Documents. In *Proc. ICDE*, San Jose, California, USA.

[Conradi and Westfechtel, 1998] Conradi, R. and Westfechtel, B. (1998). Version models for software configuration management. *ACM Comput. Surv.*

[Das Sarma et al., 2008] Das Sarma, A., Dong, X., and Halevy, A. (2008). Bootstrapping pay-as-you-go data integration systems. In *Proc. SIGMOD*, Vancouver, BC, Canada.

[David, 1994] David, G. D. (1994). Palimpsest: A data model for revision control. In *Proc. Collaborative Hypermedia Systems*.

[de Keijzer and van Keulen, 2008] de Keijzer, A. and van Keulen, M. (2008). IMPrECISE: Good-is-good-enough data integration. In *Proc. ICDE*, Cancún, México.

[De La Calzada and Dekhtyar, 2010] De La Calzada, G. and Dekhtyar, A. (2010). On measuring the quality of Wikipedia articles. In *Proc. WICOW*, Raleigh, North Carolina, USA.

[De Vries and Van Someren, 2012] De Vries, G. K. D. and Van Someren, M. (2012). Machine Learning for Vessel Trajectories Using Compression, Alignments and Domain Knowledge. *Expert Syst. Appl.*

[Dimond et al., 2013] Dimond, M., Smith, G., and Goulding, J. (2013). Improving route prediction through user journey detection. In *Proc. SIGSPATIAL*, Orlando, Florida, USA.

[Dong et al., 2007] Dong, X., Halevy, A. Y., and Yu, C. (2007). Data Integration with Uncertainty. In *Proc. VLDB*, Vienna, Austria.

[Dong et al., 2010] Dong, X. L., Berti-Equille, L., Hu, Y., and Srivastava, D. (2010). Global detection of complex copying relationships between sources. *VLDB Endow.*

[Dong et al., 2009a] Dong, X. L., Berti-Equille, L., and Srivastava, D. (2009a). Integrating conflicting data: the role of source dependence. *VLDB Endow.*

[Dong et al., 2009b] Dong, X. L., Berti-Equille, L., and Srivastava, D. (2009b). Truth discovery and copying detection in a dynamic world. *VLDB Endow.*

[Dong and Naumann, 2009] Dong, X. L. and Naumann, F. (2009). Data Fusion: Resolving Data Conflicts for Integration. *VLDB Endow.*

[Dong et al., 2012] Dong, X. L., Saha, B., and Srivastava, D. (2012). Less is more: Selecting sources wisely for integration. *VLDB Endow.*

[Estublier, 2000] Estublier, J. (2000). Software configuration management: A Roadmap. In *Proc. ICSE*, Limerick, Ireland.

[Galland et al., 2010] Galland, A., Abiteboul, S., Marian, A., and Senellart, P. (2010). Corroborating information from disagreeing views. In *Proc. WSDM*, New York, USA.

[Geiger and Halfaker, 2013] Geiger, R. S. and Halfaker, A. (2013). When the Levee Breaks: Without Bots, What Happens to Wikipedia's Quality Control Processes? In *Proc. WikiSym*, Hong Kong, China.

[Gouriten and Senellart, 2012] Gouriten, G. and Senellart, P. (2012). API Blender: A uniform interface to social platform APIs. In *Proc. WWW*, Lyon, France.

[Graham et al., 1994] Graham, R. L., Knuth, D. E., and Patashnik, O. (1994). *Concrete Mathematics: A Foundation for Computer Science*. Addison-Wesley Longman Publishing Co., Inc.

[Halfaker et al., 2013] Halfaker, A., Geiger, R. S., Morgan, J., and Riedl, J. T. (2013). The rise and decline of an open collaboration system: How Wikipedia's reaction to sudden popularity is causing its decline. *American Behavioral Scientist*.

[Kanza et al., 2014] Kanza, Y., Kravi, E., and Motchan, U. (2014). City Nexus: Discovering Pairs of Jointly-Visited Locations Based on Geo-Tagged Posts in Social Networks. In *Proc. SIGSPATIAL*, Dallas, USA.

[Khan et al., 2002] Khan, L., Wang, L., and Rao, Y. (2002). Change detection of XML documents using signatures. In *Real World RDF and Semantic Web Applications*, Honolulu, Hawai.

[Khanna et al., 2007] Khanna, S., Kunal, K., and Pierce, B. C. (2007). A formal investigation of Diff3. In *Proc. FSTTCS*, New Delhi, India.

[Kharlamov et al., 2010] Kharlamov, E., Nutt, W., and Senellart, P. (2010). Updating Probabilistic XML. In *Proc. Updates in XML*, Lausanne, Switzerland.

[Kimelfeld et al., 2009] Kimelfeld, B., Kosharovsky, Y., and Sagiv, Y. (2009). Query evaluation over probabilistic XML. *VLDB Journal*.

