Flexible and scalable methods for formal concept analysis in distributed architectures

Nicolas Leutwyler

IAEM Lorraine Doctoral School

Flexible and Scalable methods for Formal Concept Analysis in Distributed Architectures

THESIS IN COTUTELLE

publicly presented and defended on October 7th

for obtaining the

PhD of the University of Lorraine
(Automatic control, Signal and Image Processing, and Computer Engineering)

PhD of the National University of Quilmes
(Science and Technology)

by

Nicolás Leutwyler

Composition of the jury

President :	Prof. Virginie GOEPP	from INSA Strasbourg
Reviewers :	MdC. HDR. Florence LE BER	from ENGEES Strasbourg
	Prof. Néjib MOALLA	from Lyon 2 University
Examiner :	Prof. Gabriela AREVALO	from National University of Quilmes
Invitees :	Cedric Marcone	from Valraiso
	Jérémie Noyrey	from ESF
Supervisors :	MdC HDR. Mario LEZOCHE	from University of Lorraine
	Prof. Hervé PANETTO	(co-supervisor) from University of Lorraine
	Prof. Diego TORRES	(cotutelle co-supervisor) from National University of Quilmes

Centre de Recherche en Automatique de Nancy (CRAN) — CNRS UMR 7039
Centro LIFIA, Facultad de Informática, Universidad Nacional de La Plata

Acknowledgments

It is only on rare occasions, that we've got the possibility to thank those who have helped us become who we are now. I want to leverage this opportunity to express my heartfelt gratitude to everyone who contributed to this milestone, both professionally, and personally.

First and foremost, I wish to acknowledge the great value the comments of the jury had in improving the overall quality of this manuscript's final version. You all have my gratitude for being honest and just when providing your feedback.

Secondly, I would like to thank my thesis supervisors, who made their best to guide and contribute to the final quality of this work. In particular, I am very grateful with the work Hervé did in order to make the starting of the Ph.D., project possible. Thanks to Mario that trusted me from the moment *zero*, and who always encouraged me to follow my instincts when pursuing my, sometimes crazy, research ideas. And also, I am deeply thankful for the invaluable guidance of Diego, who never hesitated in challenging my ideas, helping me grow unfathomably.

I would like to express my thanks as well to the scientific staff of the University of Lorraine, National University of Quilmes, and National University of La Plata, who provided a stimulating and supportive academic environment. Their assistance in the scientific exchanges we made significantly enriched my overall experience.

Stories, usually, wouldn't be as fun or relatable if it wasn't because of the challenges the characters face, or if they didn't grow thanks to them. And most of the time, it is with the help of their brave, wise, and loyal friends, that they are capable of overcoming these challenges. This story is no different. Nothing would've been the way it is now, if it wasn't for the advices, countless beers, and the unconditional friendship of Chiara and Mariano, with whom I would feel forever in debt. Days would've been tinted in gray if I haven't had the privilege of meeting the amazing group of "Vamos a la playa" where we accompanied each other and never hesitated in giving a helping hand. Thanks to all of you guys (Ale, Belén, Clémence, Elena, Fran, Gabriel, Mario, Mica, Nico Z, and Ola, in no particular order). Regarding learning French and integrating into the French society, I reckon it would've been impossible, or at least way less fun, to do so without the -somehow unconventional- way of teaching of Audrey and Laura. Thanks to both of you guys. For that, and for being there in very tough moments. I wanted to thank Lula as well, who gave me her unwavering support when I most desperately needed it. And finally, to my *family*, my brothers by choice Fede, Jesús, Lucas, and Nico H, to my mother Nelly, to my father Miguel, and to my brother Hernan, without whom nothing of this would've made any sense.

To all of you, you have my sincere gratitude. But don't worry, this journey doesn't end here. Many more challenges, and adventures are to come, and I hope, if it's not too much to ask, that I'd be able to count with at least a portion of the amazing company you provide.

"Gone were the stylized forms used at court duels. Gone were the perfect angles and distances of sparring. And yet, it was a dazzling, daring, and dashing display, that would have won the applause of even the most jaded audience." - by Christopher Paolini, in Murtagh, page 551.

Abstract

This thesis explores adapting Formal Concept Analysis (FCA) to tackle challenges in distributed and dynamic data environments. This research addresses the need for efficient data mining techniques in Industry 4.0 and the Internet of Things (IoT), where data is vast and continuous. The primary goals include developing scalable, flexible methods to prevent the loss of significant concepts during concept lattice construction, proposing incremental algorithms for real-time data streams, and demonstrating practical applications of these methods. The thesis starts with a comprehensive review of current knowledge representation, scalability, and FCA, emphasizing the challenges of large datasets and efficient mining techniques. A systematic literature review identifies gaps in existing FCA methods, particularly for distributed architectures, highlighting the need for new approaches to manage the exponential growth of formal concepts and dynamic data streams. Novel algorithms are introduced for merging lattices to recover lost concepts and incremental methods for constructing concept lattices. These algorithms, designed for both single-threaded and parallel execution, enhance scalability and flexibility. Practical use cases, such as an e-commerce recommendation system for the French Ski School, demonstrate the real-world benefits of the proposed methods. Empirical evaluations using synthetic and real-world datasets validate the performance, showing significant improvements in handling large, distributed data. The thesis concludes by summarizing the contributions and suggesting future research directions to further enhance FCA capabilities in distributed settings.

Keywords: Formal Concept Analysis, Distributed Architectures, Conceptual Structures Merge, Recommendation Systems.

Résumé

Cette thèse explore l'adaptation de l'Analyse Formelle de Concepts (AFC) pour relever les défis posés par les environnements de données distribués et dynamiques. Cette recherche répond à la nécessité de techniques de fouille de données efficaces dans le contexte de l'Industrie 4.0 et de l'Internet des Objets (IoT), où la génération de données est vaste et continue. Les objectifs principaux incluent le développement de méthodes scalables et flexibles pour éviter la perte de concepts significatifs lors de la construction de réticulé de concepts, la proposition d'algorithmes incrémentaux adaptés aux flux de données en temps réel, et la démonstration des applications pratiques de ces méthodes. La thèse commence par une revue exhaustive de l'état de l'art, définissant les concepts clés de la représentation des connaissances, de la scalabilité et de l'AFC, en soulignant les défis inhérents à la gestion de grands ensembles de données et la nécessité de techniques de fouille de données efficaces. Une revue systématique de la littérature identifie les lacunes des méthodes d'AFC existantes, particulièrement celles applicables aux architectures distribuées, mettant en évidence la nécessité de nouvelles approches pour gérer la croissance exponentielle des concepts formels et la nature dynamique des flux de données. La thèse introduit des algorithmes novateurs pour la fusion des réticulés afin de récupérer des concepts potentiellement perdus et des méthodes incrémentales pour la construction de réticulé de concepts.

Ces algorithmes, conçus pour des exécutions mono-thread et parallélisées, améliorent leur scalabilité et flexibilité. Des cas pratiques, comme un système de recommandation pour l'École de Ski Française, démontrent les avantages réels des méthodes proposées. Des évaluations empiriques utilisant des ensembles de données synthétiques et réels valident la performance de ces méthodes, montrant des améliorations significatives dans la gestion de données larges et distribuées. La thèse conclut en résumant les contributions et en suggérant des directions pour de futures recherches afin de renforcer les capacités de l'AFC dans les environnements distribués.

Mots-clés: Analyse Formelle des Concepts, Architectures Distribuées, Combination de structures conceptuelles, Systemes de Recommandation.

Resumen

Esta tesis explora la adaptación del Análisis Formal de Conceptos (FCA, por sus siglas en inglés) para enfrentar desafíos en entornos de datos distribuidos y dinámicos. Esta investigación aborda la necesidad de técnicas eficientes de minería de datos en la Industria 4.0 y el Internet de las Cosas (IoT), donde los datos son vastos y continuos. Los objetivos principales incluyen desarrollar métodos escalables y flexibles para evitar la pérdida de conceptos significativos durante la construcción de reticulados de conceptos, proponer algoritmos incrementales para flujos de datos en tiempo real y demostrar aplicaciones prácticas de estos métodos. La tesis comienza con una revisión exhaustiva del conocimiento actual sobre representación del conocimiento, escalabilidad y FCA, enfatizando los desafíos de los grandes conjuntos de datos y las técnicas de minería de datos eficientes. Una revisión sistemática de la literatura identifica vacíos en los métodos FCA existentes, particularmente para arquitecturas distribuidas, destacando la necesidad de nuevos enfoques para gestionar el crecimiento exponencial de conceptos formales y flujos de datos dinámicos. Se introducen nuevos algoritmos para fusionar reticulados y recuperar conceptos perdidos, y métodos incrementales para construir reticulados de conceptos. Estos algoritmos, diseñados tanto para ejecución monohilo como paralela, mejoran la escalabilidad y la flexibilidad. Casos de uso prácticos, como un sistema de recomendación de comercio electrónico para la Escuela Francesa de Esquí, demuestran los beneficios del mundo real de los métodos propuestos. Las evaluaciones empíricas utilizando conjuntos de datos sintéticos y del mundo real validan el rendimiento, mostrando mejoras significativas en el manejo de datos grandes y distribuidos. La tesis concluye resumiendo las contribuciones y sugiriendo direcciones futuras de investigación para mejorar aún más las capacidades de FCA en entornos distribuidos.

Palabras clave: Análisis Formal de Conceptos, Arquitecturas Distribuidas, Combinación de estructuras conceptuales, Sistemas de Recomendación.

Contents

Chapter 1 Introduction **1**

1.1 Motivation . . . 1

1.1.1 Mining from Distributed Architectures . . . 2

1.1.2 Mining using Distributed Architectures . . . 2

1.1.3 Data Streams and Batch processing . . . 3

1.2 Scope and objectives of the thesis . . . 4

1.3 Results summary . . . 6

1.4 Outline of the thesis . . . 6

Chapter 2 State of the art **9**

2.1 Knowledge definition . . . 9

2.2 Knowledge Representation and Extraction . . . 11

2.3 Scalability, and Flexibility . . . 13

2.4 Formal Concept Analysis . . . 14

2.4.1 Basic definitions . . . 14

2.4.2 FCA in distributed architectures . . . 16

2.5 Batch and Incremental approaches in FCA . . . 18

2.6 Frequent Itemsets and Association Rules . . . 19

2.7 Finding evidence about the scientific gaps . . . 20

2.7.1 Literature Research Questions . . . 21

2.7.2 Keywords and search strings definition . . . 21

2.7.3 Database selection . . . 24

2.7.4 Study selection and quality assessment . . . 25

2.7.5 Data extraction . . . 26

2.7.6 Analysis and classification . . . 26

2.7.7 Quantitative results . . . 29

2.7.8 Analysis of correlations among categories with FCA . . . 37

2.7.9 Qualitative results . . . 40

Chapter 3 Recovering relevant lost concepts **45**
3.1 Introduction . . . 45
3.1.1 Motivation . . . 45
3.1.2 Preliminaries . . . 46
3.2 Approach to mitigate the loss of concepts . . . 49
3.3 Algorithms to merge concept lattices . . . 52
3.3.1 Using existing approaches . . . 52
3.3.2 Single-threaded merging algorithm . . . 53
3.3.3 Parallelization . . . 57
3.3.4 Implementation details and Complexity . . . 62
3.4 Conclusions . . . 64

Chapter 4 Arbitrarily distributed Formal Contexts incremental approach **67**
4.1 Introduction . . . 67
4.2 Motivation . . . 68
4.3 The algorithm of Goel and Chaudhary . . . 68
4.4 Why incremental . . . 70
4.4.1 AddIntent . . . 71
4.4.2 DeleteInstance . . . 74
4.4.3 Generalization . . . 75
4.5 AddPair . . . 75
4.6 Conclusions . . . 79

Chapter 5 Use cases **81**
5.1 ESF “mon-séjour-en-montagne” e-commerce application . . . 82
5.2 Recommendation needs . . . 83
5.2.1 Unavailable ski lessons . . . 83
5.2.2 Complementary products . . . 84
5.3 Recommending Ski Lessons from Historical Purchases . . . 85
5.3.1 Historical purchases: dataset . . . 86
5.3.2 Approach . . . 87
5.4 Recommending Complementary Products using FCA and Association Rules . 90
5.4.1 Carts: dataset . . . 90
5.4.2 Approach . . . 91
5.5 Conclusions . . . 92

Chapter 6 Evaluation **95**
6.1 Single threaded batch merge . . . 96
6.1.1 Implementation and setting for the experiments . . . 96
6.1.2 Experiments with real datasets . . . 96
6.1.3 Experiments with synthetic datasets . . . 101
6.1.4 Discussion . . . 102
6.2 Incremental minimal updates . . . 102
6.2.1 Comparison with AddIntent . . . 105
6.2.2 Comparison with Goel and Chaudhary's algorithm . . . 105
6.3 Distributed batch merge using Apache Spark . . . 109
6.3.1 Experiments on distributed clusters . . . 109
6.3.2 Remarks about the results . . . 109
6.4 Conclusions . . . 112

Chapter 7 Conclusions & Future Work **113**
7.1 Summary . . . 113
7.2 Perspectives and future directions . . . 115
7.3 Publication List . . . 116

Résumé étendu en français **117**

Resumen extendido en español **123**

Appendixs **129**

Appendix A List of Abbreviations **129**

Bibliography **131**

List of Figures

1.1 Data stream over time 4
1.2 Cyber Physical Systems distributed architecture conceptual data mining general schema, where (1) represents a flow of events growing *infinitely*, and (2), an output that in the worst case is *exponential* in relation to the input. 5

2.1 Data, information, knowledge, and wisdom (DIKW) pyramid. 10
2.2 Example of a concept lattice in a *reduced line diagram* where only *introduced* objects and attributes are shown. 16
2.3 Example of a sliding window of size $l - k$. 18
2.4 Venn diagram from a conceptual point of view, showing the relations between *data mining* (DM), *multi-relational data mining* (MRDM), *knowledge extraction* (KE), Knowledge Discovery on Databases (KDD), Concept Analysis (referring to FCA and its extensions, and abbreviated CA), *information retrieval* (IR), and *distributed architectures* (DA). 22
2.5 Number of articles included/excluded in each step of the selection process . . 26
2.6 Generic Data Mining process life cycle. 27
2.7 Articles per year 30
2.8 Articles per publication venue 30
2.9 Methods used in the articles classified in CA, MRDM and DA. 31
2.10 Heatmap showing the number of articles regarding the DM process life cycle presented in Figure 2.6. 31
2.11 Domains of articles classified by method category. 32
2.12 Taxonomy of the domains found in a granular way. 33
2.13 Input formats with more than 1 occurrence by method category. 33
2.14 Output formats with more than 1 occurrence by method category. 34
2.15 Relations between input and output formats. 34
2.16 Evaluation strategy by method in CA. 35
2.17 Evaluation strategy by method in MRDM. 35
2.18 Evaluation strategy by method in DA. 35
2.19 Clusters of articles by semantic interoperability between knowledge formats. 37
2.20 Venn diagram of association rules, showing the discussed ones in relation to the total. 38

3.1 Contiguous lattice snapshots merge. 47
3.2 Spaced lattice snapshots merge case. 47
3.3 Intersected snapshots merge case. 47
3.4 Temporal path after merging when g_{t_i} occurs for the first time between l and n 49

3.5 Temporal path after merging when g_{t_i} occurs before l . . . 49
3.6 General method for recovering potentially lost concepts on demand. . . . 50
3.7 Basic Map Reduce example. . . . 57
3.8 RDD example with 3 workers showing a possible distribution of the formal concepts coming from Table 2.2. . . . 59
3.9 Example of the IntentsDict structure to accomplish the look by intent in worst case scenario in $\mathcal{O}(k \max(|g'^m|))$ . . . 63

4.1 Concept categories during the an AddPair incremental step in a concept lattice. 77

5.1 Snapshot of a development tool for the MSEM e-commerce website. . . . 82
5.2 MSEM interaction with ESF centralized API in order to recover *available* lessons. . . . 83
5.3 Snapshot of some courses being listed for the user ready to be booked. . . . 84
5.4 Cross-selling coefficients visualization tool. . . . 85
5.5 MSEM system calling the API of the recommendation system when no lessons are available with the selected characteristics. . . . 88
5.6 Recommender system using merge of lattice snapshots strategy. . . . 89
5.7 Example of a smaller sliding windows being used, but being able to consider an older event nonetheless. . . . 93

6.1 $|\mathcal{L}| + |\mathcal{L}_s|$ vs minutes. $|G|$ ratio $1 : \frac{1}{2}$ . . . 97
6.2 $|\mathcal{L}_m|$ vs minutes. $|G|$ ratio $1 : \frac{1}{2}$ . . . 99
6.3 $|\mathcal{L}| + |\mathcal{L}_s|$ vs minutes. $|G|$ ratio $1 : 1$ . . . 99
6.4 $|\mathcal{L}_m|$ vs minutes. $|G|$ ratio $1 : 1$ . . . 100
6.5 $|\mathcal{L}_1| + |\mathcal{L}_2|$ vs time in minutes. Experiments where $|G_1| = 2|G_2|$. . . . 101
6.6 Zoomed version of Figure 6.5 in seconds and considering only the first 4 experiments. . . . 101
6.7 $|G_1| + |G_2|$ vs time in minutes. Experiments where $|G_1| = 2|G_2|$. . . . 102
6.8 Zoomed version of Figure 6.7 in seconds and considering only the first 4 experiments. . . . 102
6.9 $|\mathcal{L}_1| + |\mathcal{L}_2|$ vs time in minutes. Experiments where $|G_1| = |G_2|$ for each pair of experiments. . . . 103
6.10 Zoomed version of Figure 6.9 in seconds and considering only the first 4 experiments. . . . 103
6.11 $|G_1| + |G_2|$ vs time in minutes. Experiments where $|G_1| = |G_2|$. . . . 103
6.12 Zoomed version of Figure 6.11 in seconds and considering only the first 4 experiments. . . . 103
6.13 $|\mathcal{L}| + |\mathcal{L}_s|$ vs time in minutes with ratio $1 : \frac{1}{2}$ . . . 106
6.14 $|\mathcal{L}_m|$ vs time in minutes with ratio $1 : \frac{1}{2}$ . . . 106
6.15 $|\mathcal{L}| + |\mathcal{L}_s|$ vs time in minutes with ratio $1 : 1$ . . . 106
6.16 $|\mathcal{L}_m|$ vs time in minutes with ratio $1 : 1$ . . . 106
6.17 $|\mathcal{L}_m|$ vs time in seconds with ratio $1 : \frac{1}{2}$ with 2 workers. . . . 110
6.18 $|\mathcal{L}_m|$ vs time in seconds with ratio $1 : \frac{1}{2}$ with 3 workers. . . . 110
6.19 $|\mathcal{L}_m|$ vs time in seconds with ratio $1 : \frac{1}{2}$ with 4 workers. . . . 110
6.20 $|\mathcal{L}_m|$ vs time in seconds with ratio $1 : 1$ with 2 workers. . . . 111
6.21 $|\mathcal{L}_m|$ vs time in seconds with ratio $1 : 1$ with 3 workers. . . . 111
6.22 $|\mathcal{L}_m|$ vs time in seconds with ratio $1 : 1$ with 4 workers. . . . 111

List of Tables

2.1 People count per ski resort per hour. . . . 11
2.2 A Formal Context of ski lessons purchases. . . . 15
2.3 Query strings used for each search engine. . . . 24
2.4 Number of papers obtained on the queries execution on each database. . . . 24
2.5 Criteria for selecting papers. . . . 25
2.6 Articles by ID . . . 42
2.7 Data extraction fields divided in two subcategories: general and process. . . . 43
2.8 Subclassification of articles according to whether they are implemented *using* a DA system 1.1.2, or they mine *from* a DA system 1.1.1. . . . 44
2.9 *Problematic addressed in* multivalued field . . . 44

3.1 Some of the Apache Spark functions. . . . 58

5.1 Events of the data stream corresponding to purchases taking place in ESF external marketplaces. . . . 86
5.2 Events of the data stream corresponding to the MSEM marketplace. . . . 87
5.3 REST API for the recommendation system using the frequent itemsets notion. 89
5.4 Events of the data stream corresponding to the cross-selling recommendation system. . . . 91

6.1 Sizes for each experiment. . . . 98
6.2 Parameters and characteristics for each experiment. . . . 104
6.3 Parameters and characteristics for each experiment regarding Algorithm AddPair and Algorithm IncrementalMerge. . . . 107
6.4 Number of pairs for each experiment regarding Algorithm AddPair and Algorithm IncrementalMerge. . . . 108

1

Introduction

Contents

1.1 Motivation 1
1.1.1 Mining from Distributed Architectures 2
1.1.2 Mining using Distributed Architectures 2
1.1.3 Data Streams and Batch processing 3
1.2 Scope and objectives of the thesis 4
1.3 Results summary 6
1.4 Outline of the thesis 6

1.1 Motivation

All the way through modern history, data has been gathered and utilized to help society advance. However, data generation is unprecedented nowadays. According to a study made by the Software Alliance[1] [16], already in 2016, around 90% of the world's data has been generated from the year 2014 to 2016. In addition, there has been a significant reduction in storage expenses, attributed to the emergence of new, arguably superior technologies tailored for this purpose. Even more so, the rate in which data generation grows is expected to continue its increasing tendency.

Consequently, a myriad of disciplines have emerged to aid researchers and practitioners to be able to gather meaningful data under this unprecedented growth. Among them, we can find *data mining* (DM), which is the process of discovering knowledge or patterns from massive amounts of data [54]; information retrieval (IR), which is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers) [91]; and knowledge extraction (KE) that is the creation of knowledge from structured (e.g., relational databases, extensible markup language (XML)) and unstructured (e.g., text, documents, images) sources [127].

Moreover, these methods play a crucial role in the Industry 4.0[2] (defined in [77]) by leveraging data from various sources to derive actionable insights. Some pivotal areas that

[1] https://www.bsa.org/about-bsa

[2] Also known as the Fourth Industrial Revolution, is characterized by the integration of digital technologies into industrial processes, leading to increased automation, connectivity, and data-driven decision-making.

require data mining methods in Industry 4.0 are for example *data acquisition and sensors*, *data integration and connectivity*, *big data analytics*, and *predictive maintenance*.

In summary, in order to derive actionable insights that drive operational efficiency, improve product quality, and enable informed decision-making across the entire manufacturing ecosystem, suitable KE methods for each of the common settings present in the Industry 4.0 are needed. Additionally, by harnessing the power of advanced analytics and digital technologies, organizations can unlock new opportunities for innovation and competitiveness.

Nevertheless, among other things, the growth of Internet of Things (IoT) over the last years raised new challenges related with the large amounts of heterogeneous data it produces. Accordingly, several methods for the extraction of knowledge (e.g., K-means, Formal Concept Analysis (FCA), Regression Trees, etc.) had to be adapted to the settings and different purposes brought by it. For example, in order to be able to process *larger* datasets, without compromising too much the time spent, some algorithms to compute the set of formal concepts[3] using distributed architectures (DA) have been proposed [141]. Additionally, other approaches have emerged to be able to process data that arrives *from* distributed architectures in an infinite fashion [27].

1.1.1 Mining from Distributed Architectures

One of the challenges that arises in the context of DAs is how to extract knowledge *from* them, considering its inherent characteristics such as:

(i) The amount of data to be mined is usually orders of magnitude larger than that of non-distributed (single-node) architectures.

(ii) Data formats do not always match those coming from single-node architectures.

In particular, item (i) implies that methods ought to be not only asymptotically scalable, but to provide as well a way of dealing with situations such as when it is not *even* possible to traverse the whole dataset efficiently, e.g., the dataset gets continuously updated at the same rate that can be traversed, thus, being essentially infinite, then the algorithm simply cannot wholly traverse it. Additionally, item (ii) refers to the fact that methods need to be able to mine from the usual formats used in DAs, e.g., data streams, and distributed file systems.

1.1.2 Mining using Distributed Architectures

One of the goals of using distributed architectures for data mining methods is to allow them to scale *horizontally*, i.e., adding computation power (usually called nodes) would result in better performance. However, some algorithms are inherently single-threaded and adapting them to leverage the advantages of DAs poses certain challenges regardless of where the data comes from. This problem is exacerbated by the fact that there are several distributed computation models to consider, and their differences make the transition from one model to another non-trivial.

For instance, let us consider the threaded computation model with shared memory [120], i.e., an algorithm is run by several threads that run in parallel and have access to a certain space of memory that is shared by all of them. In this model, typically, threads compute

[3] The core component of FCA.

on their own, and at a given point modify a certain part of the shared memory. To avoid concurrency problems, concepts such as semaphores should be used [31]. Additionally, in this model, each thread runs a *sequential* program nonetheless.

In contrast, let us consider the MapReduce model [30], where there are usually two types of nodes, *workers*, that are computers with the responsibility of processing a partition of the entire data, and *drivers*, being the computers that trigger the functions to process. In synthesis, the model simplifies the distributed computation by programming in terms of two type of functions: *map*, and *reduce*. Typically, *map* would perform simple transformation in chunks of data that are distributed across nodes in the cluster, while *reduce* would combine the results in a certain way. The computation is not only distributed, but also done in parallel, since multiple workers can process a function in their specific partition of the data *at the same time*.

Then, it is evident that the models have essential differences, e.g., the first one is limited to how many cores a single computer can have, while the other is not. Additionally, while most of the MapReduce implementations are Turing complete (i.e., all decidable problems can be programmed with them [117]), it is not clear nor trivial to make the translation from one model to the other.

What is more, it is strongly believed that not all problems are efficiently parallelizable [10], i.e., a computational task is said to have efficient parallel algorithms if inputs of size n can be solved using a parallel computer with $n^{\mathcal{O}(1)}$ processors and in time $\log^{\mathcal{O}(1)} n$. This comes from the definitions of the classes NC (i.e., Nick's class [10]) and P. As demonstrated in [10], a problem is efficiently parallelizable if and only if it is in NC. Then, a major open research question is whether $P = NC$ where P is the class of all polynomial languages (i.e., that can be decided in polynomial time). Moreover, it is known that if any language $L \in P-$complete, and $L \in NC$, then $P = NC$. However, the consensus seems to suggest that $P \neq NC$, or in other words, there are polynomial problems that cannot be efficiently parallelizable.

1.1.3 Data Streams and Batch processing

When it comes to the Internet of Things (IoT), DAs share certain patterns. One of these commonalities is the usage of several physical devices that extend beyond the typical computer networks. For instance, in a factory there are typically sensors that measure and publish data continuously about the tools' temperature, the speed of the mechanical tape, or measures of the product in each of the production stages. Additionally, these devices can be changed, updated, or removed from the network, making it for a highly dynamic environment.

In a context like this, one of the challenges of knowledge extraction, is the ability to gain insights hidden in the vast and *ever-growing* seas of data available that are generated by these architectures. What is more, there is the need of gaining knowledge on how these systems evolve over time, given their dynamic nature [24]. Consequently, an abstraction capable of representing this infinite source of data is used,

Definition. 1.1: Data Stream

A *countably infinite* sequence of elements, as defined in [92].

As depicted in Figure 1.1, a data stream can be seen as a sequence of elements, where

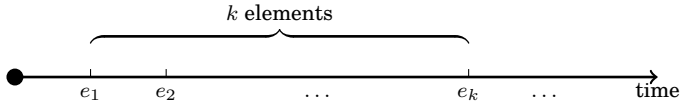

Figure 1.1: Data stream over time

their definition is left purposely loose. Nevertheless, elements are typically *tuples* representing either "atomic updates" of some state (i.e., the Δ or difference with the previous state), or directly an entire new state.

There are several technical challenges worth mentioning regarding knowledge extraction from data streams. As mentioned in [24], current approaches fail to scale to IoT volumes, and thus, new systems employing innovative mining techniques are imperative owing to the rapid velocity, diverse variety, and fluctuating variability inherent in such data. In the near future, accurately performing this task in real-time stands as the primary challenge for IoT analytics systems. These systems should employ algorithms that utilize an extremely small amount of time and memory resources, adapt to changes, and continuously learn without interruption. Additionally, these algorithms ought to be distributed and executed on top of Big Data infrastructures. Such demands arise from the challenging nature of the IoT setting.

1.2 Scope and objectives of the thesis

The theoretic idea of this thesis comes from the European project named Digital Innovation Hub for Cyber-Physical Systems (DIH4CPS)[4]. Ever since its conception, the goal of the project was to help the affiliated companies in the digitalization of the existing systems by means of the application of artificial intelligence, and data mining techniques, focused on cyber-physical systems and embedded systems. Particularly, **this thesis seeks to cover the gap in the usage of *conceptual* knowledge extraction from *low level data* when it comes from distributed architectures such as those in which cyber-physical systems are often implemented (see Figure 1.2).**

A number of well-known data mining methods exist to address different problematics regarding the knowledge extraction from low level data coming from DAs. For instance, K-means [69], K-nearest neighbors [99], regression trees [63], and so forth. However, the majority of these methods depend tightly on the structure of the data to be dealt with, and, in some cases, on modelling techniques that create an *impedance mismatch* between the source data and the data that is actually being used as their input. This creates all sorts of problems when it comes to areas such as *explainability*, and makes them less interesting for practitioners, since achieving it would require a non-negligible amount of extra work. For that reason, a method like Formal Concept Analysis [137], whose output is closely related to how humans would understand "conceptual data", is very valuable for this task.

Formal Concept Analysis has been proven to work for knowledge extraction in several areas concerning mining *from* DAs and also *using* DAs. A great deal of approaches have been proposed to solve problems related to scalability in those areas [141, 28, 132, 49]. However, these methods introduce new problems to the mix. In particular, when it comes to *infinite* data stream mining, the main problem concerns the fact that no algorithm can process infi-

[4] http://www.produtech.org/european-projects/dih4cps

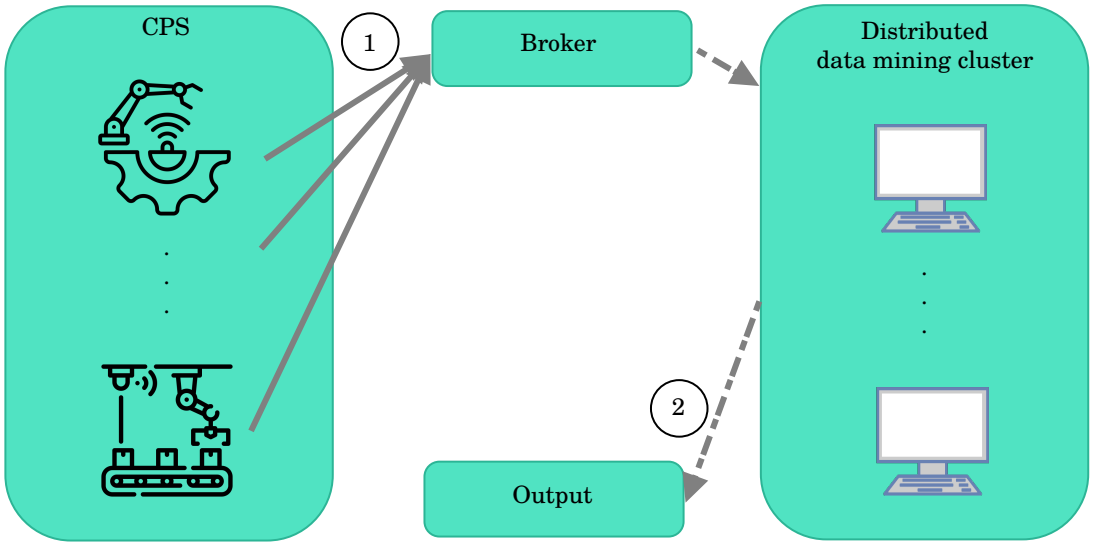

Figure 1.2: Cyber Physical Systems distributed architecture conceptual data mining general schema, where (1) represents a flow of events growing *infinitely*, and (2), an output that in the worst case is *exponential* in relation to the input.

nite data i.e., it would take infinite time only to read the input. Such a problem is commonly addressed by the usage of incremental algorithms and sliding windows in conjunction. In turn, this paradigm defines a very *specific* (even if it differs from approach to approach) way of considering only a part of the data stream, e.g., only the latest k events, or only the most recently updated formal concepts. Consequently, the possibility of **potentially losing important or relevant concepts** is introduced.

Additionally, incremental algorithms in general, except for some cases such as that of DeleteInstance [149], are defined for the case in which objects are added to the context [132, 129]. However, as one of the hypothesis of incremental algorithms is that not all information (i.e., objects in this case) is available at all times, it is also valid to assume that not even all attributes of an object are valid at all times. Thus, incremental algorithms are not flexible enough to be used in these scenarios. In particular, there is only one algorithm that can effectively process contexts that are *arbitrarily* distributed [49]. However, this algorithm falls into the batch category, thus there is the need for an incremental solution for the cases in which the context is not only arbitrarily distributed, but it is also dynamic.

Finally, the lattice construction problem is known to be $P-$complete[5], meaning, among other things, that the problem is strongly believed not to be effectively parallelizable. Nevertheless, given the time constraints that the IoT setting demands, parallelization still contributes to the scalability even if it only affects the computation by a constant.

In this context, the goals of the thesis are:

(G.1) Provide a scalable and flexible method to address the loss of potentially important concepts while constructing concept lattice using sliding windows.

(G.2) Propose an incremental method for the construction of concept lattices in the setting where not all attributes are known for a given object in each update (i.e., the formal context is arbitrarily distributed and dynamic).

[5]In fact, even determining the number of formal concepts of a certain formal context is in that class [75].

(G.3) Achieve horizontal scalability by proposing a method that leverages parallelism *using* DAs.

(G.4) Show its potential usefulness in a practical scenario.

1.3 Results summary

Regarding (G.1), a method for considering potentially lost knowledge by merging lattices has been presented. Given a general real-time lattice that is being maintained with a sliding-window technique, the method [84] consists of the storage of partial (snapshots) lattices containing particular knowledge selected in advance, and a *merge* (assembly in [130]) step between the real-time one and a snapshot. In that same vain, the proposed algorithm has been proven to compute exactly all formal concepts regarding the merge of the two underlying contexts, i.e., they are underlying because the input consists of the two lattices to be merged instead of the formal contexts. Furthermore, several experiments have been carried out to show a promising result in experimental performance compared to simply incrementally merging the lattices, in the specific case of highly correlated data.

Concerning (G.2), an incremental algorithm for *minimally updating* a concept lattice has been proposed. The algorithm fills the scientific gap of merging lattices coming from arbitrarily distributed formal contexts. Since a generalization of FCA incremental algorithms has been presented in [129], we discuss the difference with our approach. Before this work, there was only one algorithm for computing the set of formal concepts coming from an arbitrarily distributed formal context ([49]), and it falls into the *batch* category. After this work, there is one more algorithm for that purpose, but falling into the incremental one, hence, more suitable for stream processing or for dynamic settings.

In regard to (G.3), a distributed method for *merging* arbitrarily distributed lattices has been proposed. This comes as the natural extension of the single-threaded algorithm for computing the merge of lattices, and the single-threaded version of the incremental algorithm for lattice construction using *minimal* updates. In addition, this result increases the flexibility in the distributed setting, providing more insight on when it is recommendable to use merging algorithms, and when it is better to just use the incremental ones.

Finally, regarding (G.4), some of the presented methods have been put in practice in the context of an e-commerce website of the French Ski School[6]. More precisely, an FCA-Content-Based filtering ([34, 101]) method has been proposed to tackle the problem of recommending alternative ski courses and products. The method takes advantage of both flexible and scalable characteristics, i.e., it allows horizontally scaling, and moreover, recovering knowledge to target recommend based on the client's needs.

1.4 Outline of the thesis

The manuscript is generally written in first-person singular, referring to myself (Nicolás Leutwyler). However, in some specific cases I use the first-person plural "we", referring to me and the co-authors of the specific work I am mentioning in that section. For the most part, nonetheless, the plural would refer to Mario Lezoche, Diego Torres, and Hervé Panetto.

[6] ESF for its French acronym: https://www.esf.net/

The thesis is structured as follows, firstly, chapter 2 presents the state of the art regarding preliminary and necessary concepts for the understanding and development of the thesis, namely, Knowledge, Scalability, and Formal Concept Analysis. Additionally, it presents a systematic study showing evidence of the scientific gap in FCA methods related to distributed architectures, and moreover in the area of infinite data streams.

Then, in chapter 3, a method for recovering potentially lost relevant concepts is proposed by the means of merging lattices. In that regard, three different algorithms for *merging lattices* are presented. The first one, consists of the use of an already existing incremental algorithm [132] to update the largest of the lattices, and by doing that, save time on creating *some* of the existing concepts. The second one, consists of a batch algorithm that seeks to profit from the two input lattices already being computed, or in other words, to *only compute the necessary concepts*. This later algorithm is presented altogether with its proof of correctness, and its proof of worst case time complexity. Thirdly, a distributed and parallelized version of the batch algorithm (using Apache Spark) is presented, and its differences with the single threaded version are discussed.

Following, chapter 4 delves deep into FCA when the formal context is distributed across multiple nodes. It presents an incremental algorithm for updating the lattice in a minimal manner, allowing in this way to process the formal context as a data stream. The chapter includes, on top of the algorithm, a comparison with an existing batch algorithm [49], and a discussion on the advantages and disadvantages of both approaches.

Further, chapter 5 shows the real-world scenario in which the presented methods have been integrated. In particular, it presents the context, which is the e-commerce site for the French Ski School (ESF). Then, it presents the problematics they were having and wanted to address by using recommendation systems. And finally, it presents the two different followed approaches in order to address them.

Next, chapter 6 presents different sets of experiments to evaluate and compare the proposed approaches with existing (and comparable) methods. Both synthetic and real-world data sets are used. Moreover, the promising results are highlighted regarding the scalability and flexibility desirable characteristics.

And finally, chapter 7 concludes the work with the final thoughts, and sets the ground pointing out to the directions to follow in the future.

2

State of the art

Contents

2.1 Knowledge definition . . . 9
2.2 Knowledge Representation and Extraction . . . 11
2.3 Scalability, and Flexibility . . . 13
2.4 Formal Concept Analysis . . . 14
2.4.1 Basic definitions . . . 14
2.4.2 FCA in distributed architectures . . . 16
2.5 Batch and Incremental approaches in FCA . . . 18
2.6 Frequent Itemsets and Association Rules . . . 19
2.7 Finding evidence about the scientific gaps . . . 20
2.7.1 Literature Research Questions . . . 21
2.7.2 Keywords and search strings definition . . . 21
2.7.3 Database selection . . . 24
2.7.4 Study selection and quality assessment . . . 25
2.7.5 Data extraction . . . 26
2.7.6 Analysis and classification . . . 26
2.7.7 Quantitative results . . . 29
2.7.8 Analysis of correlations among categories with FCA . . . 37
2.7.9 Qualitative results . . . 40

In this chapter, the relevant state of the art regarding the scope of the thesis is presented. Firstly, the concepts of knowledge, and its extraction are presented. Then, a compendium of relevant concepts and important notation regarding scalability, flexibility, and Formal Concept Analysis is given. Later on, the methodology and results of a Systematic Literature Review [85] are presented, discussed, and analyzed using an FCA-based method including the considered articles. Furthermore, the implications and rules gathered from that analysis are examined. To conclude, the found scientific gaps are discussed.

2.1 Knowledge definition

Many epistemological debates about the meaning of knowledge have interested philosophers since the classical Greek era. For this thesis, nonetheless, we will develop this matter from an information technology (IT) point of view, rather than from the epistemological one.

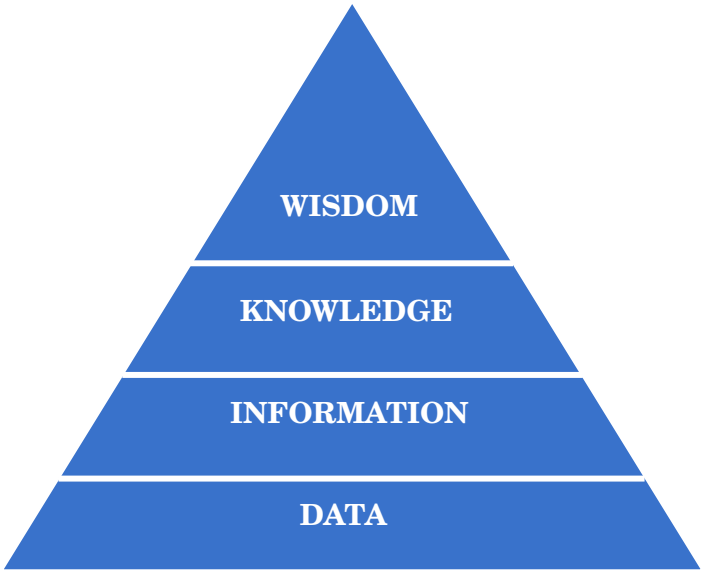

Figure 2.1: Data, information, knowledge, and wisdom (DIKW) pyramid.

In information science, it is not uncommon to find the terms of *data*, *information*, and *knowledge*. One of the possible ways of depicting them is by using the so-called DIKW pyramid Figure 2.1 ([88]) where *wisdom* also appears, although it goes outside the scope of this work. From the outset, we can see that the terms appear in a different hierarchy on the triangle, suggesting that they are not only different, but are also hierarchically inter-related. Often, data is viewed as raw numbers and facts, information is processed and contextualized data, and knowledge is authenticated information. However, these definitions vary greatly according to the dimension to be considered, e.g., context, usefulness, or interpretability. According to Alavi and Leider in [4], the key to distinguish between information and knowledge is not necessarily related to its content, structure, accuracy, or utility. Instead, knowledge is information possessed in the mind of individuals, i.e., personalized information, related to fact, procedures, concepts, interpretations, ideas, observations, and judgements.

For some authors, such as Tuomi in [126], the DIKW hierarchy is, in fact, inverse. To them, knowledge ought to exist before data or information could be formulated and before data can be measured to form information. Intrinsically, there is no "raw data" that has not been already altered by the knowledge processes that led to its identification and collection. This author argues that knowledge exists, and when it is verbalized, articulated or structured, it becomes information. Similarly, when given a fixed representation and interpretation, it becomes data. Thus, in their argument, knowledge only exists as the result of an agent stimuli. It is shaped by its requirements and its initial supply of knowledge [126]. Consequently, in [4], it is stated that knowledge becomes information once it is articulated and presented in the form of text, graphics, words, or other symbolic forms. This idea of knowledge means that agents must share a certain knowledge base to get the same understanding of data or information.

For the purpose of this work, we would deem *data* to be facts, and raw numbers. *Information* to be processed or interpreted data. And *Knowledge*, to be personalized information.

Table 2.1: People count per ski resort per hour.

Ski lift	Difficulty	Hour	Count	Ski lift	Difficulty	Hour	Count
1	Blue	9h	62	2	Blue	9h	100
		10h	150			10h	90
		11h	235			11h	80
		12h	279			12h	40
		13h	456			13h	10

Example. 2.1:

Let us suppose a system whose goal is to give directions to people in ski resorts wanting to go to a certain ski lift. In this setup, the idea is that sometimes people do not know that maybe a couple of meters nearby, there is a similar lift with the same difficulty, but with a lower queue time. The Table 2.1, shows facts regarding how many people passed through certain ski lifts for each hour per ski lift. In this context, these facts have no particular meaning on their own, and as such, they are *data*.
By aggregating the data, we could get information about at which hours ski lifts have lower queue times (because of fewer people going there). For instance, "in the morning, the ski lift 1 has a lower queue time than that of ski lift 2", and "in the afternoon, ski lift 2 has a lower queue time than ski lift 1".
To conclude, the system can filter this information to extract the necessary knowledge for its goal: "in the morning, recommend the ski lift 1. And by the afternoon, recommend the ski lift 2".

This example illustrates a possible way of modelling these three concepts, i.e., data, information, and knowledge. Nonetheless, other conceptions are valid. Moreover, notice that depending on the context, some things can be considered to be any of the three concepts. Going back to the example, we can see it as a system that has access to certain facts, i.e., people per hour per lift. From them, thanks to a process of knowledge extraction, i.e., in this particular scenario, simply aggregating and filtering, it can take decisions such as to recommend customers going to one lift instead of the other.

2.2 Knowledge Representation and Extraction

Now that we have a clearer idea of what we mean when we speak about knowledge, we will formalize it in mathematical terms to be used in a systematic fashion. In other words, if we want to compute based on the knowledge we have, we need computers to be able to understand them, hence, a formal model is needed. According to Sowa ([119]), "Knowledge representation is the application of logic and ontology to the task of constructing computable models for some domain". On the same vain, it is desirable to be able to (systematically) reason on the knowledge at hand, and thus, logic would be a good fit for such a task. However, there are numerous logics used to represent knowledge, e.g., propositional logic, first order logic, L_2 logics, and description logics [100]. In addition, not all of them are considered *decidable*, i.e., there is no algorithmic method to decide whether an arbitrary formula is valid or not. In particular, although first order logic is very powerful in terms of expres-

sivity, it is not decidable [46]. Nevertheless, it is possible to make it decidable by restricting the arity of the formulas to two [1].

Semantic networks ([14]), ontologies ([51]), and knowledge graphs ([118]) are some examples of different knowledge representation formats which have been ubiquitously used in all areas of artificial intelligence. A semantic network is a graphic notation for representing knowledge in patterns of related nodes and arcs [9]. These relations show hierarchical relationships between objects and descriptive factors. Some of the most common arcs are of the is-a or has-a type. Is-a is used to show class relationship, while has-a links are used to identify characteristics or attributes of the object nodes. Other arcs are used for showing inheritance. An ontology, on the other hand, is a formal and explicit specification of a conceptualization [52]. They define concepts, relationships between concepts, and constraints on how these concepts can be used. Typically, semantic networks and ontologies follow a structured format and, for representation, use formal languages such as RDF (Resource Description Framework) and OWL (Web Ontology Language), respectively. Finally, while knowledge graphs also consists of nodes and relationships between them, they often incorporate both structured and unstructured data sources, enabling richer and more comprehensive knowledge representation compared to semantic networks and ontologies.

Having many different types of formats for the representation of knowledge increases the ways in which knowledge extraction can be conceived. For instance, it can be to *extract a semantic network explaining the structure of this text*, or to construct an *ontology showing the conceptual structure of a database containing facts*. Consequently, a plethora of different extraction methods are used depending on the particular needs. Examples of this could be methods such as *k-means* ([69]) or *k-nearest neighbors* ([99]), whose purposes are to cluster vectors based on a certain similarity function. Other examples are *random forests* ([15]) or *neural networks* ([50]) whose goals are to learn to classify (or to do regression) based on data with certain characteristics. As a final exemplification, we have Formal Concept Analysis (FCA) based methods whose purpose (among others) is to cluster objects into different categories (formal concepts) and inheritance hierarchies (relations between concepts) based on their characteristics (more on this in section 2.4).

Nevertheless, for this work, it is essential to set a common ground on what we refer to with knowledge extraction. For us, the meaning would be the understanding of information based on common patterns to a certain goal.

Definition 2.1 (Knowledge Extraction (KE))**.** The creation of knowledge from structured (e.g., relational databases, extensible markup language (XML)) and unstructured (e.g., text, documents, images) sources ([127]).

Definition 2.2 (Data Mining (DM))**.** The process of discovering knowledge or patterns from massive amounts of data ([54]).

Definition 2.3 (Knowledge Discovery in Databases (KDD))**.** An automatic, exploratory analysis and modeling of large data repositories. KDD is the organized process of identifying valid, novel, useful, and understandable patterns from large and complex data sets ([89]).

In fact, if we compare Definition 2.1, Definition 2.2 and Definition 2.3, we realize that they are essentially referring to the same concept. In particular, creating, discovering, and identifying knowledge are just three different ways of stating the same goal. Their only difference lies in the *source* of the data, but that becomes meaningless when considering the

interoperability between data formats. Thus, in this thesis, we will use them interchangeably.

2.3 Scalability, and Flexibility

A number of different characteristics can be considered to be relevant to data mining systems. Regarding knowledge extraction in distributed architectures, nevertheless, scalability, and flexibility, will be the main focus in this work.

The software domain utilizes the term "scalability" to describe a system's (or algorithm's) ability to adapt and perform well as the size and complexity of its inputs grow. This can be thought in a number of ways. For instance, a system that requires too much memory, might scale poorly in terms of space, but, the more memory we add to it, the larger the input the system will be able to handle. Another example of the concerns of scalability is how algorithms behave as the input gets larger. However, in the general case, all possible inputs are infinite[7], and even taking different inputs of the same size, they can present different characteristics (see Example 2.2), hence, an effective way of measure scalability regardless of the characteristics of the input is needed. For example, when analyzing how an algorithm behaves related to computation time, *measuring its run-time* with certain test-cases is not ideal because it would be impossible to consider all inputs (since they are infinite), and also because it will always depend on the architecture[8], and available resources the algorithm is being run with (thus, it'd be far from a *general* method to compare scalability).

Example. 2.2:

Let us consider the Algorithm Minimum of finding the minimum element of a list containing natural numbers $\mathbb{N}$ (or $\mathbb{N} \cup \{0\}$)[a]. The idea of the algorithm is to traverse the list and updating the variable $minimum\_elem$ with the minimum value so far, thus, at the end of the loop, the variable correspond to the minimum value. Moreover, if a 0 is reached at some point, we know that no lesser element will be found (because all elements $x \in \mathbb{N}$), hence, we just stop the loop and return 0.
Now, imagine that we run the algorithm with the inputs $L_1 = \{1000, 3, 4, 80, 3, 24, 1\}$, and $L_2 = \{0, 1000, 3, 4, 80, 3\}$. Even though both inputs are the same size, the algorithm would not do the same amount of work for both of them. In fact, it is easy to see that for L_1, the algorithm will traverse *all* elements before returning 1, while for L_2, it will immediately return 0 (lines 4-5).

[a] Typically, in computer science, natural numbers include the 0, and we will follow that convention.

For all the mentioned reasons, it is necessary to have an effective and reliable way to measure scalability regardless of the architecture, the language, and the differences in between inputs of the same size. With that in mind, in computer science, it is common to consider different *general* scenarios for studying scalability, e.g., (1) the worst case, (2) average case, and (3) best case.. The cases are studied in function of the size of the input n. The so-called *big* "$\mathcal{O}$" notation is used to describe the functions based on their asymptotic behavior (see chapters 2 and 3 in [23]). The idea of this approach of analysis is to *count* how many *basic operations* are necessary to perform in the worst (average, best, etc) case. The

[7] Let us think of inputs of size 1, 2, and in general, of size n, where $n \in \mathbb{N}$.
[8] Depending on the architecture, the actual *machine code* can vary greatly.

Algorithm Minimum Algorithm for finding the minimum element of a list L containing elements $x \in \mathbb{N}$

```
1: minimum_elem ← +∞
2: for all x in L do
3:     if x = 0 then
4:         minimum_elem ← 0
5:         break
6:     end if
7:     minimum_elem ← min(x, minimum_elem)
8: end for
9: return minimum_elem
```

basic operations are defined in [23], in the widely known random-access machine (RAM) model, and it consists of a set of "reasonable" operations that can be done in constant time, e.g., *ADD*, *SUB*, *JUMP*. By doing so, it is possible to compare algorithms by their scalability, for example Algorithm Minimum is $\mathcal{O}(n)$ in the worst case, but $\mathcal{O}(1)$ in the best case. Then, if we take the version of the algorithm that does not include the optimization (lines 4-5), it would be $\mathcal{O}(n)$ in the best case as well, thus, we would prefer the first over the second one. Notice that, since the analysis is asymptotic, the fact that $\mathcal{O}(f(n))$ is better than $\mathcal{O}(g(n))$ for certain functions $f, g : \mathbb{N} \to \mathbb{N}$, does not mean that the algorithm related with the first one will run faster than the second one. It means that *from some point on* there will be an input size from which the first algorithm will start running faster (or consuming less space, depending on where the analysis was done) than the second one.

Besides how scalable algorithms, systems, or methods are, this work is interested in how *flexible* they are. This concerns characteristics such as in how many situations it is possible to apply the method, and mainly, *how hard* it is to adapt the method to different settings.

2.4 Formal Concept Analysis

2.4.1 Basic definitions

Formal Concept Analysis (FCA), introduced in [137], is a mathematical framework, also referred to as a data analysis method, for extracting knowledge from objects described by attributes. It has been applied in tasks such as the mining of high level semantics from low level events [28], and as such it is the main focus of study in this thesis.

Definition 2.4 (Formal Context)**.** A triple $\mathbb{K} = \langle G, M, I \rangle$, where G is a set of objects, M is a set of attributes, and I is an incidence matrix where iIj if $g_i \in G$ has the attribute $m_j \in M$, and $i\not{I}j$ otherwise.

