Communauté
UNIVERSITÉ Grenoble Alpes

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE GRENOBLE

Spécialité : **Informatique**

Arrêté ministérial : 7 Août 2006

Présentée par

Martin Kirchgessner

Thèse dirigée par **Sihem Amer-Yahia**
et co-encadrée par **Vincent Leroy**

préparée au sein du **Laboratoire d'Informatique de Grenoble**
et de **l'École Doctorale Mathématiques, Sciences et Technologies de l'Information, Informatique**

Fouille et classement d'ensembles fermés dans des données transactionnelles de grande échelle

Thèse soutenue publiquement le ,
devant le jury composé de :

M. Noël De Palma
Professeur à l'Université Grenoble Alpes, Président
Mme. Laure Berti-Equille
Senior Scientist at Qatar Computing Research Institute, Rapporteur
M. Florent Masseglia
Chargé de Recherche INRIA, Rapporteur
Mme. Céline Robardet
Professeur à l'INSA de Lyon, Examinatrice
M. Renaud Lachaize
Maître de conférences à l'Université Grenoble Alpes, Examinateur
M. Alexandre Termier
Professeur à l'Université de Rennes 1, Examinateur
Mme. Sihem Amer-Yahia
Directrice de recherche CNRS, Directeur de thèse
M. Vincent Leroy
Maître de conférences à l'Université Grenoble Alpes, Co-Encadrant de thèse

Abstract

Mining and ranking closed itemsets from large-scale transactional datasets

Keywords Data mining, Big Data, Parallel systems, Association rules, Quality measures

The recent increase of data volumes raises new challenges for itemset mining algorithms. In this thesis, we focus on transactional datasets (collections of items sets, for example supermarket tickets) containing at least a million transactions over hundreds of thousands items. These datasets usually follow a "long tail" distribution: a few items are very frequent, and most items appear rarely. Such distributions are often truncated by existing itemset mining algorithms, whose results concern only a very small portion of the available items (the most frequents, usually). Thus, existing methods fail to concisely provide relevant insights on large datasets. We therefore introduce a new semantics which is more intuitive for the analyst: browsing associations per item, for any item, and less than a hundred associations at once.

To address the items' coverage challenge, our first contribution is the item-centric mining problem. It consists in computing, for each item in the dataset, the k most frequent closed itemsets containing this item. We present an algorithm to solve it, TopPI. We show that TopPI computes efficiently interesting results over our datasets, outperforming simpler solutions or emulations based on existing algorithms, both in terms of run-time and result completeness. We also show and empirically validate how TopPI can be parallelized, on multi-core machines and on Hadoop clusters, in order to speed-up computation on large scale datasets.

Our second contribution is CAPA, a framework allowing us to study which existing measures of association rules' quality are relevant to rank results. This concerns results obtained from TopPI or from jLCM, our implementation of a state-of-the-art frequent closed itemsets mining algorithm (LCM). Our quantitative study shows that the 39 quality measures we compare can be grouped into 5 families, based on the similarity of the rankings they produce. We also involve marketing experts in a qualitative study, in order to discover which of the 5 families we propose highlights the most interesting associations for their domain.

Our close collaboration with Intermarché, one of our industrial partners in the Datalyse project, allows us to show extensive experiments on real, nation-wide supermarket data. We present a complete analytics workflow addressing this use case. We also experiment on Web data. Our contributions can be relevant in various other fields, thanks to the genericity of transactional datasets.

Altogether our contributions allow analysts to discover associations of interest in modern datasets. We pave the way for a more reactive discovery of items' associations in large-scale datasets, whether on highly dynamic data or for interactive exploration systems.