[Kimelfeld and Sagiv, 2008] Kimelfeld, B. and Sagiv, Y. (2008). Modeling and querying probabilistic XML data. *SIGMOD Rec.*

[Kimelfeld and Senellart, 2013] Kimelfeld, B. and Senellart, P. (2013). Probabilistic XML: Models and complexity. In *Advances in Probabilistic Databases for Uncertain Information Management*, Studies in Fuzziness and Soft Computing. Springer-Verlag.

[Koc and Tansel, 2011] Koc, A. and Tansel, A. U. (2011). A survey of version control systems. In *Proc. ICEME*, Orlando, Florida USA.

[Kögel and Maximilian, 2008] Kögel and Maximilian (2008). Towards software configuration management for unified models. In *Proc. Comparison and versioning of software models*, Leipzig, Germany.

[Kumawat et al., 2010] Kumawat, S., Scholar, M. T., and Khunteta, A. (2010). A Survey on Operational Transformation Algorithms: Challenges, Issues and Achievements. *Computer Applications*.

[Lenzerini, 2002] Lenzerini, M. (2002). Data integration: a theoretical perspective. In *Proc. SIGMOD*, Madison, Wisconsin.

[Li et al., 2015] Li, Q., Li, Y., Gao, J., Su, L., Zhao, B., Demirbas, M., Fan, W., and Han, J. (2015). A Confidence-Aware Approach for Truth Discovery on Long-Tail Data. In *Proc. VLDB*, Kohala Coast, HI.

[Li et al., 2014] Li, Q., Li, Y., Gao, J., Zhao, B., Fan, W., and Han, J. (2014). Resolving Conflicts in Heterogeneous Data by Truth Discovery and Source Reliability Estimation. In *Proc. SIGMOD*, Snowbird, Utah, USA.

[Li et al., 2012] Li, X., Dong, X. L., Lyons, K., Meng, W., and Srivastava, D. (2012). Truth Finding on the Deep Web: Is the Problem Solved? *VLDB Endow.*

[Li et al., 2010] Li, Z., Ding, B., Han, J., Kays, R., and Nye, P. (2010). Mining Periodic Behaviors for Moving Objects. In *Proc. KDD*, Washington, DC, USA.

[Lindholm et al., 2006] Lindholm, T., Kangasharju, J., and Tarkoma, S. (2006). Fast and simple XML tree differencing by sequence alignment. In *Proc. DocEng*, Amsterdam, The Netherlands.

[Liu et al., 2013] Liu, J., Ma, Z., and Yan, L. (2013). Querying and ranking incomplete twigs in probabilistic XML. *World Wide Web*.

[Liu et al., 2014] Liu, M., Fu, K., Lu, C.-T., Chen, G., and Wang, H. (2014). A Search and Summary Application for Traffic Events Detection Based on Twitter Data. In *Proc. SIGSPATIAL*, Dallas, USA.

[Ma et al., 2010] Ma, J., Liu, W., Hunter, A., and Zhang, W. (2010). An XML based framework for merging incomplete and inconsistent statistical information from clinical trials. In *Software Computing in XML Data Management*. Springer-Verlag.

[Magnani and Montesi, 2010] Magnani, M. and Montesi, D. (2010). A survey on uncertainty management in data integration. *J. Data and Information Quality*.

[Maniu et al., 2011a] Maniu, S., Cautis, B., and Abdessalem, T. (2011a). Building a signed network from interactions in Wikipedia. In *Proc. DBSocial*, Athens, Greece.

[Maniu et al., 2011b] Maniu, S., Cautis, B., and Abdessalem, T. (2011b). Casting a web of trust over Wikipedia: an interaction-based approach. In *Proc. WWW (Companion Volume)*, Hyderabad, India.

[Mehdi et al., 2014] Mehdi, A.-N., Urso, P., Balegas, V., and Perguiça, N. (2014). Merging OT and CRDT Algorithms. In *Proc. PaPEC*, Amsterdam, The Netherlands.

[Mitchell, 1979] Mitchell, T. M. (1979). *Version Spaces: An Approach to Concept Learning*. PhD thesis, Stanford, CA, USA.

[Morris, 2007] Morris, J. C. (2007). DistriWiki: A Distributed peer-to-peer wiki network. In *Proc. WikiSym*, Montréal, Québec, Canada.

[Myers, 1986] Myers, E. W. (1986). An O(ND) difference algorithm and its variations. *Algorithmica*.

[Nierman and Jagadish, 2002] Nierman, A. and Jagadish, H. V. (2002). ProTDB: probabilistic data in XML. In *Proc. VLDB*, Hong Kong, China.

[Osman, 2013] Osman, K. (2013). The Role of Conflict in Determining Consensus on Quality in Wikipedia Articles. In *Proc. WikiSym*, Hong Kong, China.

[Pandey and Munson, 2013] Pandey, M. and Munson, E. V. (2013). Version aware libreoffice documents. In *Proc. DocEng*, Florence, Italy.

[Peters, 2005] Peters, L. (2005). Change detection in XML trees: a survey. In *Proc. TSIT*.