Example. 2.3:

In Table 2.2, a Formal Context consisting of 4 ski lesson purchases, i.e., p_1, p_2, p_3, and p_4, their characteristics, i.e., **GL**, **PL**, **SKALPN**, **SKNORD**, **nbP**$<$**3**, **3**$\leq$**nbP**$<$**7**, and **7**$\leq$**nbP**. Finally, the incidence matrix is represented in the table marking with an X if gIm, and with nothing otherwise.

Table 2.2: A Formal Context of ski lessons purchases.

Purchases	GL	PL	SKALPN	SKNORD	nbP<3	3≤nbP<7	7≤nbP
p_1	X			X	X		
p_2		X		X		X	
p_3		X	X				X
p_4	X			X		X	

Typically, given a formal context $\mathbb{K}$, the framework clusters the objects based on the attributes they share. Each of these clusters resemble how humans think of concepts in the hierarchical sense, i.e., *animals* are living beings, *dogs* are *animals* and have four legs, *spiders are animals* and have *eight legs*, hence, both dogs and spiders are living beings (conceptually, that is the characteristic they have in common), however, they are not part of the same concept because they differ in at least one characteristic. In that vein, the framework defines "what is a *formal concept*" in mathematical terms, and moreover, it defines as well the sub-concept (and super-concept) relationship.

To dive into the definition of a formal concept, we must first define what is the derivation of a set of objects and attributes,

Definition 2.5 (Derivation (')). Let $'$ be the derivation operation on a set of objects $X \subseteq G$ (dually, on a set of attributes $Y \subseteq M$) given by

$$X' = \{m \in M \mid \forall g \in X \text{ st } gIm\}$$
$$Y' = \{g \in G \mid \forall m \in Y \text{ st } gIm\}$$

Example. 2.4:

The derivation of the set of attributes $\{\textbf{GL}\}$ is noted $\{\textbf{GL}\}' = \{p_1, p_4\}$, whilst the derivation of the set of objects $\{p_1, p_4\}' = \{\textbf{GL}, \textbf{SKNORD}\}$.

The Definition 2.5 is a function that we use as $' : G \to M$ or $' : M \to X$. In particular, the double derivation $'' : G \to G$ or $M \to M$. As we can see in the Example 2.4, $''$ is an increasing function, and, what is more, $X' = X'''$ ([42]) for any set $X \subseteq G \vee X \subseteq M$.

Definition 2.6 (Formal Concept). Given a formal context $\mathbb{K} = \langle G, M, I \rangle$. A formal concept is a pair $C = \langle X, Y \rangle$ such that $X \subseteq G, Y \subseteq M$, $X' = Y$, and $Y' = X$. X is called the ***extent*** and Y the ***intent***.

Example. 2.5:

An example of a formal concept from the formal context described in Example 2.3 is

$$\langle \{p_1, p_4\}, \{\textbf{GL}, \textbf{SKNORD}\} \rangle$$

We can think of a formal concept $\langle X, Y \rangle$ as the pair containing objects and attributes from the dataset, where the extent are *exactly* all the objects that share all attributes in its intent. Dually, Y are *exactly* all the attributes shared by the objects in its extent. Another way of thinking about it, maybe more in natural language terms, is that if we add any object

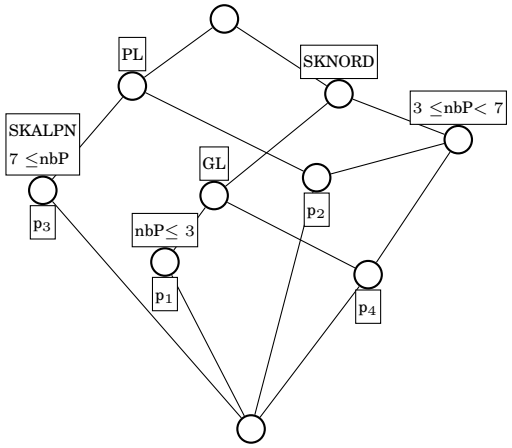

Figure 2.2: Example of a concept lattice in a *reduced line diagram* where only *introduced* objects and attributes are shown.

(or attribute) to its extent, then the intent has to be reduced, because surely there will be at least one attribute that is not shared by all the objects. In other words, since the new intent has fewer attributes, the concept will become *more general*.

On that same vain, we define the sub-concept and super-concept relationships in the following way

Definition 2.7 (Sub-concept (Super-concept))**.** We say that a concept $C_1 = \langle X_1, Y_1 \rangle$ is a ***sub-concept*** of another concept $C_2 = \langle X_2, Y_2 \rangle$ iff $Y_2 \subseteq Y_1$ (which is equivalent to $X_1 \subseteq X_2$). And we will note it $C_1 \leq C_2$. Dually, C_1 is a ***super-concept*** of C_2 iff $Y_1 \subseteq Y_2$ (which is equivalent to $X_2 \subseteq X_1$), and we will note it $C_1 \geq C_2$.

As depicted in the Definition 2.7, the sub-concept (and dually the super-concept) relationship yields a partial order between concepts. Then, the set of all formal concepts $\mathcal{C}$ and their sub-concept relationships $\leq$ form the so-called *concept lattice* $\mathcal{L} = \langle \mathcal{C}, \leq \rangle$.

Example. 2.6:

The context in Example 2.3 has 10 concepts. The *reduced* line diagram depicted in Figure 2.2 represents the concept lattice of this context. To interpret the diagram, it is important to have in mind that the top concept $\top$ contains all objects, while the bottom concept $\bot$ contains all attributes. Then, concepts inherit attributes from top to bottom, whereas they inherit objects from bottom to top. The objects and attributes are written only in the concept that introduces them. For instance, the concept $\langle \{p_1, p_4\}, \{\textbf{GL}, \textbf{SKNORD}\} \rangle$ gets the two objects from its two children, and the **SKNORD** attribute from its only parent.

2.4.2 FCA in distributed architectures

FCA has been successfully applied in DA settings, mining *using* DAs [141], mining *from* DAs [132], and both combined [28]. For instance, [141] implements two FCA algorithms using Twister Iterative *MapReduce* ([35]), leveraging its properties (*distributed* and iterative) in order to reduce computation time. The first one, called MRGanter, consists of applying

a modified version of the NextClosure ([42]) algorithm to multiple nodes each having a partition of the input in order to maximize the parallelization. Nonetheless, the algorithms are not intended to process tuples as they arrive. In fact, they are meant to divide the *entire* input into several nodes to then start the distributed computation. Hence, although they certainly enhance the computation scalability, they are not designed to mine *from* DAs. For this reason, it would fall into the category of Mining using Distributed Architectures.

Additionally, the work presented in [28] entails the implementation of a data mining method on a distributed real-time computation system (i.e., Apache Storm[9]) with the objective of big data stream analysis on smart cities. The presented system uses the stream abstraction provided by Apache Storm, i.e., an unbounded sequence of tuples that is processed and created in parallel in a distributed fashion. Schemas are defined to associate fields in the tuples. A *spout* is a source of a stream in the system. Particularly, spouts read tuples from an external source and emit them to the system (i.e., topology in Apache Storm). The processing operations are executed by *bolts*, which include, for instance, filtering, executing arbitrary functions, aggregating, and so forth. Parallelism is obtained by configuring spouts and bolts to start *executors*, that can be thought as threads being able to run in parallel, each of which can process data by executing different *tasks*. The scalability is acquired by allowing to change the number of executors on run-time. Then, let us suppose several distributed sensors producing data represented by the tuple $\langle x_1, \ldots, x_k \rangle, k \in \mathbb{N}$. The tuples will be processed by a broker (e.g., Kafka[10]) and sent to the system, which then will be emitted by a spout with a certain structure depending on the area of the city where the tuple was produced. The tuple arrives to the first bolt that is parallelized depending on the city areas, and whose purpose is to create aggregations based on the specific domain, e.g., given two measures of traffic, consider only the maximum of them. And finally, the second bolt uses the *curated tuples* that continuously arrive to compute the output incrementally. Thus, it would fall into both the categories of Mining from Distributed Architectures and Mining using Distributed Architectures.

Concluding the examples, van der Merwe et al. in [132] present an incremental conceptual data mining algorithm, i.e,. the conceptual lattice is computed as data arrives, instead of having to recompute it. The way the algorithm works is by defining an initial result $\mathcal{L}_0$ when no data has been received. Then, using the function "*add_intent*" to update the conceptual structure $\mathcal{L}_i$ to $\mathcal{L}_{i+1}$ as the i-th object arrives, $i \in \mathbb{N}$. Since each object has a finite amount of attributes, the invocations of "*add_intent*" can be understood as steps in the processing of a data stream of objects and their attributes. However, the presented algorithm is meant to be run in a single node, since it uses shared variables and loops that make it not trivially paralellizable. Therefore, it only falls into the category of Mining from Distributed Architectures.

However, all these approaches have to deal with certain inherent problems posed by the FCA framework, and other problems particular to their specific contexts.

One of the most relevant problems regarding FCA, is that the amount of formal concepts in a context can be exponential to the size of the context in the worst case. More precisely, given a context $\mathbb{K} = \langle G, M, I \rangle$, the amount of concepts that can be extracted from it is $\mathcal{O}(2^{\max(|G|,|M|)})$. This in itself is a problem even for non-distributed settings. It not only poses challenges regarding time, but also space. Consequently, a number of approaches are commonly taken into account to deal with it.

[9] Apache storm website.
[10] Kafka website.

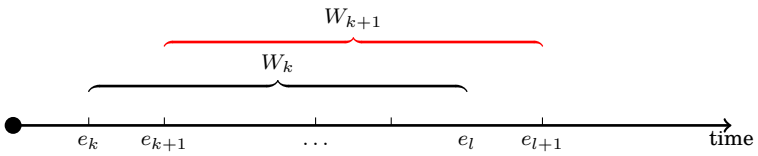

Figure 2.3: Example of a sliding window of size $l - k$.

Firstly, some method use "interest" coefficients to prone the set of formal concepts. To name the most common one, given a formal concept $C = \langle X, Y \rangle$ from a formal context $\mathbb{K} = \langle G, M, I \rangle$, we have the following,

$$\textit{Support: } \sigma(C) = \frac{|X|}{|G|} \tag{2.1}$$

being how many objects from the dataset are actually occurring in that specific concept. This idea comes from the construction of iceberg concept lattices [121], which, in turn, comes from a prior idea in frequent itemsets mining [2, 114]. In summary, a concept is said to be frequent if its intent is frequent, and an intent Y is frequent if $\frac{|Y'|}{|G|} > \textit{minsupp}$ for $\textit{minsupp} \in [0, 1]$. Hence, the support gives us a good idea of it because it actually is *how many objects are contained in the concept represented by that intent*. Or in other words, *how many occurrences do we have of that intent in the dataset*. By using this coefficient, the iceberg concept lattice consist of all frequent concepts C with a support $\sigma(C) > \textit{minsupp}$.

2.5 Batch and Incremental approaches in FCA

Furthermore, there is a problem related to highly dynamic settings such as that of IoT. A plethora of algorithms for concept lattice computation are meant to be run "just once" given a dataset. These algorithms are typically referred to as *batch* algorithms [41, 13, 73, 86, 121]. Their main characteristic is that they have access to the whole dataset and can thus take smart decisions based on that. However, if anything in the data changes, the algorithm has to be run again in order to compute the new concept lattice. To approach this problem, incremental algorithms are used [132, 103, 48, 18, 104]. These programs approach the construction of the concept lattice in an incremental manner, that is, the lattice is updated for each object. The advantage they have over batch algorithms is that they can be used with dynamic datasets such as data streams, allowing to compute *only* an update rather than the whole lattice from scratch. Nevertheless, they might still run into performance problems (both time and space) when the dataset to process is big, which clearly will always be the case when dealing with data streams.

Another common practice related to this problem is the use of sliding windows [95, 28]. It consists of considering elements in a data stream that belong to a certain window of time. For example, as depicted in Figure 2.3, given a data stream, a sliding window of size x would consider only the x last elements. Furthermore, it is called sliding, because as the stream progresses, the window can be slid to consider new elements as in the figure is shown in red. Nonetheless, it is not the only way of applying a sliding window technique, it is also possible to consider only the concepts more recently updated.

Finally, it is common as well to use FCA extensions in order to leverage their advantages for pruning even more the final lattice [26]. Examples of such extensions are Fuzzy Formal

Concept Analysis (i.e., an extension of FCA [90] where the incidence matrix is a relation $I \subseteq G \times M \to [0,1]$) and Temporal Concept Analysis [138]. However, for readability, we will introduce them as we need them in their respective chapters.

2.6 Frequent Itemsets and Association Rules

As already stated in this thesis, data mining is the process of discovering patterns in huge amounts of data. In that vain, among the interesting patterns that are usually looked for in the Industry 4.0 we can find *frequent itemsets* and *association rules*, where the latter is a more complex characterization of data and its discovery depends fundamentally on the finding of *frequent itemsets*. The problem of finding frequent itemsets [79] consists of discovering sets that appear "frequently" in a set of baskets, where the baskets are sets of items. More specifically, let us imagine a marketplace selling all kinds of goods, and a database recording each purchase as of "these goods have been bought together". Then the problem would be finding the common goods that are usually bought together.

Intuitively, a set of items that is related to many baskets is thought to be *frequent*. More formally, there is a number s, or *support threshold*. If Y is a set of items, the support of I is the number of baskets for which I is a subset. Then, we say that I is frequent if its support $\delta(I)$ is greater or equal than s.

Example. 2.7:

Let us imagine a marketplace of a ski resort selling group and private ski lessons, different types of accommodation, services such as a box Wi-Fi, equipment rent, spa service, and so forth. In the following, we have these purchases,

1. { box Wi-Fi, group ski lesson, lunch }
2. { box Wi-Fi, group ski lesson }
3. { group ski lesson, 3 stars accommodation }
4. { private ski lesson, 5 stars accommodation, spa service }
5. { box Wi-Fi, private ski lesson, spa service }

If we consider a support threshold of 2, here we have the set as a frequent itemset since it is included in all 5 baskets (also called *transactions*). Nevertheless, this is only interesting for theoretical reasons, and hence we will just mention it here once. Then, we have the frequent singletons { box Wi-Fi }, { group ski lesson }, { spa service }, { private ski lesson }, with a support of 3, 3, 2, 2, each respectively. Finally, we have the frequent itemsets with two elements which are { box Wi-Fi, group ski lesson }, and { spa service, private ski lesson } with a support of 2 the two of them.

As you might have guessed, there is a clear similarity between frequent itemsets, transactions and FCA. In fact, if we say that each transaction is an object, the set of all items in the dataset are the attributes, and the itemsets are represented in the incidence matrix, then, the set of frequent itemsets given a support threshold s is exactly the set of *intents* from a concept with a support larger than $\frac{s}{|G|}$.

One of the main applications of frequent itemsets is the analysis of actual market baskets, where a supermarket would store of every purchase in each shopping cart. In this example, the items are the products that the supermarket sells and the transactions or baskets are the sets of items in each single purchase. By identifying frequent itemsets, a retailer can discover which items are commonly purchased together. We will discover by this analysis that many people book beginner and intermediate ski lessons together, but that is of little interest, since we already knew these were popular options individually. We might discover that many people book ski lessons and equipment rentals together. That, again, should be no surprise to avid skiers, but it offers the ski resort an opportunity to do some clever marketing. They can advertise a discount on ski lessons and raise the price of equipment rentals. When people come for the discounted ski lessons, they often will remember that they need to rent equipment, and buy that too. Either they will not notice the price is high, or they will reason that it is not worth the trouble to go somewhere else for cheaper rentals.

Moreover, the analysis can help discover that some items are related even if we hardly expected them to. The usual example of this are the "diapers and beer", where in a specific instance it was discovered that people buying diapers were surprisingly likely to also buy beers (i.e., the pair was a frequent itemset with a very high support). The hypothesis is that people buying diapers are expected to have a baby at their place, and in that case, they are unlikely to be going to drink at a bar, thus, when drinking, they will do it at home. Additionally, frequent itemsets has found applications in areas such as text document analysis, and plagiarism detection.

Another way of conceiving these frequent itemsets is in the form of a pair of an antecedent and a consequent, and in that case, they are called *association rules*. An association rule of the form $A \to B$ where A and B are an itemses. The meaning of such a rule is that if all items in A appear in a transaction, then, all items in B are "likely" to appear in it as well. Formally, the likelihood is said to be the *confidence* of a rule, and it is defined as the ratio of the support of $A \cup B$ and the support of A. In other words, how often transactions containing A also contain B.

In the Industry 4.0, mining association rules is of capital interest. In fact, the problem of finding them usually considers that all transactions do not fit in memory. Hence, algorithms have to find their way in order to compute them nonetheless. In this thesis, the concepts of frequent itemsets and association rules will be of use in the chapters 3 and 5.

2.7 Finding evidence about the scientific gaps

Considering the areas and problems mentions thus far, in order to have a clearer idea what areas are already covered, solved, or overlooked, it is necessary to make an assessment of the current state of the art. For that reason, it is common to carry out literature reviews. In particular, in the case of this thesis, a specific and systematic methodology has been followed with the goal of finding what are the scientific gaps present in the field of knowledge extraction in distributed architectures, considering the challenges that come with distributed architectures, formal concept analysis, and the Industry 4.0. To do so, the methodology aims to answer a very focused and specific question, identify relevant studies, evaluate their quality, and qualitatively (or quantitatively) summarize the findings ([110]). The search carried out in this particular study is based on the guidelines introduced by Petersen et al. [106]. Generally, the methodology consists of defining the research questions that will guide the

search. Use the keywords in the questions in order to create a query string that will define the set of articles to consider in the review. Select in which databases the search will take place. Search for articles using the defined query. And finally, analyze the results. One of the reasons why it is interesting to do an SLR is that it leaves evidence on how to replicate the search using the criteria mentioned in it.

Particularly, this section starts with the definition of the literature research questions (2.7.1) and then proceeds with the definition of keywords (2.7.2) that will guide the creation of *search strings*, to then define the databases to be used (2.7.3). Afterward, the search is performed by querying the selected databases using the previously defined *search strings* and a subset of the resulting articles is selected following specific criteria (2.7.4). Finally, three more steps are conducted on the final selection of articles: data extraction (2.7.5), analysis and classification (2.7.6), and validity evaluation (2.7.6).

2.7.1 Literature Research Questions

In this subsection, the research questions are defined. To guide this study, the main literature research question is the following: (LRQ) What are the approaches on multi-relational data mining (MRDM) and Concept Analysis (CA) for knowledge extraction (KE) used for semantic interoperability in DA?
Taking LRQ into account, we derive the following specific literature-research questions:

- LRQ_1: When and where the articles have been published?
- LRQ_2: What are the methods utilized for MRDM?
- LRQ_3: What are the methods utilized for CA?
- LRQ_4: What are the methods utilized for DA?
- LRQ_5: What are the problematics the papers aim to solve?
- LRQ_6: In which domains the methods are applied?
- LRQ_7: What data format the methods allow extracting and from which ones?
- LRQ_8: What are the characteristics considered in the evaluation of the articles?
- LRQ_9: What are the data formats and methods used in each article for the semantic interoperability problem?

2.7.2 Keywords and search strings definition

Important concepts

The following concepts will be used toward the rest of the SLR,

- *Data Mining* (DM) is the process of discovering knowledge or patterns from massive amounts of data ([54]).
- *Knowledge Discovery in Databases* (KDD) is an automatic, exploratory analysis and modeling of large data repositories. KDD is the organized process of identifying valid, novel, useful, and understandable patterns from large and complex data sets ([89]).

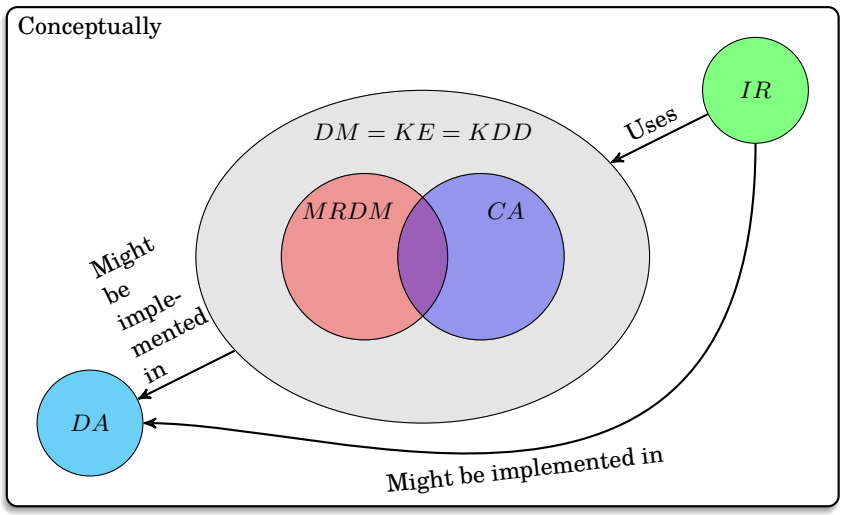

Figure 2.4: Venn diagram from a conceptual point of view, showing the relations between *data mining* (DM), *multi-relational data mining* (MRDM), *knowledge extraction* (KE), Knowledge Discovery on Databases (KDD), Concept Analysis (referring to FCA and its extensions, and abbreviated CA), *information retrieval* (IR), and *distributed architectures* (DA).

- *Knowledge Extraction* (KE) is the creation of knowledge from structured (e.g., relational databases, extensible markup language (XML)) and unstructured (e.g., text, documents, images) sources ([127]).
- *Information Retrieval* (IR) is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers) ([91]). This definition differs to the Data Mining one in the sense that, in IR, the goal is to gather information from an unstructured source in order to *satisfy* some *known* need, e.g., browsing in google. Hence, it is important to notice that IR methods might use Data Mining algorithms in their process.
- *Concept Analysis* (CA) is referred as the ensemble of the FCA method, introduced by Wille [137], and its extensions, e.g., RCA ([112]), PCA ([44]).
- *Distributed Architecture* (DA) is a collection of autonomous computing elements that appears to its users as a single coherent system ([133]).

Considering these definitions, and as discussed in depth in the article of Fayyad et al., from the AI magazine in 1996 [37], it is our understanding that DM, KDD, and KE are three names for the same concept. Moreover, IR is a concept related to the overall process that might use DM algorithms in its implementation, but also includes steps like data warehousing. Additionally, DA is a type of environment in which both DM and IR algorithms might be implemented in or not. To better represent this idea, Figure 2.4 depicts our understanding of the relations between these concepts.

The keywords and the search strings were defined relying on Population, Intervention, Comparison, and Outcomes (PICO, defined in [60]):

- Population: In the context of this literature study, the population are knowledge extraction or information retrieval studies using some form of CA (e.g., FCA, RCA, Fuzzy FCA) or applied in the MRDM context.
- Intervention: In this work, these are algorithms, methods and tools,
- Comparison: In this study, we compare the different ways of dealing with the challenges of DA by identifying and analyzing the used algorithms and their applicability in semantic interoperability.
- Outcomes: A classification of the methods in terms of their capabilities.

Considering this, the identified keywords are the following:

- Scoping the search for our domain: "Knowledge Extraction", "Data Mining", "Knowledge Discovery", and "Information Retrieval".
- Related with the population: MRDM, "concept analysis" and their synonyms.
- Terms related with the intervention: "algorithm", "method", and "tool".
- Search terms related with the comparison: semantic, semantically, interoperability, and interoperable.

Query

Following the four sets of keywords defined in the previous subsection, we divide the query string into four groups.

S_1: "*algorithm*" OR "*method*" OR "*tool*".

S_2: "*knowledge*" OR "*extraction*" OR "*data*" OR "*mining*" OR "*information*" OR "*retrieval*". This is to look for papers having not only the concepts "knowledge extraction", "data mining", and "information retrieval", but also the combination of words such as "knowledge mining", "data extraction". This group also includes other combinations, such as "mining extraction" or "knowledge data", and the isolated words, but those articles will be filtered out in a later stage (subsection 2.7.4).

S_3: "*RCA*" OR "*concept analysis*" OR "*MRDM*" OR "*multi relational*". The purpose of this group is to consider only papers that related to the frameworks FCA or some other form of CA, and MRDM.

S_4: "*semantic**" OR "*interop**". This is a group to aim the search toward those papers being about any form of semantic or interoperability. Note that the * means that all types of endings are valid. For example, *semantic* is a valid word as well as *semantically*.

Finally, the query S_1 AND S_2 AND S_3 AND S_4 was performed on the selected databases on the titles, abstracts and keywords, as we show in Table 2.3.

Database	Search
ACM	Title:((method OR algorithm OR tool) AND (knowledge extraction OR data mining OR Information Retrieval) AND (rca OR "concept analysis" OR mrdm OR "multi relational") AND (semantic* OR interop*)) OR Abstract:((method OR algorithm OR tool) AND (knowledge extraction OR data mining OR Information Retrieval) AND (rca OR "concept analysis" OR mrdm OR "multi relational") AND (semantic* OR interop*)) OR Keyword:((method OR algorithm OR tool) AND (knowledge extraction OR data mining OR Information Retrieval) AND (rca OR "concept analysis" OR mrdm OR "multi relational") AND (semantic* OR interop*))
IEEE	(method OR algorithm OR tool) AND (knowledge OR extraction OR data OR mining OR information OR retrieval) AND (rca OR "concept analysis" OR mrdm OR "multi relational") AND (semantic* OR interop*)
Scopus	TITLE-ABS-KEY ((method OR algorithm OR tool) AND (knowledge OR extraction OR data OR mining OR information OR retrieval) AND (rca OR "concept analysis" OR mrdm OR "multi relational") AND (semantic* OR interop*))
Taylor & Francis Online	(method OR algorithm OR tool) AND (knowledge OR extraction OR data OR mining OR information OR retrieval) AND (rca OR "concept analysis" OR mrdm OR "multi relational") AND (semantic* OR interop*)
Web of Science	(method OR algorithm OR tool) AND (knowledge OR extraction OR data OR mining OR information OR retrieval) AND (rca OR "concept analysis" OR mrdm OR "multi relational") AND (semantic* OR interop*)

Table 2.3: Query strings used for each search engine.

2.7.3 Database selection

For the search, we considered the recommendation in [106] that says using IEEE and ACM plus two indexing databases is sufficient. Additionally, we also included a more general database, Taylor & Francis Online, to possibly reach applications in more domains. In summary, we used the databases: ACM, IEEE_Xplore, Scopus, Taylor & Francis Online, and Web of Science. The obtained results can be seen in Table 2.4. Zotero[11], a reference management tool, was used in order to delete duplicates and to manage the large amount of references. This study has been conducted in May 2022, so all articles up until the 30th of April were considered during the search.

Database	Total	Unique	Duplicated
ACM	169	36	133
IEEE	72	16	56
Scopus	644	644	0
Taylor & Francis Online	6	0	6
Web of Science	209	12	197
Total:	1100	**708**	392

Table 2.4: Number of papers obtained on the queries execution on each database.

[11] https://www.zotero.org/

2.7.4 Study selection and quality assessment

From the resulting articles, firstly, we filtered them based on their titles and abstracts in a step named first screening. Secondly, we included or excluded papers based on a full-text reading, in a step named second screening. The criteria used for the selection can be seen in Table 2.5, which is divided in two groups, C_1 being about the type of articles to be included or excluded, and C_2 being their content. It is important to mention that, when in doubt during the title and abstract reading, articles were taken to full-text reading instead of directly excluded. One of the threats to the reliability of the review is that the selection was conducted by only one author, and hence it could have misclassified some articles (more on that in section 2.7.6). To mitigate this threat, a different author examined a subset of the final selection. Afterward, the full-text and metadata of the references of the articles that passed the first and second screening are filtered with the same criteria in a step called *backward snowballing*, introduced by Jalali and Wohlin [66], and applied in [40]. Similarly, the process of searching in the papers referencing the selection after the second screening, is called *forward snowballing*. For these steps, we used the Scolr[12] tool, which allows conducting both screening and snowball steps. The number of papers included/excluded in each of the steps can be seen in Figure 2.5.

C1 Inclusion	Exclusion
Papers written in English	Papers not written in English
Conference Papers	Non peer reviewed papers
Journal Papers	Literature reviews
Book Chapters/Sections	Proceedings
C2 Inclusion	Exclusion
About MRDM or About Formal Concept Analysis	Anything not satisfying the inclusion criteria
Applied to Distributed Architectures or Big Data or Optimization of the method	

Table 2.5: Criteria for selecting papers.

The quality assessment consisted in comparing the final 69 primary studies with an independent set of papers that ought to be in the final set ([8, 28, 29, 33]). Also, the following questions were answered to assess the quality of the selected articles:

- Is the method clearly the core part of the article?
- Are the algorithms, architectures, or methods explicitly defined?
- Are experiments or discussions conducted to evaluate the results?

Thus, studies not explicitly defining the methods, algorithms, architectures or not doing experiments or discussions on the results were excluded. The final selection of articles is presented in Table 2.6.

[12] https://scolr.lifia.ar

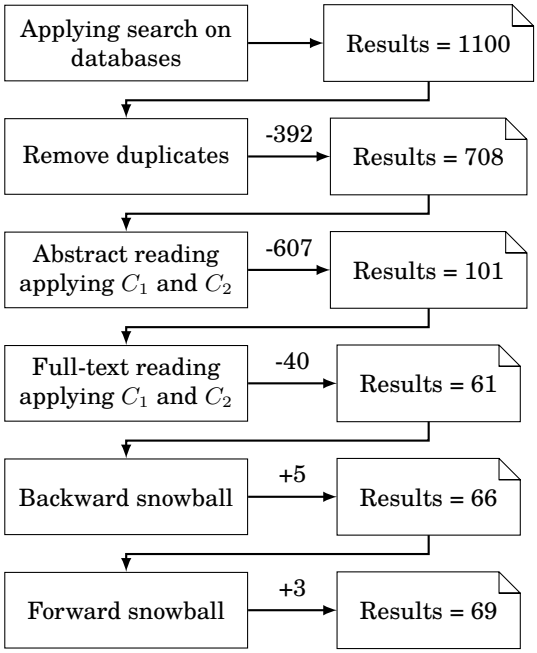

Figure 2.5: Number of articles included/excluded in each step of the selection process

2.7.5 Data extraction

The data extraction from the identified primary studies has been done considering the Table 2.7. Data extraction fields have a *data item* and a *value*, which are the name and the description of the fields respectively. Notice that although the *Article ID* and the *Article title* fields are not related with any LRQ, they were deemed useful, trivially, to differentiate the articles univocally.

2.7.6 Analysis and classification

The fields extracted from each selected primary article are tabulated and visually illustrated in subsection 2.7.7. The papers were grouped and counted by each of the fields, and since some of them are multiple (e.g., methods) the total sum is greater than the total amount of papers reviewed.

For the field *methods application*, the articles were grouped by how the methods are being used in them, which could be CA, MRDM, or DA. An article is considered to be in CA if the method proposed in it is based on FCA or any of its extensions. Additionally, an article is considered to be in MRDM if it proposes a method for extracting knowledge from any multi-relational data format. Finally, an article is considered to be in DA if the method is *implemented* using a DA system, i.e., introduced in subsection 1.1.2, or if it *extracts knowledge from a DA system*, i.e., introduced in subsection 1.1.1. As an example of how the classification was done, a paper that used FCA for knowledge extraction but implementing a way to use it in DA would have CA and DA in its *method application* field. Furthermore, the analysis was conducted considering the three main groups CA, MRDM, and DA, allowing us to draw conclusions about their different characteristics.

The *addressed in* field reflects where the problematic is addressed regarding the generic DM process lifecycle presented in Figure 2.6. This field is multivalued because an article

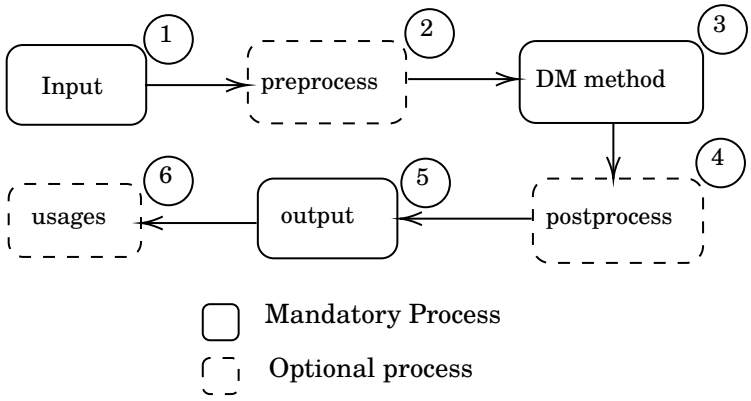

Figure 2.6: Generic Data Mining process life cycle.

can address the problematic in more than one way. This process is based on the one introduced in [142] under the name of “The steps for data mining process” in the Figure 3 (and also based on the Figure 1 in the article "From Data Mining to Knowledge Discovery in Databases" [37]). In our diagram, 1) *input* corresponds to the target data where it is needed to extract knowledge (*data* and *target data* in [142]). 2) *Preprocess* is the treatment of the target data in order to match it with the specific input of the DM method to be used (*preprocess* and *transformation* in [142]). 3) *DM method* is the method/algorithm to be used in particular, e.g., k-means, FCA, or Regression Trees (*data mining* in [142]). 4) *Postprocess*, analogously to preprocess, is the treatment of the *direct* output of the DM method in order to achieve the *expected* output, that not always matches with the one it produces. 5) *Output* corresponds to the final knowledge produced (*knowledge* in [142]). 6) *Usages* represents a step in which the produced knowledge is exploited.

The purpose of the *domains* field is to represent in which domain the paper applies the method. The considered domains are (1) *general* when the article presents a solution that is not bounded to any specific domain (although it could be applied or tested in a specific one, the solution is still presented in a general way), (2) *semantic web* when the authors aim to contribute with a semantic web technology such as ontologies or Resource Description Framework (RDF), (3) Machine Learning (*ML*) when the contribution is based to any machine learning method.

The fields *Input of the method* and *Output of the method* represent the method’s expected and produced formats respectively. For instance, if the method used is plain FCA, the input of the method is going to be a Formal Context. The purpose of this field is to determine what are the formats covered to perform data mining on the one hand, and on the other one, what are their respective output formats.

For the *evaluation strategies* that the articles use in order to evaluate their proposed solution, this work makes use of the taxonomy defined in [108]. In particular, the evaluation types considered for categorizing the evaluation strategies of articles are (1) *Accuracy*: the degree of agreement between outputs of the method and the expected outputs. (2) *Effectiveness*: the degree to which the method achieves its goal in a real situation. (3) *Efficacy*: the degree to which the method achieves its goal considered narrowly, without addressing situational concerns. (4) *Efficiency*: the maximization of the ratio between outputs and inputs of the method. (5) *Robustness*: the ability of the method to handle invalid inputs or stressful environmental conditions. (6) *Performance*: the degree to which the method ac-

complishes its functions within given constraints of time or space. Speed and throughput (the amount of output produced in a given period of time) are examples of time constraints. Memory usage is an example of space constraint. (7) *Technical feasibility*: evaluates, from a technical point of view, the ease with which a proposed method will be built and operated. (8) *Operational feasibility*: evaluates the degree to which management, employees, and other stakeholders, will support the proposed method, operate it, and integrate it into their daily practice. (9) *Learning capability*: the ability of the method to learn. (10) *Validity*: means that the method works correctly, i.e. correctly achieves its goal. (11) *Scalability*: the ability of the method to either handle growing amounts of work in a graceful manner, or to be readily enlarged. (12) *Ease of use*: the degree to which the use of the method by individuals is free of effort. (13) *Consistency*: the degree of uniformity, standardization, and freedom from contradiction among the elements of the structure of the method. (14) *Utility*: measures the value of achieving the method's goal, i.e. the difference between the worth of achieving this goal and the price paid for achieving it. The given definitions come from the appendix in [108].

Finally, the field *semantic interoperability* represents how the articles contributed to the semantic interoperability between different knowledge formats. For example, let us consider an article that develops a method to solve the problem of certain links (edges) not being present in knowledge graphs (KG), by using ML and completing the links with certain predictions. Such an article would have the value *KG* in this field, because it is contributing to the semantic interoperability between two KG, one having more links than the other based on a probabilistic method.

Validity evaluation

According to the guidelines in [106], these types of validity should be considered: descriptive, theoretical, generalizability, and interpretive validity, all of which are explained and detailed below.

Descriptive validity

Descriptive validity is how accurately and objectively observations are described. Qualitative works have a greater threat to descriptive validity than quantitative ones. In order to reduce this threat, we created a data extraction form in Table 2.7, and discussed each field in subsection 2.7.6 to help objectify the recording of data. Thus, this threat is considered under control.

Theoretical validity

The ability of being able to capture what we intend to capture is called *theoretical validity*. For instance, biases can lead us to select articles we should not and to not select others that we should.

While searching, studies could have been missed. For example, two searches in the same field, could yield different sets of articles. To address this threat, backward and forward snowball sampling of all articles has been done after the full-text reading [66] (see Figure 2.5).

Bias is also a threat in the phase of *data extraction and classification*. To address this problem, it is useful that one researcher performs the phase while the other reviews it.

However, given the fact that the process involves human judgement, the threat is unavoidable.

Generalizability

[105] introduced a distinction between external and internal generalizability, i.e., between groups or organizations, and within a group, respectively. Internal generalizability does not represent a major threat because of the wide range of articles following the same strategies. In terms of the external one, it is not a major threat as well because the approach takes into account general metrics, defined in Table 2.7, that can be applied to other fields of study.

Interpretive validity

Interpretive validity refers to the idea of the conclusion being reasonable given the data. A threat to interpretive validity is, again, researcher bias. In our case, the first author's major field of experience is algorithms and efficiency, and that could lead to a bias in the interpretation. Despite this, the rest of the authors are experts in knowledge representation and extraction, semantic interoperability, and industrial engineering, helping reduce this threat.

Repeatability

Repeatability refers to how repeatable the process is. It demands a thorough reporting of the research process. We reported the SLR process followed, and moreover, we explained the different possible threats and our actions to reduce them.

2.7.7 Quantitative results

The results of the data extraction have been gathered and put together in an open dataset in [81] to facilitate its reproducibility. In this section, using the mentioned dataset, the findings of the systematic literature review are illustrated and presented.

Frequency of Publications ($\mathbf{LRQ}_1$)

Articles were counted by year and publication venue, and the results are shown in Figure 2.7 and Figure 2.8 respectively. The year with more articles was 2016 with 13 articles, more than doubling the second most occurred years 2010, 2019, and 2020 with 6 papers each. Most of the papers were published after 2010. Moreover, the years 1996-2000, 2002, 2003, 2007, and 2008 did not have occurrences.

The growth in papers after 2009 showed that there is still an increasing interest in this topic in the scientific community. Additionally, one of the hypothesis we have about the reason this happens is the increase of challenges the surge of IoT had in those years.

In terms of publication venue, most of the papers were published in conferences and journals with 35 and 26 occurrences each respectively. Only 6 workshop articles and 2 book sections have been included in the study. Particularly, both the journals and conferences in the study are heterogeneous, ranging from computer-science specific to engineering and manufacturing ones.

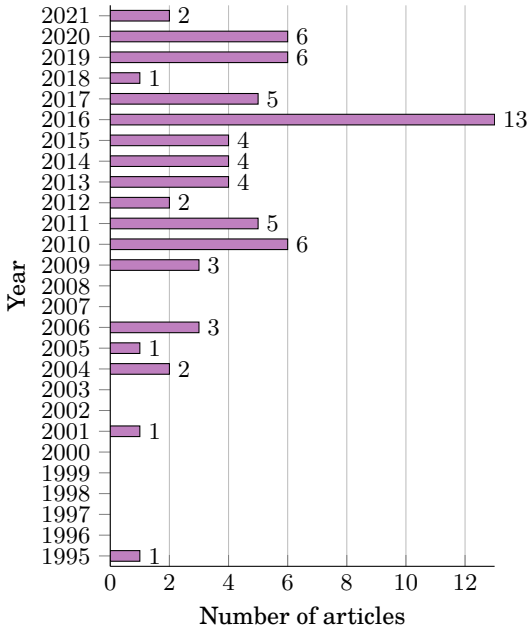

Figure 2.7: Articles per year

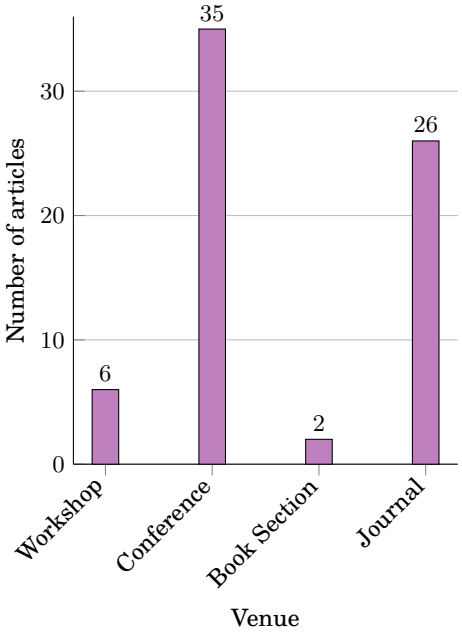

Figure 2.8: Articles per publication venue

Methods application (LRQ_2 & LRQ_3 & LRQ_4)

The used methods in the articles were counted and categorized in the main three categories in this study: CA, MRDM, and DA.

In the Figure 2.9, there is a Venn diagram with the three categories showing the amount of articles in each of the parts: only CA 17, only MRDM 19, and only DA 2. For the mixed used methods, there are 21 CA ∩ MRDM, 8 CA ∩ DA, and 2 in MRDM ∩ DA. Interestingly, there were no articles with methods doing CA ∩ MRDM ∩ DA.

It is important to notice that both CA and MRDM overall had a similar total amount of occurrences: 46 and 42 respectively. The main difference relies on the methods based on DA that are almost unsubstantial in MRDM, with only 2 articles, while CA had 8 occurrences in that regard. Furthermore, in Table 2.8, the classification on where each DA article stands regarding the items 1.1.2 and 1.1.1 is shown.

Addressed problematics (LRQ_5)

For the addressed problematics, articles were categorized according to their place in the DM process lifecycle depicted in Figure 2.6. In Figure 2.10, the amount of articles according to the maximum number of papers found in all of the areas has been depicted using a heatmap where completely white means 0 articles, and completely red means 53. We can observe that 14 articles addressed a problematic regarding the input, 3 the preprocess, 53 the DM method, 0 the postprocess, 10 the output, and 22 the usages.

The Table 2.9 shows that from the articles that addressed the problematic by defining a particular DM method, 10 also did it with a specific usage in mind. Only 1 define a particular output while still aiming to a specific use case. And 7 defined the DM method with a specific output in mind, but without the use case.

Additionally, there were 5 articles that addressed their problematic by providing a new

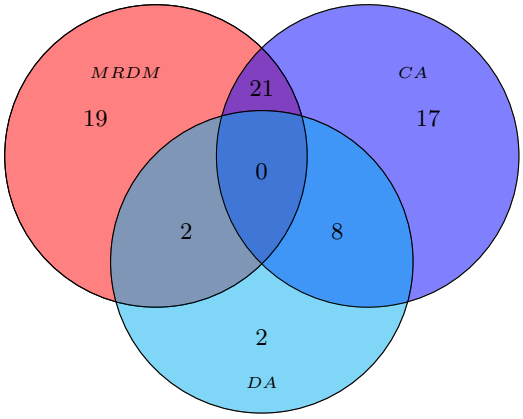

Figure 2.9: Methods used in the articles classified in CA, MRDM and DA.

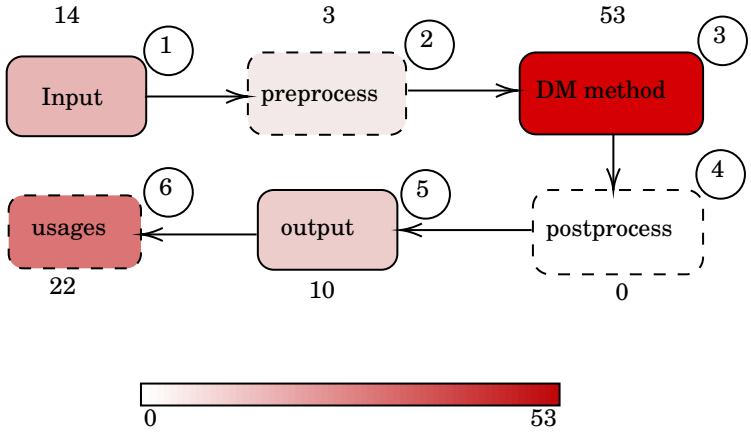

Figure 2.10: Heatmap showing the number of articles regarding the DM process life cycle presented in Figure 2.6.

type of input to an already defined DM method. 11 also defined a new DM method to consume the defined input. Only 1 defined the entire process with a particular type of input and output. And 2 articles defined a new input to solve a specific use case.

Regarding the articles whose problematic was based on the produced output, only 2 focused entirely on it, whereas 1 did it with a particular use case. Moreover, 3 articles presented a preprocess step with a DM method, and only 1 of them also defined a new output. Finally, there were 5 articles whose problematic was based on a specific use case and they did not have to define anything new but the application.

Notice that there were no articles addressing the problematic with a postprocess solution. One reason why this happened could be the nature of the article databases selected, which are more centered in computer science and not so much in applications, while the postprocess step is probably more common in specific domains that look novel ways to **utilize** DM methods rather than to **improve** them.

Domains ($\mathbf{LRQ}_6$)

The domains found in the articles by method category are depicted in Figure 2.11. From the 46 CA articles, 41 were found to be general solutions, while 4 were applied specifically to se-

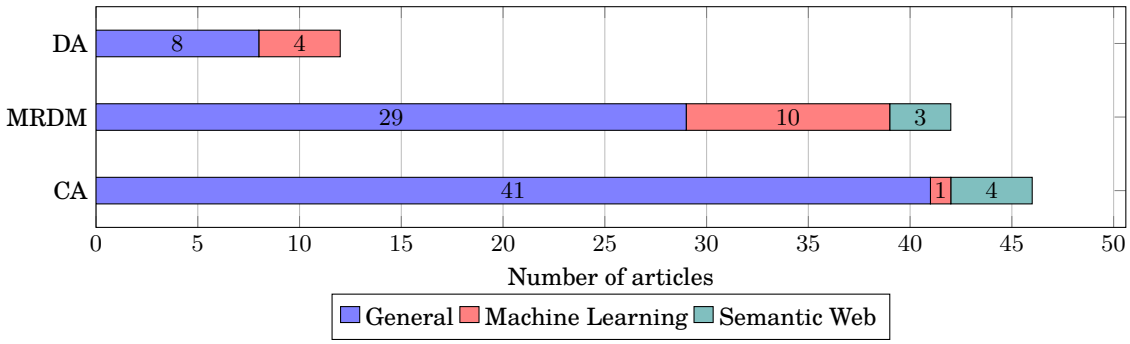

Figure 2.11: Domains of articles classified by method category.

mantic web technologies, and 1 to the machine learning domain. Additionally, from the 42 articles in MRDM, 30 were general, 9 on machine learning techniques, and 3 on semantic web technologies. Moreover, from the 12 DA articles, 8 were found to be general, 4 particularly in the domain of machine learning. More granularly, Figure 2.12 depicts in a taxonomy the subcategories in each of the three main ones. There were only two subcategories shared by *general* and *machine learning*: *financial* and *social networks*.

Input and Output formats ($\mathbf{LRQ}_7$)

The amount of input formats found in the search was larger than anticipated. In fact, almost every paper used a specific format in it. Thus, for readability purpose, in Figure 2.13 only the formats with more than 1 occurrence are displayed. The first thing that stands out of the figure is the fact that only 3 input formats have more than 3 occurrences: Formal Context, RDB, and Data Streams. Then, showing the heterogeneity of scenarios addressed in the articles, RCF has 3 occurrences, and with 2 occurrences there are the following input formats Distributed Formal Context, Fuzzy Formal Context, RDF, Documents (text), Interordinal Formal Context, Knowledge Base, Tweet Stream, and Multi-relational Graph.

The amount of output formats found in the search was as numerous as the amount of input formats. For the same reason, Figure 2.14 shows only formats with more than 1 occurrence. Differently to the input formats figure, in this one the heterogeneity is slightly more balanced, that is there are 4 dominant format types instead of 2, being *classifier* with 9 occurrences in MRDM, *concepts lattice* with 6 occurrences in CA and 2 in MRDM, *association rules* with 6 in MRDM and 3 in CA, and *ontology* with 6 in CA and 5 in MRDM. In DA, the three output formats with more than 2 occurrences are *formal concepts*, *set of tweets*, and *regression tree* with 2 occurrences each.

In Figure 2.15, the relations between input and output formats are depicted. On the left part, there are the input formats with more than 2 occurrences, and on the right hand, the output formats with more than 2 occurrences plus a node *other* representing any output format with only one occurrence. Both sides are connected by arrows conveying that for the specific input format on the arrow, there is at least one article that produces that output format.

Evaluated Characteristics of the Methods ($\mathbf{LRQ}_8$)

The evaluated characteristics of the methods in each article have been assessed and counted, yielding the results depicted in Figure 2.16, Figure 2.17, and Figure 2.18. The top evaluated

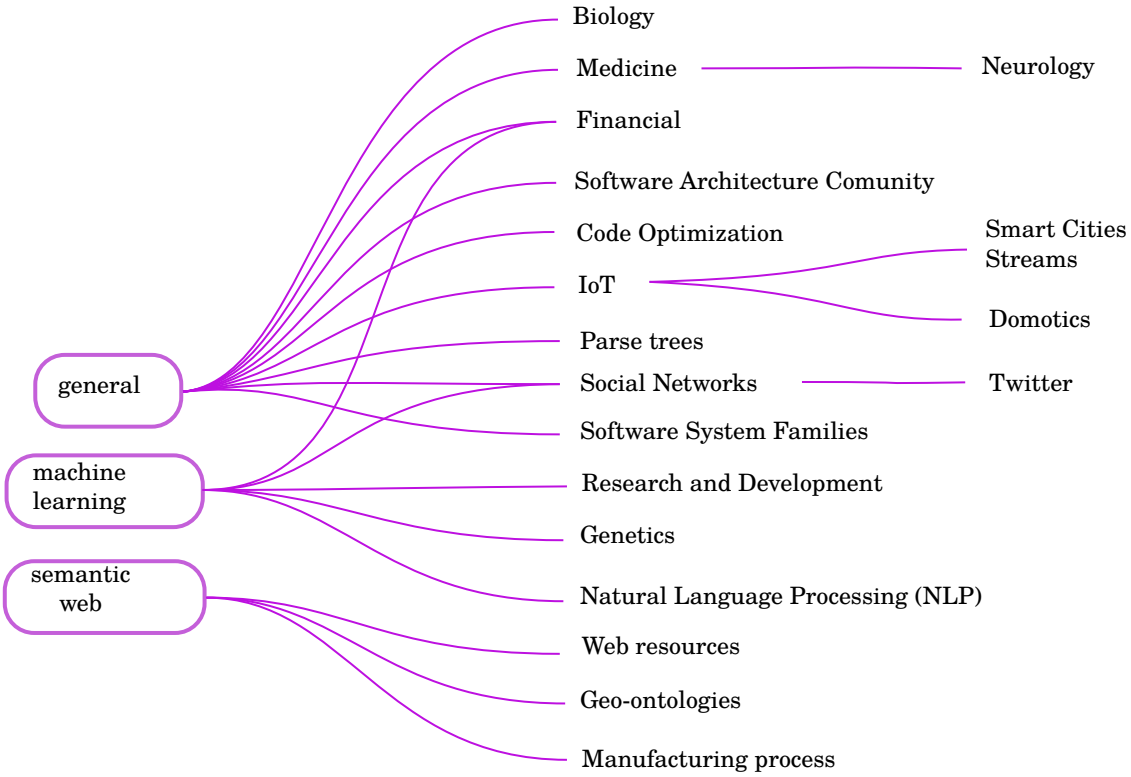

Figure 2.12: Taxonomy of the domains found in a granular way.

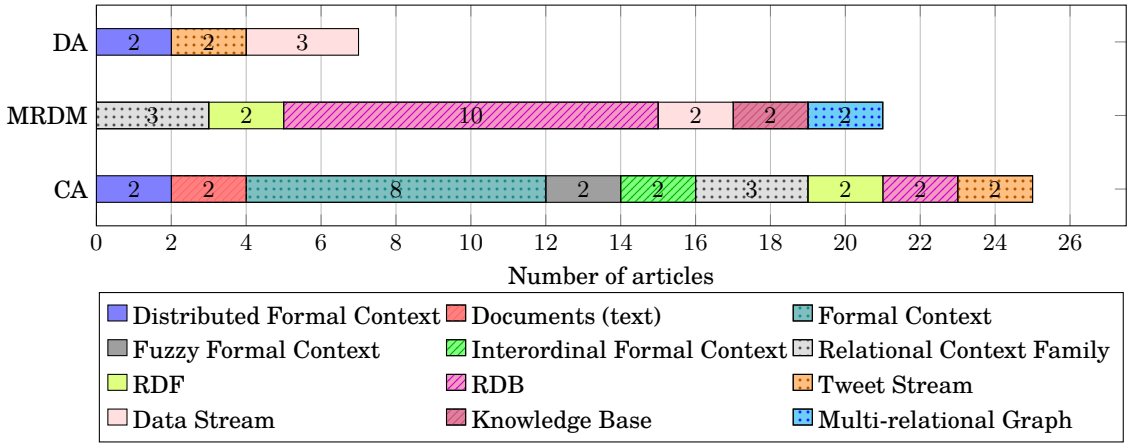

Figure 2.13: Input formats with more than 1 occurrence by method category.

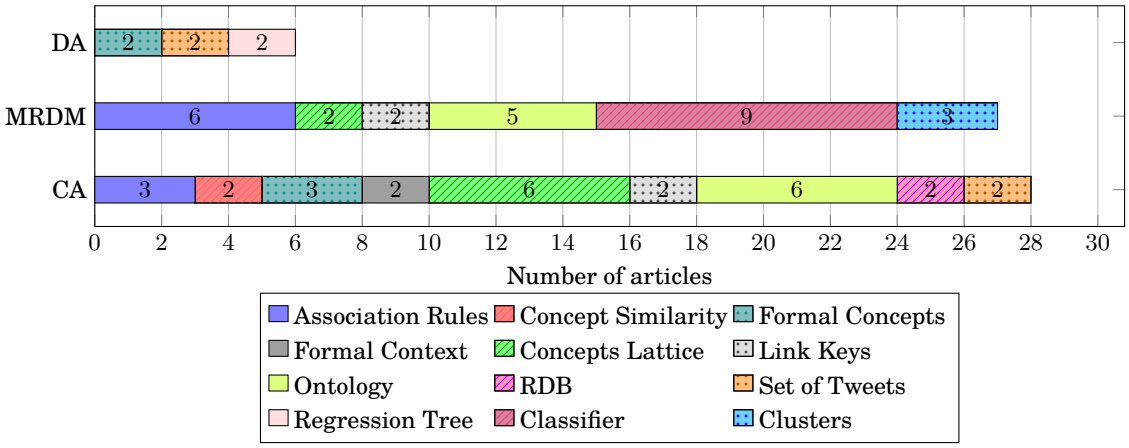

Figure 2.14: Output formats with more than 1 occurrence by method category.

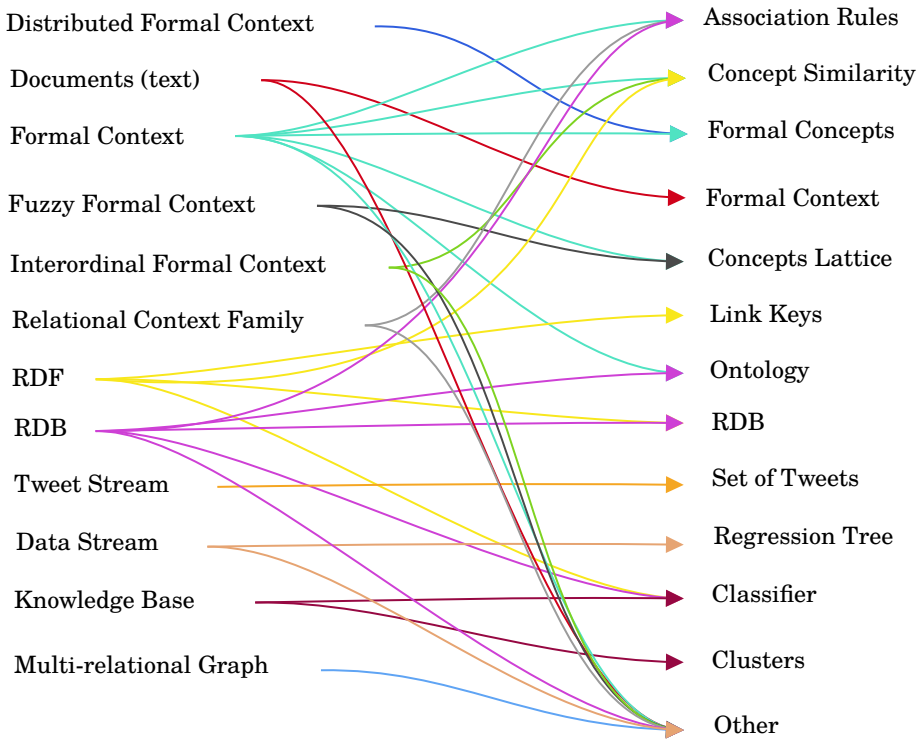

Figure 2.15: Relations between input and output formats.

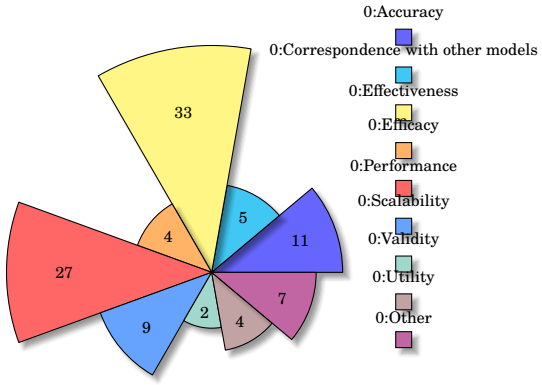

Figure 2.16: Evaluation strategy by method in CA.

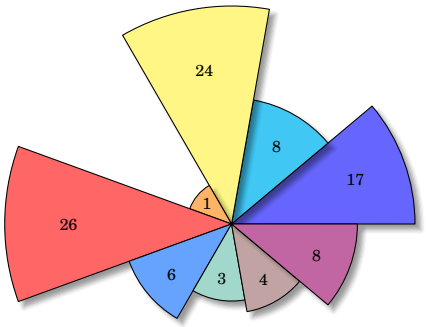

Figure 2.17: Evaluation strategy by method in MRDM.

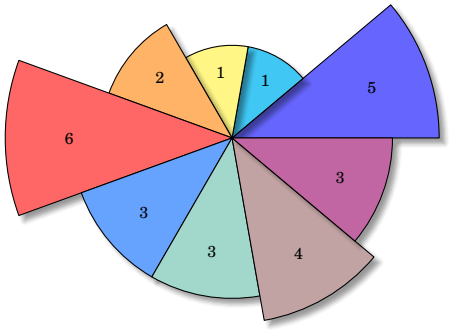

Figure 2.18: Evaluation strategy by method in DA.

characteristic was *performance*, having been evaluated in the three method categories with in a similar percentage of all evaluations. The second top evaluated characteristic overall was *effectiveness*, however it has only been evaluated in 1 of the DA articles. *Accuracy* has been widely evaluated in the three method categories with a considerable percentage, being DA the one in which the metric is the top evaluated, alongside with *performance*. *Scalability* is the next most evaluated characteristic in line, with 9, 6, and 3 occurrences in CA, MRDM, and DA respectively. *Utility* is a characteristic that, even though it has been evaluated in all method categories, it had more attention in the DA ones in terms of occurrences relative to the amount of articles. *Correspondence with other models* was not so popular between the evaluated characteristics, but it was at least fairly studied in MRDM with 8 occurrences out of the 42 articles. *Validity* and *efficacy* had a similar amount of occurrences in all method categories, nevertheless, they had more relative weight in DA, and a very low impact in CA. Finally, *other* is the category for the evaluated characteristics with 2 or fewer occurrences in all three method categories: *technical feasibility*, *robustness*, *ease of use*, *efficiency consistency*, and *learning capability*.

Different from CA and MRDM, only 1 DA article evaluated effectiveness. This might be related to the fact that DA is still not in a state as mature as that of CA and MRDM. Moreover, this is also similar to the result about having only one DA article evaluating correspondence with other models, while in CA and MRDM there are 5 and 8 respectively. On the other hand, both in the DA and MRDM fields, not all the papers evaluating performance also evaluated scalability, in fact, in those fields, only a small percentage did.

Methods for Semantic Interoperability between Knowledge Formats ($\mathbf{LRQ}_9$)

The problem of semantic interoperability between different knowledge formats has been assessed for each article. From that assessment, several clusters of articles essentially addressing the interoperability between the same knowledge formats have been extracted and depicted in Figure 2.19. Following, we summarize each cluster of papers considering whether the methods used were CA, MRDM or DA, and mention the specific methods used in each of them,

1. Tab (tabular data): The articles [111, 8] adopted CA methods to address the semantic interoperability between Formal Contexts, and Comma Separated Value (CSV) files (some articles may consider formal contexts regardless of their format, while others may specify that they are using CSV) to Formal Contexts. The methods that were utilized were FCA and Mixed FCA.

2. StL (stream to lattice): [49, 28, 29, 132, 141] adopted methods in CA and DA to address the semantic interoperability from Distributed Formal Contexts, Tuples, and Streams, to lattices or some usage of them such as evolution paths. The methods used were FCA, Incremental Fuzzy FCA, and Fuzzy Cognitive Maps.

3. FC (to formal concepts): The papers [136, 115, 59, 139, 39, 134, 98, 19, 90] adopted CA and MRDM methods to address the semantic interoperability from Fuzzy Formal Context, and Formal Concepts, to Formal Concepts. The methods used in these articles were FCA, Granular Computing, Interordinal FCA, Multi-valued FCA, set-coverage, and FCA + Inclusion Degree Theory.

4. TXT (between free text documents): [55, 93, 21, 128] utilized CA and MRDM methods to address the semantic interoperability between documents with free text. The specific methods used were FCA + R-Trees, and FCA + Latent Semantic Analysis.

5. AR (to Association Rules): The articles [109, 96, 124, 32, 114, 146] adopted CA and MRDM methods to deal with the semantic interoperability from Association Rules, Noisy Formal Context, Fuzzy Formal Context, and Relational Context Family (RCF) to Association Rules. The methods used were FCA, Fuzzy C-means Clustering, Fuzzy FCA, RCA-AOC, and Inductive Logic Programming (ILP).

6. Streams (between streams): The papers [26, 63, 27, 62, 64, 116] utilized methods in MRDM and in DA to address the semantic interoperability between different kinds of Streams such as tweets, heterogeneous, and general streams. The specific methods used were Fuzzy FCA, Regression, and Decision Trees.

7. Misc (miscellaneous formats): The articles [143, 17, 68, 94] used CA and MRDM methods to deal with the semantic interoperability between Software Programs (source code), and Software System Families and training datasets. The methods used were Convolutional Neural Networks, FCA + Pattern Structures, FCA + Spatial Indexing, and Granular Computing.

8. Onto (to ontologies): The articles [140, 11, 6, 3, 123, 25, 12, 107, 43, 78, 61, 65] adopted CA, MRDM, and DA methods to deal with the semantic interoperability from Formal Contexts, RDBs, ontologies, and web pages, to ontologies. The methods used in the articles were FCA, RCA, Fuzzy FCA, Fuzzy RCA, Rough Set Theory, and Nystörm approximation.

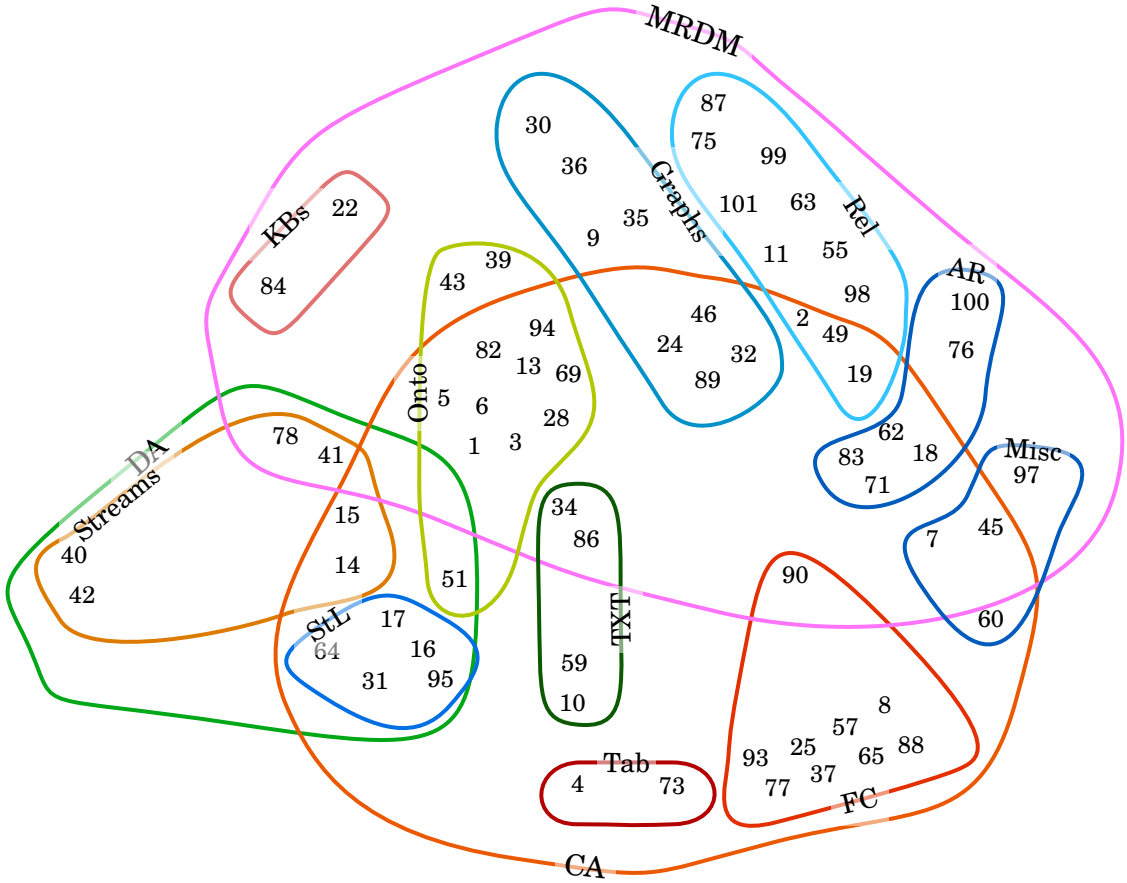

Figure 2.19: Clusters of articles by semantic interoperability between knowledge formats.

9. Graphs (between graphs): The papers [71, 38, 53, 135, 57, 45, 58, 20] utilized CA and MRDM methods to address the semantic interoperability between different types of graphs such as Conceptual, Knowledge, and multi-relational. The methods used were Graph-FCA, Homophilic FCA, RCA, Deep Learning, Neural Networks, Homophilic FCA, Principal Component Analysis, and Subgraph Matching.

10. Rel (between relational formats): The articles [147, 22, 87, 76, 144, 113, 97, 131, 145, 33, 5] adopted CA and MRDM methods to address the semantic interoperability between RDBs, and RCFs. The methods used were K-means, Semantic tableaux, Naïve Bayes, FCA, Tuple ID propagation, Multi-relational Iceberg-cubes, Bayesian belief network, Multi-relational frequent pattern discovery, and RCA-AOC.

11. KBs (between Knowledge Bases): [125, 36] utilized MRDM methods to address the semantic interoperability between different knowledge bases. The methods used there were evolutionary algorithms, and Fuzzy Clustering.

2.7.8 Analysis of correlations among categories with FCA

This subsection serves as a formal supporting element for the subsequent discussion, so that not only isolated results are considered. In the following subsections 2.7.8-2.7.8, in order

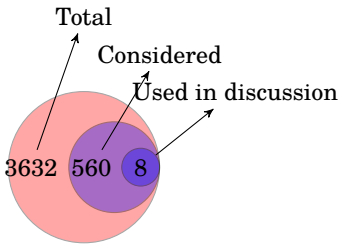

Figure 2.20: Venn diagram of association rules, showing the discussed ones in relation to the total.

to extract correlations between the different categories, each of which has been discussed in the previous section, the results are interpreted by means of association rules where each article is considered a *transaction*, and each category an *item*. To be more precise, a transaction is a list of items, and an association rule is a pair $X \to Y$ whose meaning is "a transaction with items X is likely to also have items Y". For instance, if the dataset had only two articles, the first one with the categories $\{\text{DM method}, \text{accuracy}, \text{classifier}, \ldots\}$, and the second one with the categories $\{\text{DM method}, \text{accuracy}, \text{regression tree}, \ldots\}$, one possible association rule would be $r = \{\text{DM method}, \text{accuracy}\} \to \{\text{classifier}\}$, meaning that from all the articles with the categories *DM method* and *accuracy*, some of them also have the category *classifier*. There are two metrics usually considered in order to understand how meaningful these rules are: *support* (σ) and *confidence* (κ), being the amount of transactions including the items in $X \cup Y$ over the amount of all transactions, and the amount of transactions including $X \cup Y$ over the ones including X respectively. Particularly in this example $\sigma(r) = 50\%$ because half of the articles have all the categories in the rule, and $\kappa(r) = 50\%$ because from the articles having the categories in X, half of them also have the categories in Y. From now on, the association rules will be noted as $X \to Y_{\sigma=x,\kappa=y}$, where X and Y sets of properties and x and y are the support and confidence respectively. One of the main difficulties in analyzing these rules is the amount they are in relation to the amount of items and transactions (i.e., exponential). Additionally, several rules can be redundant, for example if $X \to Z$, we could say that $X \to Z \setminus X$. In this particular case, as depicted in Figure 2.20, the total amount of rules found with Lattice Miner[13], with minimum $\sigma = 0.1\%$ and minimum $\kappa = 0\%$ were 3632. Moreover, the association rules considered for the analysis were selected from the 560 ones found with Lattice Miner, with a minimum $\sigma = 5\%$ and a minimum $\kappa = 70\%$, i.e., rules that involve at least 5% of the total articles, and that at least 70% of the articles having X also have Y.

Furthermore, three scientific gaps that have been detected during the article assessment are discussed in the subsection 2.7.9 together with possible research directions and approaches to address them.

About the input treatment

Data Mining methods can be very specific, applied to niche cases with particular characteristics. However, there are methods that are generalized to some degree and allow their usage in a variety of different scenarios. For that reason, practitioners sometimes prefer to transform their input to a "standard" one in order to use a general data mining method.

[13] https://sourceforge.net/projects/lattice-miner/

In some cases, such transformation is trivial, but in some others, whether it is because of the loss of semantics, performance, or lack of resources, it is not. Thus, there are research articles that address this matter in certain ways, such as providing a more effective transformation function, adapting the DM method in order to be able to process more input formats, or both.

Among the association rules including *input* in their hypothesis, there is the association rule $\{\text{DA, input}\} \rightarrow \{\text{DM method}\}_{\sigma\simeq 8\%,\kappa=100\%}$ implying that *all* articles with a DA method addressing a problematic regarding the input defined or adapted a particular DM method. One of the reasons why this could be happening is that input formats in the DA environment are usually not easily convertible to non-distributed ones, thus improving the interoperability of the DM algorithms in that regard requires adapting the existing algorithms to properly function. Another reason why this could happen is the lack of general DM methods in the DA environment, meaning that with the slightest change in the input, either a modification of an existing method or a completely new method has to be implemented. Another interesting rule is $\{\text{scalability, input}\} \rightarrow \{\text{CA, DM method}\}_{\sigma\simeq 7\%,\kappa=100\%}$, meaning that *all* articles that evaluated scalability as an important metric, on top of addressing an input problematic, are articles adapting or implementing new CA methods.

About the output treatment

There are several reasons why articles could work on the output of a DM method. The first and most obvious is the translation between a format to another one, considering a particular domain. The second one could be that the output does not comply with the requirements of the problem, such as needing too much time to compute it, or even not being able to load it because of its size. The latter is generally the case of CA methods, that, since most of them rely on FCA, their output is bounded to have to deal with the exponential sized output it could generate.

Among the association rules including *output* in their hypothesis, there is the implication $\{\text{output, effectiveness}\} \rightarrow \{\text{CA}\}_{\sigma\simeq 12\%,\kappa=100\%}$ which means that all articles whose problematic addressed was in the output and that considered effectiveness as an interesting metric, are articles using CA methods. $\{\text{output, MRDM}\} \rightarrow \{\text{performance}\}_{\sigma\simeq 14\%,\kappa=90\%}$ shows how important is the performance metric for articles using MRDM methods. Interestingly, $\{\text{output, CA}\} \rightarrow \{\text{MRDM}\}_{\sigma\simeq 11\%,\kappa=80\%}$, the majority of articles addressing a problematic in output included in the CA method category, are also included in the MRDM method one.

About ML and Semantic Web

$\{\text{ML, classifier}\} \rightarrow \{\text{MRDM}\}_{\sigma\simeq 10\%,\kappa=100\%}$ is one of the most interesting rules because it implies that all articles with methods in the domain of ML, providing as an output a classifier, work with MRDM methods. Additionally, this means that no pure CA article worked on the domain of ML and provided a classifier as the final output, although there are known CA methods to do so. Another one in the ML domain with an interesting meaning is $\{\text{ML, classifier, usages}\} \rightarrow \{\text{accuracy}\}_{\sigma\simeq 57\%,\kappa=100\%}$, conveying that from the mentioned articles, *all* the ones addressing the problematic with a particular usage also consider accuracy as metric to evaluate.

As for Semantic Web, the rule that stands out is $\{\text{semantic web}\} \rightarrow \{\text{CA, effectiveness}\}$ with a $\sigma \simeq 5\%$, and a $\kappa = 100\%$, implying that all semantic web articles in the study use CA methods and consider effectiveness as an important metric to evaluate. This did not

surprise us, since semantic web technologies rely on conceptual structures, and evaluating how effectively the methods can extract those concepts is understandably a metric worth considering.

2.7.9 Qualitative results

As depicted in Table 2.8, most of the DA papers mine *from* DAs subsection 1.1.1 instead of doing so by providing a distributed DM method (i.e., they were not selected because of subsection 1.1.2). Moreover, the vast majority of them used data streams as their input type format, considering certain common constraints. Firstly, the incremental fashion in which data streams usually arrive to be processed [28, 29, 63, 116, 132, 62, 64]. Secondly, the temporal characteristic of events [28, 29, 63, 116, 27, 62, 64]. Finally, the unbounded nature of data streams has been considered in [49] by mining from only a part of it, i.e., a *batch stream* in Apache Spark[14]. In [28] by pruning the formal concepts that did not have an occurrence of an object to them in a certain amount of time based on a parametric threshold λ. And lastly, in [63, 116, 62, 64] by considering a bounded part of the stream defined by a sliding window, like in [95].

During the assessment of the articles, we found that MRDM methods are still not enough addressed considering DA constraints and challenges (see the intersection between MRDM and DA in Figure 2.9). From the selected DA articles, only [63] and [116] are MRDM methods, both mining data stream as the input data type coming from DAs, but proposing a solution that treats it locally. The rest of the MRDM articles have, for the most part, RDBs as their main input type format (see Figure 2.13), which, although they could be used in DAs to store data, these articles have considered them only in non-DA related topics, and have proposed only local algorithms. Moreover, the articles not considering RDBs, worked with RCF, RDF, documents (text), or multi-relational graphs always in a non-distributed fashion, assuming that the whole structure is accessible at all times, which is often not the case in DA.

Furthermore, although CA methods had more involvement in the DA environment research (8 articles considering CA and DA i.e., [49, 28, 29, 26, 78, 27, 132, 141]), they represent still a small portion of the total. More particularly, as depicted in Figure 2.15, there were occurrences of distributed formal contexts being treated by methods to produce formal concepts, but only in the binary and traditional FCA. No distributed fuzzy formal context, nor other types of traditional CA inputs with their distributed variation, have been found in the search. In the following subsections, the identified scientific gaps are presented.

General CA method for infinite data streams

In the industry 4.0, several key components work in IoT architectures, which are inherently distributed. A big part of these architectures, use data streams as a common format for communication between their components ([24]). Moreover, some IoT architectures are designed to run infinitely, creating the need for handling certain tasks such as hot swapping ([122]). Toward those lines, in the area of CA methods, there are those that can handle the processing of data streams, usually called *incremental* ([28, 29, 132]) by only updating the output in the necessary parts with the arrival of each element of the stream. The main problem these methods have is that if the stream never stop producing elements, at some

[14] Apache Spark url.

point the update they do will be too costly. This can be solved by using sliding windows, or by removing concepts when they "get old" (e.g., no occurrence contributed to the extent of the concept in a defined time window) like in [28]. However, the problem these methods introduce is the loss of information i.e., in the sliding window case, transactions are lost independently of their meaning, while in the later case only the "oldest" are lost.

It is understandably unreasonable to expect a data mining method to be able to process an infinite amount of data without losing ***any*** information. Nevertheless, one of the questions that arises is *what to do when it is needed to recover* ***some*** *of the information lost*. Considering the mentioned incremental methods to process infinite data streams, one of the approaches that could be taken into account to fill the gap is the *merge* of lattices generated in different points in time [84], since this would allow regaining lost information on demand (e.g., depending on the available resources, or depending on the needs), taking advantage from not having to recalculate the stored lattices. This approach has different challenges, from which the highlights are (1) identify the properties of the data stream that could decrease (or increase) the cost of computing the merge, e.g., whether the attributes are known in advance, what are the structures that are already available to use without any computational cost. (2) Develop the theory to understand what are the differences between merging the stream parts and then computing the incremental algorithm on the merged data stream and directly merging the lattices.

General CA and MRDM method for infinite data streams

While some CA & MRDM methods directly use FCA and then add certain functions on top of it (e.g., RCA), others base their definition in the idea of *formal concepts*, but their computation is not necessarily related with the one in FCA (e.g., Triadic and Polyadic Concept Analysis). This makes the discussion provided in the section 2.7.9 to also hold for CA & MRDM methods, since the development of a *general* CA method for infinite data streams will not carry onto all its extensions naturally. For this reason, a general method on CA and MRDM could be considered an independent gap and has to be addressed in a specific research line, since dealing with heterogeneous sources adds many constraints and challenges to the mix, and an FCA method that allows infinite data stream processing will not be enough to cover all the needs in the MRDM domain regarding CA.

Deal with loss of information and semantics

Another challenge that arose from the article assessment is the standardization of information and semantic loss. On the one hand, in [74], Kuznesov proposed the definition of stability of a formal concept to measure the plausibility of concept-based hypothesis. On the other hand, in [28], De Maio et al. used a function that, together with the support, was used to determine how old a concept is in the case of fuzzy temporal formal concept analysis. These are two examples of how researchers decide how important (or the contrary) a concept is at a given moment. In order to support a flexible method that not only supports processing infinite data streams but also allows recovering certain information, it would be useful to have a way to measure the impact of removing a concept (or an object) from the lattice (or the formal context), as well as the impact of recovering a certain concept (or object).

Article ID	Article	Article ID	Article
1	[140]	36	[25]
2	[136]	37	[36]
3	[11]	38	[90]
4	[147]	39	[38]
5	[55]	40	[97]
6	[6]	41	[8]
7	[3]	42	[65]
8	[68]	43	[131]
9	[57]	44	[29]
10	[61]	45	[145]
11	[45]	46	[146]
12	[22]	47	[53]
13	[115]	48	[114]
14	[59]	49	[26]
15	[87]	50	[33]
16	[109]	51	[98]
17	[125]	52	[96]
18	[93]	53	[63]
19	[139]	54	[43]
20	[107]	55	[135]
21	[39]	56	[94]
22	[21]	57	[17]
23	[71]	58	[124]
24	[134]	59	[5]
25	[111]	60	[78]
26	[49]	61	[116]
27	[58]	62	[12]
28	[76]	63	[19]
29	[123]	64	[27]
30	[144]	65	[132]
31	[143]	66	[141]
32	[28]	67	[32]
33	[113]	68	[62]
34	[128]	69	[64]
35	[20]		

Table 2.6: Articles by ID

Data item	Value	RQ
General		
Article ID	DOI/ISBN	
Article Title	Name of the article	
Year of Publication	Calendar year	LRQ_1
Publication Venue	Name of publication venue	LRQ_1
Process		
Methods application	CA | MRDM | DA	LRQ_2 & LRQ_3 & LRQ_4
Addressed in	In which part of the DM process (Figure 2.6) the article is addressed.	LRQ_5
Domains	In which domains the methods are applied.	LRQ_6
Input of the method	Type of input, e.g., relational database, simple table, ontology.	LRQ_7
Output of the method	Different types of knowledge representations, e.g., association rules, ontologies.	LRQ_7
Evaluated characteristics	Evaluated characteristics of the article, e.g., accuracy, robustness.	LRQ_8
Semantic Interoperability	Formats that the article contributed to their semantic interoperability.	LRQ_9

Table 2.7: Data extraction fields divided in two subcategories: general and process.

Article	1.1.2	1.1.1
Concept discovery from un-constrained distributed context ([49])	X	X
Distributed online Temporal Fuzzy Concept Analysis for stream processing in smart cities ([28])	X	X
Making sense of cloud-sensor data streams via Fuzzy Cognitive Maps and Temporal Fuzzy Concept Analysis ([29])	X	X
Online query-focused twitter summarizer through fuzzy lattice ([26])		X
Regression on evolving multi-relational data streams ([63])		X
Two FCA-Based Methods for Reducing Energy Consumption of Sensor Nodes in Wireless Sensor Networks ([78])		X
xStreams: Recommending Items to Users with Time-evolving Preferences ([116])		X
Time Aware Knowledge Extraction for microblog summarization on Twitter ([27])		X
AddIntent: A New Incremental Algorithm for Constructing Concept Lattices ([132])		X
Distributed Formal Concept Analysis Algorithms Based on an Iterative MapReduce Framework ([141])	X	
Mining time-changing data streams ([62])		X
Learning model trees from evolving data streams ([64])		X

Table 2.8: Subclassification of articles according to whether they are implemented *using* a DA system 1.1.2, or they mine *from* a DA system 1.1.1.

Addressed problematic	**# of articles**
dm method	18
dm method, output	7
dm method, output, usages	1
dm method, usages	10
input	5
input, dm method	11
input, dm method, output	1
input, dm method, usages	2
input, usages	3
output	2
output, usages	1
preprocess, dm method	2
preprocess, dm method, output	1
usages	5

Table 2.9: *Problematic addressed in* multivalued field

3

Recovering relevant lost concepts

Contents

3.1 Introduction 45
3.1.1 Motivation 45
3.1.2 Preliminaries 46
3.2 Approach to mitigate the loss of concepts 49
3.3 Algorithms to merge concept lattices 52
3.3.1 Using existing approaches 52
3.3.2 Single-threaded merging algorithm 53
3.3.3 Parallelization 57
3.3.4 Implementation details and Complexity 62
3.4 Conclusions 64

3.1 Introduction

In this chapter, the problem of losing knowledge when creating a lattice from an infinite data stream is presented with a motivation and an example. Then, considering that it is impossible to get all knowledge condensed in a limited-sized lattice, a flexible and scalable approach for recovering a part of the lost knowledge is proposed. The approach consists on the storage of lattice snapshots to be used later in a merge step with the current lattice, depending on the needs of the practitioner. Regarding the computation of the merge lattices, an algorithm to compute the set of concepts of the merge between two lattices with disjoint objects is presented, and then contrasted with the existing ones in the literature. Finally, the need for distributing the snapshots and the merging step itself has been discussed and in that regard, three distributed and parallel algorithms are presented.

3.1.1 Motivation

Regarding data stream processing and distributed architectures, FCA has been applied in the context of smart cities [28], using techniques such as incremental algorithms [132], and sliding windows [95]. However, when we consider infinite data streams, where maintaining

a bounded-sized lattice is imperative, the process of doing so can lead to the loss of meaningful knowledge [84]. To illustrate this, let us consider the following situation (Example 3.1) in the context of data mining in smart cities,

Example. 3.1: Loosing concepts

In the algorithm proposed in [28], the lattice size is essentially controlled by the so-called *decay factor* value λ. The way it works, is by reducing the support of a concept based on how much time has passed since the last time an object has contributed to its extent, i.e., the more time since the last update, the bigger the support reduction. Formally, given a concept $C = \langle X, Y \rangle$ the support of a concept is computed as

$$\text{Supp}(C) = \frac{|X|}{|G|} e^{-\lambda(t - t_{lio})} \tag{3.1}$$

where t is the current timestamp and t_{lio} is the last timestamp in which an object has contributed to the concept C, i.e., has entered its extent.
Considering the context of the article, where the goal is to mine high level semantics events reported from sensors placed in a smart city, let us suppose a somewhat *rare* event such as a geomagnetic storm, or the likes, happening between the moments k and l. Assuming the data stream contains the updates and the data necessary to measure it, there would be a concept C in $\mathcal{L}_{k,l}$ corresponding to it. However, since that combination of fluctuations in the measures is rare, at some point in the future, say between m and n, where $k < m < n$ and $l < n$, C will be removed from $\mathcal{L}_{m,n}$ since no object (update in the stream) contributed to it. Thus, a potentially meaningful concept has been lost between m and n.

Furthermore, since data stream processing requires fast computations, it is more likely that the size of the lattice to be maintained will be dictated by *how much time is available to compute* between events in the data stream rather than *how much available space there is*. Consequently, a way of dealing with both the necessity of running infinitely, and the ability to regain a part of the lost information, is to consider an algorithm that keeps the size of the lattice bounded, and another algorithm that can perform the union or merge of the lattices computed in different points in time. Doing so would give the practitioners the flexibility to choose the size they are able to maintain given their resources, without the worry of losing *all* the information outside the lattice they are maintaining. This, however, comes at the cost of having to store the snapshots in a certain way.

3.1.2 Preliminaries

In order to speak about FCA in the context of infinite data streams, let us consider the following definitions,

Definition. 3.1: Formal Context data stream

Let G and M be the objects and attributes of a formal context respectively. The **content** of a *Formal Context data stream* in a given moment $k \in \mathbb{N}$ is defined as an indexed family $S_k = (r_j)_{j \in 0..k-1}$ where $r_j \subseteq G \times M$.

Definition. 3.2: Underlying Formal Context

Given a Formal Context data stream S_k, we define its underlying traditional (i.e., the traditional triplet of objects, attributes, and the incidence matrix) Formal Context $K = (G, M, I)$, where $G = \{\pi_1(r) \mid r \in S_k\}$, $M = \{\pi_2(r) \mid r \in S_k\}$, $gIm \iff \langle g, m\rangle \in S_k$, and both π_1 and π_2 are the functions that return the first and second element of the tuple respectively.

Definition. 3.3: Formal Context Snapshot

A **snapshot** of a Formal Context data stream between moments $k, l \in \mathbb{N}$ is the sequence of tuples $S_{k,l} = S_l \setminus S_k$. Notice that $l \leq k$ implies $S_{k,l} = \varnothing$. For convenience, we will use the notation $\mathcal{L}_{k,l}$ when speaking about the underlying lattice from $S_{k,l}$.

In particular, there are three cases we consider important to highlight for merging different lattice snapshots. The first one, depicted in Figure 3.1, is the one in which the goal is to merge $\mathcal{L}_{k,l}$ with $\mathcal{L}_{l+1,m}$. The second one, depicted in Figure 3.2, is the one where the goal is to merge $\mathcal{L}_{k,l}$ and $\mathcal{L}_{n,m}$ and $l+1 < n$, i.e., there is at least one element outside the covered range. And the last one, depicted in Figure 3.3, is the one in which the goal is to merge two snapshots $\mathcal{L}_{k,l}$, and $\mathcal{L}_{n,m}$, where $k \leq l$, $n \leq m$, and $n < l$, i.e., there are repeated elements in the two lattices. The separation in cases is thought with two things in mind (1) Computation: in case there is repeated information between the two snapshots, the computation cost of merging might be reduced in comparison with the case in which both snapshots are completely disjoint. (2) Interpretation: The result when merging two snapshots that are *not contiguous* might need a special interpretation (see Example 3.2).

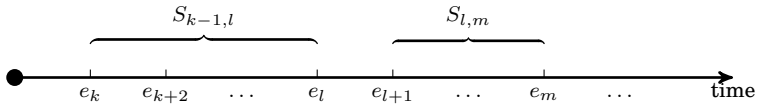

Figure 3.1: Contiguous lattice snapshots merge.

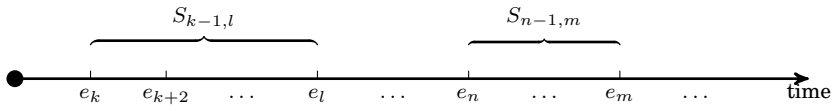

Figure 3.2: Spaced lattice snapshots merge case.

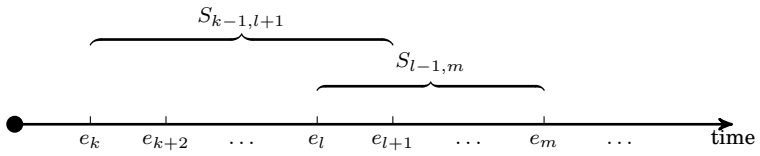

Figure 3.3: Intersected snapshots merge case.

Example. 3.2: Interpretation of Temporal Lattice when objects are lost

Let us consider the temporal extension of FCA proposed in [28], and based on [102]. This model combines FCA with Conceptual Time Systems by indexing the timestamped objects adding a time variable to the subject of the formal context, i.e., g_{t_i} with $g \in G$. In this definition, t represents a time window, $i \in \mathbb{N}$, and g_{t_i} is the latest observation g occurring in the time window assigned to that specific time variable. The larger the time window, the more general the study, the smaller, the more detailed it is. In particular,

- g represents a type of observation, e.g., change in temperature of the environment,
- t_i is the i-th element in a discretization of time with the time window t,
- g_{t_i} is the last occurrence of g in the time t_i,
- t_i precedes t_j if $i < j$ for any two $i, j \in \mathbb{N}$.

Before stating the different interpretations let us define the following concepts,

Definition. 3.4: Temporal Lattice

It is a pair $(\mathcal{L}, E^t)$ where $\mathcal{L} = (\mathcal{C}, \leq)$ is a concept lattice and E^t is a set of temporal edges (see Definition 3.5).

Definition. 3.5: Temporal Edge

(Based on the definition 3.7 in [28]) Given the set of concepts $\mathcal{C}$, a **Temporal Edge** $e^t_{ij} \in E^t$ is a pair $(C_i, C_j) \in \mathcal{C} \times \mathcal{C}$ iff there exist two objects $g_{t_s} \in C_i$ and $g_{t_k} \in C_j$ such that the time t_s precedes the time t_k.

The temporal edges have a weight associated with the time granule size function $\Delta t : E^t \to \mathbb{R}$ defined by $\Delta t(e^t_{ij}) = 1/|t_{s'} - t_{k'}|$. Furthermore, they define a function that filters edges that are not so representative, called Temporal Edge Support (TES). For the purpose of this example, we would consider the function, but without any particular definition.

Definition. 3.6: Temporal Path

(Definition 3.9 in [28]) A **temporal path** is defined as a path $\pi = (C_s, \ldots, C_t) \in \mathcal{C} \times \mathcal{C} \times \ldots \mathcal{C}$, such that there exist a temporal edge $e^t_{ij} \in E^t$ for $s \leq i \leq j \leq t$.

Considering these definitions, the *contiguous* and *intersected* merge cases should be interpreted as follows: after performing the merge, the resulting lattice represents the same as if the incremental algorithm had run from point k to m. However, that is not the case when there are elements in the middle that are lost between l and n. In OTM, there would be some g_{t_i} where g either has been previously introduced or not. On the one hand, if it was not previously introduced (i.e., $i = 1$) there would be no

impact in the temporal paths, besides losing the first part of them (see Figure 3.4). On the other hand, if g was previously introduced, it would simplify the path in the erased part by taking much less granular edges (see Figure 3.5).

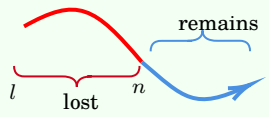

Figure 3.4: Temporal path after merging when g_{t_i} occurs for the first time between l and n

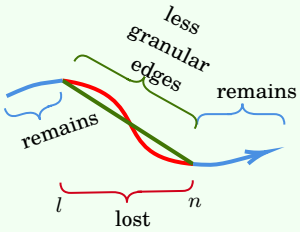

Figure 3.5: Temporal path after merging when g_{t_i} occurs before l

Regarding the model, the main problem is that although we could use incremental algorithms to calculate their lattices, at some point the k would be so large that even performing one more update would be too costly, either in time or in space. Thus, for a flexible method whose goal is to run infinitely, it is mandatory to consider "forgetting" objects, attributes, or concepts. Nevertheless, doing so implies potentially losing relevant information, meaning that it is equally necessary to have a way of merging different *snapshots* of the stream, as suggested in [49]. Although one option is to simply join the intervals, remove duplicates, and apply the algorithm to the result, it could be possible that there are not enough resources to run the algorithm with such an input. Moreover, considering that a lattice for each interval has already been computed, maybe working with them would be more efficient than recalculating everything from the ground up.

Considering this, we say that a method is *flexibly scalable* for knowledge extraction in data stream processing, if it allows running in the data stream infinitely, and it provides a way of considering "forgotten" information without the necessity to *recalculate* or *traverse* the whole lattice.

3.2 Approach to mitigate the loss of concepts

Considering that we have a system using a real-time-updated lattice to fulfil a certain purpose. If the data stream the lattice is being updated with is considered to be *infinite*, as discussed previously, it would certainly be using a technique to maintain the size of the lattice bounded, because otherwise, even one update will take more time than the time it takes for another event to happen in the data stream. Thus, the system will inevitably be using a lattice that may not have some relevant concepts. In this context, we propose to store lattice snapshots (see Definition 3.3) at carefully chosen ranges of time, in order to their later usage in combination with the current one. This will help not only recover the concepts learned in the specific moment of the snapshot, but also create links between them and the current ones.

In summary, the general method consists of having certain lattice snapshots stored, and when needed, to merge them with the current lattice to have a more complete concept hierarchy without compromising the process of online lattice update.

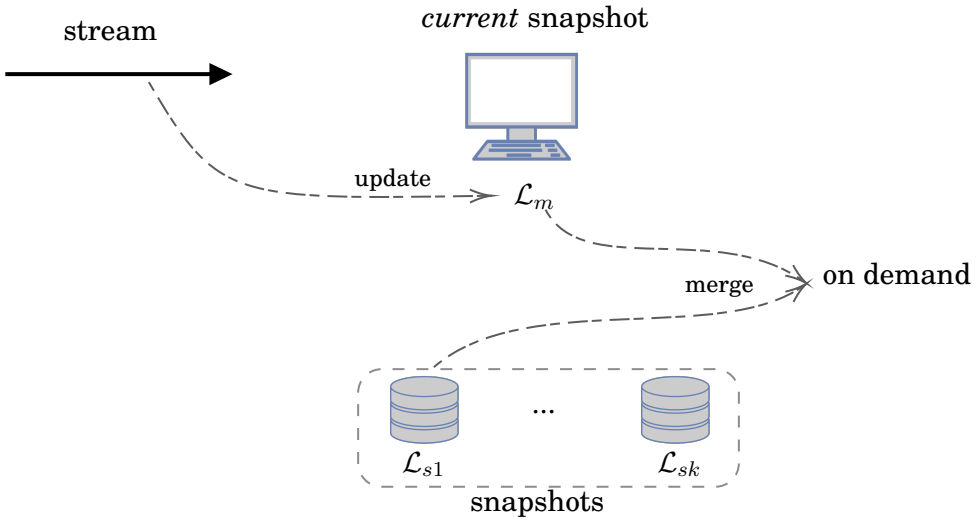

Figure 3.6: General method for recovering potentially lost concepts on demand.

Let us suppose a system that consists of a stateless[15] e-commerce website where customers fill certain required data, to then be shown several available ski lessons. Being stateless does not necessarily mean that the system does not store any information whatsoever, but in this particular case, it does not have specific information about the users. In this system, a feasible scenario is that a certain combination of parameters given by the user yield no available lessons. Hence, the goal of the recommendation system is to show to the user alternative lessons that are compatible with what they previously selected. However, since the only information available about the users when dealing with the requests is what they filled in the form (i.e., the information present in the request), the recommendation system is bound to use only those parameters in order to look for the most similar lessons to recommend.

One approach to deal with this task is to use a concept lattice of the latest k purchases and its association rules to guide the recommendations, as it would allow finding similar purchases based on their characteristics and the distance with concepts in the current lattice. In addition, since we are dealing with lattices, a merge algorithm could be used in order to recover already computed knowledge (concepts) that happened at certain key moments (depending on the needs of the practitioner or the business rules).

Example. 3.3:

Given the ski lesson purchases Formal Context, shown in Table 2.2, where each ski lesson purchase could be a group lesson (**GL**), a private lesson (**PL**), could be related to Alpine (**SKALPN**) or Nordic ski (**SKNORD**), it could have less than 3 people involved (**nbP**< 3), between 3 and 6 (**3**$\leq$**nbP**$<$**7**), or more than 6 (**7**$\leq$**nbP**), the following set of formal concepts can be derived (recall that p_i stands for the i-th purchase),

1. $\langle\{\mathrm{p}_1, \mathrm{p}_2, \mathrm{p}_3, \mathrm{p}_4\}, \varnothing\rangle$
2. $\langle\{\mathrm{p}_2, \mathrm{p}_3\}, \{\mathbf{PL}\}\rangle$

[15] All requests are independent, because the system does not have *sessions*, therefore, *a priori*, it does not know anything about the user making the request.

3. $\langle\{p_1, p_2, p_4\}, \{\textbf{SKNORD}\}\rangle$
4. $\langle\{p_3\}, \{\textbf{PL}, \textbf{SKALPN}, 7 \leq \textbf{nbP}\}\rangle$
5. $\langle\{p_1, p_4\}, \{\textbf{GL}, \textbf{SKNORD}\}\rangle$
6. $\langle\{p_2\}, \{\textbf{PL}, \textbf{SKNORD}, 3 \leq \textbf{nbP} < 7\}\rangle$
7. $\langle\{p_2, p_4\}, \{\textbf{SKNORD}, 3 \leq \textbf{nbP} < 7\}\rangle$
8. $\langle\{p_1\}, \{\textbf{GL}, \textbf{SKNORD}, \text{nbP} < 3\}\rangle$
9. $\langle\{p_4\}, \{\textbf{GL}, \textbf{SKNORD}, 3 \leq \textbf{nbP} < 7\}\rangle$
10. $\langle\varnothing, \{\textbf{GL, PL, SKALPN, SKNORD, nbP} < 3, 3 \leq \textbf{nbP} < 7, 7 \leq \textbf{nbP}\}\rangle$

And among the association rules that can be extracted, we can find:

- When there are less than 3 people involved, the activity of the lesson is Nordic Ski.
- Group Lessons are always Nordic Ski.
- When there are more than 7 people, the booked lesson is always a Private Lesson.
- Being between 3 and 7 people has a 50% chance of being a group or a private lesson.

Now, every time there is a new purchase, the lattice is incrementally updated, making room for new knowledge to emerge. For example, by adding one transaction to the dataset with the following characteristics: **GL**, **SKALPN**, $3 \leq$**nbP**< 7, the association rules would change accordingly:

- When there are less than 3 people involved, the activity of the lesson is Nordic Ski.
- ~~Group Lessons are always Nordic Ski.~~
- *A Group Lesson is Nordic Ski with 66% chance.*
- *A Group Lesson is Alpine Ski with 33% chance.*
- When there are more than 7 people, the booked lesson is always a Private Lesson.
- ~~Being between 3 and 7 people has a 50% chance of being a group or a Private Lesson.~~
- *Being between 3 and 7 people has a 33% chance of being a Private Lesson and a 66% chance of being a Group Lesson.*

When recommending, the latest k purchases concept lattice $\mathcal{L}$ is considered and merged with a certain lattice snapshot based on business logic, e.g., lattices are divided by resort. Then, the merged concept lattice $\mathcal{L}_m$ is used to generate the association rules having the characteristics in the request and having the highest confidence and support. This process flow is depicted in Figure 3.6.

The proposed method has certain limitations nonetheless. In particular, it requires additional *storage space* that will grow depending on how many snapshots the practitioner may need. Moreover, how to decide which time ranges to take snapshots is a major challenge and certainly goes outside the scope of this thesis. However, we can provide some guidelines nonetheless: first, following the hypothesis that the goal is to recover anomalies that at a certain time were impactful, the action to take is to store a snapshot of the current lattice at **that** specific time. For instance, following the Example 3.1, if suddenly there is a traffic jam and we do not know its cause, **that** might be a good time to take a snapshot to be used later when analyzing the lattice is needed, since it is very likely the concept related to the anomaly that caused it will be present there.

3.3 Algorithms to merge concept lattices

3.3.1 Using existing approaches

A trivial approach to compute $\mathcal{L}_m$ would be to simply join the contexts and then apply a lattice construction algorithm. Considering the lowest bound for an algorithm for constructing lattices being $\mathcal{O}(|\mathcal{L}||G|(|G| + |M|))$ [104]. Such an algorithm would cost in the worst case $\mathcal{O}(|\mathcal{L}_m||G_m|(|G_m| + |M_m|))$, where $G_m = G_1 \cup G_2$, and $M_m = M_1 \cup M_2$. However, *addintent* [132] has been empirically shown to work faster than the one mentioned before, even if it has a greater upper bound time complexity, i.e., $\mathcal{O}(|\mathcal{L}||G|^2 \max(|g'|))$. Moreover, in the *addintent* algorithm, (thoroughly explained in section 4.4), the first part of CreateLatticeIcrementally sets the bottom to be M and incrementally updates from there, for each addintent call, hence, without changing anything in the implementation, M_1 and M_2 ought to be equal. To allow M_1 and M_2 to be different, it suffices adding a function that adds attributes without any related object. This can be accomplished by simply adding all attributes to the bottom (a function that does that is included but not implemented in line 1 of the Algorithm IncrementalMerge). In case the bottom contains objects in its extent, create a new bottom with all attributes and the old bottom as a parent. Such an addition would not have an impact in the complexity of *addintent* considering all attributes. Thus, in order to use *addintent* to compute the merge between two lattices, the modified version should be used (see Algorithm IncrementalMerge). Notice that this version of the algorithm also adds the object to the respective extents. Moreover, notice that the algorithm receives two lattices and merges them adding object by object from the smallest lattice ($\mathcal{L}_{\text{small}}$), to the largest ($\mathcal{L}_{\text{large}}$). This is key to the main idea of leveraging the fact that a considerable part of the work has already been done, and that the algorithm needs to update only a few objects. Additionally, we can always assume that the first input is the largest and the second is the smallest because if that is not the case, we could simply swap them before calling the function.

Algorithm IncrementalMerge Merge $\mathcal{L}_{\text{large}} = \langle \mathcal{C}_1, \leq \rangle$ and $\mathcal{L}_{\text{small}} = \langle \mathcal{C}_2, \leq \rangle$

```
1: add_attributes(L_large, L_small)
2: for g ∈ L_small do
3:     attributes ← {m ∈ M_2 | gI_2m}
4:     addintent(g, attributes, L_large)
5: end for
6: return L_large
```

3.3.2 Single-threaded merging algorithm

Another way to compute a merge of lattices $\mathcal{L}_m = \text{merge}(\mathcal{L}_1, \mathcal{L}_2)$ is to do it in batch. Particularly, if only the set of concepts is needed, we presented the novel Algorithm BatchMerge in [84]. The way the algorithm works is by, first, computing the $\top$ and $\bot$ concepts and adding them to a set $\mathcal{C}_m$ (line 1). Second, it adds all concepts $C = \langle Y'^1 \cup Y'^2, Y \rangle$ such that Y is a non-empty intent in $\mathcal{L}_1$ or in $\mathcal{L}_2$ (lines 2-5). Then, for each pair of concepts $\langle C_1, C_2 \rangle$ such that $C_1 = \langle X_1, Y_1 \rangle \in \mathcal{L}_1$, $C_2 = \langle X_2, Y_2 \rangle \in \mathcal{L}_2$ and the intersection of their intents is not empty, it adds the concept $\{\langle Y'^1 \cup Y'^2, Y \rangle\}$ to the resulting set (lines 6-13).

Algorithm BatchMerge Merge $\mathcal{L}_1 = \langle \mathcal{C}_1, \leq \rangle$ and $\mathcal{L}_2 = \langle \mathcal{C}_2, \leq \rangle$

```
1: C_m ← {⊤, ⊥}
2: All_Intents ← {Y | ⟨X, Y⟩ ∈ C_1 ∪ C_2 ∧ X ≠ ∅ ∧ Y ≠ ∅}
3: for Y ∈ All_Intents do
4:     C_m ← C_m ∪ {⟨Y'^1 ∪ Y'^2, Y⟩}
5: end for
6: for ⟨X_1, Y_1⟩ ∈ C_1 do
7:     for ⟨X_2, Y_2⟩ ∈ C_2 do
8:         if Y_1 ∩ Y_2 ≠ ∅ then
9:             Y ← Y_1 ∩ Y_2
10:            C_m ← C_m ∪ {⟨Y'^1 ∪ Y'^2, Y⟩}
11:        end if
12:    end for
13: end for
14: return C_m
```

Proof of correctness

In the article [84], it is proven that, in Algorithm BatchMerge, all intents in $\mathcal{L}_m$ are correctly computed, via the Lemma 3.1 and Lemma 3.2. For readability, the algorithm and the lemmas are copied below.