[Pianese et al., 2013] Pianese, F., An, X., Kawsar, F., and Ishizuka, H. (2013). Discovering and predicting user routines by differential analysis of social network traces. In *Proc. WoWMoM*, Madrid, Spain.

[Pilato, 2004] Pilato, M. (2004). *Version Control With Subversion*. O'Reilly & Associates, Inc.

[Pochampally et al., 2014] Pochampally, R., Das Sarma, A., Dong, X. L., Meliou, A., and Srivastava, D. (2014). Fusing data with correlations. In *Proc. SIGMOD*, Snowbird, Utah, USA.

[Reichenberger and Kratky, 2009] Reichenberger, C. and Kratky, S. (2009). Object-oriented version control: Use inheritance instead of branches.

[Robin, 2002] Robin, L. F. (2002). Merging XML files: A new approach providing intelligent merge of XML data sets. In *Proc. XML Europe*, Barcelona, Spain.

[Rönnau and Borghoff, 2009] Rönnau, S. and Borghoff, U. (2009). Versioning XML-based office documents. *Multimedia Tools and Applications*.

[Rönnau and Borghoff, 2012] Rönnau, S. and Borghoff, U. (2012). XCC: change control of XML documents. *Computer Science - Research and Development*.

[Rusu et al., 2005] Rusu, L. I., Rahayu, W., and Taniar, D. (2005). Maintaining versions of dynamic XML documents. In *Proc. WISE*, New York, USA.

[Sabel, 2007] Sabel, M. (2007). Structuring wiki revision history. In *Proc. WikiSym*, Montréal, Québec, Canada.

[Simonite, 2013] Simonite, T. (2013). The decline of Wikipedia. *MIT Technology Review Magazine*.

[Sinnott, 1984] Sinnott, R. W. (1984). Virtues of the Haversine. *Sky and Telescope*.

[Suciu et al., 2011] Suciu, D., Olteanu, D., Ré, C., and Koch, C. (2011). *Probabilistic Databases*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers.

[Suzuki, 2002] Suzuki, N. (2002). A Structural Merging Algorithm for XML Documents. In *Proc. ICWI*, Lisbon, Portugal.

[Taka et al., 2013] Taka, H., Shibata, D., Wada, M., Matsumoto, H., and Hatanaka, K. (2013). Construction of a marine traffic monitoring system around the world. In *Proc. OCEANS*, San Diego, USA.

[Thao and Munson, 2014] Thao, C. and Munson, E. (2014). Using versioned trees, change detection and node identity for three-way XML merging. *Computer Science - Research and Development*.

[Thao and Munson, 2011] Thao, C. and Munson, E. V. (2011). Version-aware XML documents. In *Proc. DocEng*, Mountain View, California, USA.

[van Keulen and de Keijzer, 2009] van Keulen, M. and de Keijzer, A. (2009). Qualitative effects of knowledge rules and user feedback in probabilistic data integration. *VLDB Journal*.

[Van Keulen et al., 2005] Van Keulen, M., de Keijzer, A., and Alink, W. (2005). A Probabilistic XML Approach to Data Integration. In *Proc. ICDE*, Tokyo, Japan.

[Voss, 2005] Voss, J. (2005). Measuring Wikipedia. In *Proc. ISSI*, Stockholm, Sweden.

[Waguih and Berti-Equille, 2014] Waguih, D. A. and Berti-Equille, L. (2014). Truth Discovery Algorithms: An Experimental Evaluation. *CoRR*.

[Wang et al., 2003] Wang, Y., DeWitt, D. J., and Cai, J.-Y. (2003). X-Diff: An Effective Change Detection Algorithm for XML Documents. In *Proc. ICDE*, Bangalore, India.

[Weiss et al., 2010] Weiss, S., Urso, P., and Molli, P. (2010). Logoot-Undo: Distributed Collaborative Editing System on P2P Networks. *IEEE Transactions on Parallel & Distributed Systems*.

[Wolfson, 2002] Wolfson, O. (2002). Moving Objects Information Management: The Database Challenge. In *Proc. NGITS*, Caesarea, Israel.

[Yin et al., 2008] Yin, X., Han, J., and Yu, P. S. (2008). Truth discovery with multiple conflicting information providers on the web. *IEEE Trans. on Knowl. and Data Eng.*

[Yuan et al., 2014] Yuan, H., Qian, Y., Yang, R., and Ren, M. (2014). Human mobility discovering and movement intention detection with GPS trajectories. *Decision Support Systems*.

[Zhang and Jagadish, 2013] Zhang, J. and Jagadish, H. V. (2013). Revision provenance in text documents of asynchronous collaboration. In *Proc. ICDE*, Brisbane, Australia.

[Zhao et al., 2012] Zhao, B., Rubinstein, B. I. P., Gemmell, J., and Han, J. (2012). A bayesian approach to discovering truth from conflicting sources for data integration. *VLDB Endow.*

[Zhao et al., 2014] Zhao, Z., Cheng, J., and Ng, W. (2014). Truth Discovery in Data Streams: A Single-Pass Probabilistic Approach. In *Proc. CIKM*, Shangai, China.