Lemma. 3.1:

Let $K_1 = \langle G_1, M_1, I_1 \rangle, K_2 = \langle G_2, M_2, I_2 \rangle, K_s = K_1 \mid K_2$ be three formal contexts where $G_1 \cap G_2 = \varnothing$. Let $\mathcal{L}_1 = \langle \mathcal{C}_1, \leq \rangle$, $\mathcal{L}_2 = \langle (\mathcal{C}_2, \leq \rangle, \mathcal{L}_s = \mathcal{L}_1 \mid \mathcal{L}_2 = \langle \mathcal{C}_m, \leq \rangle$ be their respective lattices. Then, given a concept $\langle X, Y \rangle = C_1 \in \mathcal{C}_1$, such that $Y \neq \varnothing$ and $X \neq \varnothing$, there exist a $\langle Z, Y \rangle \in \mathcal{C}_m$, where $X \subseteq Z$.

Proof. For reading purposes, we will say $Y = Y_1 = Y_2 = Y_s$. Y_1' will be used when speaking about the derivation in the context of $\mathcal{L}_1$. Similarly, Y_2' will be used when speaking about the derivation in the context of $\mathcal{L}_2$, and Y_s' when speaking about the derivation in the context of $\mathcal{L}_s$. Analogously, we will use the same notation for the set of objects $X = X_1 = X_2 = X_s$.
Since (X, Y) is a formal concept in $\mathcal{C}_1$, $X_1' = Y$ and $Y_1' = X$.

1. If X_1 at least has one element, it would be an element of G_1, i.e., $g \in G_1$. Then $X_s' = Y$, because the merge does not change the attributes held by any of the G_1

objects.

2. $Y'_s = \{g \in G_1 \cup G_2 \mid gI_sm, \forall m \in Y_s\} = \{g \in G_1 \mid gI_sm, \forall m \in Y\} \cup \{g \in G_2 \mid gI_sm, \forall m \in Y\} = Y'_1 \cup Y'_2$. Then $Y''_s = (Y'_1 \cup Y'_2)' = (X \cup Y'_2)' = X' \cap Y''_2$.

3. Since $Y'_s = Y'_1 \cup Y'_2$ (because of step 2), then $Y'_1 \subseteq Y'_s$. Therefore, $X \subseteq Y'_s$.

4. Since $X' = X'_s = Y$ (because of the step 1)., $Y''_s = X' \cap Y''_2 = Y \cap Y''_2 \implies Y''_s \subseteq Y$.

5. Since $Y \neq \varnothing$, let $m \in Y$. Let us suppose that $m \notin Y''_s$, then, there exist an object $x \in Y'_s$ such that $x \not I_s m$, which is **absurd** because $Y'_s = \{g \in G_1 \mid gI_sm, \forall m \in Y\} \cup \{g \in G_2 \mid gI_sm, \forall m \in Y\}$, $I_s = I_1 \cup I_2$ and objects are disjoint, thus gI_1m implies gI_sm. Analogously, gI_2m implies gI_sm. Therefore, $m \in Y''_s$, which means $Y \subseteq Y''_s$.

6. Since $Y''_s \subseteq Y$ and $Y \subseteq Y''_s$, then $Y = Y_s = Y''_s$ (steps 4 and 5). Thus, $(Z, Y) = (Y'_s, Y_s) \in \mathcal{C}_s$, and $X \subseteq Z$ (step 3).

□

Lemma. 3.2:

Let $K_1 = \langle G_1, M_1, I_1\rangle, K_2 = \langle G_2, M_2, I_2\rangle, K_s = K_1 \mid K_2$ be three formal contexts where $G_1 \cap G_2 = \varnothing$. Let $\mathcal{L}_1 = \langle \mathcal{C}_1, \leq\rangle, \mathcal{L}_2 = \langle \mathcal{C}_2, \leq\rangle, \mathcal{L}_s = \mathcal{L}_1 \mid \mathcal{L}_2 = \langle \mathcal{C}_m, \leq\rangle$ be their respective lattices. Let $\langle X, Y\rangle \in \mathcal{C}_m, Y \neq \varnothing, Y \neq M_1 \cup M_2$ be a formal concept such that its intent Y is not empty nor the whole set of attributes, and it is not an intent of $\mathcal{C}_1$ nor in $\mathcal{C}_2$, then $Y = Z_1 \cap Z_2$ where Z_1 is an intent in $\mathcal{C}_1$ and Z_2 is an intent in $\mathcal{C}_2$.

Proof. Let $Y'_1 = \{g \in G_1 \mid gI_1m, \forall m \in Y\}$ and $Y'_2 = \{g \in G_2 \mid gI_2m, \forall m \in Y\}$. By definition $Y' = Y'_1 \cup Y'_2$. Since Y is not an intent in $\mathcal{C}_1$ nor in $\mathcal{C}_2$, both Y'_1 and Y'_2 yield a set of objects whose derivatives are larger than Y, i.e., $Y \subset Y''_1$ and $Y \subset Y''_2$. Then $Y = Y''_1 \cap Y''_2$. Since Y is an intent of $\mathcal{C}_s$, $Y = Y''$ and $X = X''$. Moreover, $Y'_1 \subset X$ and $Y'_2 \subset X$ because otherwise, their derivative could not possibly yield more attributes. Let us suppose that Y''_1 is not an intent of $\mathcal{C}_1$, then $Y''_1 \subset Y''''_1$ which is absurd because $B' = B'''$ for any attribute set in the same context (proofed in section 1.1, Proposition 10 of [42]). Similarly, Y''_2 is an intent in $\mathcal{C}_2$. If we rewrite $Y''_1 = Z_1$ and $Y''_2 = Z_2$, we have that $Y = Z_1 \cap Z_2$. □

In summary, Lemma 3.1 expresses that all intents present either in $\mathcal{L}_1$ or $\mathcal{L}_2$ are also present in $\mathcal{L}_m$, whereas Lemma 3.2 says that any intent in the merged lattice that is not present in either of the lattices $\mathcal{L}_1$ or $\mathcal{L}_2$, is the intersection of two intents $Y_1 \in \mathcal{C}_1, Y_2 \in \mathcal{C}_2$. However, these two lemmas alone are not enough to prove the full correctness of the algorithm. To be able to say that it computes exactly $\mathcal{C}_m$, i.e., the set of all formal concepts in the merged lattice, it is also needed to show that **all** non-empty intersections of intents in $\mathcal{C}_1$ and $\mathcal{C}_2$ respectively are also intents in $\mathcal{C}_m$, since otherwise between line 6 and line 13 it would be possible to add an intent not belonging to $\mathcal{C}_m$. In the following, we present a lemma, two corollaries, and a theorem to complete the correctness proof.

Before stating the following claims, let us establish the notation that will be used in the respective proofs. Given a lattice $\mathcal{L}_x$ from a formal context $K_x = \langle G_x, M_x, I_x\rangle$ and a set of

attributes Y, we define the derivation of Y with respect to $\mathcal{L}_x$ as,

$$Y^{'x} = \{g \in G_x \mid \forall m \in Y,\ gI_x m\} \tag{3.2}$$

Analogously, the derivation of the set of objects X with respect to $\mathcal{L}_x$ is defined as,

$$X^{'x} = \{m \in M_x \mid \forall g \in X,\ gI_x m\} \tag{3.3}$$

Now, let us complete the correctness proof,

Lemma. 3.3:

Let $K_1 = \langle G_1, M_1, I_1\rangle, K_2 = \langle G_2, M_2, I_2\rangle, K_s = K_1 \mid K_2$ be three formal contexts where $G_1 \cap G_2 = \varnothing$. Let $\mathcal{L}_1 = \langle \mathcal{C}_1, \leq\rangle$, $\mathcal{L}_2 = \langle \mathcal{C}_2, \leq\rangle, \mathcal{L}_s = \mathcal{L}_1 \mid \mathcal{L}_2 = \langle \mathcal{C}_m, \leq\rangle$ be their respective lattices. Let $\langle X_1, Y_1\rangle \in \mathcal{C}_1$, $\langle X_2, Y_2\rangle \in \mathcal{C}_2$ be two formal concepts such that $Y_1 \cap Y_2 \neq \varnothing \wedge Y_1 \cap Y_2 \neq M_1 \cup M_2$, then $Y = Y_1 \cap Y_2$ is an intent in C_m.

Proof. Let $Y = Y_1 \cap Y_2$,
On the one hand,

$$\begin{aligned} Y^{'m} &= \{g \in G_1 \cup G_2 \mid \forall a \in Y,\ gI_m a\} \\ &= \{g \in G_1 \mid \forall a \in Y,\ gI_m a\} \cup \{g \in G_2 \mid \forall a \in Y,\ gI_m a\} \\ &= Y^{'1} \cup Y^{'2} \end{aligned} \tag{3.4}$$

On the other hand,

$$\begin{aligned} Y^{'m'm} &= (Y^{'1} \cup Y^{'2})^{'m} \\ &= Y^{'1'm} \cap Y^{'2'm} \\ &= \{a \in M_1 \cup M_2 \mid \forall g \in Y^{'1},\ gI_m a\} \cap \\ &\quad \{a \in M_1 \cup M_2 \mid \forall g \in Y^{'2},\ gI_m a\} \\ &= \{a \in M_1 \cup M_2 \mid \forall g \in (Y_1^{'1} \cup Y_2^{'1}),\ gI_m a\} \cap \\ &\quad \{a \in M_1 \cup M_2 \mid \forall g \in (Y_1^{'2} \cup Y_2^{'2}),\ gI_m a\} \\ &= (Y_1^{'1} \cup Y_2^{'1})^{'m} \cap (Y_1^{'2} \cup Y_2^{'2})^{'m} \\ &= Y_1^{'1'm} \cap Y_2^{'1'm} \cap Y_1^{'2'm} \cap Y_2^{'2'm} \\ &= Y_1 \cap Y_2^{'1'm} \cap Y_1^{'2'm} \cap Y_2 \end{aligned}$$

(since Y_1 is an intent in $\mathcal{C}_1$ and vice versa)

Then, $Y^{'m'm} = Y_1 \cap Y_2^{'1'm} \cap Y_1^{'2'm} \cap Y_2 \subseteq Y_1 \cap Y_2 = Y$. Since the closure operation $''$ is increasing, i.e., $X \subseteq X''$ for any X, then $Y \subseteq Y^{'m'm}$. Therefore, by double inclusion, $Y = Y^{'m'm}$. □

Corollary. 3.1:

Let $K_1 = \langle G_1, M_1, I_1 \rangle, K_2 = \langle G_2, M_2, I_2 \rangle, K_s = K_1 \mid K_2$ be three formal contexts where $G_1 \cap G_2 = \varnothing$. Let $\mathcal{L}_1 = \langle \mathcal{C}_1, \leq \rangle, \mathcal{L}_2 = \langle \mathcal{C}_2, \leq \rangle, \mathcal{L}_s = \mathcal{L}_1 \mid \mathcal{L}_2 = \langle \mathcal{C}_m, \leq \rangle$ be their respective lattices. Then, Algorithm BatchMerge computes exactly all intents in $\mathcal{C}_m$.

Proof. Regarding the bottom and top concepts, Algorithm BatchMerge computes them in line 1. Furthermore, the algorithm adds to the result all intents in $\mathcal{C}_1$ and $\mathcal{C}_2$ (except for the bottom and top ones) between lines 3 and 5. Then, by Lemma 3.1, if an intent is not computed by the algorithm, it must be an intent that is not present in $\mathcal{C}_1$ or $\mathcal{C}_2$. Following, because of Lemma 3.2, the only possible intents that match that description are of the form $Y = Y_1 \cap Y_2$ where $Y_1 \in \mathcal{C}_1$ and $Y_2 \in \mathcal{C}_2$. However, the algorithm adds all intersections between $\mathcal{C}_1$ and $\mathcal{C}_2$ in the lines 6-13. Hence, all intents are present in $\mathcal{C}_m$. Additionally, because of Lemma 3.3, all intersection of intents $Y_1 \cap Y_2$ such that $Y_1 \in \mathcal{C}_1$ and $Y_2 \in \mathcal{C}_2$ are intents in $\mathcal{C}_m$, lines 6-13 could not have added any entry that is not an intent in $\mathcal{C}_m$. □

Corollary. 3.2:

Let $K_1 = \langle G_1, M_1, I_1 \rangle, K_2 = \langle G_2, M_2, I_2 \rangle, K_s = K_1 \mid K_2$ be three formal contexts where $G_1 \cap G_2 = \varnothing$. Let $\mathcal{L}_1 = \langle \mathcal{C}_1, \leq \rangle, \mathcal{L}_2 = \langle \mathcal{C}_2, \leq \rangle, \mathcal{L}_s = \mathcal{L}_1 \mid \mathcal{L}_2 = \langle \mathcal{C}_m, \leq \rangle$ be their respective lattices. Then, Algorithm BatchMerge computes exactly all extents in $\mathcal{C}_m$.

Proof. By Corollary 3.1, we know the algorithm computes exactly all intents. For each intent Y, the computed extent is $X = Y'^1 \cup Y'^2$, which is equal to Y'^m (Equation 3.4). Then,

$$\begin{aligned} X'^{m'm} &= Y'^{m'm'm} \\ &= Y'^m && (Y \text{ is an intent of } \mathcal{C}_m) \\ &= X \end{aligned}$$

$\therefore X$ is the extent of the formal concept (X, Y). □

Theorem. 3.1: The BatchMerge algorithm is correct.

Let $K_1 = \langle G_1, M_1, I_1 \rangle, K_2 = \langle G_2, M_2, I_2 \rangle, K_s = K_1 \mid K_2$ be three formal contexts where $G_1 \cap G_2 = \varnothing$. Let $\mathcal{L}_1 = \langle \mathcal{C}_1, \leq \rangle, \mathcal{L}_2 = \langle \mathcal{C}_2, \leq \rangle, \mathcal{L}_s = \mathcal{L}_1 \mid \mathcal{L}_2 = \langle \mathcal{C}_m, \leq \rangle$ be their respective lattices. Then, Algorithm BatchMerge computes exactly the set of formal concepts $\mathcal{C}_m$.

Proof. Let $(X, Y) \in \mathcal{P}(G_1 \cup G_2) \times \mathcal{P}(M_1 \cup M_2)$ be a pair in the $\mathcal{C}_m$ computed by Algorithm BatchMerge. Then, by Corollary 3.1, Y is an intent, and X is its corresponding extent, by Corollary 3.2. In other words, $Y'm = X$ and $X' = Y$. Hence, (X, Y) is a formal concept. Since all intents and all extents are computed, then the returning set of the algorithm is exactly the set of all formal concepts. □

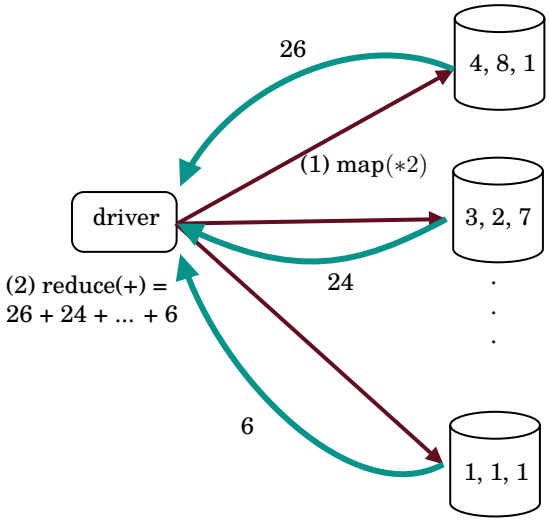

Figure 3.7: Basic Map Reduce example.

3.3.3 Parallelization

As already mentioned in this thesis, the problem of computing an entire concepts lattice is believed to be *P*-complete, or in other words, not efficiently parallelizable. This means that looking for algorithms that, by parallelization, could improve the upper bound of known algorithms by more than a constant has a great interest in this research field, since whether $P = NC$ is on the line. However, doing so is believed to be near impossible. Having said that, this does not mean that taking advantage of parallelization models is not worth it, in fact, even if a parallel algorithm only improves the performance by a constant, it can be considered useful in a practical sense. In that vein, this subsection proposes a solution that takes advantage of a distributed architecture in order to achieve parallelization.

To conceive the algorithm, the MapReduce computation model will be used, thus, is firstly needed to set a common ground on its key notions. Generally, we can think of it as a computational model where two operations can be performed: *map* and *reduce*. Map refers to a function applied to every element of the input, and reduce is the combination of the results performed by the map step. Typically, the *map* is the operation that is expected to leverage concurrency, while *reduce* is the one that "combines" the results computed in several nodes. In Figure 3.7, a basic example is given, where a driver node first performs a map$(*2)$ which makes each node to multiply every value (assuming they are all numbers) by two. Then, it performs a reduce$(+)$ which combines the results of the map, returning the sum of the results performed in each of the nodes.

To implement the algorithm, Apache Spark[16] will be used, and for that reason, first we will introduce certain concepts. The way the framework achieves parallelism is by using *transformations* (map) and *actions* (reduce). The core abstraction used in the framework is a read-only collection of data that can be partitioned across a subset of Spark cluster machines and form the main working component, which is widely called Resilient Distributed Dataset (RDD). An RDD can be thought as an Abstract Data Type (ADT) that provides a certain interface. To name a few examples, see Table 3.1. For more details, it would be

[16]https://spark.apache.org/

recommended to the reader to refer to the RDD official guide[17].

Name	Parameters	Explanation
		Transformations
map	RDD,f	Apply a function to every element
filter	RDD,f	Filter element based on a specific boolean function
distinct	RDD	Returns a new RDD without duplicates
intersection	RDD, RDD	Returns a new RDD with the elements in common given two RDDs
cartesian	RDD,RDD	When called on two datasets A, B of types T and U respectively, returns the dataset $A \times B$, which has a type $\langle T, U \rangle$.
flatMap	RDD,f	When called on a dataset with elements of type T, it applies $f : T \rightarrow List[U]$ and returns a new dataset with elements of type U.
		Actions
reduce	RDD,f	Aggregate the elements of the dataset using a function f
collect	RDD	Returns a *list* with all elements into memory

Table 3.1: Some of the Apache Spark functions.

In order to merge lattices using Apache Spark, it is necessary to define how is a lattice defined in terms of RDDs. For the purpose of this work, only the set of concepts will be considered as such, mainly based on the work of Goel and Chaudhary [49]. In particular, the RDD used as a set of concepts is a key-value collection, where each element is a pair $\langle k, v \rangle$ representing a formal concept, k represents its *intent* as a sorted tuple, whereas v is the list of objects representing its *extent*.

Example. 3.4:

To illustrate how the formal concepts are distributed using Apache Spark, let us consider the ones coming from Table 2.2, having been enumerated in Example 3.3. Let us also consider that the Apache Spark cluster instance has 3 workers, and thus, the RDD is distributed across them. Although there are certain key parameters that will determine how an RDD is distributed, such as the *number of partitions*, they will not be discussed in this work. Considering that, a possible way in which the set of formal concepts can be distributed is depicted in Figure 3.8, where the attributes are represented from the numbers 0 to 6, being **GL**$= 0$, **PL**$= 1$ and so forth.

Map reduce algorithm considering only intents

In this context, the algorithms that will be considered take two key-value RDDs, say rdd_1 and rdd_2, representing a set of formal concepts as described in Example 3.4. First, if we focus only on the intents, we could first *map* the two RDDs getting their keys. Then, it would be enough to use the *Cartesian* map function between the two resulting RDDs to have a new one with the pairs of intents $\langle Y_1, Y_2 \rangle$. Finally, based on Theorem 3.1, it would be enough to

[17] https://spark.apache.org/docs/latest/rdd-programming-guide.html

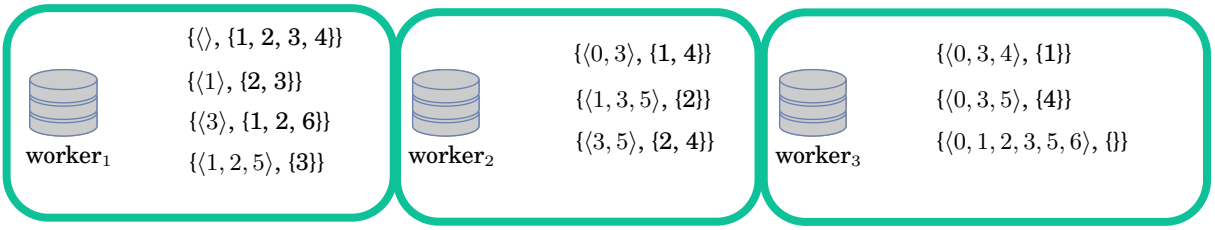

Figure 3.8: RDD example with 3 workers showing a possible distribution of the formal concepts coming from Table 2.2.

flatMap the Cartesian product of intents with the function $\lambda\, Y_1, Y_2.\ \{Y_1, Y_2, Y_1 \cap Y_2\}$[18] if the intersection is not empty nor equal to either Y_1 or Y_2. In terms of MapReduce, the algorithm can be written like in Algorithm ParallelMergeOnlyIntents (recall that given a tuple, the function π_i returns its i-th element). Lines 1 and 2 compute two RDDs containing only the intents. Then, line 3 defines a function that, given a pair of intents, returns exactly the two intents and their intersection in case it is non-empty and different to Y_1 and Y_2. Finally, the line 12 returns a new RDD resulting from applying MERGE_INTENTS to each element of the Cartesian product of the two sets of intents.

Algorithm ParallelMergeOnlyIntents Merge intents of rdd_1 and intents of rdd_2

```
1:  intents_1 ← rdd_1.map(λ P. π_1(P))
2:  intents_2 ← rdd_2.map(λ P. π_1(P))
3:  function MERGE_INTENTS(p)
4:      Y_1 ← π_1(p)
5:      Y_2 ← π_2(p)
6:      res ← {Y_1, Y_2}
7:      if Y_1 ∩ Y_2 ≠ ∅ ∧ Y_1 ∩ Y_2 ≠ Y_1 ≠ Y_2 then
8:          res ← res ∪ {Y_1 ∩ Y_2}
9:      end if
10:     return res
11: end function
12: return intents_1.cartesian(intents_2).flatMap(MERGE_INTENTS).distinct()
```

Now, remembering the section 2.6, where it is stated that the set of all intents corresponds to the frequent itemsets, Algorithm ParallelMergeOnlyIntents can be used as a batch merge algorithm to retrieve frequent itemsets when the whole set is necessary. However, contrary to Algorithm BatchMerge, it does not compute the extents which can be crucial for some applications.

Map reduce algorithm considering extents

In order to compute the extents as well, while there is no guarantee that, given two formal concepts $\langle X_1, Y_1\rangle$ and $\langle X_2, Y_2\rangle$, the *intersection* concept is $\langle X_1 \cup X_2, Y_1 \cap Y_2\rangle$ is actually a formal concept, i.e., $X_1 \cup X_2 = (Y_1 \cap Y_2)'^m$, because $(Y_1 \cap Y_2)'^m$ can have objects that are not

[18] In computer science, for its simplicity, it is common to use lambda calculus for *inline* functions, whose grammar corresponds to *expr* := *var* | λ *var*. *expr* | *expr expr*.

present in $X_1 \cup X_2$, by doing the Cartesian product, every extent will be correctly computed nonetheless as it is proven below,

Theorem. 3.2: MERGE_CONCEPTS on the Cartesian product of lattices to be merged will yield at least a correct concept.

Let $C = \langle X, Y \rangle$ be a formal concept in $\mathcal{L}_m$ such that $Y \neq \varnothing$ and it can be written as $Y_1 \cap Y_2$ for two intents in $\mathcal{L}_1$ and $\mathcal{L}_2$ respectively. Then, at least one concept $\langle X_1 \cup X_2, Y_1 \cap Y_2 \rangle$ for each pair of concepts $\langle X_1, Y_1 \rangle$, and $\langle X_2, Y_2 \rangle$ in the Cartesian product of concepts in $\mathcal{L}_1$ and $\mathcal{L}_2$ will be equal to C.

Proof. Let us suppose that for all pair of concepts $C_1 = \langle X_1, Y_1 \rangle$, $C_2 = \langle X_2, Y_2 \rangle$, it happens that $\langle X_1 \cup X_2, Y_1 \cap Y_2 \rangle \neq C$. Since by hypothesis we know that $Y \neq \varnothing$ and it can be written as $Y_1 \cap Y_2$, then it must be because $X_1 \cup X_2 \neq X$. Since Y_1 and Y_2 are intents in their respective lattices and their intersection is Y, we know that all objects in X_1 and X_2 share Y, then certainly $X_1 \cup X_2 \subseteq X$. Consequently, the inequality must come from the fact that there is an object $g \in X$ such that $g \notin X_1 \cup X_2$. However, since $\langle X, Y \rangle$ is a concept in $\mathcal{L}_m$, we know that $gI_m a$ for all $a \in Y$. Moreover, since all pairs in the Cartesian products are being considered, let us take specifically the concepts $\langle X_1, Y_1 \rangle$ and $\langle X_2, Y_2 \rangle$ where Y_1 and Y_2 are the generators of Y. Then, since $Y \subseteq M_1$ and $Y \subseteq M_2$, g must be either in X_1, or X_2, which is absurd because that would mean exactly the opposite to what we assumed.
$\therefore \langle X_1 \cup X_2, Y_1 \cap Y_2 \rangle = C$, for some pair of concepts $C_1 = \langle X_1, Y_1 \rangle$, $C_2 = \langle X_2, Y_2 \rangle$ in the Cartesian product. □

On the other hand, it is important to notice that computing ${Y_1'}^m \cup {Y_2'}^m$ could be trivially done when having access to RDDs containing the pairs $\langle g, m \rangle$ corresponding to each of the sets of formal concepts. However, depending on the ratio of the number of formal concepts and the amount of pairs in the data streams, it could be convenient to follow the strategy of traversing the Cartesian product instead, because, while many settings fall into the category of "the amount of formal concepts is considerably larger the size of the formal context", there are some settings where the opposite could happen. Considering this latter assumption, and the property proven in Theorem 3.2, and Theorem 3.1, we propose the Algorithm ParallelMergeConcepts. The way the algorithm works is by defining the function MERGE_CONCEPTS, with two concepts in $\mathcal{L}_1$ and $\mathcal{L}_2$ respectively as an input, and a list of concepts computed between the lines 5-17. Then, it uses the *Cartesian* function provided by Apache Spark between the two RDDs corresponding to $\mathcal{L}_1$ and $\mathcal{L}_2$, to then apply the function *reduceByKey* which will select only the extent with most elements in the Cartesian product for each pair with the same intent (i.e., largest$(X_1, X_2) = X_1$ if $|X_1| \geq X_2$, and largest$(X_1, X_2) = X_2$ otherwise). Finally, with a *map* function, it inverts the pairs so that they again have the extent in the first component and the intent in the second one.

Reduce the amount of elements to consider in the Cartesian product

Another approach that can be taken into account is one that is closer to the Algorithm BatchMerge, where *firstly* all intents in $\mathcal{L}_1$ and $\mathcal{L}_2$ are considered, and then only the ones that have a non-empty intersection between their intents. This approach would have the advantage that the Cartesian product will happen only between the necessary intents, in-

Algorithm ParallelMergeConcepts Merge two sets of formal concepts rdd_1 and rdd_2

function MERGE_CONCEPTS(p)
    $\langle X_1, Y_1 \rangle \leftarrow \pi_1(p)$
    $\langle X_2, Y_2 \rangle \leftarrow \pi_2(p)$
    $res \leftarrow \varnothing$
    if $Y_1 \cap Y_2 = \varnothing$ **then**
        $res \leftarrow res \cup \{\langle Y_1, X_1 \rangle, \langle Y_2, X_2 \rangle\}$
    else if $Y_1 = Y_2$ **then**
        $res \leftarrow res \cup \{\langle Y_1, X_1 \cup X_2 \rangle\}$
    else
        if $Y_1 \cap Y_2 = Y_1$ **then**
            $res \leftarrow res \cup \{\langle Y_2, X_2 \rangle, \langle Y_1, X_1 \cup X_2 \rangle\}$
        else if $Y_1 \cap Y_2 = Y_2$ **then**
            $res \leftarrow res \cup \{\langle Y_1, X_1 \rangle, \langle Y_2, X_1 \cup X_2 \rangle\}$
        else
            $res \leftarrow res \cup \{\langle Y_1, X_1 \rangle, \langle Y_2, X_2 \rangle, \langle Y_1 \cap Y_2, X_1 \cup X_2 \rangle\}$
        end if
    end if
    return res
end function
return rdd_1.cartesian(rdd_2)
        .flatMap(MERGE_CONCEPTS)
        .reduceByKey($\lambda\ X_1, X_2.$ largest(X_1, X_2)) ▷ Returns the largest extent in $\mathcal{O}(1)$.
        .map($\lambda\ p.\ \langle \pi_2(p), \pi_1(p) \rangle$) ▷ Reverse the elements of each pair.

stead of all of them. To do so, Algorithm ParallelMergeConceptsWithUnion, which reduces the amount of elements to apply the MERGE_CONCEPTS function, is proposed. In the lines 1 and 2, pairs in p_rdd_1 and p_rdd_2 are reversed so that the intent can be used as the keys[19]. The set of all intents is computed in line 3. Then, between lines 4-8, rdd_1 and rdd_2 are filtered so that only intents containing at least one element in the common attributes are considered. Between lines 9-17 the function for merging pairs in the Cartesian product is defined. Finally, the union of the elements in the Cartesian product between the two filtered RDDs having been merged and all intents is computed, reduced by intent with largest of their extents, and then reversed so that the pairs are in the form of $\langle extent, intent \rangle$ again.

Algorithm ParallelMergeConceptsWithUnion Merge two sets of formal concepts p_rdd_1 and p_rdd_2

1: $\text{rdd}_1 \leftarrow \text{p\_rdd}_1.\text{flatMap}(\lambda\ p.\ \langle \pi_2(p), \pi_1(p) \rangle)$
2: $\text{rdd}_2 \leftarrow \text{p\_rdd}_2.\text{flatMap}(\lambda\ p.\ \langle \pi_2(p), \pi_1(p) \rangle)$
3: $all\_intents \leftarrow \text{rdd}_1.\text{union}(\text{rdd}_2)$
4: $attributes_1 \leftarrow \text{rdd}_1.\text{flatMap}(\lambda\ p.\ \pi_2(p))$
5: $attributes_2 \leftarrow \text{rdd}_2.\text{flatMap}(\lambda\ p.\ \pi_2(p))$
6: $all\_attributes \leftarrow attributes_1.\text{intersection}(attributes_2)$
7: $filtered\_rdd_1 \leftarrow \text{rdd}_1.\text{filter}(\lambda\ p.\ (\pi_1(p) \cap all\_attributes) \neq \varnothing)$
8: $filtered\_rdd_2 \leftarrow \text{rdd}_2.\text{filter}(\lambda\ p.\ (\pi_1(p) \cap all\_attributes) \neq \varnothing)$
9: **function** MERGE_FILTERED_CONCEPTS(p)
10: $\quad \langle X_1, Y_1 \rangle \leftarrow \pi_1(p)$
11: $\quad \langle X_2, Y_2 \rangle \leftarrow \pi_2(p)$
12: $\quad res \leftarrow \varnothing$
13: $\quad$ **if** $Y_1 \cap Y_2 \neq \varnothing$ **then**
14: $\quad\quad res \leftarrow res \cup \{\langle Y_1 \cap Y_2, X_1 \cup X_2 \rangle\}$
15: $\quad$ **end if**
16: $\quad$ **return** res
17: **end function**
18: **return** $filtered\_rdd_1.\text{cartesian}(filtered\_rdd_2)$
$\quad .\text{flatMap}(\text{MERGE\_FILTERED\_CONCEPTS})$
$\quad .\text{union}(all\_intents)$
$\quad .\text{reduceByKey}(\lambda\ X_1, X_2.\ \text{largest}(X_1, X_2))$
$\quad .\text{map}(\lambda\ p.\ \langle \pi_2(p), \pi_1(p) \rangle)$

3.3.4 Implementation details and Complexity

Algorithm IncrementalMerge calls *addintent* $|G_2|$ times which yield a computational time complexity of $O(|\mathcal{L}_m||G_m||G_2|\ \max(|g'|))$, as opposed to calling *addintent* from scratch (i.e., once for every object in G_m) which would have a complexity of $O(|\mathcal{L}_m||G_m|^2\ \max(|g'|))$. Since $|G_2| \leq |G_m|$, then $O(|\mathcal{L}_m||G_m||G_2|\ \max(|g'|)) \subseteq O(|\mathcal{L}_m||G_m|^2\ \max(|g'|))$, which attains to the goal of Algorithm IncrementalMerge, which was to leverage the fact the largest lattice has already been computed.

Algorithm BatchMerge computes the bottom and top formal concepts separately, which

[19]This is so to have a clear similarity with Algorithm BatchMerge, however, depending on the dimensions, the extents could be used as the keys instead.

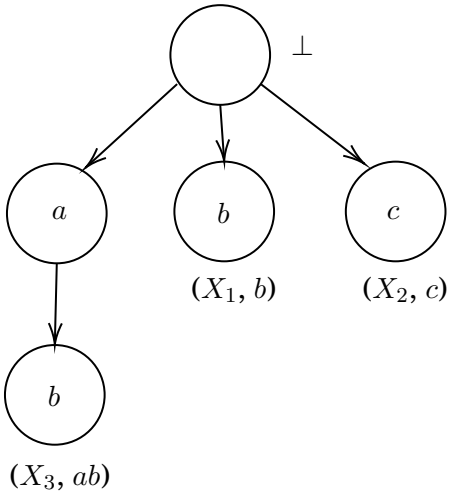

Figure 3.9: Example of the IntentsDict structure to accomplish the look by intent in worst case scenario in $\mathcal{O}(k \max(|g'^{m}|))$

has a computational cost of $\mathcal{O}(|G_1 \sqcup G_2||M_1 \cup M_2|)$. To compute lines 3-5, for each concept $C_2 = \langle X_2, Y \rangle$ in $\mathcal{L}_2$, it looks for the canonical generator of Y, $C_1 = \langle X_1, Z \rangle$ in $\mathcal{L}_1$, which can be done in $\mathcal{O}(|G_1| \max(|g'^{1}|))$, and add the pair $C_m = \langle X_1 \cup X_2, Y \rangle$ to a *trie* data structure called IntentsDict. In case no such C_1 exists, it simply adds $C_m = \langle X_2, Y \rangle$. The main purpose of the trie is to be able to retrieve concepts by intent in $\mathcal{O}(k \max(|g'^{m}|))$, where k is the size of the largest attribute, assuming they are strings. The way this data structure works is by considering a total order of the attributes (e.g., lexicographical order), and then a formal concept $\langle X, Y \rangle$, where $Y = m_1, \ldots, m_k$ would be stored under the nodes $m_1, \ldots, m_k$ (see Figure 3.9). Overall, this first iteration costs $\mathcal{O}(|\mathcal{L}_2||G_1| \max(|g'^{1}|))$. Then, we do the same for all concepts in $\mathcal{L}_1$ with a complexity of $\mathcal{O}(|\mathcal{L}_1||G_2| \max(|g'^{2}|))$. Up until this point, IntentsDict has all intents in $\mathcal{L}_1$ and $\mathcal{L}_2$, with their extents updated.

Then, to compute lines 6-13, for each pair of concepts C_1, C_2 in $\mathcal{L}_1$ and $\mathcal{L}_2$ respectively, the intersection of its intents is calculated. In case the intersection $Y \neq \varnothing$ and $Y \notin$ IntentsDict, we look for the concepts $CI_1 = \langle X_1, Z_1 \rangle$ and $CI_2 = \langle X_2, Z_2 \rangle$ in $\mathcal{L}_1$ and $\mathcal{L}_2$ respectively, such that CI_1 and CI_2 are canonical generators of Y, and add a new concept $\langle X_1 \cup X_2, Y \rangle$. These two loops have a cost of $\mathcal{O}(|\mathcal{L}_1||\mathcal{L}_2| \max(|G_1| \max(|g'^{1}|), |G_2| \max(|g'^{2}|))$. Which, in turn, although less sharp, could be simplified to $\mathcal{O}(|\mathcal{L}_1||\mathcal{L}_2|(|G_1||M_1| + |G_2||M_2|))$.

Finally, all concepts in $\mathcal{C}_m$ are in IntentsDict so they are simply returned. Thus, the final complexity of the algorithm is

$$\begin{aligned} &\mathcal{O}(|\mathcal{L}_2||G_1| \max(|g'^{1}|)) + \mathcal{O}(|\mathcal{L}_1||G_2| \max(|g'^{2}|)) + \\ &\mathcal{O}(|\mathcal{L}_1||\mathcal{L}_2|(|G_1| \max(|g'^{1}|) + |G_2| \max(|g'^{2}|)) = \\ &\mathcal{O}(|\mathcal{L}_1||\mathcal{L}_2|(|G_1| \max(|g'^{1}|) + |G_2| \max(|g'^{2}|)) \end{aligned} \tag{3.5}$$

The complexity of the parallel algorithms can be reduced basically to the complexity of performing the Cartesian product between the two sets of concepts, and then for each of the pairs performing the merge function. The Cartesian product between two sets U, V is $|U||V|$, and then the three merge functions provided in the three algorithms rely only on intersections and unions of intents and extents in the sets of concepts, i.e., $\mathcal{O}(|\mathcal{L}_1||\mathcal{L}_2|(|G_1| \log|G_2| + |G_2| \log G_1 + |M_1| \log|M_2| + |M_2| \log|M_1|))$. However, the difference between these algorithms,

besides the fact that they can be performed by a number of workers, is that they do not rely on any lookup of generators. Nevertheless, the effectiveness of the parallelization must be evaluated empirically, specially considering that datasets have different characteristics affecting how algorithms perform greatly.

3.4 Conclusions

This chapter tackles the challenge of knowledge loss when constructing a lattice from a never-ending stream of data. Recognizing the limitations of capturing all knowledge within a finite lattice, we propose a general method to recover some of this lost information. This method involves storing snapshots of the lattice at specific points. Later, when needed, these snapshots can be merged with the current lattice. The Algorithm BatchMerge has been introduced for efficiently performing these merges, and it has been theoretically compared to existing approaches. Finally, acknowledging the computational demands of snapshot distribution and merging, three novel distributed and parallel algorithms to handle these tasks have been presented (algorithms ParallelMergeOnlyIntents, ParallelMergeConcepts, and ParallelMergeConceptsWithUnion).

In this context, the proposed FCA-based method provides *flexibility* to practitioners that need to perform knowledge extraction in highly dynamic settings such as the ones presented in the Industry 4.0. More specifically, distributed architectures generating massive amounts of data each second, cyber-physical systems high level semantic mining, knowledge extraction in the context of smart cities temporal events, and so forth. Although selecting which snapshots to store is still a challenging task, now the problem of mining concepts from the infinite data streams can be reduced to how to combine certain partial lattices containing useful information, with the latest available knowledge available.

Additionally, the method contributes to the *scalability* of data mining in the context of distributed architectures, because before the sliding window to consider had to be larger in order to reduce the possibility of losing relevant concepts, but with this method, it is possible to consider a slightly smaller sliding window size (thus being able to compute more events in the same timeframe), and then recover the relevant concepts on the merge step.

All presented algorithms rely on the proven properties Lemma 3.1, Lemma 3.2, and Lemma 3.3, which can be thought as a consequence of the fact that lattices are closed under intersection. Particularly, they take advantage of the fact that the two input lattices have already been computed and are stored somewhere. They compute on the one hand the concepts in the merged lattice sharing intents with the two input lattices. And then, they compute the concepts having an intent resulting from a non-empty intersection between two intents in the two input lattices.

In addition, the presented distributed algorithms leverage the RDD abstraction to parallelize different parts of the computation using different strategies. Particularly, ParallelMergeOnlyIntents only computes the intents, which can be more efficient when actual instances are of no importance, whereas ParallelMergeConcepts also computes the extents using the Cartesian product between concepts in the two input lattices. What is more, ParallelMergeConceptsWithUnion seeks to reduce the number of elements to traverse in the Cartesian product by only considering the intents whose intersection is not empty. To do so, it has to compute the intersection between M_1 and M_2, which might not be ideal in some scenarios where the amount of attributes is exceedingly large.

The remaining parts of the study require the presented algorithms to be contrasted

with existing algorithms, and with each other (in the case of the parallel ones), following a number of experiments to understand their behavior in different scenarios. Moreover, real world use cases are desired to be shown in order to weight its actual practicality. These continuations can be read in 5 and 6.

4

Arbitrarily distributed Formal Contexts incremental approach

Contents

4.1 Introduction **67**
4.2 Motivation **68**
4.3 The algorithm of Goel and Chaudhary **68**
4.4 Why incremental **70**
4.4.1 AddIntent 71
4.4.2 DeleteInstance 74
4.4.3 Generalization 75
4.5 AddPair **75**
4.6 Conclusions **79**

4.1 Introduction

In this chapter, an algorithm to incrementally update a lattice from an arbitrarily distributed context is presented. As opposed to the only existing algorithm to compute from arbitrarily distributed formal contexts, this algorithm adds flexibility to the setting by not having to recompute the whole lattice from scratch in case the context, on top of being arbitrarily distributed, is also dynamic, meaning that objects and attributes might be added over time.

The structure of the chapter is as follows: firstly, section 4.2 provides a motivation and an overview of the state-of-the-art *particular to this chapter*. Second, section 4.3 introduces the algorithm of Goel and Chaudhary, which is the one that highlights the importance of computing from arbitrarily distributed formal contexts, and additionally, it shows the areas in which that algorithm is lackluster, setting a common ground for the algorithm proposed in this chapter. Thirdly, section 4.4 highlights the key challenges faced in incremental algorithms, and it introduces the existing algorithms used in our approach. Fourth, section 4.5 presents the algorithm, and delves deeper into the implementation, its complexity, advantages and disadvantages compared to the algorithm of Goel and Chaudhary. Finally, section 4.6 outlines the contribution, and provides some conclusions on the matter.

4.2 Motivation

Several algorithms exist to compute the concept lattice of a formal context (i.e., objects and their attributes). These algorithms are usually categorized into two categories: *batch* and *incremental*. The first one comprises algorithms that have full knowledge of the whole input (i.e., the formal context), and compute the concept lattice with it [7, 104]. The latter includes algorithms that do not know the whole set of objects in advance, and update the current lattice as they arrive [132]. Moreover, contrary to batch algorithms, incremental algorithms have the advantage that they can be used in stream processing applications, because they do not need to recompute everything from scratch every time a new object is added to the context. However, incremental algorithms consider that the set of all attributes (or the set of all objects) is known from the beginning, which might not be the case in all scenarios.

For instance, in the case of marketplaces, it is commonplace nowadays that the information of items to be sold are physically stored in several (and potentially different) databases. Moreover, considering as well that marketplaces could sell items from different tenants with a number of security policies, it is also frequent to not have other option than to store transactions in their databases. This lead to a setting where, from the FCA perspective, it is desirable to compute the concept lattice from distributed contexts. Distributed algorithms, in that regard, usually need to have the whole formal context replicated in all nodes (a.k.a., workers), which can be a hindrance if it is too big, or even infinite as in the case of data streams. In some cases, the workers need to share it as a part of the algorithm, and can cause a huge network overload. Considering this, the notion of arbitrarily distributed formal context has emerged, and it consists of being able to *partition* the formal context in any way.

Goel and Chaudhary in [49] present an algorithm to compute a concept lattice in a distributed environment, where the formal context is arbitrarily distributed. In other words, since there are no constraints in how the formal context (i.e., object, attributes, and the incidence relationship) is distributed across nodes, not only all objects are unknown in advance, but neither the attributes are. Additionally, the algorithm leverages the advantages of the Apache Spark batch streams, making it suitable to analyze *static* snapshots of the stream. Therefore, the algorithm they present falls into the batch category, even if the batch comes from a data stream.

Valtchev and Missaoui in [129] present a generalization of incremental methods based on the two dual assembling operations called *subposition* and *apposition* respectively. The first one corresponds to the assembly of contexts sharing the same *attributes*, whereas the latter to contexts sharing the same *objects*. In that generalization, incremental methods are extended in a way that they allow multiple objects (or attributes) to be added at once, instead of only one at a time.

However, since to the best of our knowledge no algorithm exists to *incrementally* compute a concept lattice from *arbitrarily distributed formal contexts*, in this chapter an algorithm to do so is presented. Moreover, its computational complexity bound is studied. In addition, a discussion on different approaches to address the problematic is provided.

4.3 The algorithm of Goel and Chaudhary

The following section dives into the Goel and Chaudary algorithm (introduced in [49]). It will explore the benefits this algorithm offers for handling arbitrarily distributed formal con-

texts. However, it will also reveal its limitations: the algorithm processes data in batches and experiences an exponential increase in computational complexity as the number of attributes grows, regardless of the data density within the context.

Standalone and single threaded algorithms to mine formal concepts from a given dataset are restricted in applicability as the size of the context increases. One of the reasons why this happens is that lattice construction algorithms are inherently exponential in the worst case: the amount of formal concepts is bound by $2^{O(|G|+|M|)}$. Consequently, interest in finding distributed solutions that would reduce the time for mining as well as enable distributed storage using low-cost networked computing nodes has been spurred by these limitations.

For instance, it has been successfully used in a distributed environment in the article of De Maio et al. in 2017 [28], but in conjunction with the extensions called Fuzzy Formal Concept Analysis (i.e., an extension of FCA [90] where the incidence matrix is a relation $I \subseteq G \times M \rightarrow [0,1]$) and Temporal Concept Analysis ([138]). This work takes advantage of the fuzzy and temporal characteristics of the extensions in order to deal with the exponential growth of the fuzzy lattice. In addition, numerous works have been conducted to address the problem of scalability by leveraging the MapReduce paradigm. However, these algorithms encounter constraints that hinder their scalability the larger the formal context size gets. One of these constraints involves the necessity to store the entire context on every node, while the other is weighed down by significant communication overhead at the conclusion of each iteration. In practical scenarios, we encounter extensive datasets that exceed the storage capacity of a single machine. For instance, consider a dataset comprising emails sent by customers to a company, which may be utilized to identify correlations among customers. Another illustration could be an e-commerce platform that provides a diverse array of products to its customers, aiming to categorize customers based on their specific product browsing behaviors.

In the light of this context, in Goel and Chaudhary's article [49], the authors present an algorithm for computing concept lattices from an unconstrained (i.e., arbitrarily distributed) formal context. The algorithm utilizes the Apache Spark MapReduce framework and leverages its core abstraction called Resilient Distributed Dataset (RDD). In their approach, the formal context is only used at the beginning of the algorithm to generate a distributed collection of key-value pairs (object-attribute if $|G| < |M|$, or attribute-object otherwise). Then, they generate a new key-value pair from an existing one representing the formal concepts of size k, with $k \in \mathbb{N}$. The process is repeated until $k = |G|$ and, as a result, all formal contexts have been generated.

More precisely, the algorithm proposed in [49] can be summarised as follows,

Input:

A key-value pair P of object-attribute (analogously, attribute-object), i.e., $\langle g, m \rangle$ such that $g \in |G| \ \wedge \ m \in |M|$ if $|G| < |M|$, $\langle g, m \rangle$ otherwise.

Output:

An RDD $\mathcal{C}$ with pairs $\langle k, v \rangle$ where k is an extent (or intent if $|M| < |G|$), v an intent (or extent), and $\langle k, v \rangle \in \mathcal{C}$ iff $\langle k, v \rangle$ is a formal concept from the underlying formal context $\mathbb{K} = \langle \{\pi_1(p) \mid \forall\, p \in \mathrm{P}\}, \{\pi_2(p) \mid \forall\, p \in \mathrm{P}\}, \mathrm{P} \rangle$.

Procedure:

Since mining concepts on a formal context or its transpose yields the same result, the authors chose to aggregate data by objects or by attributes, depending on which one has the lower count. To make this decision, they utilize the countApproximate function present in Apache Spark, that does not need to actually count *all* the elements on each dimension. Once the dimension is chosen, and assuming for simplicity that the objects are more than the attributes, the idea of the algorithm relies on generating candidate formal concepts by key size (i.e., attributes in this case).

The generation of candidate formal concepts of size 1 (referred as CC_1) is performed by aggregating pairs on the key as an *id* in the following way,

$$CC_1 = \{\langle g, M\rangle \mid g \in G \ \wedge \ M = \{m \mid \langle g, m\rangle \in \mathrm{P}\}\} \tag{4.1}$$

Then, an iterative process is defined based on the candidate set of formal concepts CC_N consisting of key-value pairs (as defined in Equation 4.1) $\langle K_N, V\rangle$ of keys with size N. First, let us define K_{N+1} by considering K_N to be lexicographically ordered, i.e., $k_i \leq k_{i+1} \forall i \in [1, l)$ for $K_N = \{k_1, k_2, \ldots, k_l\}$. Given two keys of size N with their first N elements in common, $P_N = \{e_1, e_2, \ldots, e_{n-1}, a\}$ and $Q_N = \{e_1, e_2, \ldots, e_{n-1}, b\}$, a new key of size $N+1$ would be generated as follows:

$$K_{N+1} = e_1, e_2, \ldots, e_{n-1}, a, b \tag{4.2}$$

Then, C_{N+1} is defined by combining all possible combinations of lexicographically ordered keys in CC_N ($\frac{l(l-1)}{2}$ combinations where $|CC_N| = l$) by using the formula Equation 4.2. And finally, given $P_N = \{e_1, e_2, \ldots, e_{n-1}, a\}$ and $Q_N = \{e_1, e_2, \ldots, e_{n-1}, b\}$ two keys of size N, and V_1, V_2 their values, then, the set of pairs of the form $\langle e_1, e_2, \ldots, e_{n-1}, a, b, V_1 \cap V_2\rangle$ defines CC_{N+1}.

Now, the set of valid formal concepts of size N are defined using CC_N and CC_{N+1}. Let $V_X = \{\pi_2(p) \mid p \in CC_{N+1}\}$ be the set of values of CC_{N+1}. Then, the set of valid formal concepts of size N is

$$VC_N = \{\langle K_N, V\rangle \mid \langle K_N, V\rangle \in CC_N \text{ and } V \notin V_X\} \tag{4.3}$$

The final result is given by all the valid formal concepts VC_i with $i \in \mathbb{N}$ and $1 \leq i < |M|$ (or $|G|$ if there were more attributes than objects). The bottom and top concepts are computed separately.

4.4 Why incremental

As seen in the previous section, given a data stream of object-attribute pairs, there exist at least one algorithm [49] to construct the set of formal concepts from it, considering the nature of the stream being arbitrarily distributed. However, the algorithm needs a stream snapshot to be selected before starting the construction process, and thus, it is a batch algorithm. As such, it needs to re-start if the snapshot changes, even if it is only by adding one more pair to the snapshot. This means that, even though it can be used with data streams, it is not a suitable option for *online* data stream processing.

Moreover, the generalization of the incremental update method proposed in [129], addresses the problem of having set-wise object updates instead of object-wise, meaning that a whole subset of objects $\delta O = \{o_{i+1}, \ldots, o_{i+l}\}$ are to be added at a time. However, this still does not cover the idea of adding only a pair $\langle g, m\rangle$ where both g and m are *potentially unknown*.

In the following, three algorithms relevant to the work presented in the chapter will be described.

4.4.1 AddIntent

With the aim of incrementally adding the object g to a lattice $\mathcal{L}$ incremental algorithms to the best of our knowledge consider that either M (or G) is known in advance, and then the incremental steps happen on the elements that are unknown. An incremental update is usually considered between $\mathcal{L}_i$ and $\mathcal{L}_{i+1}$ where the difference between the two lattices is that g is being added to $\mathcal{L}_i$ with all attributes g', and additionally $g \notin G_i$. Then, the techniques these algorithms use in order to efficiently traverse the lattice and do not do unnecessary computations usually rely on having the line diagram ($\preceq$ lower and upper neighbors in [42]) updated after each incremental step. Moreover, as described in [47], in this type of algorithm, given an update from the lattice $\mathcal{L}_i$ to $\mathcal{L}_{i+1}$, they typically separate the concepts into the categories (notice that AI in this context stands for AddIntent)

AI.1 *New*: a concept $\langle C, D\rangle \in \mathcal{L}_{i+1}$ where D is not an intent in $\mathcal{L}_i$.

AI.2 *Modified*: a concept $\langle A, B\rangle \in \mathcal{L}_i$ such that $B \subseteq g'$ since g has to be added to its extent in $\mathcal{L}_{i+1}$.

AI.3 *Generator*: a concept $\langle A, B\rangle \in \mathcal{L}_i$ such that given a *new* concept $\langle C, D\rangle \in \mathcal{L}_{i+1}$, $D = B \cap g' \neq B$. And

AI.4 *old*: any other concept.

Every *new concept* $\langle C, D\rangle$ has at least one generator, but it may have several. However, there is a unique most general generator which is typically called the *canonical* generator of $\langle C, D\rangle$. The rest of them are called *non-canonical*. The primary challenge of incremental construction lies in the identification of all *modified* concepts and the determination of all canonical generators for new concepts (to ensure each new concept is generated precisely once). Efficient algorithms aim to minimize the effort spent on searching through unmodified and non-canonical generators. Some algorithms such as AddIntent, defined in [132], address this challenge by recursively traversing the diagram graph of $\mathcal{L}$.

Finding canonical generators is therefore a crucial part of incremental algorithms. In general, a concept $\langle X, Y\rangle$ can only generate concepts with an intent $Y \cup g$ for some g, and moreover, AddIntent leverages the following two properties:

Proposition. 4.1: Proposition 1 in [132]

If $\langle B', B\rangle$ is a canonical generator of a new concept $\langle F', F\rangle$, and $\langle D', D\rangle$ is a non-canonical generator of $\langle F', F\rangle$, then any concept $\langle H', H\rangle$ such that $H \subset D$ and $H \not\subset B$ is neither modified nor a canonical generator of any new concept.

Proposition. 4.2: Proposition 2 in [132]

If $\langle D', D\rangle$ is an old concept and $D \cap g' = B$, then any concept $\langle H', H\rangle$ such that $H \subset D$ and $H \not\subset B$ is neither modified nor a canonical generator of any new concept.

Hence, concepts $\langle H', H\rangle$ can be ignored when searching for canonical generators and modified concepts. Particularly, since AddIntent maintains the line diagram (from most to least general), it excludes these concepts by simply traversing the graph from the $\perp$ concept.

The general algorithm to look for a canonical generator can be described as follows

Algorithm GetCanonicalGenerator Algorithm to get the canonical generator of the set of attributes Y in the lattice $\mathcal{L}$, from the concept C

```
1:  is_maximal ← true
2:  canonical_generator ← C
3:  while is_maximal do
4:      is_maximal ← false
5:      for parent ∈ parents(canonical_generator) do
6:          if Y ⊆ intent(parent) then
7:              is_maximal ← true
8:              canonical_generator ← parent
9:          end if
10:     end for
11: end while
12: return canonical_generator
```

Example. 4.1:

Looking for the canonical generator of the intent $Y = \{\text{GL}\}$ in the lattice shown in Figure 2.2, the algorithm would start looking from the most general concept, i.e., $\perp$, which contains all $m \in M$. Since $Y \subseteq M$, it would check all its parents, which in this case are 3, and take any of them such that Y is included in their intent, in this case it could be the concept C with the intent $\{\text{GL}, 3 \leq \text{nbP} < 7, \text{SKNORD}\}$. It will repeat this process until no parents of the current generator has an intent including Y. In this example, the found canonical generator would be the one with the intent $\{\text{GL}, \text{SKNORD}\}$.

Analyzing Algorithm GetCanonicalGenerator, $\mathcal{O}(|G|)$ parents could get traversed in each of the iterations of the *while* (line 3). Assuming that the most general C that can be used is the $\perp$ concept, the *while* statement could run at most $\max(|g'|)$ iterations, since each parent has at least one attribute less in its intent each time. Thus, considering that the inclusion check in line 6 can be done in $\mathcal{O}(\max(|g'|) \log \max(|g'|))$, one worst case bound for the algorithm is $\mathcal{O}(|G| \max(|g'|)^2 \log \max(|g'|))$. However, in there are a number of optimizations that can improve this bound. For example, if precomputing the number of attributes in common that each concept in $\mathcal{L}$ has with the intent to be added (g'), the comparison can be done in $\mathcal{O}(1)$, since $|g' \cap parent.I| == |g'| \implies g' \subseteq parent.I$. Hence, in that case, it is more suitable to use the bound $\mathcal{O}(|G| \max(|g'|))$.

Then, given the object g with an intent g' to be added to the lattice $\mathcal{L}_i$, the idea of AddIn-

tent is to *recursively* add new concepts whenever it is necessary (from the canonical generators). For clarity, and for reference, the algorithm is copied in Algorithm AddIntent.

Algorithm AddIntent Algorithm to add the object g with the intent Y in the lattice $\mathcal{L}$, from the generator concept C

```
1:  canonical_generator_concept ← GetCanonicalGenerator(g', C)
2:  if intent(canonical_generator_concept) == Y then
3:      return canonical_generator_concept
4:  end if
5:  new_parents ← ∅
6:  for candidate ∈ parents(canonical_generator_concept) do
7:      if intent(candidate) ⊄ Y then
8:          candidate ← AddIntent(intent(candidate) ∩ Y, L, candidate)
9:      end if
10:     add_parent ← true
11:     for parent ∈ new_parents do
12:         if intent(candidate) ⊆ intent(parent) then
13:             add_parent ← false
14:             break
15:         else if intent(parent) ⊆ intent(candidate) then
16:             new_parents ← new_parents \ {parent}
17:         end if
18:     end for
19:     if add_parent then
20:         new_parents ← new_parents ∪ {candidate}
21:     end if
22: end for
23: new_concept ← ⟨extent(canonical_generator_concept), Y⟩
24: L ← L ∪ {new_conceept}
25: for parent ∈ new_parents do
26:     remove_link(parent, canonical_generator_concept, L)
27:     set_link(parent, new_concept, L)
28: end for
29: set_link(new_concept, anonical_generator_concept, L)
30: return new_concept
```

First, in line 1, the algorithm looks for the canonical generator of intent Y, and if a concept with such an intent already exist, it simply terminates (line 3). Otherwise, it means that no concept with the intent Y exists, and hence, a new concept with that intent will be created and returned (lines 23-30).

Afterwards, between lines 6-18, the parents of *new_concept* are looked up by traversing the parents of the *canonical_generator_concept*. Then, a *candidate* is modified if its intent is a subset of Y. Otherwise, the *candidate* to be considered is the one returned by the recursive call of AddIntent from that concept and the intent $candidate.I \cap Y$ (line 8). Then, the candidate is added to the *new_parents* list if it has a maximal intent. Additionally, if some concept among its parents is more general than *candidate*, then the parent is removed from the list. This is to ensure that, in the end, *new_parents* has exactly the list of new parents

of $new\_concept$ (line 25). Notice that the algorithm does not update extents. In the original work. The authors decided to do it outside the algorithm, in the lattice construction part, which basically would add g to the extent of the $new\_concept$ returned by the first call of AddIntent and all concepts above it.

Regarding the complexity, the algorithm including all optimizations can be computed in $\mathcal{O}(|\mathcal{L}||G|^2 \max(|g'|))$. This is because a single call of AddIntent (without the recursive call) is bounded by $\mathcal{O}(|G|^2 \max(|g'|))$, and additionally, AddIntent would be recursively called at most only once per concept.

4.4.2 DeleteInstance

Another incremental approach that is taken into account is the one in which instances (or objects) are being *incrementally* removed from a lattice. Several algorithms have been proposed to compute a lattice $\mathcal{L}_{i+1}$ from a $\mathcal{L}_i$ where the difference is that $\mathcal{L}_{i+1}$ has had g removed from it. In particular, the one considered in this work is the so-called FastDeletion [149]. As what happens in the setting of incrementally adding objects, when certain objects have to be forgotten, it makes more sense to incrementally remove them, rather than to construct a lattice from scratch. In this scenario, the incremental process can be described into the following sets of concepts,

DI.1 *Deleted*: a concept $\langle A, B\rangle \in \mathcal{L}_i$ if B is not an intent in $\mathcal{L}_{i+1}$.

DI.2 *Modified*: a concept $\langle A, B\rangle \in \mathcal{L}_i$ such that $g \in A$ and B is still an intent in $\mathcal{L}_{i+1}$.

DI.3 *Old*: a concept $\langle A, B\rangle \in \mathcal{L}_i$ if it remains intact in $\mathcal{L}_{i+1}$

DI.4 *Destructor*: a concept $\langle A, B\rangle$ is a destructor of a *deleted* concept $\langle C, D\rangle$ if $C \setminus \{g\} = A$.

Using these sets, the algorithm is quite straightforward, but it has some subtleties nonetheless. In particular, after finding the canonical generator of g', all concepts below it are *old*, since they do not have g. From there, all concepts above it can be either *modified* or *deleted*. However, deleted concepts might need some links to be created between their children and their parents before actually being removed. Therefore, identifying the links to be created before removing a deleted concept is one of the most important steps in the incremental deletion of an instance.

Considering this, the following pseudo-algorithm can be used to incrementally delete the instance g from $\mathcal{L}_i$.

Step.1 Find the lowest concept containing the deleted object g, and add it to a set of *candidates*.

Step.2 Pop a concept C from *candidates* and append all unvisited parents of C to the *candidates* list.

Step.3 If C is a deleted concept, remove it from the lattice and go to Step.4. Otherwise, mark all parents of c as modified concepts and go to Step.2.

Step.4 Set links between a child of C and its destructor if necessary.

The FastDeletion algorithm utilizes, among others, one property in order to efficiently perform the Step.2 and Step.3. In particular, when judging if a concept $C = \langle A, B \rangle$ is *deleted* or not, all previous algorithms traversed its whole set of children. Since that process need to compare $A \setminus \{g\}$ for each of C's children, its complexity bound would be $\mathcal{O}(|G||M|)$ in the worst case. However, FastDeletion needs to compare $A \setminus \{g\}$ with only one of its children, yielding a complexity bound of $\mathcal{O}(|G| + |M|)$ instead. To achieve that improvement, the property leveraged was the following,

Proposition. 4.3: Proposition 3 in [149]

Let $\mathcal{L}_i$ and $\mathcal{L}_{i+1}$ be the concept lattice of a given context before and after deleting an object g, respectively. Let $\langle X, Y \rangle$ be a formal concept in $\mathcal{L}_i$. Then $\langle X, Y \rangle$ is a deleted concept if and only if there is one and only one *old* concept $\langle X_0, Y_0 \rangle$ among all the children of $\langle X, Y \rangle$ and $X_0 = X \setminus \{g\}$.

Then, the FastDeletion algorithm performs the steps described in 4.4.2 in $\mathcal{O}(|\mathcal{L}||G||M|^2)$.

4.4.3 Generalization

So far, we have gone through the incremental methods for adding an object g into a certain lattice $\mathcal{L}_i$, deleting an instance g from a lattice $\mathcal{L}_i$. Now, we will review the generalization of the incremental construction of lattices proposed in [129].

In that article, the goal is to allow the insertion of several objects at a time instead of only one, as what happens in AddIntent for instance. The way they propose to do so is by *assembling* lattices. More precisely, instead of inserting objects one by one into $\mathcal{L}_i$, the proposal is to first compute the lattice $\delta\mathcal{L}$ corresponding to the subset of objects to be added δO and then construct $\mathcal{L}_{i+1}$ from the *merge* between $\mathcal{L}_i$ and $\delta\mathcal{L}$.

In particular, the strategy they adopt in order to perform the merge, or assembly as they call it, is to use the *direct product* of lattices $\mathcal{L}_\times = \mathcal{L}_1 \times \mathcal{L}_2$ which is itself a lattice. The nodes of $\mathcal{L}_\times$ are pairs of concepts $\langle C_1, C_2 \rangle$, where C_i appears in $\mathcal{L}_i$ for $i = 1, 2, \ldots, k$ and its order relation $\leq_\times$ is the product of both lattice orders. In this setting, $\mathcal{L}_i$ are called *partial* lattices, and the underlying lattice of the merge of all partial lattices is called *global*. Then, there is a one-to-one relationship between the $\mathcal{L}_\times$ and the global lattice $\mathcal{L}_{i+1}$. It is important to mention as well that the partial lattices in this work share the same set of attributes, unlike what we presented in chapter 3. Additionally, the algorithm they propose takes care of maintaining the links between concepts.

4.5 AddPair

In the last section, a batch algorithm that performs a distributed batch algorithm to compute the set of formal concepts from an arbitrarily distributed formal context, an algorithm that incrementally adds objects with a certain intent to a lattice, an incremental algorithm to remove objects, and the generalization of incremental algorithms in FCA to allow multiple objects to be added to a lattice at once were presented. However, these methods do not translate trivially to the **incremental** construction of lattices coming from **dynamic and arbitrarily distributed formal contexts**, because they either are batch algorithms, or are incremental, but need to have full knowledge of at least one of the dimensions of the

dataset (i.e., generally attributes). For that reason, in this section, we present an *incremental* algorithm to compute a set of formal concepts and its line diagram in that setting.

With the aim of consume a data stream as described in [49], as object-attribute pairs, and assuming a dynamic setting where attributes and objects can be added at any given time, the method proposed will incrementally update a lattice $\mathcal{L}_i$ from a pair $\langle g, m\rangle$ coming from a distributed data stream. With this conception of "incrementally adding pairs" in mind, the process of incrementally constructing the lattice can be described with the following sets of concepts,

AP.1 *New*: a concept $\langle C, D\rangle \in \mathcal{L}_{i+1}$ where D is not an intent in $\mathcal{L}_i$ and C is not an extent in $\mathcal{L}_i$.

AP.2 *Modified*: a concept $\langle A, B\rangle \in \mathcal{L}_i$ such that $B \subseteq g'$ since g has to be added to its extent in $\mathcal{L}_{i+1}$ or $A \subseteq m'$ since m has to be added to its intent in $\mathcal{L}_{i+1}$.

AP.3 *Generator*: a concept $\langle A, B\rangle \in \mathcal{L}_i$ such that given a *new* concept $\langle C, D\rangle \in \mathcal{L}_{i+1}$, $D = B \cap g' \neq B$ or $C = A \cap m' \neq A$ (these second ones will be referred to as generator concept^{-1}). And

AP.4 *old*: the rest.

So, let us firstly identify what are the different scenarios when adding $\langle g, m\rangle$ to $\mathcal{L}_i$ where g might or might not be in G_i, m might or might not be in M_i, and $\langle g, m\rangle \notin I_i$.

First, let us consider the case where $g \notin G_i$, and $m \notin M_i$. In this case, the only concepts $\langle A, B\rangle \in \mathcal{L}_i$ that could be *modified* are the bottom and the top. In fact, if $\bot_i = \langle \varnothing, M_i\rangle$, then, it is a modified concept and its new version would be $\bot_{i+1} = \langle \varnothing, M_i \cup \{m\}\rangle$. Analogously, if $\top_i = \langle G_i, \varnothing\rangle$, its modified version would be $\top_{i+1} = \langle G_i \cup \{g\}, \varnothing\rangle$. However, if $\bot_i = \langle X, M_i\rangle$, $X \neq \varnothing$, then it is the canonical generator of the *new* concept $\bot_{i+1}$. Similarly, if the intent of $\top_i$ is not empty, then it will be the canonical generator^{-1} of the *new* concept $\top_{i+1} = \langle G_i \cup \{g\}, \varnothing\rangle$.

Second, if $g \notin G_i$, but $m \in M_i$ the cases are, in fact, exactly the same as in AddIntent (AI.1-AI.4). Analogously, if $m \notin M_i$, but $g \in G_i$, the dual version of the AddIntent algorithm (e.g., we can call it "AddExtent") would suffice, hence, the cases are also dual (see the additions of AP.1-AP.4 with respect to AI.1-AI.4).

Finally, let us consider the case when $g \in G_i$ and $m \in M_i$. As we can see in Figure 4.1, in $\mathcal{L}_i$, assuming that b is the generator concept^{-1} of g, whereas d is the generator concept of m, we can see in **red** all concepts containing g in their extents, and in **light-blue**, all concepts containing m. Moreover, concepts *below* b might or might not contain m in their intent, whilst concepts *above* d might or might not contain g in their extent. For instance, in this particular scenario, w contains m in its intent because it is a sub-concept of d. What we know, from a procedural point of view, is that the **red** concepts *may be* modified because their intent now might include m, and **light-blue** concepts *may be* modified analogously because their extent might have to include g now.

Considering that finding one generator can be done in $\mathcal{O}(|G|^2|M|)$[20], while, dually, finding a generator^{-1}, in $\mathcal{O}(|G||M|^2)$, we could argue that an algorithm to find the generator of m and the generator^{-1} of g would cost $\mathcal{O}(|G|^2|M| + |G||M|^2)$, or its simplified version

[20]This bound depends on several aspects, notably, the worst case happens when the algorithm start traversing the lattice from the $\bot$ (or dually from the $\top$). However, this can be considerably faster if we start from higher in the hierarchy (or lower in the case of generator^{-1})

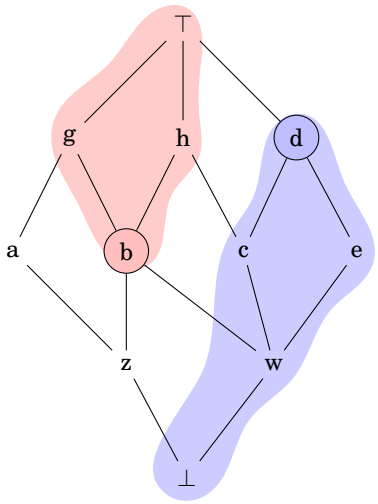

Figure 4.1: Concept categories during the an AddPair incremental step in a concept lattice.

$\mathcal{O}(\max(|G|^2|M|, |G||M|^2))$. Additionally, in the worst case, either above the generator^{-1} of g or below the generator of m, there could be $|\mathcal{L}_i|$ concepts.

In the light of that, we propose Algorithm AddPair to incrementally add a pair $\langle g, m\rangle$ to a lattice $\mathcal{L}_i$ such that both g and m may or may not be present in G_i and M_i respectively. The algorithm consists in the combination of two other incremental algorithms: AddIntent [132] and DeleteInstance [149]. Basically, if $g \in \mathcal{L}_i$, between lines 1-5 the algorithm sets a variable old_intent with the value of g' (i.e., in $\mathcal{L}_i$, thus it does not include m yet), then, it deletes the object g (i.e., equivalent to deleting the row in which that instance appears in the formal context), and finally, it sets a variable new_intent as $g' \cup \{m\}$. Otherwise, $new\_intent = \{m\}$, since g is new in $\mathcal{L}_{i+1}$ and its only related attribute is m (line 6). Then, at line 7, $new\_intent = g'$ regardless of whether $g \in G_i$ or not. Continuing, in line 8, in case $m \notin M_i$, the algorithm updates the bottom using Algorithm UpdateBottom, which simply creates a new bottom containing all attributes (as it should be) when there exists at least an object in $\perp$, or it updates the existing one otherwise. Then, we are in the case in which $m \in M$, and $g \notin G$, hence, we can use AddIntent given that we also have g' in the new_intent variable (line 11). Finally, the resulting $\mathcal{L}$ is returned which corresponds to $\mathcal{L}_{i+1}$.

As stated before, one of the challenges of incremental construction algorithms is efficiently identifying the *modified* concepts. Since the amount of them in the worst case is $|\mathcal{L}_i|$, and finding the generator^{-1} of g and the generator of m would cost $\mathcal{O}(\max(|G|^2|M|, |G||M|^2))$. Furthermore, for each of the modified concepts, it would be necessary to find its generator (or dually, its generator^{-1}), because by adding g to its extent (or m to its intent), it is possible that there already exist a concept in $\mathcal{L}_i$ with that extent (or intent). Hence, the best we could hope for is bounded[21] by,

$$\begin{aligned}\mathcal{O}(|\mathcal{L}|\ \max(|G|^2|M|, |G||M|^2)) &= \mathcal{O}(|\mathcal{L}|\ (|G|^2|M| + |G||M|^2)) \\ &= \mathcal{O}(|\mathcal{L}||G||M|\ (|G| + |M|)) \\ &= \mathcal{O}(|\mathcal{L}||G||M|\ \max(|G|, |M|))\end{aligned} \tag{4.4}$$

Then, if we analyze Algorithm AddPair, the complexity between the lines 1-7 is the

[21] Notice that $\mathcal{O}$ is an *upper* bound.

Algorithm AddPair Algorithm to incrementally add the attribute m to the object g in the lattice $\mathcal{L}$

1: **if** $g \in \mathcal{L}.G$ **then**
2: $old\_intent \leftarrow g'$
3: $delete\_instance(g, \mathcal{L})$
4: $new\_intent \leftarrow old\_intent \cup \{m\}$
5: **else**
6: $new\_intent \leftarrow \{m\}$
7: **end if**
8: **if** $m \notin L.M$ **then**
9: $update\_bottom(m, \mathcal{L})$
10: **end if**
11: $add\_intent(g, m, \mathcal{L})$
12: **return** $\mathcal{L}$

Algorithm UpdateBottom Algorithm to update the $\perp$ concept in $\mathcal{L}$ considering that $m \notin M$

1: $\mathcal{L}.M \leftarrow \mathcal{L}.M \cup \{m\}$
2: **if** $\mathcal{L}.\perp.E = \varnothing$ **then**
3: $\mathcal{L}.\perp.I \leftarrow \mathcal{L}.M$
4: **else**
5: $new\_bottom \leftarrow \langle \mathcal{L}.\perp.E, \mathcal{L}.M \rangle$
6: set_link$(\mathcal{L}.\perp, new_bottom)$
7: $\mathcal{L}.\perp \leftarrow new\_bottom$
8: **end if**
9: **return** $\mathcal{L}$

complexity of DeleteInstance, i.e., $\mathcal{O}(|\mathcal{L}||G||M|^2)$ [149]. Then, the complexity between the lines 8-12 is the maximum between Algorithm UpdateBottom and AddIntent. Since the complexity of Algorithm UpdateBottom is negligible compared to that of AddIntent, the complexity between these lines ends up being $\mathcal{O}(|\mathcal{L}||G|^2|M|)$[22] ([132]). Further, the total complexity of Algorithm AddPair is

$$\begin{aligned}\mathcal{O}((|\mathcal{L}||G|^2|M| + |\mathcal{L}||G||M|^2)) &= \mathcal{O}(|\mathcal{L}||G||M|\ (|G| + |M|)) \\ &= \mathcal{O}(|\mathcal{L}||G||M|\ \max(|G|, |M|))\end{aligned} \tag{4.5}$$

Then, it is noticeable that Equation 4.5 is exactly the same bound as Equation 4.4, which in turn is the best we can hope for when updating every modified object exactly once.

Finally, let us give an asymptotic bound for the Goel and Chaudhary's Algorithm. Recalling Equation 4.1, which in their implementation is performed with a *reduceByKey* function on the distributed pairs. This first step does not play a considerable role in the complexity of the algorithm, and for that reason, we will leave it aside. The most crucial part of this algorithm for the asymptotic behavior is the calculation of C_{N+1} from C_N, since it generates $\mathcal{O}(|C_N|^2)$ (or more precisely $\frac{|C_N|(|C_N|-1)}{2}$, which is quadratic nonetheless) new candidate concepts. This is repeated for every C_N with $1 \leq N < \max(|g'|)$ (or $\max(|m'|)$). Since in the worst case $\max(|g'|) = |M|$, and considering that $|C_1| \leq |M|$ (or $|G|$ when there are more attributes than objects), a big O bound for these steps would be

$$\mathcal{O}\left(\sum_{i=1}^{|M|-1} |M|^{2^i}\right) \tag{4.6}$$

Although $|\mathcal{L}| = \mathcal{O}(2^{\,max(|G|,|M|)})$ in the worst case, in practical scenarios, when there is highly correlated data, the amount of concepts could be much smaller than its exponential bound. However, Equation 4.6 reveals that the complexity of Goel and Chaudhary's Algorithm, although scalable and parallelizable, suffers from the *exponential* generation of candidate concepts, *regardless* of the sparsity of the formal context, e.g., if only one object in the formal context has its $g' \approx |M|$, then the algorithm would necessarily compute Equation 4.6 candidate concepts up to $|M|$.

4.6 Conclusions

In this chapter, the importance of incremental algorithms to mine from arbitrarily distributed algorithms has been discussed. Along with it, algorithms that contribute to the state and generalization of these incremental algorithms have been reviewed. Finally, a novel algorithm for incrementally updating a lattice $\mathcal{L}$ has been presented, yielding a computational complexity of $\mathcal{O}(|\mathcal{L}||G||M|\ \max(|G|, |M|))$. Furthermore, the results of the theoretical analysis suggest that Algorithm AddPair should be a better fit than the algorithm of Goel and Chaudhary regarding complexity, when the formal context is *sparse* (which is often the case in practice). However, both algorithms have several differences that make the comparison very hard to perform. Firstly, being *batch* vs *incremental*, and secondly, being parallel vs single threaded. The existing batch algorithm have the upper hand in scalability regarding the fact that it is an inherently distributed algorithm, or in other words, if

[22] With certain optimizations, a tighter bound is $\mathcal{O}(|\mathcal{L}||G|^2\max(|g'|))$ where $\max(|g'|)$ is the maximum amount of attributes related to an object. However, to simplify the math, we will use the simpler version of the two.

needed, more nodes can be added to the used cluster in order to compute the lattice faster. Furthermore, when it comes to flexibility, Algorithm AddPair is preferable because it does not require to restart the computation when new objects or attributes are added. Yet, ideally, a fully distributed solution to the incremental computation of arbitrarily distributed formal context must be explored.

5

Use cases

Contents

5.1 ESF "mon-séjour-en-montagne" e-commerce application 82
5.2 Recommendation needs 83
5.2.1 Unavailable ski lessons 83
5.2.2 Complementary products 84
5.3 Recommending Ski Lessons from Historical Purchases 85
5.3.1 Historical purchases: dataset 86
5.3.2 Approach 87
5.4 Recommending Complementary Products using FCA and Association Rules 90
5.4.1 Carts: dataset 90
5.4.2 Approach 91
5.5 Conclusions 92

This chapter focuses on illustrating how the proposed *merge* algorithm can be applied in a dynamic and distributed environment. Certain use-cases of the algorithms and methods presented in chapter 3 and chapter 4 are discussed. Nevertheless, all cases revolve around the use case of a system that consists of a stateless[23] e-commerce website where customers fill certain required data, to then be shown available ski lessons for them to book.

As mentioned in the chapter 1, this thesis has been carried out in collaboration with the French Ski School[24] and the Digital Innovation Hub (DIH4CPS[25]), whose main mission is to accompany enterprises in their digitalization process, and to encourage them to use Industry 4.0 practices such as the usage of data mining techniques to improve their processes. As a result of this collaboration, the studied works in this thesis related to Formal Concept Analysis have been put in practice considering the needs of ESF and the capabilities of the proposed methods.

In particular, the chapter is organized as follows, section 5.1 introduces the main software component that all use cases are related to. section 5.2 delves into the need of applying

[23] The system does not save state related to previous transactions, therefore, it does not know anything about the user.

[24] ESF as for its acronym in French, which means École du ski français

[25] https://dih4cps.eu/

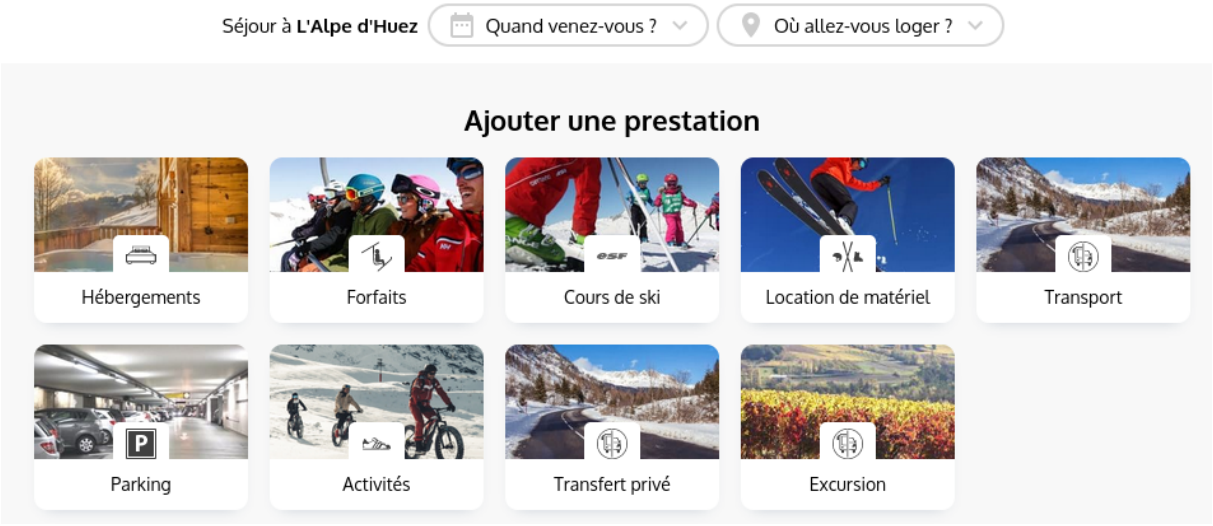

Figure 5.1: Snapshot of a development tool for the MSEM e-commerce website.

data-based recommendations in two particular parts of their processes, i.e., the recommendation of ski lessons, and the recommendation of alternative products. The section 5.3 discusses the modelling of the dataset of ski lesson purchases in order to recommend "similar" ski lessons in the case of no specific available lessons matching the whole set of parameters set by the user. In section 5.4, a method for recommending alternative products based on numeric coefficients using FCA and Association Rules is presented. Finally, section 5.5 concludes with an outline of the work done in the chapter.

5.1 ESF "mon-séjour-en-montagne" e-commerce application

The core component of the use-cases we tackled during this thesis revolve around the "mon-séjour-en-montagne" (MSEM) website, which literally translates to "*my stay in the mountains*". The main goal of the site is to provide users with a complete set of products to manage and plan their stay in the mountains. In particular, it includes not only ski lessons (which is the main product of the ESF, naturally), but also accommodation, equipment booking, ski-passes with certain durations, restaurant menus, Wi-Fi boxes, and so forth.

In Figure 5.1 a widget used for the development of the MSEM website is shown. Translating it to English, the image consists of a label saying in which *resort* the stay is going to be planned, and two dropdown inputs asking the users *when* are they planning to go, and *where* inside the resort are they going to stay. This is the minimum information that the ESF software needs to start looking for items to be shown to the users.

Particularly, the system consists of two components, as it is common nowadays, a *frontend* and a *backend* are responsible for the user interface (UI) and the application public interface that integrates the necessary read-only and write operations for the website to properly work, respectively. The backend, which is where the main focus of this work will be set upon, interacts with several external systems to provide its endpoints. Most importantly, when looking for *ski-lessons*, in particular, it communicates with a system that centralizes the information about all the different schools located across all resorts. This simple interaction is depicted in Figure 5.2.

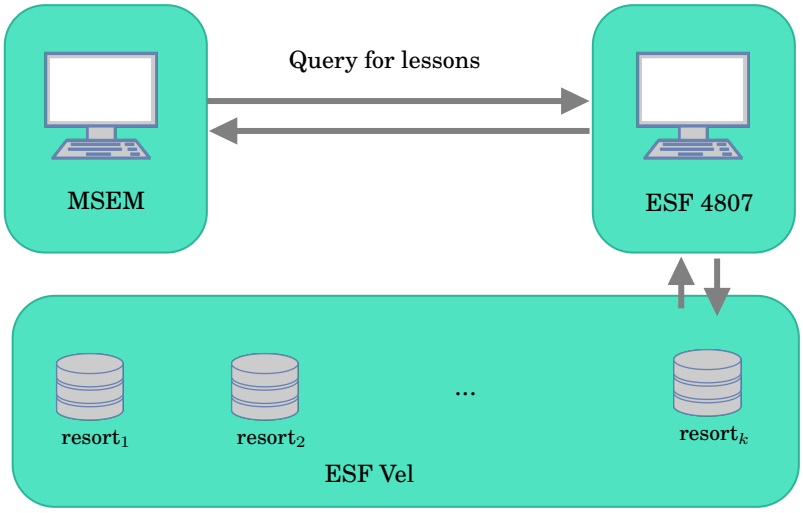

Figure 5.2: MSEM interaction with ESF centralized API in order to recover *available* lessons.

Then, a transaction in the e-ecommerce site will follow a life cycle like in the following example,

Example. 5.1: Transaction lifecycle in MSEM

A user sets the dates of his stay to be between the 2^{nd} and 9^{th} of January in the resort of Courchevel. Here, the MSEM site first makes an API call to the ESF 4807 system in order to retrieve what type of products they have available in that resort during the selected week. Then, with that information, the user can select what is he specifically looking for, for instance, a *group ski lesson*. Then, the MSEM system will add that information to the context, and query again to the external API in order to retrieve the available and necessary information on the next stage, which in this case would be the *levels*, i.e., which level would the user like the group lesson to be given in. Finally, the user selects which level is he/she expecting the lesson to be placed in, and just there the MSEM site will have enough information to do a full query into the external system in order to recover the entire available lessons to be shown to the user, as it is illustrated in Figure 5.3.

However, there are certain alternative paths the transaction can take that open the possibilities for a number of problems that will be discussed in the next sections.

5.2 Recommendation needs

5.2.1 Unavailable ski lessons

In this system, a feasible scenario is that a certain combination of parameters given by the user yield no available lessons (or items for that matter). In that case, the system at the moment only shows to the user a screen saying “no available lessons were found”. Needless to say, this is far from optimal considering the expected behavior of e-commerce

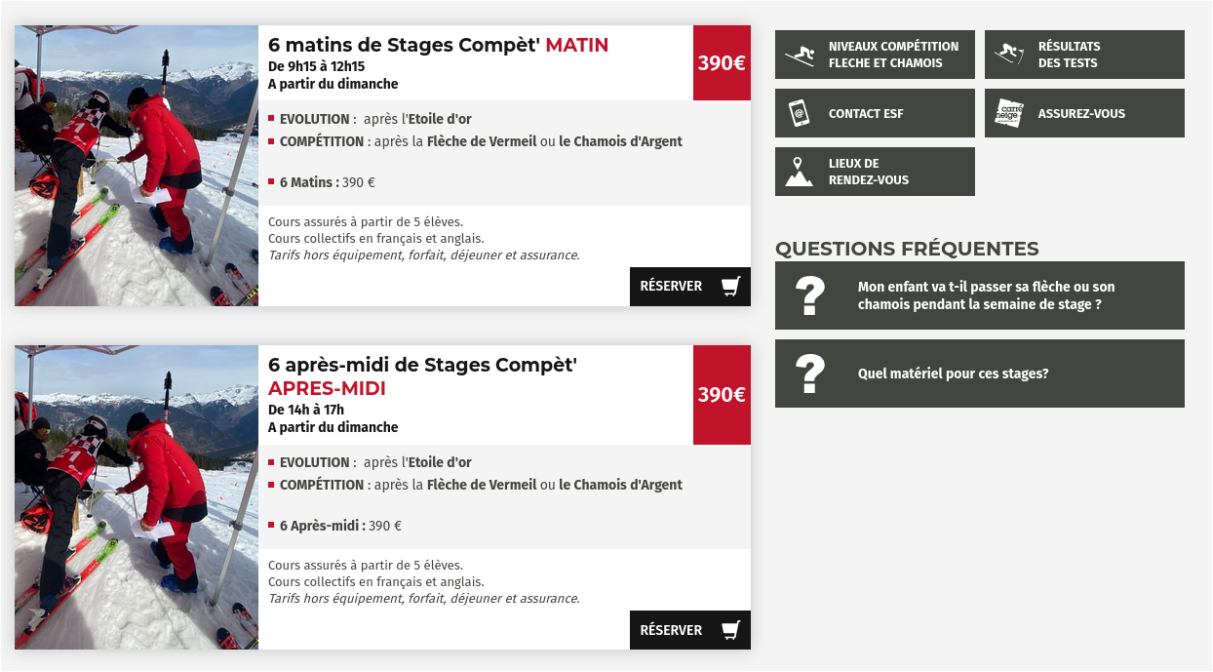

Figure 5.3: Snapshot of some courses being listed for the user ready to be booked.

sites nowadays: the tendency for most e-commerce sites is to provide users with alternative options to prevent the user to go look for other options on their own.

Additionally, understanding the interests and behaviors of users is extremely important for tailoring e-commerce websites to meet customer needs. User behavior data is usually stored in Web server logs. Traditionally, the analysis of this data has relied on data mining techniques that provide a static characterization of user behavior, often overlooking the sequence of their actions. Consequently, integrating a process-oriented perspective of users' sessions could be highly valuable for uncovering more complex behavioral patterns [56]. However, as mentioned before, the system that is being dealt with in this case is stateless, thus, at the moment of a request, the information available is very limited. In fact, it consists of only the parameters that the customer is looking for.

Nonetheless, the setting still needs a way of providing the customer with alternative options (lessons in this particular case) that are compatible with what they previously selected. Hence, the problem can be modeled as a recommendation problem, where given a certain set of parameters, the goal is to find available lessons that are "similar" to what the user looks for.

5.2.2 Complementary products

Another situation that is strategical for ESF to improve in their e-commerce site is the *discoverability* of complementary products. As already mentioned and shown in Figure 5.1, the main goal of the site is to provide the full range of options/products related to an entire stay in the mountains. But sometimes customers never even consider buying products other than the ski lessons themselves, or they simply buy/book these complementary products in other sites.

For that reason, the enterprise has put in place a recommendation system whose goal is

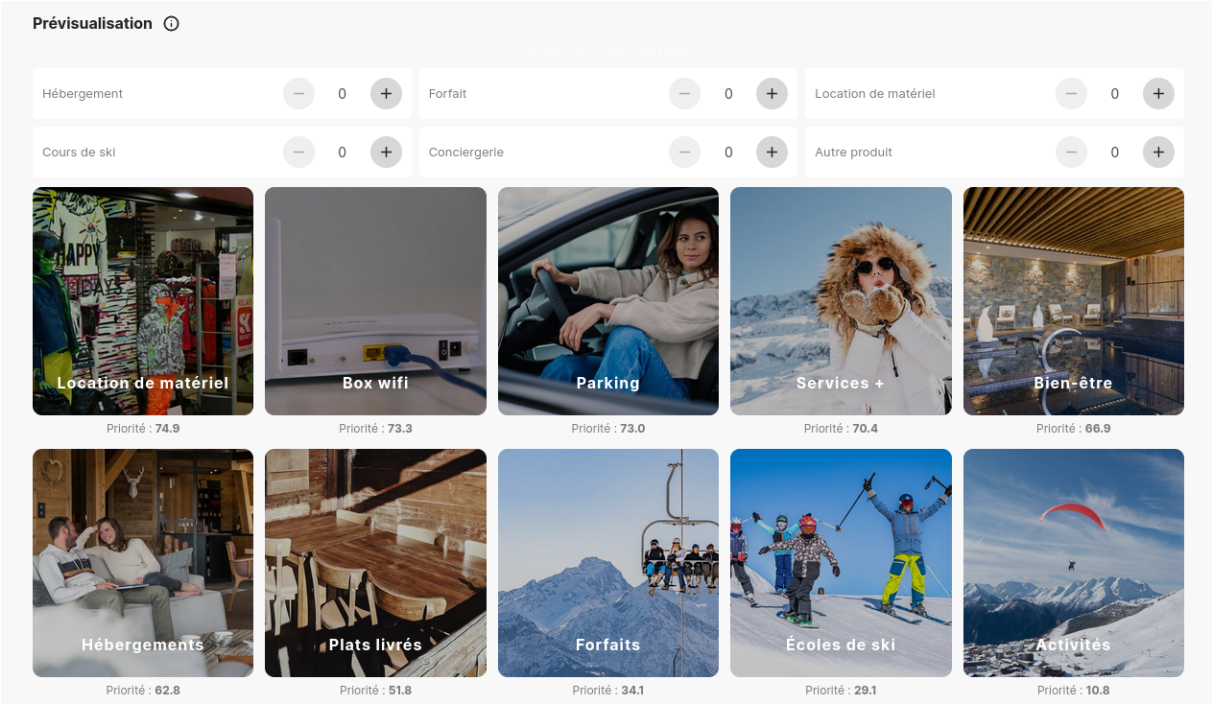

Figure 5.4: Cross-selling coefficients visualization tool.

to suggest complementary products such as Wi-Fi boxes, accommodation, ski passes, and so forth, based on *intuition* and somewhat *common knowledge*. More specifically, the system receives a *cart* filled with certain products, and for each of the complementary products, it returns a *coefficient* $c \in [0, 1]$ indicating how strongly that product has to be suggested. An example of a tool for visualizing the coefficients is depicted in Figure 5.4. The tool consists of a preview of the recommendations based on a cart, that is defined with "how many" products of each category are there in the cart, e.g., 3 ski lessons (cours de ski), and 3 equipment bookings (location de matériel), and so forth. Then, below the number of elements in each cart, the coefficients for each of the categories to be suggested are listed. In this particular example, the top recommendations would be equipment booking and right next to it, box Wi-Fi.

The problem that arises from this scenario is that intuition has inherent limitations, since it fails to be adapted to each of the resorts, whose types of common customers can vary meaningfully. In other words, the current system might work acceptably for some resorts, but not in general. Consequently, it would be ideal to have a method that (1) can be adapted to the different cases of every resort (or other criteria), and that (2) can dynamically adapt to the changes in user behavior over time.

5.3 Recommending Ski Lessons from Historical Purchases

One approach to deal with this problem is to use known facts about lessons to guide the recommendations. For instance, certain lessons are known to be incompatible based on their activity types and the resorts they are being taught in, and therefore they could be excluded when getting the possible recommendations. However, as mentioned before, resorts across

all France, and in general for that matter, can present different characteristics, many of which can be hidden in the data related to them.

In particular, in the ESF, all lesson purchases are stored as a set of characteristics i.e., the lesson in particular, or more generally as a list of lessons bought together, i.e., they were bought in the same cart in the e-commerce site. Since the site is stateless, at the moment of receiving a request for particular lessons, there is no available information about the user that is triggering the request, hence, the method to be proposed has to decide what to recommend based on the request itself plus the data available about historical purchases. Although there is a great deal of different methods to *learn* patterns from purchases and transactions, most of them require intricate or even niche modelling techniques to transform the input data into the input required by the method, which often is numeric. Thus, the process of explainability that is usually needed when recommending items in e-commerce sites becomes a challenge on its own. In this context, the proposed method leverages frequent itemsets (introduced in section 2.6) to identify hidden patterns in historical data. This approach is particularly well-suited because the impedance between frequent itemsets and the data is significantly lower compared to other methods. In other words, frequent itemsets align much better with the data, making them a strong theoretical foundation for recommendation algorithms that address the aforementioned problem.

5.3.1 Historical purchases: dataset

The recommendations considered for this problem are based on historical purchases. Notably, the dataset is a set of lessons that have been bought together in a certain timespan. In addition to the static version, it can be also seen as an ever-growing dataset from online purchases made by customers, and as such, it can be used for online learning. The entire data set consists of essentially three data-streams related to the event of a purchase. The events of first one are triggered by external purchases that took place in an ESF marketplace other than MSEM website. Its events consist of the elements described in Table 5.1.

Name	Description
Id	An ID indicating to which transaction the event is related to.
Kind	Indicates if the purchase corresponds to a group, private, competition, or a nursery ski lesson.
Level	In case the even corresponds to a lesson, this column indicates the level the lesson.
Discipline	It could be Alpine ski, Nordic ski, Snowboard, Hiking, Nordic Skate, handicap ski, deep skate, biathlon, and so forth.
Duration	A number indicating how many minutes each lesson takes.
NumberOfPeople	How many people fit in the lesson.
Price	The price of the lesson.

Table 5.1: Events of the data stream corresponding to purchases taking place in ESF external marketplaces.

The second one, corresponds to purchases that took place at the MSEM marketplace. In addition to exactly the same set of elements in Table 5.1, the events also possess the elements described in Table 5.2.

Name	Description
Lang	A discrete value (a.k.a., a multicolumn) indicating the language of preference of the purchase.
Lift	A boolean value indicating whether the purchase contains a ski pass to a lift.
Assu	A boolean value indicating whether the purchase contains an insurance package.
SkiRental	A boolean value corresponding to the booking of equipment for skiing.
Hotel	A boolean value indicating whether the purchase contains the booking for a hotel.
Transfer	A boolean value regarding the purchase including the transfer to from their location to the ski lift.

Table 5.2: Events of the data stream corresponding to the MSEM marketplace.

And finally, there is a last data stream that corresponds to "custom" purchases manually loaded into the system by the customer service staff. In this data stream, the events, instead of tuples, are JSON[26] objects. However, even though they have a semi-unstructured nature (i.e., entries are manually written by the staff, accounting for misspelled names/-words, different naming conventions, and fields not being present.), it is possible to parse them in real time getting a tuple with compatible columns as the ones previously described.

Within these three data streams lies an *underlying* formal context (recall its definition in Definition 3.2), using the *union* of all attributes, even if they are not necessarily present in each of the events (hence, the need for flexibility in this specific case). **Such a formal context will grow infinitely over time because of the nature of the data-stream coming from purchases that are constantly happening**. However, as mentioned before, the more purchases an algorithm has to process, the more computation time and space it takes to update the lattice (i.e., the problem of building a lattice is known to be #P-complete [75]). Thus, although it would be desired to maintain a knowledge-base considering *all* purchases, what is usually done is to consider a subset of them. Whether it is by taking into account only the latest k purchases, or the most interesting ones, it is inevitable to lose some information at some point when growing infinitely (as deeply discussed in chapter 3).

5.3.2 Approach

The approach we propose needs a new software component to be added in order to aid the MSEM system to deal with the case in which there are no lessons available with the selected characteristics. The new component presents an API to the MSEM system to ask for recommendations based on certain characteristics when needed, as depicted in Figure 5.5.

More particularly, the proposed recommendation system maintains a sound and useful concept lattice (to then be used for needed recommendations) corresponding to the sliding window of the latest k purchases ($\mathcal{L}$ in Figure 5.6). This is accomplished by incrementally adding each of the purchases in real time. Then, regarding the knowledge that might be lost when updating the lattice with only the latest purchases, the lattice to be used in the recommendation is the result of the *merge* between $\mathcal{L}$ and a lattice snapshot (i.e., as defined in [84] and depicted as $\mathcal{L}_s i$ in Figure 5.6 for $1 \leq i \leq n$ and $i, n \in \mathbb{N}$) that has been previously labelled as having certain valuable knowledge about a particular feature of the business, or

[26] Java Script Object Notation. A standard format used on the web to interchange information https://www.json.org/json-en.html.

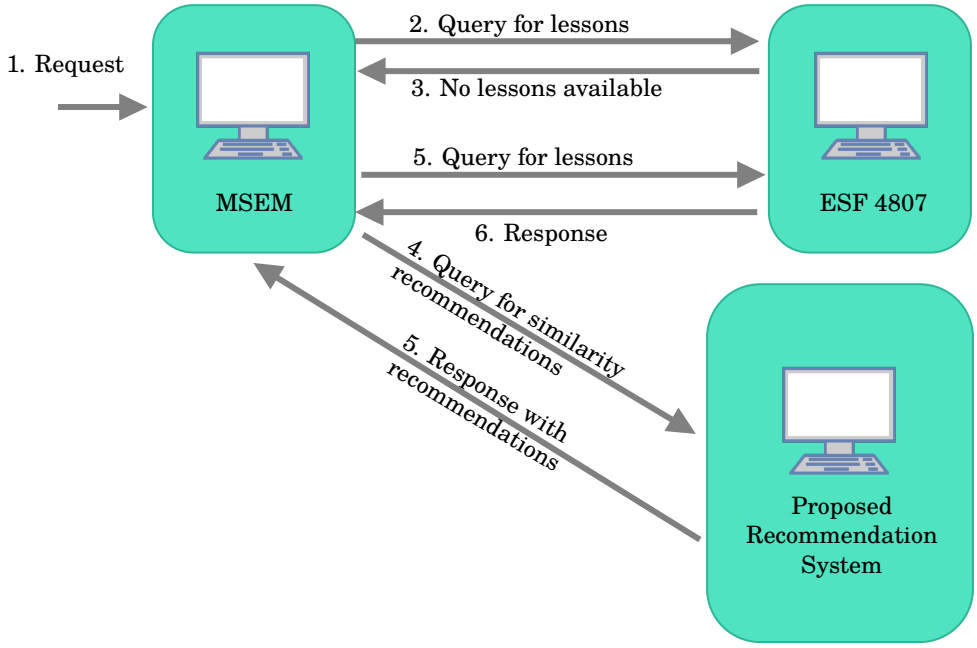

Figure 5.5: MSEM system calling the API of the recommendation system when no lessons are available with the selected characteristics.

an event with certain characteristics. Although choosing how to best take lattice snapshots goes outside the scope of this thesis, some examples are illustrated in the Example 5.2.

Example. 5.2: Different moments in which to take snapshots

The system could take snapshots every time a specific sale is made. Additionally, when a particular event such as the COVID-19 pandemic happens. Or in particular moments of each season.

With all that in consideration, the recommender system's API allows the addition of new purchases to the knowledge base, and, based on configuration, it updates the *current lattice* $\mathcal{L}$ accordingly. Then, given an indexed database of lattice snapshots, there is and endpoint to decide which one to use in order during the merge step. The minimal REST[27] API for this system to work is described in Table 5.3.

Assuming that for each purchase in the data streams a call to the *add_purchase* endpoint is triggered, the latest k lattice $\mathcal{L}$ would contain the knowledge about the latest or "current" trends. Then, as it is illustrated in Figure 5.6, when a request for a recommendation is done, the system performs a merge between $\mathcal{L}$ and the specified (in the request) $\mathcal{L}_s i$, let us call it $\mathcal{L}_m$. Then, given the $l \in \mathbb{N}$ characteristics $A = \{c_1, c_2, \ldots, c_l\}$, the system would look for association rules $A \to B$ and recommend $A \cup B$ in decreasing order on the product between its confidence and support. Notice that if lessons with A characteristics were not available, then $A \cup B$ will not be available either. Hence, one of the challenges of the recommendation system is to find an $A' \subset A$ that is semantically as close as possible to A and only then look for the association rules $A' \to B$ such that $A' \cup B \neq A$.

For this particular case, characteristics have an assigned weight $\theta(c_i)$ that is predefined

[27]Representational state transfer https://www.ibm.com/topics/rest-apis.

Name	Method	Description
add_purchase	POST	Given a set of standardized characteristics representing a purchase, it updates $\mathcal{L}$ by adding the corresponding intent, and deleting the oldest purchase.
add_snapshot	POST	Given a lattice snapshot, it adds the new entry in the snapshots DB.
snapshots	GET	It returns the IDs of all snapshots (commonly paginated).
get_recommendation	GET	Given a set of characteristics coming from the MSEM, the ID of a snapshot to use, and an integer n, it returns the n "most similar" sets of characteristics.

Table 5.3: REST API for the recommendation system using the frequent itemsets notion.

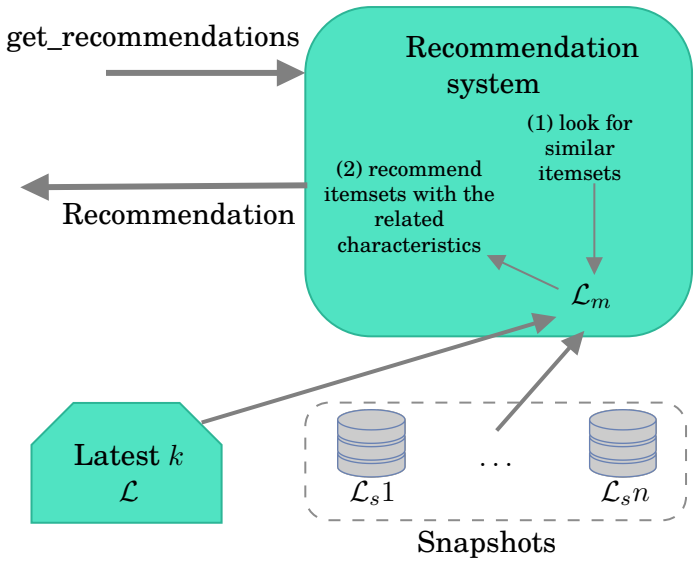

Figure 5.6: Recommender system using merge of lattice snapshots strategy.

based on business logic. For instance, the *kinds* (e.g., SKALPN, SKNORD, etc) have the highest weight among all characteristics, because, for the business, two lessons with different kinds should be considered "very different". Then, association rules $R = A' \to B$ are searched in decreasing order of the sum of the weights in the antecedent multiplied by its confidence and support (see Equation 5.1).

$$\Big(\sum_{x \in A'} \theta(x) \Big) \kappa(R) \sigma(R) \tag{5.1}$$

With this in mind, the MSEM receives a response with a collection of new sets of characteristics (i.e., $A' \cup B$), sorted by decreasing order in Equation 5.1, to then query the ESF 4807 for availability, and hopefully, to show to the user as alternative and semantically similar lessons. Example 5.3 illustrates a particular case of how these recommendations work.

Example. 5.3: Particular recommendations given a user request

Given a user request looking for a ski lesson with the following parameters {**GL**, **SKALPN**, **nbP**<**3**, **INSTRUCTOR=Arnaud**}, let us suppose that no available lesson is found matching all of them. Additionally, let us imagine that at the moment there is a high probability of storms happening at some point in the day. If in those situations, Arnaud **usually** takes *only private lessons*, a snapshot taken in the past, when there were those conditions, would most likely have concepts containing that pattern. And hence, if using that snapshot in the merge step, the recommendations would likely include a ski lesson only varying the lesson type from **GL** to **PL**. However, if no merge step is included, the "current" lattice would lack that pattern that only happens in that specific scenario.

5.4 Recommending Complementary Products using FCA and Association Rules

Similar to the approach taken in section 5.3, to address the previously described problem of discoverability, a data stream of *purchases* will be used as the source of data for a data-based recommendation system. However, the difference lies in the fact that these purchases are actually *carts* with a number of items each. Moreover, to minimize the cost of integrating the new system into the existing infrastructure, the solution has to be compatible with their previous, intuition-based system, that assigns coefficients $c \in [0, 1]$ to "the most likely product to recommend" based on the current contents of the cart.

5.4.1 Carts: dataset

For this purpose, the data stream to be used consists of the following elements,

Consider that at any moment, new elements could be added to the data stream, since it is being viewed as a real time data stream. Hence, the need for a *flexible* algorithm to learn from it.

The underlying formal context used to define the lattice to then extract frequent itemsets from consists of a discretization of the fields that are **int** type. The used ranges for the discretization were decided based on business logic, although, in this case, it could have been left as "one column per different number", since there are not many different variations among the values. However, this might depend also on the target to recommend and the analysis in particular that is needed for it. For the purpose of this thesis, this is considered to be out of scope nonetheless.

The particular discretization for each of the **int** elements was converting them into 'multi-columns" considering certain ranges. For example, n_skilesson was discretized into the attributes $[0, 1)$[28], $[1, 2)$ to $[2, 3)$ and $[3, \infty)$. Likewise, the other elements followed a similar process. A snapshot of a derived context concerning a certain timespan of the data stream can be seen in https://gitlab.com/cps-phd-leutwyler-nicolas/thesis_documents/-/blob/main/cross-selling/context.csv. The context follows the CSV norm of having one line per incidence with the ID of an object in the first column, and the ID of an attribute in

[28]For an integer $n \in \mathbb{N}$, we say that $n \in [x, y)$ when $x \leq n < y$

Name	Type	Description
n_skipass	Int	How many ski passes are there in the cart.
lodging_adults	Int	How many lodging bookings for adults are there in the cart.
lodging_children	Int	How many lodging bookings for children are there in the cart.
n_lodging	Int	How many lodging booking units are there in the cart.
n_concergierie	Int	Did they book a concierge? How many of them?
n_rental	Int	Equipment booking units.
n_skirental_products	Int	Equipment booking products (e.g., basic kit for children).
n_skilesson	Int	How many *group* ski lessons are there in the cart.
n_custom_skilesson	Int	How many *private* ski lessons are there in the cart.
has_wifi	Boolean	Is there a box Wi-Fi in the cart?
has_serviceplus	Boolean	Is there the service plus in the cart?
has_parking	Boolean	Did they book a parking spot?
has_activities	Boolean	Did they book special activities?
has_bien-être	Boolean	Did they book the spa experience?

Table 5.4: Events of the data stream corresponding to the cross-selling recommendation system.

the second one, meaning that object has that specific attribute. Notice that the discretized multi-columns are not mutually exclusive as depicted in the Example 5.4.

Example. 5.4:

Let us imagine that a particular cart has 3 group ski lessons attached, then, its related object in the underlying formal context would have the attributes n_skilesson_1, n_skilesson_2, and n_skilesson_3-5.

The purpose of this decision in the modelling of the context was to include the likelihood of a particular set of items having $x \in \mathbb{N}$ amount of a particular one such as a ski lesson, to have $x + 1$. Thus, in addition to recommending only complementary products, the system can also recommend adding more items of the same product based on the particular trends in the dataset. However, the drawback that comes with it is the addition of several trivial association rules such as $\{n\_skilesson\_2\} \rightarrow \{n\_skilesson\_1\}$ that will always have 100% confidence.

5.4.2 Approach

The process of maintaining an "updated" lattice, and the process of recovering knowledge from lattice snapshots are equivalent to the ones presented in section 5.3. In this case, however, the goal is to match the recommendations with different numerical coefficients based on how likely it is for a product to be selected next given a specific cart.

To do so, given a cart with C products, the followed approach consists of filtering the

association rules $R = A \rightarrow B$ such that $\varphi(C) \subseteq A$, where φ is a function that maps products into attributes of the context (see Example 5.5). Then, any of the items in B could be recommended based on the support and confidence of R.

Example. 5.5:

For example, given a cart with 4 group ski lessons, and 2 ski passes, $C = \{n\_skilesson\_4, n\_skipass\_2\}$, its mapping function to attributes in the context would be $\varphi(C) = \{n\_skilesson\_3-5, n\_skipass\_2\}$.

Moreover, the approach only considering the mentioned association rules suffers from the problem that it will never recommend a "new" product if there are no occurrences of that itemset in the dataset at the moment of doing the recommendation (remember that the dataset is constantly evolving). To mitigate this, the probabilities of choosing a product given an empty cart are considered with a weighted average in the final probability. For instance, if given the products C, there are no itemsets $\varphi(C) \cup \{box\_wifi\}$, but the probability of choosing a Box Wi-Fi given an empty cart is $x\%$, the probability to consider recommending that product would be $\frac{w_l * x}{w_p + w_l}$ where w_p is the weight given to the probabilities given by the association rules, w_l is the weight given to the probabilities given an empty cart, and $w_p > w_l$.

5.5 Conclusions

The online learning of concept lattices has been implemented in certain domains that utilize distributed architectures. It involves the essential use of techniques such as sliding windows to maintain a bounded size. This approach is necessary because the computational cost would otherwise become unmanageable at some point. Nevertheless, there is an inherent problem to this approach which is the potential lost of meaningful knowledge (i.e., concepts). This chapter explores two recommendation systems employing an architecture that capitalizes on the benefits of storing lattice snapshots. These snapshots are then leveraged in contrast to an updated online lattice to recover a portion of the knowledge that would otherwise be lost.

The proposed architecture offers *flexibility* by imposing minimal constraints to its implementation. No matter how the lattice snapshots are distributed, nor where the online lattice is maintained, as long as the objects of the context (i.e., events in the data stream) are disjoint. Attributes can be equal, disjoint, or have certain elements in common. In addition, it offers *scalability* regarding how much knowledge can be considered when recommending by using smaller sliding windows (and thus being able to process more events by time unit with the same resources), because a part of it will be recovered in the merge stage. As exemplified in Figure 5.7, let us imagine a concept that appeared during a certain event that has already fallen out of the sliding window, in our approach, this event could be recovered with a snapshot that was taken into account at that moment, however without the merge step, the only way to do so, would be to have a larger sliding window.

The first covered use case consists of a system for recommending ski lessons from historical purchases. The way it works, given an API that allows updating an online lattice (that we called $\mathcal{L}$) with purchases being made in the different marketplaces, is by recommending lessons that are similar to the one selected by the user, based on association rules that

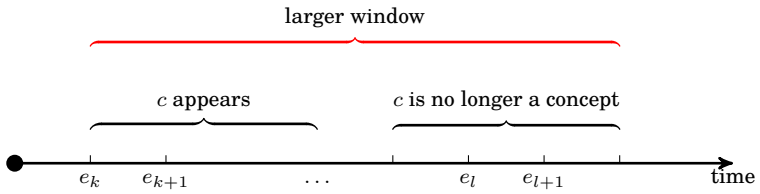

Figure 5.7: Example of a smaller sliding windows being used, but being able to consider an older event nonetheless.

come from a "richer" lattice $\mathcal{L}_m$, that is the result of the merge between $\mathcal{L}$ and some snapshot $\mathcal{L}_s$. The recommended purchases are similar to the one in the request because they are *semantically close* to it in the conceptual and hierarchical structure provided by FCA. This semantic distance is powered by not only the lattice diagram but also by business known facts considered in Equation 5.1. Even though the recommendations are sensible in the sense that given a set of characteristics A, they are $A' \to B$ and $A' \subseteq A$, it is still possible that some recommended lessons will not be available either. Thus, there is still room for improvement in the approach.

The second use case shares many characteristics with the first one. However, the main difference lies in the fact that it considers association rules $A' \to B$ such that $\varphi(C) \subseteq A'$ given a cart C. Additionally, the final output of the algorithm had to be a probability $0 \leq p \leq 1$ for each of the products of the MSEM. Since the problem to be addressed was that of the *discoverability* of products, apart from considering association rules in the prediction of the next product to recommend, the different ratios of each of the products considering an empty cart were used to create recommendations that, if only based on data, would not exist.

The purpose of this chapter is to illustrate the usefulness of the proposed "merge" algorithm in a dynamic and distributed setting. Conversely, it is by no means to compare the different metrics used in recommendation systems, and thus, the aim of the chapter does not lie in whether the recommendations have a high *accuracy* [67], or a high *F-score* (which are typical metrics used in supervised learning [70]). However, it is important to acknowledge that understanding the performance considering those metrics is a much-needed step into the practical usage of FCA in distributed architectures, and for that reason we deem it a worth research direction to pursue in the future. Nevertheless, certain studies have been carried out concerning the effectiveness of recommending using lattices and frequent itemsets [148, 72], and they can be extrapolated to this case, since, in the end, it can be understood as recommending with $\mathcal{L}_m$ all along. Thus, the advantages of the proposed method regarding distributed architectures might only lie in its computational performance gains.

Moreover, one of the aspects that is completely related to this thesis is that of *scalability*, which refers to how this architecture would behave when inputs become larger. As a solid theoretical starting point on the subject, the computational complexity of the algorithm has been shown in chapter 3. Nevertheless, while theoretical analysis provides valuable insights, the need for empirical verification cannot be overstated. Such testing would shed light on the behavior of diverse data streams. For this reason, the next chapter will cover a number of different empirical scenarios, providing experiments for each of them.

6

Evaluation

Contents

6.1 Single threaded batch merge 96
6.1.1 Implementation and setting for the experiments 96
6.1.2 Experiments with real datasets 96
6.1.3 Experiments with synthetic datasets 101
6.1.4 Discussion 102
6.2 Incremental minimal updates 102
6.2.1 Comparison with AddIntent 105
6.2.2 Comparison with Goel and Chaudhary's algorithm 105
6.3 Distributed batch merge using Apache Spark 109
6.3.1 Experiments on distributed clusters 109
6.3.2 Remarks about the results 109
6.4 Conclusions 112

In this chapter, several experiments are carried out to provide an empirical understanding about the behavior of the presented algorithms. Additionally, certain comparisons with existing algorithms and methodologies are considered as well. In particular, the goal of the evaluation is to understand what are the empirical computational gains (in time) of the algorithms proposed in chapter 3, and chapter 4. To do so, the metric considered in all experiments was the *wall time*[29] each algorithm took with a given input. Moreover, a key concern is how efficiently the algorithms handle increasing input sizes (i.e., scalability). To address this, the experiments were specifically designed to analyze the growth of computation time.

The chapter is structured as follows, section 6.1 presents a comparison between the single threaded implementations of the merge algorithm discussed in chapter 3. It includes a real dataset coming from the use case introduced in section 5.3, and a synthetic dataset with the goal of covering a wide range of characteristics of the formal contexts.

Following, section 6.2 uses the same datasets to benchmark the construction of a concept lattice from scratch using the algorithm presented in chapter 4. In addition, it provides a discussion about the implications of the results. Finally, it concludes with some final remarks and future work directions.

[29]The time the algorithm takes from the moment the **user** runs it, until the user gets the response.

As the last part of the evaluation, and to be consistent with the experiments, in the section 6.3, the distributed version of the datasets are used to give empirical evidence about the performance related to the distributed algorithm to merge lattices presented in chapter 3.

6.1 Single threaded batch merge

6.1.1 Implementation and setting for the experiments

To facilitate a comparative analysis between the two single threaded algorithms BatchMerge (i.e., the approach proposed in this thesis), and IncrementalMerge (i.e., an approach using *existing* methods), it is essential to ensure a high degree of similarity in their implementation. Thus, *C++* implementations were used for each of the algorithms[30] following similar practices (e.g., passing result references instead of creating structures from scratch, and using virtual methods). In addition, for each of the comparatives, the used input dataset should be exactly the same, having the same sizes, densities, and the same characteristics. Considering this, in all experiments, a lattice snapshot with $|\mathcal{L}|$ concepts is used as the online "latest k purchases", while another one, with $|\mathcal{L}_s|$ is considered to be the snapshot to be merged. The parameters taken into account for the experiments were the sum of the lattice sizes $|\mathcal{L}| + |\mathcal{L}_s|$, and the size of the final merged lattice $|\mathcal{L}_m|$.

Experiments were carried out on an 11th Gen Intel® Core™ i7-11850H @ 2.50GHz × 16 with 32 GB RAM, and run using Jupyter Notebooks[31] to generate the graphs and to leave a footprint of reproducible sets of experiments. The main parameters that were considered in all experiments were the size of the two input lattices $|\mathcal{L}|$, and $|\mathcal{L}_s|$, the ratio between their sizes (e.g., $1:1$ if they have roughly the same size, $1:\frac{1}{2}$ if the second one has half the size of the other), the amount of objects in the context, and the density (i.e., the max($|g'|$)).

6.1.2 Experiments with real datasets

The real world dataset used to perform these experiments is the one introduced in chapter 5, and more precisely, in subsection 5.3.1. Since the *online* datasets do not have a *fixed* size, the experiments were performed on a fixed *subset* of it (i.e., $\mathcal{D}$) containing $1\,915\,285$ rows (i.e., one for each purchase instance), from which one of the columns is the *resort* where the purchased lesson is given. Then, one pair of lattices $\mathcal{L}$, and $\mathcal{L}_s$ was generated with different sizes based on the limitations of the resort (e.g., some resorts have fewer purchases than others, others are more sparse). Moreover, the ratio between $|G|$ and $|G_s|$ has also been taken into account (i.e., the ratio between the amount of purchases in each of the underlying formal contexts).

Firstly, we considered the objects' size ratios of approximately $1:\frac{1}{2}$ and $1:\frac{1}{3}$, yielding the graphs depicted in Figure 6.1, and Figure 6.2. Afterward, the objects' size ratios of approximately $1:1$ and $1:\frac{3}{4}$ were used, giving the final graphs shown in Figure 6.3, and Figure 6.4. In Table 6.1, all parameters and results for each test and algorithm are presented. The name of each experiment comes with the suffix *x_y* representing the ratio between $|G|$ and $|G|_s$. Ultimately, the outcome of every experiment is the duration dedicated to executing both Algorithm IncrementalMerge and Algorithm BatchMerge. The Jupyter Notebook for

[30] Implementation of the algorithms: https://gitlab.com/Lwr/fca_algorithms_cpp
[31] https://jupyter.org/

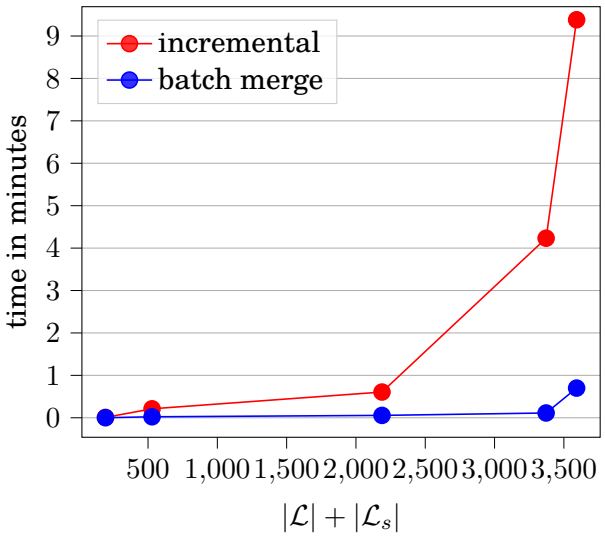

Figure 6.1: $|\mathcal{L}| + |\mathcal{L}_s|$ vs minutes. $|G|$ ratio $1 : \frac{1}{2}$

all experiments can be found in https://gitlab.com/cps-phd-leutwyler-nicolas/thesis_documents/-/blob/main/experiments/real_world_experiments_using_purchases.ipynb.

For the first comparison, the resort La Grave was used. Its latest 503 purchases were taken into account and represented in their respective $\mathcal{L}$, whereas $\mathcal{L}_s$ corresponds to the latest 302 with an offset of 200, i.e., 200 purchases before the first occurrence in $\mathcal{L}$. In this execution, the execution of the incremental approach was roughly 5 times slower than that of the merge one, i.e., 511ms and 112ms respectively.

Regarding the second comparison, the considered resort was Les Saisies, while the lattices used were the one corresponding to the latest 2130 purchases as $\mathcal{L}$, and the latest 1000 with an offset of 400, which from now on would remain constant. This comparison resulted in an even larger difference, being the merge almost 9 times faster than the incremental counterpart.

In respect to the third experiment, which is related to purchases made for the Alpe d'Huez resort, the two lattices considered were the underlying lattice of the latest 1937 purchases, and the corresponding one of the latest 985. This experiment continued the trend of increasing the time gap between both runs: the merge algorithm run in 3.37s whereas the incremental in 36.4s.

The last comparison with an approximate ratio of $1 : \frac{1}{2}$ considered the Courchevel resort. As for the purchases, it took the last 6814 ones as the base for $\mathcal{L}$, and the latest 3971 with an offset of 400 as the base for $\mathcal{L}_s$. Here, the difference between the algorithms grew staggeringly to 37 times less for the batch merge.

The fifth experiment had an approximate ratio of $1 : \frac{1}{3}$, and used the Meribel resort. The lattices taken as input were the one computed from the latest 14326 as $\mathcal{L}$ and the one coming from the latest 4416 with the already fixed offset of 400. In this case, the difference gap was not as impressive as in the fourth one, but still considerably large: the incremental approach took approximately 13 times more than the batch merge one.

Regarding the experiment E_1_1_1, the used resort was La Grave. $\mathcal{L}$ was the same as the one selected in E_1_0.5_1, while $\mathcal{L}_s$ corresponds to the latest 502 purchases with the offset of 400. In this comparison, the time gap between the two algorithms was of almost 8 times,

Experiments	$\lvert\mathcal{L}\rvert$	$\lvert G\rvert$	$\max(\lvert g'\rvert)$	$\lvert\mathcal{L}_s\rvert$	$\lvert G_s\rvert$	$\max(\lvert g'_s\rvert)$	$\lvert\mathcal{L}_m\rvert$	IncrementalMerge time	BatchMerge time
E.*I.*$\frac{1}{2}_1$	118	503	9	75	302	9	139	511ms	**112ms**
E.*I.*$\frac{1}{2}_2$	367	2130	8	162	1000	9	408	12.7s	**1.44s**
E.*I.*$\frac{1}{2}_3$	1341	1937	10	847	985	10	1651	36.4s	**3.37s**
E.*I.*$\frac{1}{2}_4$	3272	6819	11	100	3971	7	3305	4min 14s	**6.78s**
E.*I.*$\frac{1}{3}_1$	3199	14326	11	393	4416	10	3298	9min 23s	**42s**
E_*1*_*1*$_1$	118	503	9	85	502	9	142	1.81s	**220ms**
E_*1*_*1*$_2$	367	2130	8	209	2000	9	428	1min 2s	**3.85s**
E_*1*_*1*$_3$	1341	1937	10	1307	1954	10	1964	2min 21s	**9.14s**
E_*1*_*1*$_4$	3272	6819	11	432	5386	9	3637	11min 30s	**19.3s**
E.*I.*$\frac{3}{4}_1$	3199	14326	11	605	11315	10	3448	1h 43min 10s	**2min 26s**

Table 6.1: Sizes for each experiment.

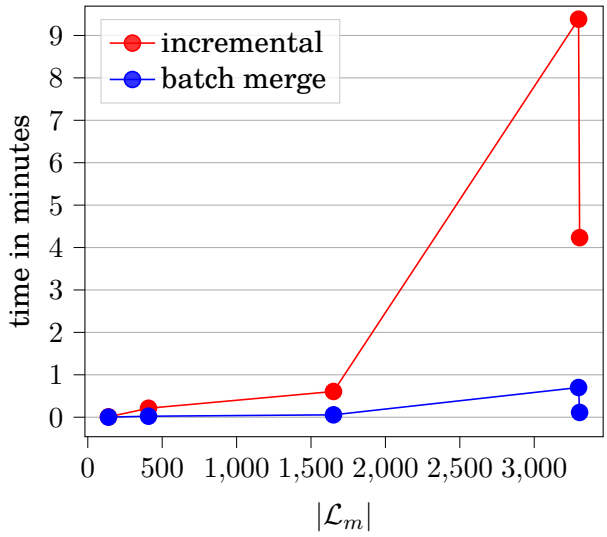

Figure 6.2: $|\mathcal{L}_m|$ vs minutes. $|G|$ ratio $1 : \frac{1}{2}$

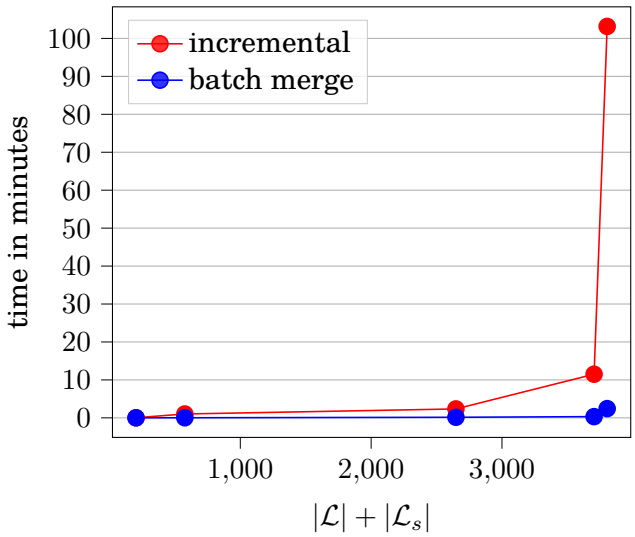

Figure 6.3: $|\mathcal{L}| + |\mathcal{L}_s|$ vs minutes. $|G|$ ratio $1 : 1$

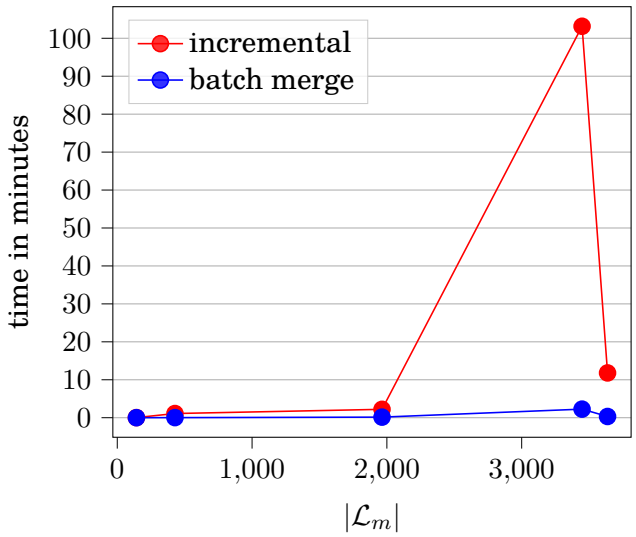

Figure 6.4: $|\mathcal{L}_m|$ vs minutes. $|G|$ ratio $1:1$

being larger than the one using a ratio of $1:\frac{1}{2}$.

The second comparison with a ratio of $1:1$ considered the resort Les Saisies, the same lattice $\mathcal{L}$ as in E_1_0.5$_2$, and the latest 2000 purchases with the fixed offset as $\mathcal{L}_s$. Invariably, the merge algorithm outperforms the competition in this scenario. Moreover, the gap is also larger than the $1:\frac{1}{2}$ ratio counterpart: 15 times vs. 9 in the two different ratios respectively.

The third experiment used the resort of l'Alpe d'Huez, and, although $\mathcal{L}$ remains unchanged, $\mathcal{L}_s$ corresponds to the latest 1954 purchases after 400 purchases difference with the first in $\mathcal{L}$. In this case, the increase of the gap comparing with the experiment with smaller ratio was from approximately 10 times in the $1:\frac{1}{2}$ to roughly 15 times in this one.

The last comparison with the ratio of $1:1$ considered the Courchevel resort, used the same $\mathcal{L}$ as E_1_0.5$_4$, and used the latest 5386 purchases with an offset of 400 as the $\mathcal{L}_s$. Here, the incremental method running time was of 11min 30s, whilst that of the batch merge one was 19.3s, making the latter about 35 times faster than the first one.

The last experiment, with a ratio of $1:\frac{3}{4}$, the Meribel resort, the unchanged $\mathcal{L}$ from its counterpart, and the $\mathcal{L}_s$ using the latest 11315 purchases with the fixed offset of 400. In this comparison, the time gap between the merge and incremental approaches reached a staggering difference of 42 times.

As depicted in Figure 6.1, and Figure 6.3, the time gap increases as the size of $\mathcal{L}$ and that of $\mathcal{L}_s$ grow. However, this is not true for the size of $\mathcal{L}_m$ as it can be observed in Figure 6.2, and Figure 6.4. This is very likely related to the amount of objects in each lattice, as well as their ratio, since the incremental approach relies considerably more on how many rows are there to process than the approach of merging considering directly the concepts in each lattice. Additionally, it is also possible to observe the increase in execution time needed for both algorithms as the ratio of objects present in the lattices to be merged grows. Nevertheless, that difference, for the batch merge approach, is not nearly as pronounced as the one found in the incremental one.

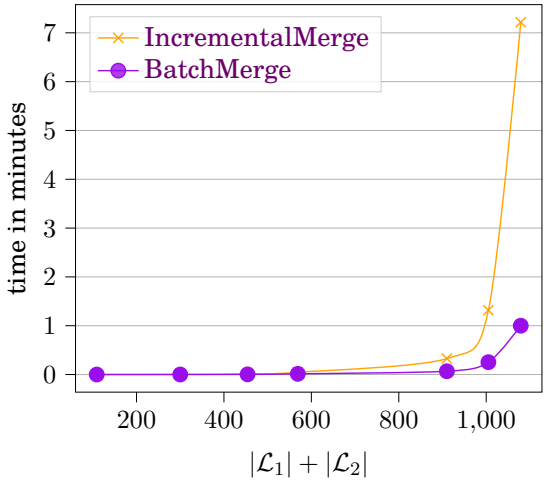

Figure 6.5: $|\mathcal{L}_1| + |\mathcal{L}_2|$ vs time in minutes. Experiments where $|G_1| = 2|G_2|$.

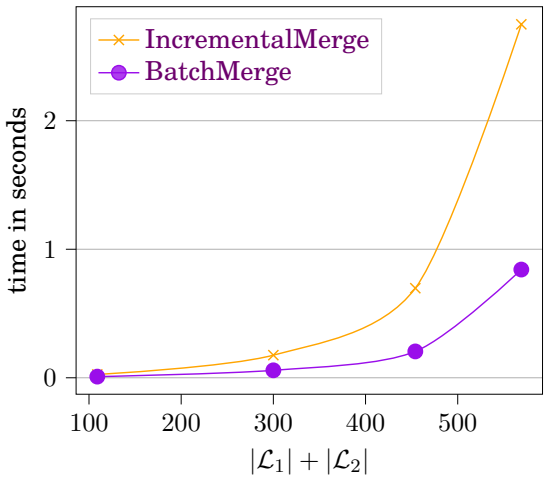

Figure 6.6: Zoomed version of Figure 6.5 in seconds and considering only the first 4 experiments.

6.1.3 Experiments with synthetic datasets

Regarding the synthetic datasets, seven pairs of contexts with an objects size ratio of $1 : \frac{1}{2}$ (i.e., $\frac{1}{2}|G_1| \approx |G_2|$), and seven more pairs with an objects size ratio of $1 : 1$ were randomly generated. All contexts have a constant size of attributes $|M| = 50$, and a $\max(|g'|) = 10$. Then, for each pair of contexts, their respective lattices were computed, merged with the two algorithms and their computation times contrasted. A Jupyter Notebook with all experiments can be found in https://gitlab.com/cps-phd-leutwyler-nicolas/concepts24-experiments/-/blob/main/concepts_experiments.ipynb.

With an object size ratio of $1 : \frac{1}{2}$:

In Figure 6.5, the 7 pairs of runs of both algorithms have been compared using the sum of lattice sizes contrasted with the run time. In Figure 6.6, only the first four experiments are depicted to have a clearer view that the difference in performance remains equivalent in that magnitude (i.e., seconds) as well. Similarly, Figure 6.7 and Figure 6.8 show the runs contrasted with the sum of object sizes $|G_1|$ and $|G_2|$. Algorithm IncrementalMerge was outperformed by Algorithm BatchMerge in all instances. The difference in run time between the algorithms ranged from 2.8 times, the smallest difference, to 7.2 times, the largest one.

With an object size ratio of $1 : 1$:

Regarding these other 7 pairs of experiments, in Figure 6.9, the runs of both algorithms have been also compared using the sum of lattice sizes contrasted with the run time. Analogously to the previous experiments, in Figure 6.10, only the first four experiments are depicted. In a similar vein, Figure 6.11 and Figure 6.12 show the runs contrasted with the sum of object sizes $|G_1|$ and $|G_2|$. Following the trend, Algorithm BatchMerge outperformed Algorithm IncrementalMerge across all instances. However, in this case the difference in run time

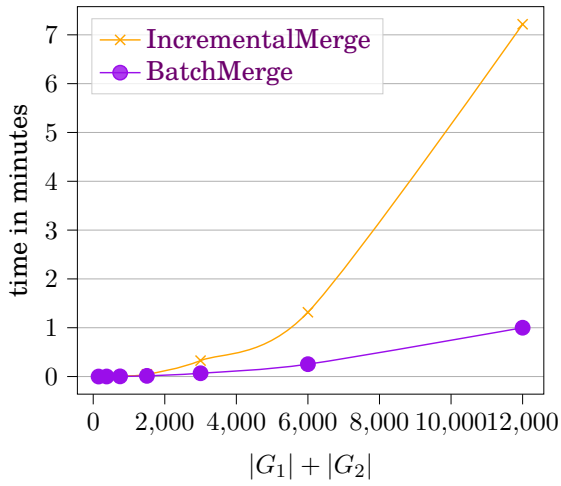

Figure 6.7: $|G_1|+|G_2|$ vs time in minutes. Experiments where $|G_1| = 2|G_2|$.

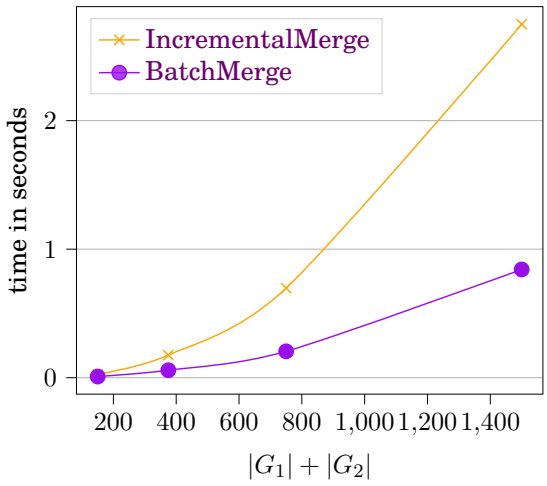

Figure 6.8: Zoomed version of Figure 6.7 in seconds and considering only the first 4 experiments.

between the algorithms was even larger, ranging from 4.5 times, the smallest difference, to 14.6 times, the largest one.

In Table 6.2, all the parameters of the experiments are shown. For each experiment, the name is an E followed by the ratio of object sizes and a sub-index indicating the number of the experiment. For instance, E.*1.*$\frac{1}{2}_1$ corresponds to the first experiment with an object size ratio of $1 : \frac{1}{2}$, whereas E.*1.1*$_5$ is the fifth experiment with a ratio of $1 : 1$.

6.1.4 Discussion

A set of experiments were carried out comparing Algorithm BatchMerge with a lattice construction algorithm (Algorithm IncrementalMerge) that is meant to leverage the fact that one of the lattices (i.e., the largest one) has already been computed. All experiments have shown that Algorithm BatchMerge outperforms Algorithm IncrementalMerge as the sparsity and the amount of objects in each lattice to be merged grow. In addition, experiments revealed that the difference in performance appears to grow linearly with the difference between object size ratios, e.g., if we double the objects size ratio between two runs, the difference in time will likely double as well.

Both real and synthetic datasets have shown similar results. However, it is important to mention that given the nature of the nested *for* statements in Algorithm BatchMerge, the denser the context gets, the less effective this difference will be. What is more, since in the worst case the amount of concepts is *exponentially larger* than the size of the formal context, the strategy of traversing $|\mathcal{L}||\mathcal{L}_s|$ pairs of concepts, when the ratio is close $1 : 1$, would be considerably more expensive than computing Algorithm IncrementalMerge.

6.2 Incremental minimal updates

While both Goel and Chaudhary, and AddPair algorithms offer distinct advantages, a direct comparison is hindered by their fundamental design choices. Goel and Chaudhary's algorithm operates in batches, processing all data at once, while Algorithm AddPair tackles data

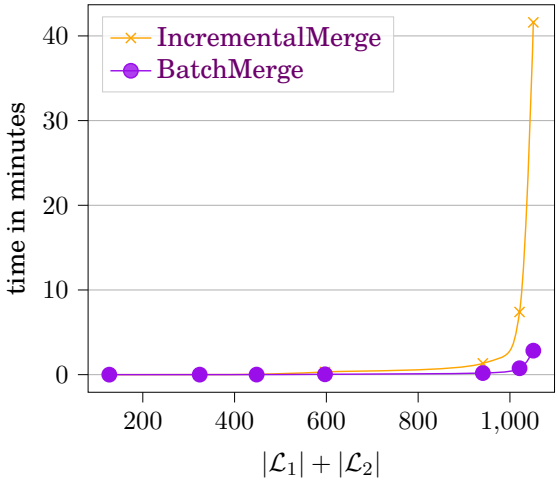

Figure 6.9: $|\mathcal{L}_1| + |\mathcal{L}_2|$ vs time in minutes. Experiments where $|G_1| = |G_2|$ for each pair of experiments.

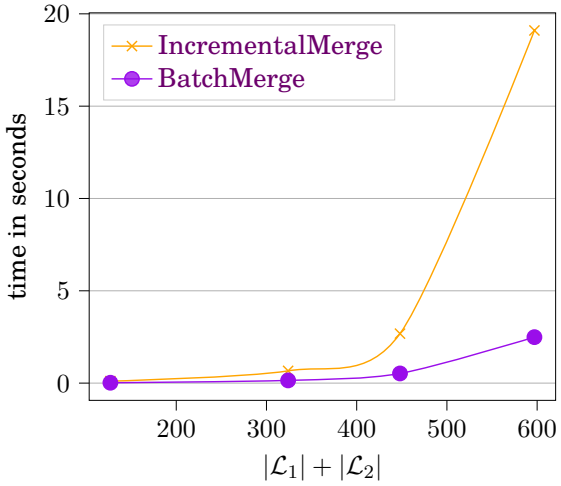

Figure 6.10: Zoomed version of Figure 6.9 in seconds and considering only the first 4 experiments.

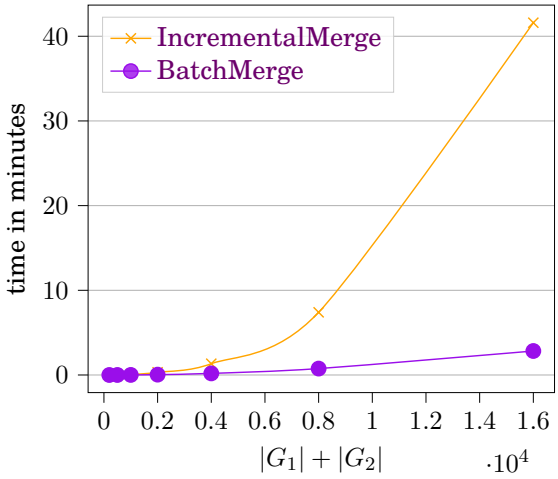

Figure 6.11: $|G_1| + |G_2|$ vs time in minutes. Experiments where $|G_1| = |G_2|$.

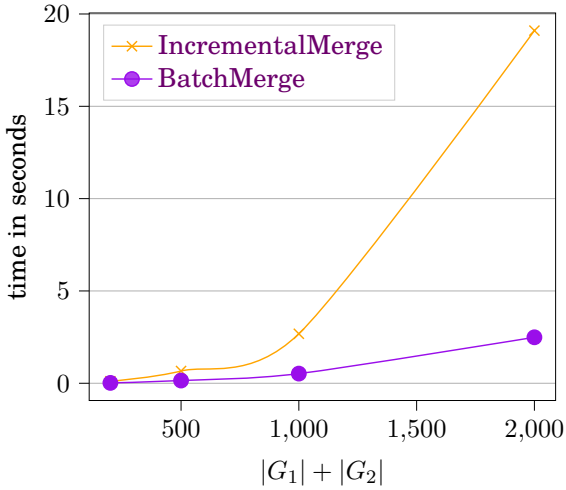

Figure 6.12: Zoomed version of Figure 6.11 in seconds and considering only the first 4 experiments.

Experiments	$\lvert\mathcal{L}_1\rvert$	$\lvert G_1\rvert$	$\lvert\mathcal{L}_2\rvert$	$\lvert G_2\rvert$	$\lvert\mathcal{L}_m\rvert$	IncrementalMerge time	BatchMerge time
E.1.$\frac{1}{2}_1$	62	100	47	50	183	26.5ms	**9.34ms**
E.1.$\frac{1}{2}_2$	150	250	150	125	472	176ms	**58.8ms**
E.1.$\frac{1}{2}_3$	235	500	219	250	693	697ms	**205ms**
E.1.$\frac{1}{2}_4$	279	1000	290	500	874	2.75s	**842ms**
E.1.$\frac{1}{2}_5$	456	2000	454	1000	1309	19.6s	**4.04s**
E.1.$\frac{1}{2}_6$	500	4000	505	2000	1459	1min 19s	**15.2s**
E.1.$\frac{1}{2}_7$	562	8000	517	4000	1544	7min 13s	**1min**
E.1.1$_1$	63	100	64	100	208	107ms	**20.2ms**
E.1.1$_2$	157	250	167	250	508	664ms	**147ms**
E.1.1$_3$	225	500	223	500	659	2.68s	**523ms**
E.1.1$_4$	274	1000	323	1000	900	19.1s	**2.49s**
E.1.1$_5$	473	2000	478	2000	1421	1min 20s	**11.3s**
E.1.1$_6$	484	4000	537	4000	1420	7min 24s	**45.8s**
E.1.1$_7$	512	8000	539	8000	1545	41min 36s	**2min 50s**

Table 6.2: Parameters and characteristics for each experiment.

incrementally, adapting as new information arrives. Additionally, Goel and Chaudhary's algorithm leverages parallelization for faster computations, whereas AddPair operates on a single thread. These differences create unique strengths and weaknesses in scalability and flexibility. In spite of the mentioned differences, this section delves deeper the evaluation of Algorithm AddPair.

Before introducing the experiments, let us recall that the nature of AddPair is to update a lattice considering only one pair of one object and one attribute at a time. This means that, for a lattice construction algorithm, it would have to run several times for each object and its intent (i.e., g and g', assuming that all objects converge at some point in time). Thus, it would be more suitable to use it as a side routine when there is no way around having to update an already existing object. Nevertheless, for the record, in subsection 6.2.1, all real world experiments will be replicated comparing Algorithm AddPair with Algorithm IncrementalMerge since the latter is the most similar (i.e., and hence, comparable) to it.

Additionally, in subsection 6.2.2, a comparison will be made on Algorithm AddPair and the algorithm of Goel and Chaudhary by considering the amount of pairs, concepts, and amount of attributes of a known dataset, and a synthetic one. The comparison is made with the goal of having an empirical clue on how our algorithm behaves in comparison to what the authors of [49] reported using the advantages of distributed computing. But it is important to remember that the comparison is unfair in the context of all the already mentioned essential differences between the two algorithms.

6.2.1 Comparison with AddIntent

As depicted in Figure 6.13, Figure 6.14, Figure 6.15, Figure 6.16, the experimental runtime of Algorithm AddPair is always above that of Algorithm IncrementalMerge, with the exception of only one instance in which Algorithm AddPair outperformed the competition. However, the fact that it is so close in performance to the incremental algorithm that adds object by object with full knowledge of all its attributes is a *positive result* since it means that it could be used in its stead, since it has the advantage of *flexibility* (it can be used to update lattices with data coming from arbitrarily distributed formal contexts) discussed in 4. Additionally, Table 6.3 shows that the runtime difference between the algorithms shrinks as the number of objects ($|G|$) increases. Eventually, for the $1 : \frac{1}{3}$ ratio of objects in the two lattices, Algorithm AddPair outperforms the other algorithm (shown in bold in Table 6.3). This suggests that Algorithm AddPair becomes increasingly advantageous as the number of objects to be incorporated grows.

6.2.2 Comparison with Goel and Chaudhary's algorithm

The main challenge when comparing a single-threaded algorithm with a distributed one is to create the conditions in which they are comparable. However, in this case, we will limit the study to the simple comparison of *raw execution time*. In Goel and Chaudhary's work [49], they reported how much time their algorithm took to process contexts snapshots with different amounts of pairs (objects and attributes), formal concepts, distinct attributes, and distinct objects.

To run comparable instances, firstly, the amount of distinct objects, attributes, and formal concepts were counted in the experiments run in subsection 6.2.1. Secondly, the SPECT[32] dataset was converted into object-attribute pairs as to match the input format

[32]A dataset that describes diagnosing of cardiac Single Proton Emission Computed Tomography (SPECT)

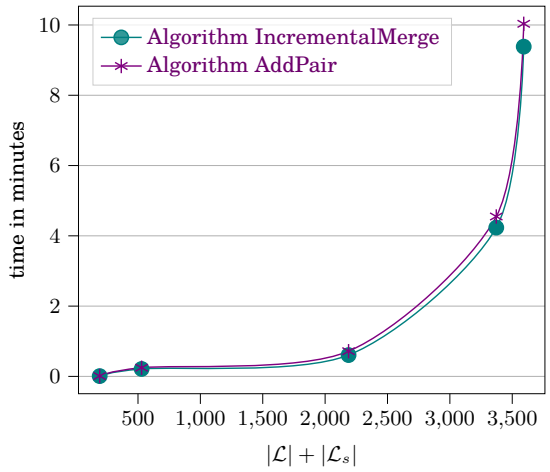

Figure 6.13: $|\mathcal{L}| + |\mathcal{L}_s|$ vs time in minutes with ratio $1 : \frac{1}{2}$

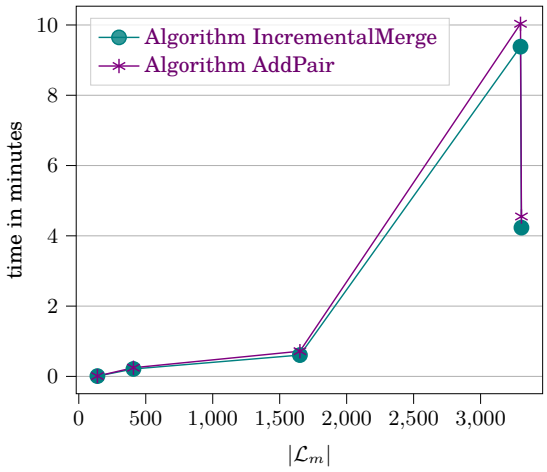

Figure 6.14: $|\mathcal{L}_m|$ vs time in minutes with ratio $1 : \frac{1}{2}$

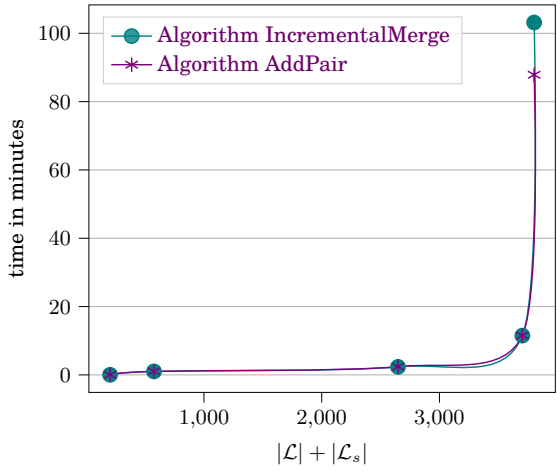

Figure 6.15: $|\mathcal{L}| + |\mathcal{L}_s|$ vs time in minutes with ratio $1 : 1$

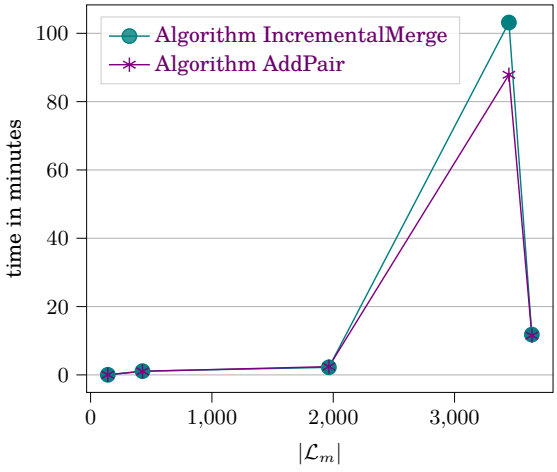

Figure 6.16: $|\mathcal{L}_m|$ vs time in minutes with ratio $1 : 1$

Experiments	$\lvert\mathcal{L}_1\rvert$	$\lvert G_1\rvert$	$\lvert\mathcal{L}_2\rvert$	$\lvert G_2\rvert$	$\lvert\mathcal{L}_m\rvert$	IncrementalMerge time	AddPair time
E.$1.\frac{1}{2}_1$	235	500	219	250	693	**511ms**	614ms
E.$1.\frac{1}{2}_2$	279	1000	290	500	874	**12.7s**	14.8ms
E.$1.\frac{1}{2}_3$	456	2000	454	1000	1309	**36.4s**	43s
E.$1.\frac{1}{2}_4$	500	4000	505	2000	1459	**4min 14s**	4min 33s
E.$1.\frac{1}{3}_1$	562	8000	517	4000	1544	**9min 23s**	10min 2s
E.1.1_1	225	500	223	500	659	**1.81s**	1.83s
E.1.1_2	274	1000	323	1000	900	**1min 1s**	1min 3s
E.1.1_3	473	2000	478	2000	1421	**2min 21s**	2min 25s
E.1.1_4	484	4000	537	4000	1420	**11min 30s**	11min 32s
E.$1.\frac{3}{4}_1$	512	8000	539	8000	1545	1h 43min 10s	**1h 27min 52s**

Table 6.3: Parameters and characteristics for each experiment regarding Algorithm AddPair and Algorithm IncrementalMerge.

Experiments	Number of pairs added to $\mathcal{L}_1$
E.*1.*$\frac{1}{2}_1$	2140
E.*1.*$\frac{1}{2}_2$	7089
E.*1.*$\frac{1}{2}_3$	6163
E.*1.*$\frac{1}{2}_4$	19038
E.*1.*$\frac{1}{3}_1$	30184
E.*1.1*$_1$	3575
E.*1.1*$_2$	14187
E.*1.1*$_3$	12138
E.*1.1*$_4$	27275
E.*1.*$\frac{3}{4}_1$	76528

Table 6.4: Number of pairs for each experiment regarding Algorithm AddPair and Algorithm IncrementalMerge.

expected by Algorithm AddPair, and then run on an 11th Gen Intel® Core™ i7-11850H @ 2.50GHz × 16 with 32 GB RAM computer. On the other hand, a synthetic dataset was used to replicate the characteristics of the Mushroom dataset[33] used in the work of Goel and Chaudhary. All these experiments can be found in a Jupyter Notebook in the following url: https://gitlab.com/cps-phd-leutwyler-nicolas/thesis_documents/-/blob/main/experiments/add_pair_experiments.ipynb.

As depicted in Table 6.4, we can observe that instances E.*1.*$\frac{1}{2}_1$, and E.*1.*$\frac{1}{2}_2$, and E.*1.1*$_1$ have under 5 thousand object-attribute pairs, which is close to the amount of pairs in the SPECT dataset (i.e., 2042). If we compare the runtimes of those instances with the ones reported in [49], we observe that Algorithm AddPair performs faster even compared with the distributed version using 4 workers (i.e., computational units that run in parallel). This might happen because of the difference in the amount of formal concepts in each instance, which SPECT have more (21550 vs fewer than 1000).

Moreover, when running Algorithm AddPair with all pairs in the SPECT dataset, the results are vastly superior to the ones reported in [49] with 4 workers. In particular, the running time for constructing the entire concept lattice is **7.98s**, against more than 1m 30s for Goel and Chaudhary's algorithm.

Finally, Algorithm AddPair has been run with a synthetic dataset that was created to be similar in size and density to the Mushroom dataset. In this scenario, the algorithm **could not finish** before the timeout of 1h, compared to the approximately 10 minutes that the algorithm of Goel and Chaudhary took. This shows that Algorithm AddPair is not suitable for dense contexts such as that of the Mushroom one.

images https://archive.ics.uci.edu/dataset/95/spect+heart

[33] A dataset that includes descriptions of hypothetical samples corresponding to 23 species of gilled mushrooms in the Agaricus and Lepiota Family https://archive.ics.uci.edu/dataset/73/mushroom.

6.3 Distributed batch merge using Apache Spark

6.3.1 Experiments on distributed clusters

In order to evaluate the distributed algorithms for merging lattices presented in chapter 3, subsection 3.3.3, and to be consistent with the previous evaluation methods, the real-world datasets used in section 6.1 were used as well for these experiments. The algorithms ParallelMergeOnlyIntents, ParallelMergeConcepts, and ParallelMergeConceptsWithUnion were run in EMR clusters[34] with 2, 3, and 4 workers (4 vCore, 16GB RAM each). To contrast, they were compared to the algorithm of Goel and Chaudhary (run on the same cluster), since it is the only one that is able to process from arbitrarily distributed formal contexts. Since that algorithm falls into the batch category, it cannot leverage $\mathcal{L}$ and $\mathcal{L}_s$ having been already computed, hence, the algorithm consists of considering the union of the two sets of object-attribute pairs to be merged. Moreover, as raw times are the key feature being considered in these experiments, the runtimes yield by Algorithm BatchMerge in each of the experiments is included in the final graphs for reference. The Jupyter Notebooks for 2, 3 and 4 workers can be consulted in https://gitlab.com/cps-phd-leutwyler-nicolas/thesis_documents/-/blob/main/experiments/.

Example. 6.1: How real world experiments were carried out

Considering two lattices $\mathcal{L}$ and $\mathcal{L}_s$, the ParallelMergeOnlyIntents, ParallelMergeConcepts, and ParallelMergeConceptsWithUnion algorithms were run on $\mathcal{L}$ and $\mathcal{L}_s$, while Goel and Chaudhary's algorithm was run on the underlying formal context data stream of pairs of the merged lattice $\mathcal{L}_m$. Afterward, for the record, the results were compared with the runtimes of Algorithm BatchMerge.

Regarding the instances with an object ratio of $1 : \frac{1}{2}$, all runtimes are depicted in Figure 6.17, Figure 6.18, and Figure 6.19. As expected, algorithms run faster the more workers are available. Additionally, with this ratio, Goel and Chaudhary's algorithm is the one that took the most time with all 2, 3, and 4 workers, whereas the ParallelMergeOnlyIntents algorithm is the one that took the least. Interestingly, the gains in performance and scalability in contrast to the single-threaded BatchMerge algorithm, start showing after $|\mathcal{L}|+|\mathcal{L}_s|$ grows pass 2300 concepts.

Runtimes in relation to the instances with an object ratio of $1 : 1$, are depicted in Figure 6.20, Figure 6.21, and Figure 6.22. In the experiments with two workers, it is particularly notorious that the Algorithm ParallelMergeConcepts performs poorly, since its runtime was above the single-threaded runs in all instances. However, after increasing the workers from 2 to 3 (and also to 4), the performance improved greatly, showing off the advantages of horizontal scaling.

6.3.2 Remarks about the results

All the proposed distributed algorithms in this thesis showed an increase in raw performance considering that the two lattices to be merged were already computed. In particular, the key conclusions about the algorithms are, firstly, that ParallelMergeOnlyIntents ran faster than the competition in all instances. Secondly, the optimization on the Algorithm

[34] Amazon Web Services clusters https://aws.amazon.com/emr/

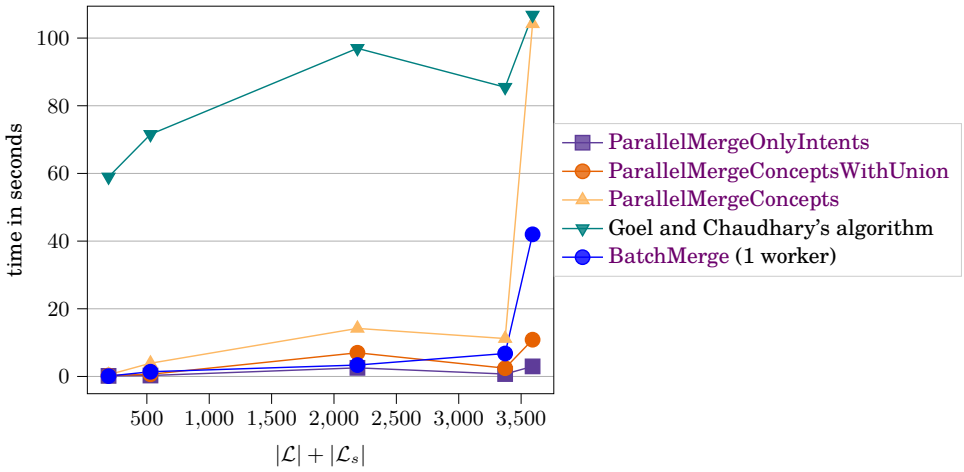

Figure 6.17: $|\mathcal{L}_m|$ vs time in seconds with ratio $1 : \frac{1}{2}$ with 2 workers.

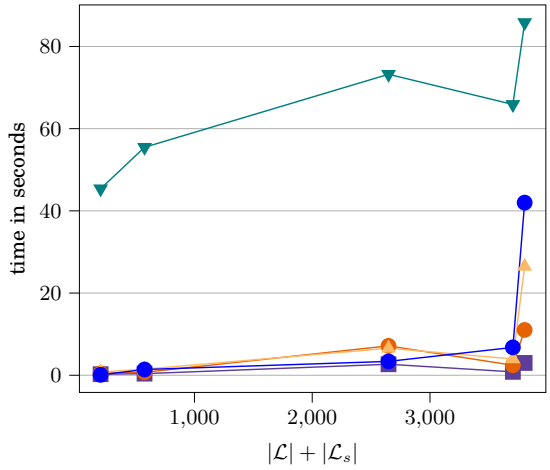

Figure 6.18: $|\mathcal{L}_m|$ vs time in seconds with ratio $1 : \frac{1}{2}$ with 3 workers.

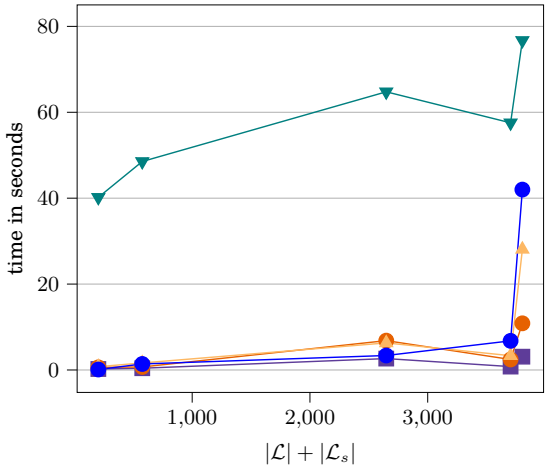

Figure 6.19: $|\mathcal{L}_m|$ vs time in seconds with ratio $1 : \frac{1}{2}$ with 4 workers.

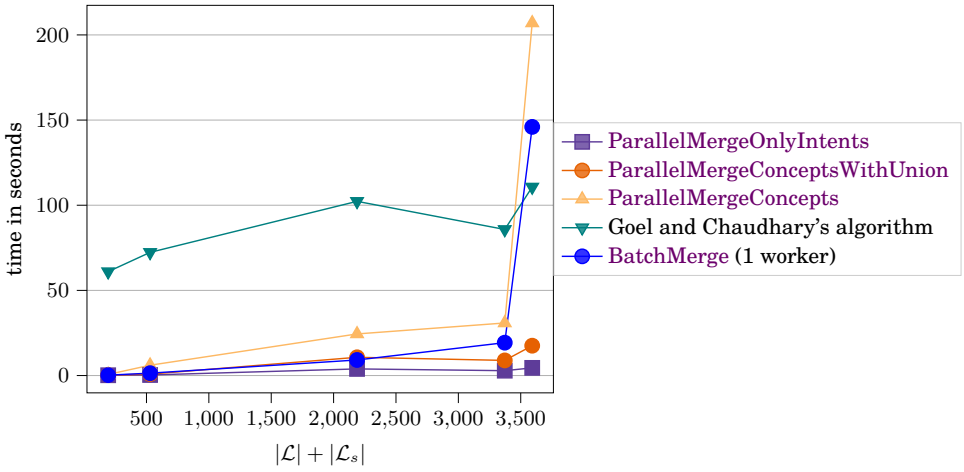

Figure 6.20: $|\mathcal{L}_m|$ vs time in seconds with ratio $1:1$ with 2 workers.

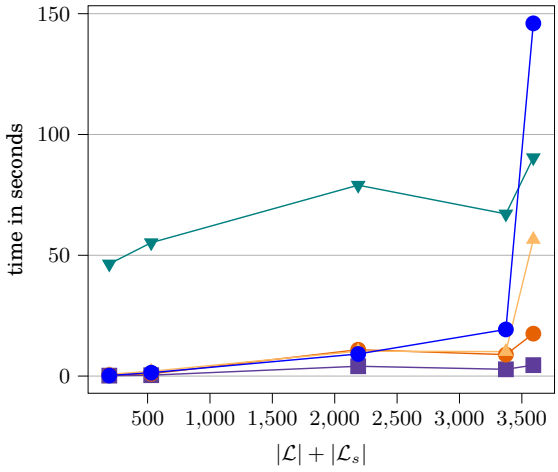

Figure 6.21: $|\mathcal{L}_m|$ vs time in seconds with ratio $1:1$ with 3 workers.

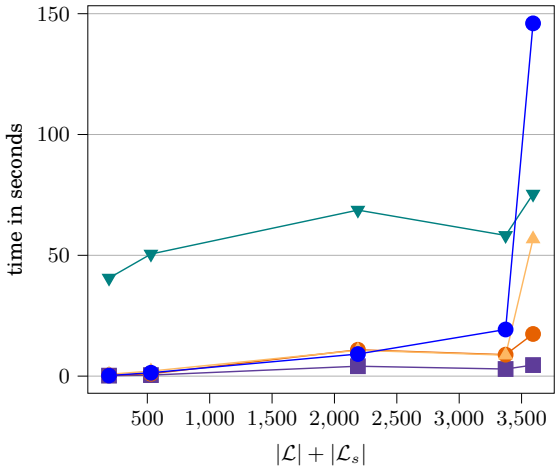

Figure 6.22: $|\mathcal{L}_m|$ vs time in seconds with ratio $1:1$ with 4 workers.

ParallelMergeConceptsWithUnion where only pairs of concepts whose intent intersection is non-empty were considered in the Cartesian product worked as intended, since it was the algorithm computing the sets of concepts (including extents) that ran the fastest in the largest instances.

Following, increasing the workers count helped the algorithms run faster, however, there seemed to be a plateau when scaling from 3 to 4 workers, where the increase was not as pronounced. Regarding this, there are several variables that could affect the performance of distributed algorithms, being network latency among the most relevant.

6.4 Conclusions

Several experiments were carried out in this chapter to provide an empirical understanding of the behavior of the presented algorithms. Additionally, comparisons with existing algorithms and methodologies were also considered.

In section 6.1 a comparison between the algorithms BatchMerge and IncrementalMerge has been done using real world datasets coming from the use case introduced in subsection 5.3.1, and a synthetic dataset with the goal of covering a wide range of characteristics of the formal contexts. The results show significant gains in performance in all experiments considering their characteristics of being sparse contexts. Moreover, results suggest that Algorithm BatchMerge scales better than Algorithm IncrementalMerge the sparser the contexts get.

Following, section 6.2 provides experiments on performance regarding the Algorithm AddPair in comparison to the Algorithm IncrementalMerge and the algorithm of Goel and Chaudhary. The comparison was carried out using the same real world datasets as the ones presented in subsection 5.3.1 to benchmark the construction of a concept lattice from scratch using the algorithm presented in chapter 4. The results show that Algorithm AddPair runtimes were almost the same as those of Algorithm IncrementalMerge, whereas it surpassed the reported runtimes of the Goel and Chaudhary algorithm considering similar datasets.

Finally, in section 6.3 distributed versions of the real world datasets were used to compare the proposed distributed merge algorithms to the single-threaded one and the Goel and Chaudhary's algorithm. The results evidenced performance superiority in almost all instances regarding the distributed algorithms, whereas the horizontal scaling showed to start being worth the network overhead (i.e., running on a cluster can be more expensive than running on a single thread in a single computer for small instances) after a lattice size $|\mathcal{L}_m| = 1500$.

7

Conclusions & Future Work

Contents

7.1 Summary 113
7.2 Perspectives and future directions 115
7.3 Publication List 116

7.1 Summary

In this thesis, the application of Formal Concept Analysis (FCA) in distributed and dynamic data environments has been addressed, considering certain emerging challenges. This research addresses the need for efficient data mining techniques in Industry 4.0 and, more particularly, the Internet of Things (IoT), where data is vast and continuous.

Firstly, in chapter 2, a systematic literature review has been carried out in order to provide evidence on the scientific gaps concerning the knowledge extraction from heterogeneous data sources in distributed environments. In that study, the results of the data extraction have been presented, including the methods utilized in MRDM, CA, and DA, the addressed problematics from the point of view of the introduced process lifecycle, the domains, the input and output formats and the relationships between each other, and finally the evaluated method characteristics. Moreover, a set of association rules regarding the categorization carried out with the set of articles has been presented by considering the input and output treatment in the process lifecycle, and the ML and Semantic Web domain categories. In addition, three scientific gaps have been identified, from which one of them has been addressed in this thesis, namely, the need for an FCA-based method that accounts for both the infinite characteristic of data streams in dynamic distributed architectures, and for the loss of knowledge when using sliding windows.

Secondly, in chapter 3, a scalable, and flexible method to help prevent the loss of significant concepts during concept lattice construction, has been proposed. The method consists in the utilization of lattice snapshots as a backup of interesting concepts, and their later usage in a *merge* step that has to be *fast*, leveraging the time that has priorly been spent in the computation of the snapshots. In the chapter, a batch algorithm (i.e., Algorithm BatchMerge) to merge lattices is presented and proven correct. Then, a single-threaded implementation using a *trie* data structure to store intents is explained and contrasted with an incremental merge algorithm (i.e., Algorithm IncrementalMerge). The results re-

garding their computational (time) complexity done in the chapter suggested that Algorithm BatchMerge should perform better than Algorithm IncrementalMerge for highly sparse contexts.

Additionally, chapter 3 presented three distributed algorithms following the idea of Algorithm BatchMerge. On the one hand, an algorithm to merge only intents (i.e., Algorithm ParallelMergeOnlyIntents), which can be useful for applications seeking to use association rules only. On the other hand, two algorithms to compute the whole set of formal concepts (i.e., Algorithm ParallelMergeConcepts and Algorithm ParallelMergeConceptsWithUnion), where, they offer more scalability than their single-threaded version BatchMerge, because they leverage parallelism by using the MapReduce computation model.

Third, in chapter 4, an incremental algorithm for mining formal contexts coming from distributed sources has been presented. In particular, since most algorithms require the context to be partitioned in a "particular" way (e.g., every worker has to know all attributes, thus, only objects can be partitioned), or require the entire context to be replicated in all nodes of the architecture, there is *not enough flexibility* for practitioners to actually mine from contexts that are *arbitrarily distributed*. Considering that there is particularly one distributed algorithm that addresses this problem in a *batch* fashion, the thesis proposed an *incremental* solution that uses two already existing algorithms, AddIntent and DeleteInstance, and analyzed the complexity compared to the maximum amount of look-ups for modified concepts. The results suggest that the proposed algorithm should outperform the *batch* version because it does not suffer from exponential growth in the attributes, contrary to its competition.

Fourthly, in chapter 5, a number of practical use cases have been put in place to demonstrate the real-world benefits of the proposed methods. Most notably, a recommendation system for the e-commerce site of the French Ski School has been implemented with the idea of using FCA as a means to compute and then recommend based on frequent itemsets, an idea that has been proven to work already multiple times for recommending items that "usually go together". In this case, the novelty lies in the intermediate "merge" step that allows recovering certain (and maybe desired) concepts. For instance, if the ESF wants to put a specific type of lesson on sale, it can use a snapshot containing the lattice corresponding to a timespan where a similar sale has been active. This can be also forced by using business logic, however, it is not as adaptive as the data-based method, that will have knowledge actually related to each of the resorts and the timestamps the practitioner choose.

Finally, in chapter 6, the presented algorithms were subject of a number of experiments, using both real-world and synthetic datasets, to evaluate their empirical performance regarding the growth of the input. All experiments were conceived with the goal of simulating a situation in which there are two lattices to be merged, with different object-object ratio (i.e., the difference between the amount of objects in the two lattices). In all experiments, the empirical performance of Algorithm BatchMerge has shown promising results compared to that of Algorithm IncrementalMerge, when dealing with *sparse contexts*. However, the results showed as well that Algorithm BatchMerge performs poorly compared to the usage of the incremental addition of "object by object" solution the more dense the context get.

Regarding the Algorithm AddPair, it showed to perform extremely similar to Algorithm AddIntent even though, intuitively, it should perform worse since it would be called $|g'|$ times for each g instead of only once. Thus, as Algorithm AddPair is more flexible than its counterpart (i.e., it can be used to construct lattices from arbitrarily distributed contexts unlike Algorithm AddIntent), its usage in sparse contexts would be preferable.

Lastly, distributed versions of the real world datasets were used to compare the proposed

distributed merge algorithms (i.e., ParallelMergeOnlyIntents, ParallelMergeConcepts, and ParallelMergeConceptsWithUnion) to the single-threaded one (BatchMerge) and Goel and Chaudhary's algorithm, presented in [49]. Experiments showed that the distributed algorithms performed significantly better in nearly all cases. However, using a cluster (horizontal scaling) only became advantageous when dealing with larger problems. For smaller problems (lattice size below $|\mathcal{L}_m| = 1500$), the extra network traffic generated by the cluster outweighed the benefits, making it more efficient to run on a single computer.

7.2 Perspectives and future directions

Several fronts have been left open to explore after the mentioned contributions. Regarding the theoretical aspect of the work, algorithms for merging filtered snapshot lattices (e.g., shrank by a support threshold) could further improve the performance, thus doing so would be a desirable and direct continuation of the work. Moreover, the analysis of the complexity of the "minimal" incremental algorithm for lattice construction coming from arbitrarily distributed contexts may be lower if considering certain properties of the *cover* of a concept, which might not need $\mathcal{O}(|G|^2|M|)$ for each concept in $\mathcal{L}$, leading to an improvement in its worst case time complexity bound.

Regarding the MapReduce approach to compute the merge of lattices, a *distributed* algorithm to compute as well the line diagram is needed for the applications that might make use of it. To do so, a seemingly promising direction is to use the distributed graph implementation GraphX[35], that is an Apache Spark implementation, and extend the algorithms that were proposed in this thesis. Moreover, the covers (i.e., upper neighbors of a concept in the line diagram) can be found by using the function FOUND-UPPER-COVERS-BIS, presented in [129]. However, the challenge lies in finding a proper way to transform that function into a distributed one that actually leverages parallelism.

In regard to the practical aspects of the thesis, an in-depth benchmark is needed to perform on the parallel version of the merging algorithm in order to understand what are the actual gains of the parallelization using Apache Spark. Particularly, there are several aspects of the framework itself that makes a theoretical analysis much more complex. For instance, certain map functions require a *shuffle* in the RDD, which means distributing data across cluster workers to enable parallel processing. This process typically occurs when data is unevenly distributed, needs to be organized in a particular manner for processing, or when a single node lacks sufficient memory to store all the necessary data. Sometimes it is worth using shuffle-needing map operations because of the time gains afterwards, but sometimes it is not. Thus, understanding which is preferable in the specific case of merging lattices would require testing and comparing different approaches.

Regarding use cases in distributed architectures, the application of the method in an actual case of IoT would be desirable, since the ones provided in this work were adapted from the e-commerce site to be used and thought as ever-growing data-streams, but there are certain characteristics that might have not been considered in such an adaptation. Additionally, response-times can change between real time learning in an IoT setting than in an e-commerce one by orders of magnitude, hence, the setting might require algorithms and methods that perform even faster, conserving the flexibility provided by what we discussed in the work.

[35] https://spark.apache.org/graphx/

In addition, IoT systems are often related to heterogeneous data sources. Although plain FCA can be used to mine from them, there are more specialized tools to deal with the situation in which those heterogeneous sources are also interrelated. In particular, Relational Concept Analysis introduced in [112], is a widely known extension of FCA to deal with the multi-relational data mining. Since the core algorithm iteratively computes concept lattices of each of the contexts in the dataset[36], its scalability could also see benefits in applying *merge* techniques similar to the ones described in this thesis.

Another area that is left open to research is the *selection* of snapshots. In other words, *when* is it useful to store a lattice snapshot. Additionally, other questions that arise related to the fine-tuning of the method to recover potentially useful concepts are (1) *how much* does it affect to the result including the merge step vs not including it, (2) what is the *ideal lattice size ratio* to consider when merging taking into account the previous question. In particular, our recommendation regarding how to address these questions is to look into contexts where *anomalies* are as important (or even more) as recurrent patterns (e.g., medicine). We believe that contexts like that one can provide valuable insights to help to answer these questions.

7.3 Publication List

The work carried out during this thesis led to the following publications:

- [83] - N. Leutwyler, M. Lezoche, D. Torres, and H. Panetto. Multi-relational and Concept Analysis based Knowledge extraction in the Industry 4.0: A systematic mapping. *IFAC-PapersOnLine*, 56(2):7318–7329, Jan. 2023. ISSN 2405-8963. doi: 10.1016/j.ifacol.2023.10.345. URL https://www.sciencedirect.com/science/article/pii/S2405896323007127

- [84] - N. Leutwyler, M. Lezoche, D. Torres, and H. Panetto. Towards a Flexible and Scalable Data Stream Algorithm in FCA. In M. Ojeda-Aciego, K. Sauerwald, and R. Jäschke, editors, *Graph-Based Representation and Reasoning*, Lecture Notes in Computer Science, pages 104–117, Cham, 2023. Springer Nature Switzerland. ISBN 978-3-031-40960-8. doi: 10.1007/978-3-031-40960-8_9

- [85] - N. Leutwyler, M. Lezoche, C. Franciosi, H. Panetto, L. Teste, and D. Torres. Methods for concept analysis and multi-relational data mining: a systematic literature review. *Knowledge and Information Systems*, May 2024. ISSN 0219-3116. doi: 10.1007/s10115-024-02139-x. URL https://doi.org/10.1007/s10115-024-02139-x

Other secondary contributions, were also done during the same period:

- [80] - N. Leutwyler, M. Lezoche, H. Panetto, and D. Torres. Extending RCA algorithm to consider ternary relations. In *ICIST 2022 Proceedings*, pages 203–209. Information Society of Serbia - ISOS, 2022. URL http://www.eventiotic.com/eventiotic/library/paper/714

- [82] - N. Leutwyler, M. Lezoche, H. Panetto, and D. Torres. Generic software for benchmarking Formal Concept Analysis: Orange3 integration. *Electronic Journal of SADIO (EJS)*, 22(3):28–40, Aug. 2023. ISSN 1514-6774. URL https://publicaciones.sadio.org.ar/index.php/EJS/article/view/597. Number: 3

[36] RCA input consists of several Formal Contexts interrelated with tables indicating relations between objects in the different contexts

Résumé étendu en français

Introduction

Tout au long de l'histoire moderne, les données ont été collectées et utilisées pour faire progresser la société. Cependant, la génération de données est aujourd'hui sans précédent. Selon une étude réalisée par la Software Alliance[37] [16], en 2016 déjà, environ 90% des données mondiales avaient été générées entre 2014 et 2016. De plus, on observe une réduction significative des coûts de stockage, attribuée à l'émergence de nouvelles technologies, sans doute supérieures, adaptées à cet usage. Plus encore, le taux de croissance de la génération de données devrait continuer à augmenter.

En conséquence, face à cette croissance sans précédent, une multitude de disciplines ont émergé pour aider les chercheurs et les praticiens à collecter des données pertinentes. Parmi celles-ci, on retrouve : l'exploration de données (Data Mining, DM) : processus de découverte de connaissances ou de schémas à partir de volumes massifs de données [54] ; la recherche d'information (Information Retrieval, IR) : consiste à trouver du matériel (généralement des documents) de nature non structurée (généralement du texte) qui répond à un besoin d'information au sein de vastes collections (généralement stockées sur des ordinateurs) [91] ; l'extraction de connaissances (Knowledge Extraction, KE) : création de connaissances à partir de sources structurées (par exemple, bases de données relationnelles, langage de balisage extensible (XML)) et non structurées (par exemple, texte, documents, images) [127].

Cette thèse porte sur l'application de l'Analyse formelle de concepts (AFC) aux environnements de données distribués et dynamiques, en tenant compte de certains défis émergents. Cette recherche répond au besoin de techniques de fouille de données efficaces dans l'Industrie 4.0 et, plus particulièrement, dans l'Internet des objets (IoT), où les données sont vastes et continues.

Cadre et objectifs

Le fondement théorique de cette thèse provient du projet européen nommé Pôle d'innovation numérique pour les systèmes cyber-physiques (DIH4CPS). Dès sa conception, l'objectif de ce projet était d'aider les entreprises affiliées dans la numérisation de leurs systèmes existants en appliquant l'intelligence artificielle et les techniques de fouille de données, en se concentrant sur les systèmes cyber-physiques et les systèmes embarqués. Plus précisément, **cette thèse vise à combler le vide existant dans l'utilisation de l'extraction de connaissances conceptuelles à partir de données de bas niveau lorsqu'elles proviennent d'architectures distribuées, comme celles dans lesquelles les systèmes cyber-physiques sont souvent implémentés**.

[37] https://www.bsa.org/about-bsa

Un certain nombre de méthodes de fouille de données bien connues existent pour traiter différentes problématiques liées à l'extraction de connaissances à partir de données de bas niveau provenant d'AD. Par exemple, les K-moyennes [69], les K-plus proches voisins [99], les arbres de régression [63], etc. Cependant, la plupart de ces méthodes dépendent fortement de la structure des données à traiter et, dans certains cas, de techniques de modélisation qui créent un « décalage d'impédance » entre les données sources et les données qui sont réellement utilisées comme entrées. Cela crée toutes sortes de problèmes dans des domaines tels que l'« explicabilité », et les rend moins intéressantes pour les praticiens, car y parvenir nécessiterait un travail supplémentaire non négligeable. C'est pourquoi une méthode comme l'analyse formelle de concepts [137], dont la sortie est étroitement liée à la façon dont les humains comprendraient les « données conceptuelles », est très précieuse pour cette tâche.

L'analyse formelle de concepts s'est avérée efficace pour l'extraction de connaissances dans plusieurs domaines liés au minage à partir d'AD et en utilisant des AD. De nombreuses approches ont été proposées pour résoudre les problèmes d'évolutivité dans ces domaines [141, 28, 132, 49]. Cependant, ces méthodes introduisent de nouveaux problèmes. En particulier, en ce qui concerne le minage de flux de données infini, le principal problème est qu'aucun algorithme ne peut traiter des données infinies, c'est-à-dire que la lecture seule de l'entrée prendrait un temps infini. Ce problème est généralement résolu par l'utilisation conjointe d'algorithmes incrémentiels et de fenêtres glissantes. À son tour, ce paradigme définit une façon très spécifique (même si elle diffère d'une approche à l'autre) de ne considérer qu'une partie du flux de données, par exemple, uniquement les k événements les plus récents, ou uniquement les concepts formels les plus récemment mis à jour. Par conséquent, on introduit **la possibilité de perdre potentiellement des concepts importants ou pertinents**.

De plus, les algorithmes incrémentiels en général, à l'exception de quelques cas comme celui de DeleteInstance [149], sont définis pour le cas où des objets sont ajoutés au contexte [132, 129]. Cependant, étant donné que l'une des hypothèses des algorithmes incrémentiels est que toutes les informations (c'est-à-dire les objets dans ce cas) ne sont pas disponibles à tout moment, il est également valable de supposer que même tous les attributs d'un objet ne sont pas valides en permanence. Par conséquent, les algorithmes incrémentiels ne sont pas assez flexibles pour être utilisés dans ces scénarios. En particulier, il n'existe qu'un seul algorithme capable de traiter efficacement des contextes arbitrairement distribués [49]. Cependant, cet algorithme appartient à la catégorie des algorithmes par lots, d'où la nécessité d'une solution incrémentielle pour les cas où le contexte est non seulement distribué arbitrairement, mais également dynamique.

Enfin, le problème de construction de réticulés est connu pour être P-complet[38]. Cela signifie, entre autres, que l'on pense fortement que le problème ne peut pas être efficacement parallélisé. Néanmoins, étant donné les contraintes de temps imposées par l'environnement de l'IdO (IoT), la parallélisation contribue toujours à l'évolutivité, même si elle n'affecte le calcul que par une constante.

Dans ce contexte, les objectifs de la thèse sont les suivants :

(G.1) Proposer une méthode évolutive et flexible pour remédier à la perte de concepts potentiellement importants lors de la construction d'un réticulé de concepts à l'aide de fenêtres glissantes.

[38] En fait, même la détermination du nombre de concepts formels d'un certain contexte formel appartient à cette classe [75]

(G.2) Proposer une méthode incrémentielle pour la construction de réticulés de concepts dans le cas où tous les attributs ne sont pas connus pour un objet donné à chaque mise à jour (c'est-à-dire, le contexte formel est arbitrairement distribué et dynamique).

(G.3) Obtenir une évolutivité horizontale en proposant une méthode qui exploite le parallélisme *à l'aide* des AD.

(G.4) Démontrer son utilité potentielle dans un scénario pratique.

Résultats

Premièrement, dans le chapitre chapter 2, une revue systématique de la littérature a été réalisée afin de fournir des preuves sur les lacunes scientifiques concernant l'extraction de connaissances à partir de sources hétérogènes dans des environnements distribués. Cette étude présente les résultats de l'extraction de données, y compris les méthodes utilisées en MRDM, CA et DA, les problématiques abordées du point de vue du cycle de vie du processus introduit, les domaines, les formats d'entrée et de sortie et leurs relations mutuelles, et enfin les caractéristiques des méthodes évaluées. De plus, un ensemble de règles d'association concernant la catégorisation effectuée avec l'ensemble des articles a été présenté en considérant le traitement des entrées et des sorties dans le cycle de vie du processus, ainsi que les catégories des domaines ML et Web Sémantique. En outre, trois lacunes scientifiques ont été identifiées, dont l'une a été abordée dans cette thèse, à savoir le besoin d'une méthode basée sur la FCA qui prend en compte à la fois la caractéristique infinie des flux de données dans les architectures distribuées dynamiques, et la perte de connaissances lors de l'utilisation des fenêtres glissantes.

Deuxièmement, dans chapter 3, une méthode scalable et flexible pour aider à prévenir la perte de concepts significatifs lors de la construction de réticulés conceptuels a été proposée. La méthode consiste en l'utilisation de snapshots de réticulés comme sauvegarde des concepts intéressants, et leur utilisation ultérieure dans une étape de *fusion* qui doit être *rapide*, tirant parti du temps préalablement passé dans le calcul des snapshots. Le chapitre présente un algorithme par lots (c'est-à-dire, Algorithm BatchMerge) pour fusionner des réticulés, qui est présenté et prouvé correct. Ensuite, une implémentation à un seul thread utilisant une structure de données *trie* pour stocker les intentions est expliquée et comparée à un algorithme de fusion incrémental (c'est-à-dire, Algorithm IncrementalMerge). Les résultats concernant leur complexité (en temps) calculée dans le chapitre suggèrent que Algorithm BatchMerge devrait performer mieux que Algorithm IncrementalMerge pour des contextes très épars.

De plus, chapter 3 a présenté trois algorithmes distribués suivant l'idée de Algorithm BatchMerge. D'une part, un algorithme pour fusionner uniquement les intentions (c'est-à-dire, Algorithm ParallelMergeOnlyIntents), qui peut être utile pour les applications cherchant uniquement à utiliser des règles d'association. D'autre part, deux algorithmes pour calculer l'ensemble complet des concepts formels (c'est-à-dire, Algorithm ParallelMergeConcepts et Algorithm ParallelMergeConceptsWithUnion), offrant une plus grande scalabilité que leur version à un seul thread BatchMerge, car ils exploitent le parallélisme en utilisant le modèle de calcul MapReduce.

Troisièmement, dans chapter 4, un algorithme incrémental pour l'exploration de contextes formels provenant de sources distribuées a été présenté. En particulier, puisque la plupart

des algorithmes nécessitent que le contexte soit partitionné de manière “particulière” (par exemple, chaque travailleur doit connaître tous les attributs, donc seuls les objets peuvent être partitionnés), ou nécessitent que le contexte entier soit répliqué dans tous les nœuds de l'architecture, il y a *un manque de flexibilité* pour les praticiens afin de réellement explorer des contextes qui sont *distribués arbitrairement*. Étant donné qu'il existe un algorithme distribué qui aborde ce problème de manière *batch*, la thèse propose une solution *incrémentale* qui utilise deux algorithmes déjà existants, AddIntent et DeleteInstance, et analyse la complexité par rapport au nombre maximum de recherches pour les concepts modifiés. Les résultats suggèrent que l'algorithme proposé devrait surpasser la version *batch* car il ne souffre pas de la croissance exponentielle des attributs, contrairement à ses concurrents.

Quatrièmement, dans chapter 5, plusieurs cas d'utilisation pratiques ont été mis en place pour démontrer les avantages réels des méthodes proposées. Plus précisément, un système de recommandation pour le site de commerce électronique de l'École de Ski Française a été mis en œuvre avec l'idée d'utiliser la FCA comme moyen de calculer et ensuite recommander sur la base de fréquents ensembles d'articles, une idée qui a déjà été prouvée pour recommander des articles qui “vont généralement ensemble”. Dans ce cas, la nouveauté réside dans l'étape intermédiaire de “fusion” qui permet de récupérer certains (et peut-être désirés) concepts. Par exemple, si l'ESF souhaite mettre en vente un type spécifique de leçon, elle peut utiliser un snapshot contenant le réticulé correspondant à une période où une vente similaire a été active. Cela peut également être forcé en utilisant une logique commerciale, cependant, ce n'est pas aussi adaptatif que la méthode basée sur les données, qui aura des connaissances réellement liées à chacun des stations et les horodatages choisis par le praticien.

Enfin, dans chapter 6, les algorithmes présentés ont été soumis à un certain nombre d'expériences, utilisant à la fois des ensembles de données réels et synthétiques, pour évaluer leur performance empirique en fonction de la croissance de l'entrée. Tous les expériences ont été conçues dans le but de simuler une situation où il y a deux réticulés à fusionner, avec des ratios objet-objet différents (c'est-à-dire, la différence entre le nombre d'objets dans les deux réticulés). Dans toutes les expériences, la performance empirique de Algorithm BatchMerge a montré des résultats prometteurs par rapport à celle de Algorithm IncrementalMerge, lorsqu'il s'agit de *contextes épars*. Cependant, les résultats ont également montré que Algorithm BatchMerge performe mal comparé à l'utilisation de la solution incrémentale “objet par objet” plus le contexte devient dense.

Concernant Algorithm AddPair, il a montré une performance extrêmement similaire à celle de Algorithm AddIntent bien que, intuitivement, il devrait performer moins bien puisqu'il serait appelé $|g'|$ fois pour chaque g au lieu de seulement une fois. Ainsi, comme Algorithm AddPair est plus flexible que son homologue (c'est-à-dire, il peut être utilisé pour construire des réticulés à partir de contextes distribués arbitrairement contrairement à Algorithm AddIntent), son utilisation dans des contextes épars serait préférable.

Enfin, des versions distribuées des ensembles de données réels ont été utilisées pour comparer les algorithmes de fusion distribués proposés (c'est-à-dire, ParallelMergeOnlyIntents, ParallelMergeConcepts, et ParallelMergeConceptsWithUnion) avec l'algorithme à un seul thread (BatchMerge) et l'algorithme de Goel et Chaudhary. Les expériences ont montré que les algorithmes distribués ont performé significativement mieux dans presque tous les cas. Cependant, l'utilisation d'un cluster (scalabilité horizontale) ne devient avantageuse que lorsqu'on traite de problèmes plus importants. Pour les problèmes plus petits (taille du réticulé inférieure à $|\mathcal{L}_m| = 1500$), le trafic réseau supplémentaire généré par le cluster a surpassé les bénéfices, rendant l'exécution sur un seul ordinateur plus efficace.

Perspectives et directions futures

Plusieurs axes restent ouverts pour exploration après les contributions mentionnées. En ce qui concerne l'aspect théorique du travail, les algorithmes pour fusionner les réticulés instantanés filtrés (par exemple, réduits par un seuil de support) pourraient améliorer encore les performances ; faire cela serait donc une continuation souhaitable et directe du travail. De plus, l'analyse de la complexité de l'algorithme "minimal" incrémental pour la construction de réticulés provenant de contextes distribués de manière arbitraire pourrait être inférieure en tenant compte de certaines propriétés de la *couverte* d'un concept, ce qui pourrait ne pas nécessiter $\mathcal{O}(|G|^2|M|)$ pour chaque concept dans $\mathcal{L}$, conduisant à une amélioration de la limite de complexité temporelle dans le pire des cas.

Concernant l'approche MapReduce pour calculer la fusion des réticulés, un algorithme *distribué* pour calculer également le diagramme de lignes est nécessaire pour les applications qui pourraient l'utiliser. Pour ce faire, une direction apparemment prometteuse est d'utiliser l'implémentation distribuée de graphes GraphX[39], qui est une implémentation d'Apache Spark, et d'étendre les algorithmes proposés dans cette thèse. De plus, les couverts (c'est-à-dire, les voisins supérieurs d'un concept dans le diagramme de lignes) peuvent être trouvés en utilisant la fonction FOUND-UPPER-COVERS-BIS, présentée dans [129]. Cependant, le défi réside dans la recherche d'un moyen approprié pour transformer cette fonction en une fonction distribuée qui exploite réellement le parallélisme.

En ce qui concerne les aspects pratiques de la thèse, un benchmark approfondi est nécessaire pour comprendre quels sont les gains réels de la parallélisation en utilisant Apache Spark. En particulier, il existe plusieurs aspects du cadre lui-même qui rendent l'analyse théorique beaucoup plus complexe. Par exemple, certaines fonctions de mappage nécessitent un *shuffle* dans le RDD, ce qui signifie distribuer les données entre les travailleurs du cluster pour permettre un traitement parallèle. Ce processus se produit généralement lorsque les données sont réparties de manière inégale, doivent être organisées d'une manière particulière pour le traitement, ou lorsqu'un seul nœud manque de mémoire suffisante pour stocker toutes les données nécessaires. Parfois, il vaut la peine d'utiliser des opérations de mappage nécessitant un shuffle en raison des gains de temps ultérieurs, mais parfois ce n'est pas le cas. Ainsi, comprendre ce qui est préférable dans le cas spécifique de la fusion des réticulés nécessiterait de tester et de comparer différentes approches.

En ce qui concerne les cas d'utilisation dans les architectures distribuées, l'application de la méthode dans un cas réel de l'IoT serait souhaitable, puisque ceux fournis dans ce travail ont été adaptés pour être utilisés et pensés comme des flux de données en constante croissance, mais certaines caractéristiques pourraient avoir été manquantes dans une telle adaptation. De plus, les temps de réponse peuvent varier de plusieurs ordres de grandeur entre l'apprentissage en temps réel dans un contexte IoT et dans un contexte de commerce électronique, donc le contexte pourrait nécessiter des algorithmes et des méthodes qui fonctionnent encore plus rapidement, tout en conservant la flexibilité fournie par ce que nous avons discuté dans le travail.

De plus, les systèmes IoT sont souvent liés à des sources de données hétérogènes. Bien que la FCA pure puisse être utilisée pour extraire des connaissances à partir de celles-ci, il existe des outils plus spécialisés pour traiter la situation où ces sources hétérogènes sont également interconnectées. En particulier, l'Analyse Conceptuelle Relationnelle introduite dans [112] est une extension bien connue de la FCA pour traiter l'exploration de données

[39] https://spark.apache.org/graphx/

multi-relationnelles. Étant donné que l'algorithme central calcule itérativement les réticulés conceptuels de chacun des contextes dans le jeu de données[40], sa scalabilité pourrait également bénéficier de l'application de techniques de *fusion* similaires à celles décrites dans cette thèse.

Une autre zone de recherche ouverte est la *sélection* des instantanés. En d'autres termes, *quand* est-il utile de stocker une instantanée de la réticulé. De plus, d'autres questions se posent concernant l'ajustement du méthode pour récupérer les concepts potentiellement utiles : (1) *combien* cela affecte-t-il le résultat d'inclure le pas de fusion par rapport à ne pas l'inclure, (2) quel est le *ratio de taille de réticulé idéal* à considérer lors de la fusion en tenant compte de la question précédente. En particulier, notre recommandation pour aborder ces questions est de se pencher sur des contextes où les *anomalies* sont aussi importantes (voire plus) que les motifs récurrents (par exemple, la médecine). Nous croyons que des contextes comme celui-ci peuvent fournir des informations précieuses pour aider à répondre à ces questions.

[40]Les entrées de RCA se composent de plusieurs Contextes Formels interconnectés avec des tableaux indiquant les relations entre les objets dans les différents contextes

Resumen extendido en español

Introducción

A lo largo de la historia moderna, se han recopilado y utilizado datos para ayudar al avance de la sociedad. Sin embargo, el crecimiento en la generación de datos hoy en día no tiene precedentes. Según un estudio realizado por la Alianza de Software[41] [16], ya en 2016, alrededor del 90 % de los datos del mundo se habían generado entre los años 2014 y 2016. Además, ha habido una reducción significativa en los costos de almacenamiento, atribuida al surgimiento de nuevas tecnologías, posiblemente superiores, diseñadas para este propósito. Aún más, se espera que la tasa de crecimiento en la generación de datos continúe con su tendencia creciente.

Consecuentemente, una multitud de disciplinas han surgido para ayudar a investigadores y profesionales a poder recopilar datos significativos en medio de este crecimiento sin precedentes. Entre ellas, podemos encontrar la *minería de datos* (DM), que es el proceso de descubrir conocimientos o patrones a partir de grandes cantidades de datos [54]; la recuperación de información (IR), que consiste en encontrar material (generalmente documentos) de naturaleza no estructurada (usualmente texto) que satisfaga una necesidad de información dentro de grandes colecciones (normalmente almacenadas en computadoras) [91]; y la extracción de conocimiento (KE), que es la creación de conocimiento a partir de fuentes estructuradas (por ejemplo, bases de datos relacionales, lenguaje de marcado extensible (XML)) y no estructuradas (por ejemplo, texto, documentos, imágenes) [127].

Además, estos métodos juegan un papel crucial en la Industria 4.0[42] (definida en [77]) al aprovechar datos de diversas fuentes para obtener información procesable. Algunas áreas clave que requieren métodos de minería de datos en la Industria 4.0 son, por ejemplo, *adquisición de datos y sensores*, *integración de datos y conectividad*, *análisis de grandes volúmenes de datos*, y *mantenimiento predictivo*.

Teniendo esto en cuenta, esta tesis trata sobre la aplicación del Análisis Formal de Conceptos (FCA) en entornos de datos distribuidos y dinámicos, teniendo en cuenta algunos desafíos emergentes. Esta investigación responde a la necesidad de técnicas de minería de datos eficaces en la Industria 4.0 y, más específicamente, en el Internet de las cosas (IoT), donde los datos son vastos y continuos.

[41] https://www.bsa.org/about-bsa

[42] También conocida como la Cuarta Revolución Industrial, se caracteriza por la integración de tecnologías digitales en los procesos industriales, lo que lleva a una mayor automatización, conectividad y toma de decisiones basada en datos.

Marco teórico y objetivos

La idea teórica de esta tesis proviene del proyecto europeo denominado Digital Innovation Hub for Cyber-Physical Systems (DIH4CPS)[43]. Desde su concepción, el objetivo del proyecto ha sido ayudar a las empresas afiliadas en la digitalización de los sistemas existentes mediante la aplicación de inteligencia artificial y técnicas de minería de datos, enfocadas en sistemas ciber-físicos y sistemas embebidos. En particular, **esta tesis busca cubrir la brecha en el uso de la extracción de conocimiento *conceptual* a partir de *datos de bajo nivel* cuando provienen de arquitecturas distribuidas, como aquellas en las que a menudo se implementan los sistemas ciber-físicos.**

Existen varios métodos de minería de datos bien conocidos para abordar diferentes problemáticas relacionadas con la extracción de conocimiento a partir de datos de bajo nivel provenientes de arquitecturas distribuidas. Por ejemplo, K-means [69], K-nearest neighbors [99], árboles de regresión [63], entre otros. Sin embargo, la mayoría de estos métodos dependen en gran medida de la estructura de los datos con los que se trabaja y, en algunos casos, de técnicas de modelado que crean un *desajuste de impedancia* entre los datos de origen y los datos que realmente se utilizan como entrada. Esto genera todo tipo de problemas en áreas como la *explicabilidad* y los hace menos atractivos para los profesionales, ya que lograrlo requeriría una cantidad considerable de trabajo adicional. Por esa razón, un método como el Análisis Formal de Conceptos [137], cuyo resultado está estrechamente relacionado con la forma en que los humanos entenderían los "datos conceptuales", es muy valioso para esta tarea.

El Análisis Formal de Conceptos ha demostrado ser efectivo para la extracción de conocimiento en varias áreas relacionadas con la minería de datos *provenientes de* arquitecturas distribuidas (DA) y también *usando* DA. Se han propuesto numerosos enfoques para resolver problemas relacionados con la escalabilidad en estas áreas [141, 28, 132, 49]. Sin embargo, estos métodos introducen nuevos problemas. En particular, cuando se trata de la minería de flujos de datos *infinitos*, el principal problema radica en el hecho de que ningún algoritmo puede procesar datos infinitos, es decir, tomaría un tiempo infinito solo para leer la entrada. Tal problema se aborda comúnmente mediante el uso conjunto de algoritmos incrementales y ventanas deslizantes. A su vez, este paradigma define una manera muy *específica* (aunque varía de un enfoque a otro) de considerar solo una parte del flujo de datos, por ejemplo, solo los últimos k eventos, o solo los conceptos formales más recientemente actualizados. En consecuencia, se introduce la posibilidad de **perder potencialmente conceptos importantes o relevantes**.

Además, los algoritmos incrementales en general, excepto en algunos casos como el de DeleteInstance [149], están definidos para el caso en que se agregan objetos al contexto [132, 129]. Sin embargo, dado que una de las hipótesis de los algoritmos incrementales es que no toda la información (es decir, los objetos en este caso) está disponible en todo momento, también es válido asumir que ni siquiera todos los atributos de un objeto son válidos en todo momento. Por lo tanto, los algoritmos incrementales no son lo suficientemente flexibles para ser utilizados en estos escenarios. En particular, solo existe un algoritmo que puede procesar contextos que están *arbitrariamente* distribuidos de manera efectiva [49]. Sin embargo, este algoritmo pertenece a la categoría de procesamiento por lotes, por lo que se necesita una solución incremental para los casos en los que el contexto no solo está arbitrariamente distribuido, sino que también es dinámico.

[43] http://www.produtech.org/european-projects/dih4cps

Finalmente, se sabe que el problema de la construcción de reticulados es P-completo[44], lo que significa, entre otras cosas, que se cree firmemente que el problema no es efectivamente paralelizable. Sin embargo, dadas las restricciones de tiempo que el entorno del IoT exige, la paralelización contribuye a la escalabilidad, incluso si solo afecta el cálculo por una constante.

En este contexto, los objetivos de esta tesis son:

(G.1) Proporcionar un método escalable y flexible para abordar la pérdida de conceptos potencialmente importantes al construir un reticulado de conceptos utilizando ventanas deslizantes.

(G.2) Proponer un método incremental para la construcción de reticulados conceptos en el contexto en el que no se conocen todos los atributos de un objeto en cada actualización (es decir, el contexto formal está arbitrariamente distribuido y es dinámico).

(G.3) Lograr escalabilidad horizontal mediante la propuesta de un método que aproveche el paralelismo *usando* arquitecturas distribuidas.

(G.4) Demostrar su utilidad potencial en un escenario práctico.

Resultados

Primero, en el capítulo chapter 2, se llevó a cabo una revisión sistemática de la literatura con el fin de proporcionar evidencia sobre las brechas científicas relacionadas con la extracción de conocimiento a partir de fuentes heterogéneas en entornos distribuidos. En ese estudio, se han presentado los resultados de la extracción de datos, incluyendo los métodos utilizados en MRDM, CA y DA, las problemáticas abordadas desde el punto de vista del ciclo de vida del proceso introducido, los dominios, los formatos de entrada y salida y las relaciones entre ellos, y finalmente, las características de los métodos evaluados. Además, se ha presentado un conjunto de reglas de asociación respecto a la categorización realizada con el conjunto de artículos, considerando el tratamiento de entrada y salida en el ciclo de vida del proceso, y las categorías de dominios de ML y Web Semántica. Adicionalmente, se han identificado tres brechas científicas, de las cuales una ha sido abordada en esta tesis, a saber, la necesidad de un método basado en FCA que tenga en cuenta tanto la característica infinita de los flujos de datos en arquitecturas distribuidas dinámicas como la pérdida de conocimiento al utilizar ventanas deslizantes.

En segundo lugar, en el chapter 3, se propuso un método escalable y flexible para ayudar a prevenir la pérdida de conceptos significativos durante la construcción de reticulados de conceptos. El método consiste en la utilización de "fotografías" (*snapshots*) de reticulados como respaldo de conceptos interesantes, y su uso posterior en un paso de *fusión* que debe ser *rápido*, aprovechando el tiempo previamente invertido en el cálculo de dichos snapshots. En el capítulo, se presenta y demuestra la corrección de un algoritmo por lotes (es decir, Algorithm BatchMerge) para fusionar reticulados. Luego, se explica una implementación monohilo utilizando una estructura de datos *trie* para almacenar intents de concepts (i.e., sus attributos) y se contrasta con un algoritmo de fusión incremental (es decir, Algorithm

[44] De hecho, incluso determinar el número de conceptos formales de un cierto contexto formal está en esa clase [75].

IncrementalMerge). Los resultados sobre su complejidad computacional (temporal) presentados en el capítulo sugieren que Algorithm BatchMerge debería desempeñarse mejor que Algorithm BatchMerge en contextos altamente dispersos.

Además, en el chapter 3, se presentaron tres algoritmos distribuidos siguiendo la idea de Algorithm BatchMerge. Por un lado, un algoritmo para fusionar únicamente intents (es decir, Algorithm ParallelMergeOnlyIntents), que puede ser útil para aplicaciones que buscan utilizar solo reglas de asociación. Por otro lado, dos algoritmos para calcular el conjunto completo de conceptos formales (es decir, Algorithm ParallelMergeConcepts y Algorithm ParallelMergeConceptsWithUnion), los cuales ofrecen una mayor escalabilidad que su versión monohilo BatchMerge, ya que aprovechan el paralelismo mediante el modelo de computación MapReduce.

En tercer lugar, en chapter 4, se presentó un algoritmo incremental para la minería de contextos formales provenientes de fuentes distribuidas. En particular, dado que la mayoría de los algoritmos requieren que el contexto esté particionado de una manera "particular" (por ejemplo, cada trabajador debe conocer todos los atributos, por lo que solo se pueden particionar los objetos), o requieren que todo el contexto se replique en todos los nodos de la arquitectura, *no hay suficiente flexibilidad* para que los profesionales puedan minar realmente contextos que están *arbitrariamente distribuidos*. Considerando que existe un algoritmo distribuido que aborda este problema de manera *por lotes*, la tesis propone una solución *incremental* que utiliza dos algoritmos ya existentes, AddIntent y DeleteInstance, y analiza la complejidad en comparación con la cantidad máxima de búsquedas de *conceptos modificados*. Los resultados sugieren que el algoritmo propuesto debería superar a la versión *por lotes* porque no sufre de crecimiento exponencial en los atributos, a diferencia de su competencia.

En cuarto lugar, en chapter 5 se presentaron varios casos de uso prácticos para demostrar los beneficios reales de los métodos propuestos. En particular, se implementó un sistema de recomendaciones para el sitio de comercio electrónico de la Escuela Francesa de Esquí (ESF) con la idea de utilizar el Análisis Formal de Conceptos como medio para calcular y luego recomendar en función de conjuntos de ítems frecuentes, una idea que ya ha demostrado ser efectiva múltiples veces para recomendar artículos que "generalmente van juntos". En este caso, la novedad radica en el paso intermedio de "fusión" que permite recuperar ciertos conceptos que han sido perdidos y que quizas era deseable tenerlos en consideración igualmente. Por ejemplo, si la ESF quiere poner en oferta un tipo específico de lección, puede utilizar un snapshot que contenga el reticulado correspondiente a un período de tiempo en el que una oferta similar ha estado activa. Esto también se puede forzar mediante lógica de negocio, sin embargo, no es tan adaptable como el método basado en datos, que tendrá conocimiento realmente relacionado con cada uno de los centros y las marcas de tiempo que elija el profesional o practicante.

Finalmente, en chapter 6, los algoritmos presentados fueron sometidos a una serie de experimentos, utilizando tanto conjuntos de datos reales como sintéticos, para evaluar su rendimiento empírico en relación con el crecimiento de la entrada. Todos los experimentos fueron diseñados con el objetivo de simular una situación en la que hay dos reticulados para fusionar, con diferentes relaciones objeto-objeto (es decir, la diferencia entre la cantidad de objetos en os dos reticulados). En todos los experimentos, el rendimiento empírico de Algorithm BatchMerge mostró resultados prometedores en comparación con el de Algorithm IncrementalMerge, al tratar con *contextos dispersos*. Sin embargo, los resultados también mostraron que Algorithm BatchMerge tiene un rendimiento deficiente en comparación con el uso de la adición incremental "objeto por objeto" a medida que el contexto se vuelve más

denso.

En cuanto a Algorithm AddPair, mostró un rendimiento extremadamente similar al de Algorithm AddIntent a pesar de que, intuitivamente, debería tener un rendimiento peor ya que se llamaría $|g'|$ veces para cada g en lugar de solo una vez. Por lo tanto, dado que Algorithm AddPair es más flexible que su contraparte (es decir, puede utilizarse para construir reticulados a partir de contextos arbitrariamente distribuidos, a diferencia de Algorithm AddIntent), su uso en contextos dispersos sería preferible.

Finalmente, se utilizaron versiones distribuidas de los conjuntos de datos del mundo real para comparar los algoritmos de fusión distribuidos propuestos (es decir, ParallelMergeOnlyIntents, ParallelMergeConcepts y ParallelMergeConceptsWithUnion) con el de un solo hilo (o monohilo) (BatchMerge) y el algoritmo de Goel y Chaudhary. Los experimentos mostraron que los algoritmos distribuidos tuvieron un rendimiento significativamente mejor en casi todos los casos. Sin embargo, el uso de un clúster (escalado horizontal) solo resultó ventajoso al tratar con problemas más grandes. Para problemas más pequeños (tamaño de retícula por debajo de $|\mathcal{L}_m| = 1500$), el tráfico de red adicional generado por el clúster estuvo significó más que los beneficios, haciendo que fuera más eficiente ejecutar en una sola computadora.

Perspectivas y direcciones futuras

Existen varios frentes abiertos para explorar después de las contribuciones mencionadas. En cuanto al aspecto teórico del trabajo, los algoritmos para fusionar reticulados de snapshots filtradas (por ejemplo, reducidas por un umbral de soporte) podrían mejorar aún más el rendimiento, por lo que hacerlo sería una continuación deseable y directa del trabajo. Además, el análisis de la complejidad del algoritmo incremental "mínimo" para la construcción de reticulados a partir de contextos arbitrariamente distribuidos podría ser menor si se consideran ciertas propiedades de la *cobertura* de un concepto, que podría no necesitar $\mathcal{O}(|G|^2|M|)$ para cada concepto en $\mathcal{L}$, lo que llevaría a una mejora en el límite superior del tiempo de complejidad en el peor caso.

En cuanto al enfoque de MapReduce para calcular la fusión de reticulados, se necesita un algoritmo *distribuido* para calcular también el diagrama de líneas para las aplicaciones que puedan hacer uso de él. Para ello, una dirección aparentemente prometedora es utilizar la implementación distribuida de grafos GraphX[45], que es una implementación de Apache Spark, y extender los algoritmos que se propusieron en esta tesis. Además, las coberturas (es decir, los vecinos superiores de un concepto en el diagrama de líneas) se pueden encontrar utilizando la función FOUND-UPPER-COVERS-BIS, presentada en [129]. Sin embargo, el desafío radica en encontrar una forma adecuada de transformar esa función en una distribuida que realmente aproveche el paralelismo.

En cuanto a los aspectos prácticos de la tesis, se necesita un benchmark detallado para evaluar la versión paralela del algoritmo de fusión con el fin de comprender cuáles son las ganancias reales de la paralelización usando Apache Spark. En particular, hay varios aspectos del marco que hacen que un análisis teórico sea mucho más complejo. Por ejemplo, ciertas funciones de mapeo requieren un *shuffle* en el RDD, lo que significa distribuir datos entre los trabajadores del clúster para permitir el procesamiento paralelo. Este proceso ocurre típicamente cuando los datos están distribuidos de manera desigual, necesitan ser organizados de una manera particular para su procesamiento o cuando un nodo único no

[45] https://spark.apache.org/graphx/

tiene suficiente memoria para almacenar todos los datos necesarios. A veces vale la pena usar operaciones de mapeo que requieren shuffle debido a las ganancias de tiempo posteriores, pero a veces no. Por lo tanto, entender cuál es la opción preferible en el caso específico de fusión de reticulados requeriría probar y comparar diferentes enfoques.

En cuanto a los casos de uso en arquitecturas distribuidas, sería deseable aplicar el método en un caso real de IoT, ya que los proporcionados en este trabajo fueron adaptaron del sitio de comercio electrónico para ser utilizados y pensados como flujos de datos en constante crecimiento, pero podrían haber faltado ciertas características importantes en dicha adaptación. Adicionalmente, los tiempos de respuesta pueden variar entre el aprendizaje en tiempo real en un entorno IoT y en uno de comercio electrónico por órdenes de magnitud, por lo que el entorno podría requerir algoritmos y métodos que funcionen aún más rápido, conservando la flexibilidad proporcionada por lo discutido en el trabajo.

Además, los sistemas de IoT a menudo están relacionados con fuentes de datos heterogéneas. Aunque el AFC simple puede usarse para minar de ellas, existen herramientas más especializadas para tratar con la situación en la que esas fuentes heterogéneas también están interrelacionadas. En particular, el Análisis Relacional de Conceptos introducido en [112] es una extensión ampliamente conocida del AFC para tratar con la minería de datos multirrelacionales. Dado que el algoritmo central calcula iterativamente los reticulados de conceptos de cada uno de los contextos en el conjunto de datos[46], su escalabilidad también podría beneficiarse de la aplicación de técnicas de *fusión* similares a las descritas en esta tesis.

Otra área que queda abierta a la investigación es la *selección* de snapshots. En otras palabras, *cuándo* es útil almacenar un snapshot del reticulado. Además, surgen otras preguntas relacionadas con el ajuste fino del método para recuperar conceptos potencialmente útiles: (1) *¿cuánto* afecta al resultado incluir el paso de fusión frente a no incluirlo? (2) ¿cuál es el *radio de tamaño ideal entre los reticulados* a considerar al fusionar, teniendo en cuenta la pregunta anterior? En particular, nuestra recomendación sobre cómo abordar estas preguntas es investigar contextos donde las *anomalías* sean tan importantes (o incluso más) que los patrones recurrentes (por ejemplo, en medicina). Creemos que contextos como ese pueden proporcionar información valiosa para ayudar a responder estas preguntas.

[46]La entrada de RCA consiste en varios Contextos Formales interrelacionados con tablas que indican relaciones entre objetos en los diferentes contextos

A

List of Abbreviations

- **API** Application Public Interface
- **CA** Concept Analysis
- **CSV** Comma Separated Values
- **DA** distributed architecture
- **DM** data mining
- **ESF** École du ski français
- **FCA** Formal Concept Analysis
- **FFCA** Fuzzy Formal Concept Analysis
- **GB** Gigabyte
- **GHz** Gigahertz
- **KDD** Knowledge Discovery in Databases
- **KE** Knowledge Extraction
- **MRDM** Multi-relational data mining
- **MSEM** Mon séjour en montagne
- **RAM** Random Access Memory
- **RCA** Relational Concept Analysis
- **RDD** Resilient Distributed Dataset
- **RDF** Resource Description Framework
- **TFCA** Temporal Formal Concept Analysis

Bibliography

[1] S. O. Aanderaa. On the Decision Problem for Formulas in which all Disjunctions are Binary*. In J. E. Fenstad, editor, *Studies in Logic and the Foundations of Mathematics*, volume 63 of *Proceedings of the Second Scandinavian Logic Symposium*, pages 1–18. Elsevier, Jan. 1971. doi: 10.1016/S0049-237X(08)70839-5. URL https://www.sciencedirect.com/science/article/pii/S0049237X08708395.

[2] R. Agrawal, T. Imieliński, and A. Swami. Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD international conference on Management of data*, SIGMOD '93, pages 207–216, New York, NY, USA, June 1993. Association for Computing Machinery. ISBN 978-0-89791-592-2. doi: 10.1145/170035.170072. URL https://doi.org/10.1145/170035.170072.

[3] S. Akmal and R. Batres. A Methodology for Developing Manufacturing Process Ontologies. *Journal of Japan Industrial Management Association*, 64:303–316, Jan. 2013. doi: 10.11221/jima.64.303.

[4] M. Alavi and D. E. Leidner. Review: Knowledge management and knowledge management systems: conceptual foundations and research issues. *MIS Quarterly*, 25 (1):107–136, Mar. 2001. ISSN 0276-7783. doi: 10.2307/3250961. URL https://doi.org/10.2307/3250961.

[5] S. Albahli and A. Melton. TripleFCA: FCA-Based Approach to Enhance Semantic Web Data Management. In *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)*, volume 1, pages 625–630, June 2016. doi: 10.1109/COMPSAC.2016.212. ISSN: 0730-3157.

[6] A. Aloui and A. Grissa. A New Approach for Flexible Queries Using Fuzzy Ontologies. In A. T. Azar and S. Vaidyanathan, editors, *Computational Intelligence Applications in Modeling and Control*, Studies in Computational Intelligence, pages 315–342. Springer International Publishing, Cham, 2015. ISBN 978-3-319-11017-2. doi: 10.1007/978-3-319-11017-2_13. URL https://doi.org/10.1007/978-3-319-11017-2_13.

[7] S. Andrews. In-Close, a Fast Algorithm for Computing Formal Concepts. page 15, Moscow, 2009.

[8] S. Andrews and C. Orphanides. Knowledge Discovery through Creating Formal Contexts. page 460, Nov. 2010. doi: 10.1109/INCOS.2010.53. Journal Abbreviation: Proceedings - 2nd International Conference on Intelligent Networking and Collaborative Systems, INCOS 2010 Publication Title: Proceedings - 2nd International Conference on Intelligent Networking and Collaborative Systems, INCOS 2010.

[9] J. E. Aronson. Expert Systems. In H. Bidgoli, editor, *Encyclopedia of Information Systems*, pages 277–289. Elsevier, New York, Jan. 2003. ISBN 978-0-12-227240-0. doi: 10.1016/B0-12-227240-4/00067-8. URL https://www.sciencedirect.com/science/article/pii/B0122272404000678.

[10] S. Arora and B. Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 1 edition, Apr. 2009. ISBN 978-0-521-42426-4 978-0-511-80409-0. doi: 10.1017/CBO9780511804090. URL https://www.cambridge.org/core/product/identifier/9780511804090/type/book.

[11] M. Atencia, J. David, J. Euzenat, A. Napoli, and J. Vizzini. A guided walk into link key candidate extraction with relational concept analysis. In *ISWC 2019 - 18th International Semantic Web Conference*, pages 1–9, Auckland, New Zealand, Oct. 2019. No commercial editor. URL https://hal.archives-ouvertes.fr/hal-02984963.

[12] M. Atencia, J. David, J. Euzenat, A. Napoli, and J. Vizzini. Link key candidate extraction with relational concept analysis. *Discrete Applied Mathematics*, 273:2–20, Feb. 2020. ISSN 0166-218X. doi: 10.1016/j.dam.2019.02.012. URL https://www.sciencedirect.com/science/article/pii/S0166218X19300952.

[13] J. P. Bordat. Calcul pratique du treillis de Galois d'une correspondance. *Informatiques et Sciences Humaines*, 96:31–47, 1986.

[14] R. J. Brachman. ON THE EPISTEMOLOGICAL STATUS OF SEMANTIC NETWORKS*. In N. V. Findler, editor, *Associative Networks*, pages 3–50. Academic Press, Jan. 1979. ISBN 978-0-12-256380-5. doi: 10.1016/B978-0-12-256380-5.50007-4. URL https://www.sciencedirect.com/science/article/pii/B9780122563805500074.

[15] L. Breiman. Random Forests. *Machine Learning*, 45(1):5–32, Oct. 2001. ISSN 1573-0565. doi: 10.1023/A:1010933404324. URL https://doi.org/10.1023/A:1010933404324.

[16] BSA. What's the Big Deal with Data? 2016. URL http://download.microsoft.com/documents/en-us/sam/bsadatastudy_en.pdf.

[17] J. Carbonnel, M. Huchard, and C. Nebut. Towards Complex Product Line Variability Modelling: Mining Relationships from Non-Boolean Descriptions. *Journal of Systems and Software*, 156:341–360, Oct. 2019. doi: 10.1016/j.jss.2019.06.002. URL https://hal.archives-ouvertes.fr/hal-02146375. Publisher: Elsevier.

[18] C. Carpineto and G. Romano. GALOIS: an order-theoretic approach to conceptual clustering. In *Proceedings of the Tenth International Conference on International Conference on Machine Learning*, ICML'93, pages 33–40, San Francisco, CA, USA, July 1993. Morgan Kaufmann Publishers Inc. ISBN 978-1-55860-307-3.

[19] Z. Chang-sheng, R. Jing, H. Hai-long, L. Long-chang, and Y. Bing-ru. An Algorithm on Generating Lattice Based on Layered Concept Lattice. *Indonesian Journal of Electrical Engineering and Computer Science*, 11(8):4477–4483, Aug. 2013. ISSN 2502-4760. doi: http://dx.doi.org/10.11591/telkomnika.v11i8.3063. URL https://ijeecs.iaescore.com/index.php/IJEECS/article/view/2509. Number: 8.

[20] S. Choudhury, L. Holder, J. Feo, and G. Chin. Fast search for dynamic multi-relational graphs. In *Proceedings of the Workshop on Dynamic Networks Management and Mining*, DyNetMM '13, pages 1–8, New York, NY, USA, June 2013. Association for Computing Machinery. ISBN 978-1-4503-2209-6. doi: 10.1145/2489247.2489251. URL https://doi.org/10.1145/2489247.2489251.

[21] V. Co, C. Taramasco, and H. Astudillo. Cheating to achieve Formal Concept Analysis over a large formal context. volume 959, Oct. 2011. Journal Abbreviation: CEUR Workshop Proceedings Publication Title: CEUR Workshop Proceedings.

[22] P. Cordero, M. Enciso, A. Mora, and I. Pérez de Guzmán. A tableaux-like method to infer all minimal keys. *Logic Journal of the IGPL*, 22(6):1019–1044, Dec. 2014. ISSN 1368-9894. doi: 10.1093/jigpal/jzu025. Conference Name: Logic Journal of the IGPL.

[23] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms*. MIT Press, Cambridge, Massachusetts London, England, third edition edition, 2009. ISBN 978-0-262-03384-8 978-0-262-53305-8.

[24] G. De Francisci Morales, A. Bifet, L. Khan, J. Gama, and W. Fan. IoT Big Data Stream Mining. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 2119–2120, New York, NY, USA, Aug. 2016. Association for Computing Machinery. ISBN 978-1-4503-4232-2. doi: 10.1145/2939672.2945385. URL https://doi.org/10.1145/2939672.2945385.

[25] C. De Maio, G. Fenza, M. Gallo, V. Loia, and S. Senatore. Formal and relational concept analysis for fuzzy-based automatic semantic annotation. *Applied Intelligence*, 40(1):154–177, Jan. 2014. ISSN 0924-669X. doi: 10.1007/s10489-013-0451-7. URL https://doi.org/10.1007/s10489-013-0451-7.

[26] C. De Maio, G. Fenza, V. Loia, and M. Parente. Online query-focused twitter summarizer through fuzzy lattice. In *2015 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, pages 1–8, Aug. 2015. doi: 10.1109/FUZZ-IEEE.2015.7337927.

[27] C. De Maio, G. Fenza, V. Loia, and M. Parente. Time Aware Knowledge Extraction for microblog summarization on Twitter. *Information Fusion*, 28:60–74, Mar. 2016. ISSN 1566-2535. doi: 10.1016/j.inffus.2015.06.004. URL https://www.sciencedirect.com/science/article/pii/S156625351500055X.

[28] C. De Maio, G. Fenza, V. Loia, and F. Orciuoli. Distributed online Temporal Fuzzy Concept Analysis for stream processing in smart cities. *Journal of Parallel and Distributed Computing*, 110:31–41, Dec. 2017. ISSN 0743-7315. doi: 10.1016/j.jpdc.2017.02.002. URL https://www.sciencedirect.com/science/article/pii/S0743731517300503.

[29] C. De Maio, G. Fenza, V. Loia, and F. Orciuoli. Making sense of cloud-sensor data streams via Fuzzy Cognitive Maps and Temporal Fuzzy Concept Analysis. *Neurocomputing*, 256:35–48, Sept. 2017. ISSN 0925-2312. doi: 10.1016/j.neucom.2016.06.090. URL https://www.sciencedirect.com/science/article/pii/S0925231217304125.

[30] J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, Jan. 2008. ISSN 0001-0782. doi: 10.1145/1327452.1327492. URL https://dl.acm.org/doi/10.1145/1327452.1327492.

[31] E. W. Dijkstra. Cooperating Sequential Processes. In P. B. Hansen, editor, *The Origin of Concurrent Programming: From Semaphores to Remote Procedure Calls*, pages 65–138. Springer, New York, NY, 2002. ISBN 978-1-4757-3472-0. doi: 10.1007/978-1-4757-3472-0_2. URL https://doi.org/10.1007/978-1-4757-3472-0_2.

[32] X. Dolques, F. Le Ber, and M. Huchard. AOC-posets: a scalable alternative to Concept Lattices for Relational Concept Analysis. *CEUR Workshop Proceedings*, 1062, Oct. 2013.

[33] X. Dolques, F. Le Ber, M. Huchard, and C. Grac. Performance-friendly rule extraction in large water data-sets with AOC posets and relational concept analysis. *International Journal of General Systems*, 45(2):187–210, Feb. 2016. ISSN 0308-1079. doi: 10.1080/03081079.2015.1072927. URL https://doi.org/10.1080/03081079.2015.1072927. Publisher: Taylor & Francis _eprint: https://doi.org/10.1080/03081079.2015.1072927.

[34] P. du Boucher-Ryan and D. Bridge. Collaborative Recommending using Formal Concept Analysis. In M. Bramer, F. Coenen, and T. Allen, editors, *Research and Development in Intelligent Systems XXII*, pages 205–218, London, 2006. Springer. ISBN 978-1-84628-226-3. doi: 10.1007/978-1-84628-226-3_16.

[35] J. Ekanayake, H. Li, B. Zhang, T. Gunarathne, S.-H. Bae, J. Qiu, and G. Fox. Twister: a runtime for iterative MapReduce. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, pages 810–818, Chicago Illinois, June 2010. ACM. ISBN 978-1-60558-942-8. doi: 10.1145/1851476.1851593. URL https://dl.acm.org/doi/10.1145/1851476.1851593.

[36] N. Fanizzi, C. d'Amato, and F. Esposito. Fuzzy Clustering for Categorical Spaces. In J. Rauch, Z. W. Raś, P. Berka, and T. Elomaa, editors, *Foundations of Intelligent Systems*, Lecture Notes in Computer Science, pages 161–170, Berlin, Heidelberg, 2009. Springer. ISBN 978-3-642-04125-9. doi: 10.1007/978-3-642-04125-9_19.

[37] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. From Data Mining to Knowledge Discovery in Databases. *AI Magazine*, 17(3):37–37, Mar. 1996. ISSN 2371-9621. doi: 10.1609/aimag.v17i3.1230. URL https://ojs.aaai.org/index.php/aimagazine/article/view/1230. Number: 3.

[38] S. Ferré and P. Cellier. Graph-FCA in Practice. page 107, July 2016. doi: 10.1007/978-3-319-40985-6_9. URL https://hal.inria.fr/hal-01405491.

[39] A. Formica. Concept Similarity in Formal Concept Analysis with Many-Valued Contexts. *COMPUTING AND INFORMATICS*, 40(3):469–488, Nov. 2021. ISSN 2585-8807. doi: 10.31577/cai_2021_3_469. URL https://www.cai.sk/ojs/index.php/cai/article/view/2021_3_469. Number: 3.

[40] C. Franciosi, B. Iung, S. Miranda, and S. Riemma. Maintenance for Sustainability in the Industry 4.0 context: a Scoping Literature Review. *IFAC-PapersOnLine*, 51 (11):903–908, Jan. 2018. ISSN 2405-8963. doi: 10.1016/j.ifacol.2018.08.459. URL https://www.sciencedirect.com/science/article/pii/S2405896318315866.

[41] B. Ganter. Two Basic Algorithms in Concept Analysis. In L. Kwuida and B. Sertkaya, editors, *Formal Concept Analysis*, Lecture Notes in Computer Science, pages 312–340, Berlin, Heidelberg, 2010. Springer. ISBN 978-3-642-11928-6. doi: 10.1007/978-3-642-11928-6_22.

[42] B. Ganter and R. Wille. *Formal Concept Analysis: Mathematical Foundations*. Springer, Berlin, Heidelberg, 1999. ISBN 978-3-540-62771-5 978-3-642-59830-2. doi: 10.1007/978-3-642-59830-2. URL http://link.springer.com/10.1007/978-3-642-59830-2.

[43] Z.-Y. Gao, Y.-Q. Liang, and S.-H. Qiao. Relational Database Ontology Discovery Method Based on Formal Concept Analysis. pages 727–735. Atlantis Press, Dec. 2016. ISBN 978-94-6252-303-6. doi: 10.2991/mme-16.2017.101. URL https://www.atlantis-press.com/proceedings/mme-16/25871532. ISSN: 2352-5401.

[44] George Voutsadakis. Polyadic Concept Analysis. *Order*, 19(3):295–304, Sept. 2002. ISSN 1572-9273. doi: 10.1023/A:1021252203599. URL https://doi.org/10.1023/A:1021252203599.

[45] X. Glorot, A. Bordes, J. Weston, and Y. Bengio. A Semantic Matching Energy Function for Learning with Multi-relational Data, Mar. 2013. URL http://arxiv.org/abs/1301.3485. arXiv:1301.3485 [cs].

[46] K. Godel. *On Formally Undecidable Propositions of Principia Mathematica and Related Systems*. Dover Publications, 1992. ISBN 0-486-66980-7. URL http://www.ddc.net/ygg/etext/godel.

[47] R. Godin and R. Missaoui. An incremental concept formation approach for learning from databases. 11, 1995. doi: https://doi.org/10.1111/j.1467-8640.1995.tb00031.x.

[48] R. Godin, R. Missaoui, and H. Alaoui. Learning algorithms using a Galois lattice structure. In *[Proceedings] Third International Conference on Tools for Artificial Intelligence - TAI 91*, pages 22–29, Nov. 1991. doi: 10.1109/TAI.1991.167072. URL https://ieeexplore.ieee.org/document/167072.

[49] V. Goel and B. D. Chaudhary. Concept Discovery from Un-Constrained Distributed Context. In *Proceedings of the 4th International Conference on Big Data Analytics - Volume 9498*, BDA 2015, pages 151–164, Berlin, Heidelberg, Dec. 2015. Springer-Verlag. ISBN 978-3-319-27056-2. doi: 10.1007/978-3-319-27057-9_11. URL https://doi.org/10.1007/978-3-319-27057-9_11.

[50] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. URL http://www.deeplearningbook.org.

[51] T. R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199–220, June 1993. ISSN 1042-8143. doi: 10.1006/knac.1993.1008. URL https://www.sciencedirect.com/science/article/pii/S1042814383710083.

[52] N. Guarino, D. Oberle, and S. Staab. What Is an Ontology? In S. Staab and R. Studer, editors, *Handbook on Ontologies*, pages 1–17. Springer, Berlin, Heidelberg, 2009. ISBN 978-3-540-92673-3. doi: 10.1007/978-3-540-92673-3_0. URL https://doi.org/10.1007/978-3-540-92673-3_0.

[53] S. Guesmi, C. Trabelsi, and C. Latiri. Multidimensional community discovering in heterogeneous social networks. *Concurrency and Computation: Practice and Experience*, 33(1):e5809, 2021. ISSN 1532-0634. doi: 10.1002/cpe.5809. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.5809. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpe.5809.

[54] J. Han. Data Mining. In L. LIU and M. T. ÖZSU, editors, *Encyclopedia of Database Systems*, pages 595–598. Springer US, Boston, MA, 2009. ISBN 978-0-387-39940-9. doi: 10.1007/978-0-387-39940-9_104. URL https://doi.org/10.1007/978-0-387-39940-9_104.

[55] W. He, S. Li, and X. Yang. A Hybrid Approach for Reducing Textual Formal Context Based on Thesaurus. In *2015 11th International Conference on Computational Intelligence and Security (CIS)*, pages 146–149, Dec. 2015. doi: 10.1109/CIS.2015.43.

[56] S. Hernández, P. Álvarez, J. Fabra, and J. Ezpeleta. Analysis of Users' Behavior in Structured e-Commerce Websites. *IEEE Access*, PP:1–1, May 2017. doi: 10.1109/ACCESS.2017.2707600.

[57] M. Hildebrandt, S. S. Sunder, S. Mogoreanu, M. Joblin, A. Mehta, I. Thon, and V. Tresp. A Recommender System for Complex Real-World Applications with Nonlinear Dependencies and Knowledge Graph Context. In P. Hitzler, M. Fernández, K. Janowicz, A. Zaveri, A. J. Gray, V. Lopez, A. Haller, and K. Hammar, editors, *The Semantic Web*, Lecture Notes in Computer Science, pages 179–193, Cham, 2019. Springer International Publishing. ISBN 978-3-030-21348-0. doi: 10.1007/978-3-030-21348-0_12.

[58] M. Hildebrandt, S. S. Sunder, S. Mogoreanu, I. Thon, V. Tresp, and T. Runkler. Configuration of Industrial Automation Solutions Using Multi-relational Recommender Systems. In U. Brefeld, E. Curry, E. Daly, B. MacNamee, A. Marascu, F. Pinelli, M. Berlingerio, and N. Hurley, editors, *Machine Learning and Knowledge Discovery in Databases*, Lecture Notes in Computer Science, pages 271–287, Cham, 2019. Springer International Publishing. ISBN 978-3-030-10997-4. doi: 10.1007/978-3-030-10997-4_17.

[59] T. Ho. An Approach to Concept Formation Based on Formal Concept Analysis. *IEICE Transactions on Information and Systems*, May 1995.

[60] X. Huang, J. Lin, and D. Demner-Fushman. Evaluation of PICO as a knowledge representation for clinical questions. *AMIA ... Annual Symposium proceedings. AMIA Symposium*, 2006:359–363, 2006. ISSN 1942-597X.

[61] Y. Huang, M. Nickel, V. Tresp, and H.-P. Kriegel. A scalable kernel approach to learning in semantic graphs with applications to linked data. volume 685, pages 3–13, 2010. ISSN: 1613-0073.

[62] G. Hulten, L. Spencer, and P. Domingos. Mining time-changing data streams. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '01, pages 97–106, New York, NY, USA, Aug. 2001. Association for Computing Machinery. ISBN 978-1-58113-391-2. doi: 10.1145/502512.502529. URL https://doi.org/10.1145/502512.502529.

[63] E. Ikonomovska and S. Džeroski. Regression on evolving multi-relational data streams. In *Proceedings of the 2011 Joint EDBT/ICDT Ph.D. Workshop*, PhD '11, pages 1–7, New York, NY, USA, Mar. 2011. Association for Computing Machinery. ISBN 978-1-4503-0696-6. doi: 10.1145/1966874.1966875. URL https://doi.org/10.1145/1966874.1966875.

[64] E. Ikonomovska, J. Gama, and S. Džeroski. Learning model trees from evolving data streams. *Data Mining and Knowledge Discovery*, 23(1):128–168, July 2011. ISSN 1573-756X. doi: 10.1007/s10618-010-0201-y. URL https://doi.org/10.1007/s10618-010-0201-y.

[65] N. Jain and R. Krestel. Learning Fine-Grained Semantics for Multi-Relational Data. page 5, 2020.

[66] S. Jalali and C. Wohlin. Systematic literature studies: database searches vs. backward snowballing. In *Proceedings of the ACM-IEEE international symposium on Empirical software engineering and measurement*, ESEM '12, pages 29–38, New York, NY, USA, Sept. 2012. Association for Computing Machinery. ISBN 978-1-4503-1056-7. doi: 10.1145/2372251.2372257. URL https://doi.org/10.1145/2372251.2372257.

[67] G. James, D. Witten, T. Hastie, and R. Tibshirani. Statistical Learning. In G. James, D. Witten, T. Hastie, and R. Tibshirani, editors, *An Introduction to Statistical Learning: with Applications in R*, pages 15–57. Springer US, New York, NY, 2021. ISBN 978-1-07-161418-1. doi: 10.1007/978-1-0716-1418-1_2. URL https://doi.org/10.1007/978-1-0716-1418-1_2.

[68] Z. Jian and L. Kong. A novel algorithm for classification rule discovery based on concept granule structure. *Journal of Digital Information Management*, 14(2):73–80, 2016. ISSN 0972-7272.

[69] X. Jin and J. Han. K-Means Clustering. In C. Sammut and G. I. Webb, editors, *Encyclopedia of Machine Learning*, pages 563–564. Springer US, Boston, MA, 2010. ISBN 978-0-387-30164-8. doi: 10.1007/978-0-387-30164-8_425. URL https://doi.org/10.1007/978-0-387-30164-8_425.

[70] K. Järvelin and J. Kekäläinen. Cumulated gain-based evaluation of IR techniques. *ACM Trans. Inf. Syst.*, 20(4):422–446, Oct. 2002. ISSN 1046-8188. doi: 10.1145/582415.582418. URL https://doi.org/10.1145/582415.582418.

[71] N. Khediri and W. Karoui. Community Detection in Social Network with Node Attributes Based on Formal Concept Analysis. In *2017 IEEE/ACS 14th International Conference on Computer Systems and Applications (AICCSA)*, pages 1346–1353, Oct. 2017. doi: 10.1109/AICCSA.2017.200. ISSN: 2161-5330.

[72] H. Kunjachan, M. Hareesh, and K. Sreedevi. Recommendation Using Frequent Itemset Mining in Big Data. In *2018 Second International Conference on Intelligent Computing and Control Systems (ICICCS)*, pages 561–566, June 2018. doi: 10.1109/ICCONS.2018.8662905. URL https://ieeexplore.ieee.org/document/8662905.

[73] S. Kuznetsov. A fast algorithm for computing all intersections of objects in a finite semi-lattice. *Automatic Documentation and Mathematical Linguistics*, 27:11–21, Jan. 1993.

[74] S. Kuznetsov. On stability of a formal concept. *Ann. Math. Artif. Intell.*, 49:101–115, Aug. 2007. doi: 10.1007/s10472-007-9053-6.

[75] S. O. Kuznetsov and S. A. Obiedkov. Comparing performance of algorithms for generating concept lattices. *Journal of Experimental & Theoretical Artificial Intelligence*, 14(2-3):189–216, Apr. 2002. ISSN 0952-813X, 1362-3079. doi: 10.1080/09528130210164170. URL http://www.tandfonline.com/doi/abs/10.1080/09528130210164170.

[76] J. Kötters and P. W. Eklund. Conjunctive query pattern structures: A relational database model for Formal Concept Analysis. *Discrete Applied Mathematics*, 273:144–171, Feb. 2020. ISSN 0166-218X. doi: 10.1016/j.dam.2019.08.019. URL https://www.sciencedirect.com/science/article/pii/S0166218X19303920.

[77] H. Lasi, P. Fettke, T. Feld, and M. Hoffmann. Industry 4.0. *Business & Information Systems Engineering*, 6(4):239–242, Oct. 2014. URL https://aisel.aisnet.org/bise/vol6/iss4/5.

[78] Y. Lei, M. Qu, C. Lei, Z. Kong, J. Tian, and S. Wang. Two FCA-Based Methods for Reducing Energy Consumption of Sensor Nodes in Wireless Sensor Networks. *Scientific Programming*, 2022, July 2022. doi: 10.1155/2022/8520447.

[79] J. Leskovec, A. Rajaraman, and J. D. Ullman. *Mining of Massive Datasets*. Cambridge University Press, second edition, 2014. ISBN 978-1-108-75131-5. URL http://mmds.org.

[80] N. Leutwyler, M. Lezoche, H. Panetto, and D. Torres. Extending RCA algorithm to consider ternary relations. In *ICIST 2022 Proceedings*, pages 203–209. Information Society of Serbia - ISOS, 2022. URL http://www.eventiotic.com/eventiotic/library/paper/714.

[81] N. Leutwyler, M. Lezoche, H. Panetto, and D. Torres. Systematic Literature Review - Selected Articles - Data Extraction, Oct. 2023. URL https://doi.org/10.5281/zenodo.10036717.

[82] N. Leutwyler, M. Lezoche, H. Panetto, and D. Torres. Generic software for benchmarking Formal Concept Analysis: Orange3 integration. *Electronic Journal of SADIO (EJS)*, 22(3):28–40, Aug. 2023. ISSN 1514-6774. URL https://publicaciones.sadio.org.ar/index.php/EJS/article/view/597. Number: 3.

[83] N. Leutwyler, M. Lezoche, D. Torres, and H. Panetto. Multi-relational and Concept Analysis based Knowledge extraction in the Industry 4.0: A systematic mapping. *IFAC-PapersOnLine*, 56(2):7318–7329, Jan. 2023. ISSN 2405-8963. doi: 10.1016/j.ifacol.2023.10.345. URL https://www.sciencedirect.com/science/article/pii/S2405896323007127.

[84] N. Leutwyler, M. Lezoche, D. Torres, and H. Panetto. Towards a Flexible and Scalable Data Stream Algorithm in FCA. In M. Ojeda-Aciego, K. Sauerwald, and R. Jäschke, editors, *Graph-Based Representation and Reasoning*, Lecture Notes in Computer Science, pages 104–117, Cham, 2023. Springer Nature Switzerland. ISBN 978-3-031-40960-8. doi: 10.1007/978-3-031-40960-8_9.

[85] N. Leutwyler, M. Lezoche, C. Franciosi, H. Panetto, L. Teste, and D. Torres. Methods for concept analysis and multi-relational data mining: a systematic literature review. *Knowledge and Information Systems*, May 2024. ISSN 0219-3116. doi: 10.1007/s10115-024-02139-x. URL https://doi.org/10.1007/s10115-024-02139-x.

[86] C. Lindig. Fast Concept Analysis. In *Working with Conceptual Structures - Contributions to ICCS 2000*, pages 152–161. Shaker Verlag, Aug. 2000. Place: Aachen, Germany.

[87] H. Liu, X. Yin, and J. Han. An efficient multi-relational Naïve Bayesian classifier based on semantic relationship graph. pages 39–48, Aug. 2005. doi: 10.1145/1090193.1090200.

[88] Longlivetheux. English: DIKW pyramid: data, information, knowledge, and wisdom, Jan. 2015. URL https://commons.wikimedia.org/wiki/File:DIKW_Pyramid.svg.

[89] O. Maimon and L. Rokach. Introduction to Knowledge Discovery in Databases. In O. Maimon and L. Rokach, editors, *Data Mining and Knowledge Discovery Handbook*, pages 1–17. Springer US, Boston, MA, 2005. ISBN 978-0-387-25465-4. doi: 10.1007/0-387-25465-X_1. URL https://doi.org/10.1007/0-387-25465-X_1.

[90] A. Majidian, T. Martin, and M. Cintra. *Fuzzy Formal Concept Analysis and Algorithm*. Jan. 2011. Pages: 7.

[91] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, July 2008. ISBN 978-0-511-80907-1. doi: 10.1017/CBO9780511809071. URL https://www.cambridge.org/highereducation/books/introduction-to-information-retrieval/669D108D20F556C5C30957D63B5AB65C. Publisher: Cambridge University Press.

[92] A. Margara and T. Rabl. Definition of Data Streams. In S. Sakr and A. Y. Zomaya, editors, *Encyclopedia of Big Data Technologies*, pages 648–652. Springer International Publishing, Cham, 2019. ISBN 978-3-319-77525-8. doi: 10.1007/978-3-319-77525-8_188. URL https://doi.org/10.1007/978-3-319-77525-8_188.

[93] B. Martin and P. Eklund. Asymmetric Page Split Generalized Index Search Trees for Formal Concept Analysis. In F. Esposito, Z. W. Raś, D. Malerba, and G. Semeraro, editors, *Foundations of Intelligent Systems*, Lecture Notes in Computer Science, pages 218–227, Berlin, Heidelberg, 2006. Springer. ISBN 978-3-540-45766-4. doi: 10.1007/11875604_25.

[94] B. Martin and P. Eklund. Spatial indexing for scalability in FCA. In *Proceedings of the 4th international conference on Formal Concept Analysis*, ICFCA'06, pages 205–220, Berlin, Heidelberg, Feb. 2006. Springer-Verlag. ISBN 978-3-540-32203-0. doi: 10.1007/11671404_14. URL https://doi.org/10.1007/11671404_14.

[95] T. Martin, G. Francoeur, and P. Valtchev. CICLAD: A Fast and Memory-efficient Closed Itemset Miner for Streams. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1810–1818, Aug. 2020. doi: 10.1145/3394486.3403232. URL http://arxiv.org/abs/2007.01946. arXiv:2007.01946 [cs, stat].

[96] V. Martynovich and E. Vityaev. *Recovering Noisy Contexts with Probabilistic Formal Concepts*. Oct. 2016.

[97] N. R. Mashhadi, M. Jalali, and M. V. Jahan. Inference of mobile users'social relationships using Bayesian belief network. In *2015 International Congress on Technology, Communication and Knowledge (ICTCK)*, pages 232–240, Nov. 2015. doi: 10.1109/ICTCK.2015.7582676.

[98] A. Mouakher and S. Ben Yahia. QualityCover: Efficient binary relation coverage guided by induced knowledge quality. *Information Sciences*, 355-356:58–73, Aug. 2016. ISSN 0020-0255. doi: 10.1016/j.ins.2016.03.009. URL https://www.sciencedirect.com/science/article/pii/S0020025516301566.

[99] A. Mucherino, P. J. Papajorgji, and P. M. Pardalos. k-Nearest Neighbor Classification. In A. Mucherino, P. J. Papajorgji, and P. M. Pardalos, editors, *Data Mining in Agriculture*, pages 83–106. Springer, New York, NY, 2009. ISBN 978-0-387-88615-2. doi: 10.1007/978-0-387-88615-2_4. URL https://doi.org/10.1007/978-0-387-88615-2_4.

[100] B. Nebel. Logics for Knowledge Representation. In J. D. Wright, editor, *International Encyclopedia of the Social & Behavioral Sciences (Second Edition)*, pages 319–321. Elsevier, Oxford, Jan. 2015. ISBN 978-0-08-097087-5. doi: 10.1016/B978-0-08-097086-8.43053-9. URL https://www.sciencedirect.com/science/article/pii/B9780080970868430539.

[101] E. Nenova, D. I. Ignatov, and A. V. Konstantinov. An FCA-based Boolean Matrix Factorisation for Collaborative Filtering, Oct. 2013. URL http://arxiv.org/abs/1310.4366. arXiv:1310.4366 [cs, stat].

[102] R. Neouchi, A. Y. Tawfik, and R. A. Frost. Towards a Temporal Extension of Formal Concept analysis. In E. Stroulia and S. Matwin, editors, *Advances in Artificial Intelligence*, Lecture Notes in Computer Science, pages 335–344, Berlin, Heidelberg, 2001. Springer. ISBN 978-3-540-45153-2. doi: 10.1007/3-540-45153-6_33.

[103] E. Norris. An algorithm for computing the maximal rectangles in a binary relation. *Revue Roumaine de Mathématiques Pures et Appliquées*, 23, Jan. 1978.

[104] L. Nourine and O. Raynaud. A fast algorithm for building lattices. *Information Processing Letters*, 71(5-6):199 – 204, Sept. 1999. doi: 10.1016/S0020-0190(99)00108-8. URL https://hal.archives-ouvertes.fr/hal-01765512. Publisher: Elsevier.

[105] K. Petersen and C. Gencel. Worldviews, Research Methods, and their Relationship to Validity in Empirical Software Engineering Research. In *2013 Joint Conference of the 23rd International Workshop on Software Measurement and the 8th International Conference on Software Process and Product Measurement*, pages 81–89, Oct. 2013. doi: 10.1109/IWSM-Mensura.2013.22.

[106] K. Petersen, S. Vakkalanka, and L. Kuzniarz. Guidelines for conducting systematic mapping studies in software engineering: An update. *Information and Software Technology*, 64:1–18, Aug. 2015. ISSN 0950-5849. doi: 10.1016/j.infsof.2015.03.007. URL https://www.sciencedirect.com/science/article/pii/S0950584915000646.

[107] Q. Ping, Z. Zhongxiang, G. Hualing, and W. Ju. Attribute Exploration Algorithms on Ontology Construction. In Z. Shi, S. Vadera, A. Aamodt, and D. Leake, editors, *Intelligent Information Processing V*, IFIP Advances in Information and Communication Technology, pages 234–244, Berlin, Heidelberg, 2010. Springer. ISBN 978-3-642-16327-2. doi: 10.1007/978-3-642-16327-2_29.

[108] N. Prat, I. Comyn-Wattiau, and J. Akoka. A Taxonomy of Evaluation Methods for Information Systems Artifacts. *Journal of Management Information Systems*, 32 (3):229–267, July 2015. ISSN 0742-1222, 1557-928X. doi: 10.1080/07421222.2015.1099390. URL http://www.tandfonline.com/doi/full/10.1080/07421222.2015.1099390.

[109] T. T. Quan, L. N. Ngo, and S. C. Hui. An Effective Clustering-based Approach for Conceptual Association Rules Mining. In *2009 IEEE-RIVF International Conference on Computing and Communication Technologies*, pages 1–7, July 2009. doi: 10.1109/RIVF.2009.5174619.

[110] P. Robinson and J. Lowe. Literature reviews vs systematic reviews. *Australian and New Zealand Journal of Public Health*, 39(2):103–103, 2015. ISSN 1753-6405. doi: 10.1111/1753-6405.12393. URL https://onlinelibrary.wiley.com/doi/abs/10.1111/1753-6405.12393. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/1753-6405.12393.

[111] J. M. Rodriguez-Jimenez, P. Cordero, M. Enciso, and S. Rudolph. Concept lattices with negative information: A characterization theorem. *Information Sciences*, 369: 51–62, Nov. 2016. ISSN 0020-0255. doi: 10.1016/j.ins.2016.06.015. URL https://www.sciencedirect.com/science/article/pii/S0020025516304364.

[112] M. Rouane-Hacene, M. Huchard, A. Napoli, and P. Valtchev. Relational Concept Analysis: Mining Concept Lattices From Multi-Relational Data. *Annals of Mathematics and Artificial Intelligence*, 67, Jan. 2013. doi: 10.1007/s10472-012-9329-3.

[113] D. Seid and S. Mehrotra. Efficient relationship pattern mining using multi-relational iceberg-cubes. In *Fourth IEEE International Conference on Data Mining (ICDM'04)*, pages 515–518, Nov. 2004. doi: 10.1109/ICDM.2004.10059.

[114] H. Seki, Y. Honda, and S. Nagano. On Enumerating Frequent Closed Patterns with Key in Multi-relational Data. In B. Pfahringer, G. Holmes, and A. Hoffmann, editors, *Discovery Science*, Lecture Notes in Computer Science, pages 72–86, Berlin, Heidelberg, 2010. Springer. ISBN 978-3-642-16184-1. doi: 10.1007/978-3-642-16184-1_6.

[115] Y. She, W. Wang, X. He, Y. Du, and Y. Liu. A three-valued logic approach to partially known formal concepts. *Journal of Intelligent & Fuzzy Systems*, 37(2):3053–3064, Jan. 2019. ISSN 1064-1246. doi: 10.3233/JIFS-190111. URL https://content.iospress.com/articles/journal-of-intelligent-and-fuzzy-systems/ifs190111. Publisher: IOS Press.

[116] Z. F. Siddiqui, E. Tiakas, P. Symeonidis, M. Spiliopoulou, and Y. Manolopoulos. xStreams: Recommending Items to Users with Time-evolving Preferences. In *Proceedings of the 4th International Conference on Web Intelligence, Mining and Semantics (WIMS14)*, WIMS '14, pages 1–12, New York, NY, USA, June 2014. Association

for Computing Machinery. ISBN 978-1-4503-2538-7. doi: 10.1145/2611040.2611051. URL https://doi.org/10.1145/2611040.2611051.

[117] M. Sipser. *Introduction to the theory of computation*. Cengage Learning, Australia Brazil Japan Korea Mexiko Singapore Spain United Kingdom United States, third edition, international edition edition, 2013. ISBN 978-1-133-18779-0 978-1-133-18781-3 978-0-357-67058-3.

[118] J. Sowa. *Conceptual Structures: Information Processing in Mind and Machine The Systems Programming Series*. Jan. 1984.

[119] J. F. Sowa. *Knowledge representation: logical, philosophical and computational foundations*. Brooks/Cole Publishing Co., USA, Sept. 1999. ISBN 978-0-534-94965-5.

[120] W. Stallings. *Computer organization and architecture: designing for performance*. Pearson-Prentice Hall, Boston, tenth edition edition, 2016. ISBN 978-0-13-410161-3.

[121] G. Stumme, R. Taouil, Y. Bastide, N. Pasquier, and L. Lakhal. Computing iceberg concept lattices with Titanic. *Data & Knowledge Engineering*, 42(2):189–222, Aug. 2002. ISSN 0169023X. doi: 10.1016/S0169-023X(02)00057-5. URL https://linkinghub.elsevier.com/retrieve/pii/S0169023X02000575.

[122] W. Tabisz, M. Jovanovic, and F. Lee. Present and future of distributed power systems. In *[Proceedings] APEC '92 Seventh Annual Applied Power Electronics Conference and Exposition*, pages 11–18, Feb. 1992. doi: 10.1109/APEC.1992.228437.

[123] M. Tasnim, D. Collarana, D. Graux, and M.-E. Vidal. Chapter 8 Context-Based Entity Matching for Big Data. In V. Janev, D. Graux, H. Jabeen, and E. Sallinger, editors, *Knowledge Graphs and Big Data Processing*, Lecture Notes in Computer Science, pages 122–146. Springer International Publishing, Cham, 2020. ISBN 978-3-030-53199-7. doi: 10.1007/978-3-030-53199-7_8. URL https://doi.org/10.1007/978-3-030-53199-7_8.

[124] A. G. Touzi. Towards a Discovering Knowledge Comprehensible and Exploitable by the End-User. In *2010 Second International Conference on Advances in Databases, Knowledge, and Data Applications*, pages 126–134, Apr. 2010. doi: 10.1109/DBKDA.2010.36.

[125] M. D. Tran, C. d'Amato, B. T. Nguyen, and A. G. B. Tettamanzi. An evolutionary algorithm for discovering multi-relational association rules in the semantic web. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '17, pages 513–520, New York, NY, USA, July 2017. Association for Computing Machinery. ISBN 978-1-4503-4920-8. doi: 10.1145/3071178.3079196. URL https://doi.org/10.1145/3071178.3079196.

[126] I. Tuomi. Data is more than knowledge: Implications of the reversed knowledge hierarchy for knowledge management and organizational memory. *Journal of Management Information Systems*, 16:103–117, Jan. 1999. doi: 10.1080/07421222.1999.11518258.

[127] J. Unbehauen, S. Hellmann, S. Auer, and C. Stadler. Knowledge Extraction from Structured Sources. In S. Ceri and M. Brambilla, editors, *Search Computing: Broadening Web Search*, Lecture Notes in Computer Science, pages 34–52. Springer, Berlin, Heidelberg, 2012. ISBN 978-3-642-34213-4. doi: 10.1007/978-3-642-34213-4_3. URL https://doi.org/10.1007/978-3-642-34213-4_3.

[128] A. V S and A. S. Extracting Conceptual Relationships and Inducing Concept Lattices from Unstructured Text. *Journal of Intelligent Systems*, 28, Jan. 2017. doi: 10.1515/jisys-2017-0225.

[129] P. Valtchev and R. Missaoui. Building Concept (Galois) Lattices from Parts: Generalizing the Incremental Methods. In H. S. Delugach and G. Stumme, editors, *Conceptual Structures: Broadening the Base*, Lecture Notes in Computer Science, pages 290–303, Berlin, Heidelberg, 2001. Springer. ISBN 978-3-540-44583-8. doi: 10.1007/3-540-44583-8_21.

[130] P. Valtchev, R. Missaoui, and P. Lebrun. A partition-based approach towards constructing Galois (concept) lattices. *Discrete Mathematics*, 256(3):801–829, Oct. 2002. ISSN 0012-365X. doi: 10.1016/S0012-365X(02)00349-7. URL https://www.sciencedirect.com/science/article/pii/S0012365X02003497.

[131] C. R. Valêncio, F. T. Oyama, P. Scarpelini Neto, A. C. Colombini, A. M. Cansian, R. C. G. de Souza, and P. L. P. Corrêa. MR-Radix: a multi-relational data mining algorithm. *Human-centric Computing and Information Sciences*, 2(1):4, Mar. 2012. ISSN 2192-1962. doi: 10.1186/2192-1962-2-4. URL https://doi.org/10.1186/2192-1962-2-4.

[132] D. van der Merwe, S. Obiedkov, and D. Kourie. AddIntent: A New Incremental Algorithm for Constructing Concept Lattices. In P. Eklund, editor, *Concept Lattices*, Lecture Notes in Computer Science, pages 372–385, Berlin, Heidelberg, 2004. Springer. ISBN 978-3-540-24651-0. doi: 10.1007/978-3-540-24651-0_31.

[133] M. van Steen and A. S. Tanenbaum. A brief introduction to distributed systems. *Computing*, 98(10):967–1009, Oct. 2016. ISSN 1436-5057. doi: 10.1007/s00607-016-0508-7. URL https://doi.org/10.1007/s00607-016-0508-7.

[134] V. Vychodil. Computing sets of graded attribute implications with witnessed non-redundancy. *Information Sciences*, 351:90–100, July 2016. ISSN 00200255. doi: 10.1016/j.ins.2016.03.004. URL http://arxiv.org/abs/1511.01640. arXiv:1511.01640 [cs].

[135] M. Wajnberg, M. Lezoche, A. Blondin Masse, P. Valtchev, H. Panetto, and L. Tyvaert. Semantic interoperability of large systems through a formal method: Relational Concept Analysis. In *16th IFAC Symposium on Information Control Problems in Manufacturing, INCOM 2018*, volume 51 of *IFAC-Papersonline*, pages 1397–1402, Bergamo, Italy, June 2018. doi: 10.1016/j.ifacol.2018.08.330. URL https://hal.archives-ouvertes.fr/hal-01813398. Issue: 11.

[136] X. Wei, W. Liang, Q. Chen, and X. Li. A calculation method of concept similarity base on inclusion degree theory. In *2010 2nd IEEE International Conference on Information Management and Engineering*, pages 501–505, Apr. 2010. doi: 10.1109/ICIME.2010.5477960.

[137] R. Wille. Restructuring Lattice Theory: An Approach Based on Hierarchies of Concepts. In I. Rival, editor, *Ordered Sets*, NATO Advanced Study Institutes Series, pages 445–470, Dordrecht, 1982. Springer Netherlands. ISBN 978-94-009-7798-3. doi: 10.1007/978-94-009-7798-3_15.

[138] K. E. Wolff. Temporal Concept Analysis. In *ICCS-2001 International Workshop on Concept Lattices-Based Theory, Methods and Tools for Knowledge Discovery in Databases, Stanford University, Palo Alto (CA)*, pages 91–107, 2001.

[139] X. Wu, J. Zhang, and R. Lu. Attribute Logic Formula Description of Granule and Its Application to Build Concept Lattice. *IEEE Access*, 8:12592–12606, 2020. ISSN 2169-3536. doi: 10.1109/ACCESS.2020.2964834. Conference Name: IEEE Access.

[140] J. Xiao and Z. He. A Concept Lattice for Semantic Integration of Geo-Ontologies Based on Weight of Inclusion Degree Importance and Information Entropy. *Entropy*, 18:399, Nov. 2016. doi: 10.3390/e18110399.

[141] B. Xu, R. de Fréin, E. Robson, and M. Ó Foghlú. Distributed Formal Concept Analysis Algorithms Based on an Iterative MapReduce Framework. In F. Domenach, D. I. Ignatov, and J. Poelmans, editors, *Formal Concept Analysis*, Lecture Notes in Computer Science, pages 292–308, Berlin, Heidelberg, 2012. Springer. ISBN 978-3-642-29892-9. doi: 10.1007/978-3-642-29892-9_26.

[142] A. Yang, W. Zhang, J. Wang, K. Yang, Y. Han, and L. Zhang. Review on the Application of Machine Learning Algorithms in the Sequence Data Mining of DNA. *Frontiers in Bioengineering and Biotechnology*, 8, 2020. ISSN 2296-4185. URL https://www.frontiersin.org/articles/10.3389/fbioe.2020.01032.

[143] G. Ye, Z. Tang, H. Wang, D. Fang, J. Fang, S. Huang, and Z. Wang. Deep Program Structure Modeling Through Multi-Relational Graph-based Learning. In *Proceedings of the ACM International Conference on Parallel Architectures and Compilation Techniques*, PACT '20, pages 111–123, New York, NY, USA, Sept. 2020. Association for Computing Machinery. ISBN 978-1-4503-8075-1. doi: 10.1145/3410463.3414670. URL https://doi.org/10.1145/3410463.3414670.

[144] X. Yin, J. Han, J. Yang, and P. S. Yu. CrossMine: Efficient Classification Across Multiple Database Relations. In J.-F. Boulicaut, L. De Raedt, and H. Mannila, editors, *Constraint-Based Mining and Inductive Databases*, Lecture Notes in Computer Science, pages 172–195, Berlin, Heidelberg, 2006. Springer. ISBN 978-3-540-31351-9. doi: 10.1007/11615576_9.

[145] W. Zhang. Mining Multi-Level Multi-Relational Frequent Patterns Based on Conjunctive Query Containment. In *2009 WRI Global Congress on Intelligent Systems*, volume 2, pages 436–440, May 2009. doi: 10.1109/GCIS.2009.290. ISSN: 2155-6091.

[146] W. Zhang. Multi-Relational Data Mining Based on Higher-Order Inductive Logic Programming. In *2009 WRI Global Congress on Intelligent Systems*, volume 2, pages 453–458, May 2009. doi: 10.1109/GCIS.2009.289. ISSN: 2155-6091.

[147] D. Zhao and X. Liu. A Genetic K-means Membrane Algorithm for Multi-relational Data Clustering. page 959, Jan. 2016. ISBN 978-3-319-31853-0. doi: 10.1007/978-3-319-31854-7_106.

[148] C. Zou, D. Zhang, J. Wan, M. M. Hassan, and J. Lloret. Using Concept Lattice for Personalized Recommendation System Design. *IEEE Systems Journal*, 11(1):305–314, Mar. 2017. ISSN 1937-9234. doi: 10.1109/JSYST.2015.2457244. URL https://ieeexplore.ieee.org/document/7177069. Conference Name: IEEE Systems Journal.

[149] L. Zou, Z. Zhang, J. Long, and H. Zhang. A fast incremental algorithm for deleting objects from a concept lattice. *Knowledge-Based Systems*, 89:411–419, Nov. 2015. ISSN 0950-7051. doi: 10.1016/j.knosys.2015.07.022. URL https://www.sciencedirect.com/science/article/pii/S0950705115002762.