Résumé

Mots-clés Fouille de données, Grandes masses de données, Systèmes parallèles, Règles d'association, Mesures de qualité

Les algorithmes actuels pour la fouille d'ensembles fréquents sont dépassés par l'augmentation des volumes de données. Dans cette thèse nous nous intéressons plus particulièrement aux données transactionnelles (des collections d'ensembles d'objets, par exemple des tickets de caisse) qui contiennent au moins un million de transactions portant sur au moins des centaines de milliers d'objets. Les jeux de données de cette taille suivent généralement une distribution dite en "longue traine": alors que quelques objets sont très fréquents, la plupart sont rares. Ces distributions sont le plus souvent tronquées par les algorithmes de fouille d'ensembles fréquents, dont les résultats ne portent que sur une infime partie des objets disponibles (les plus fréquents). Les méthodes existantes ne permettent donc pas de découvrir des associations concises et pertinentes au sein d'un grand jeu de données. Nous proposons donc une nouvelle sémantique, plus intuitive pour l'analyste: parcourir les associations *par objet*, au plus une centaine à la fois, et ce *pour chaque objet* présent dans les données.

Afin de parvenir à couvrir tous les objets, notre première contribution consiste à définir la *fouille centrée sur les objets*. Cela consiste à calculer, pour chaque objet trouvé dans les données, les k ensembles d'objets les plus fréquents qui le contiennent. Nous présentons un algorithme effectuant ce calcul, TopPI. Nous montrons que TopPI calcule efficacement des résultats intéressants sur nos jeux de données. Il est plus performant que des solutions naives ou des émulations reposant sur des algorithmes existants, aussi bien en termes de rapidité que de complétude des résultats. Nous décrivons et expérimentons deux versions parallèles de TopPI (l'une sur des machines multi-coeurs, l'autre sur des grappes Hadoop) qui permettent d'accélerer le calcul à grande échelle.

Notre seconde contribution est CAPA, un système permettant d'étudier quelle mesure de qualité des règles d'association serait la plus appropriée pour trier nos résultats. Cela s'applique aussi bien aux résultats issus de TopPI que de jLCM, notre implémentation d'un algorithme récent de fouille d'ensembles fréquents fermés (LCM). Notre étude quantitative montre que les 39 mesures que nous comparons peuvent être regroupées en 5 familles, d'après la similarité des classements de règles qu'elles produisent. Nous invitons aussi des experts en marketing à participer à une étude qualitative, afin de déterminer laquelle des 5 familles que nous proposons met en avant les associations d'objets les plus pertinentes dans leur domaine.

Notre collaboration avec Intermarché, partenaire industriel dans le cadre du projet Datalyse, nous permet de présenter des expériences complètes et portant sur des données réelles issues de supermarchés dans toute la France. Nous décrivons un flux d'analyse complet, à même de répondre à cette application. Nous présentons également des expériences portant sur des données issues d'Internet; grâce à la généricité du modèle des ensembles d'objets, nos contributions peuvent s'appliquer dans d'autres domaines.

Nos contributions permettent donc aux analystes de découvrir des associations

d'objets au milieu de grandes masses de données. Nos travaux ouvrent aussi la voie vers la fouille d'associations interactive à large échelle, afin d'analyser des données hautement dynamiques ou de réduire la portion du fichier à analyser à celle qui intéresse le plus l'analyste.

Remerciements

Je tiens tout d'abord à remercier mes parents pour leur indéfectible soutien ces 29 dernières années. Merci à Sihem Amer-Yahia et Vincent Leroy pour leur investissement tout au long de ma thèse, ainsi qu'à Alexandre Termier et Marie-Christine Rousset pour m'avoir mis le pied à l'étrier scientifique. Merci enfin à Etienne Millon de m'avoir relu et aidé à dresser LaTeX.
Beaucoup d'autres ont contribué de manière aussi indispensable qu'indirecte aux travaux qui suivent ; la *mixtape* p.97 vous est dédiée.

Thèse effectuée au LIG dans le cadre du *programme d'investissement d'avenir* Datalyse. Manuscrit transmis aux rapporteurs le Jeudi 2 Juin 2016.

Contents

CHAPTER 1

Introduction

In 1960, the introduction of the IBM 1401 manual [25] celebrates the apparition of magnetic storage, replacing unit-record (punched cards):

> "As the volume of data to be processed increases, as the decision-making process is refined to the point where it requires more and more information, as the time available for decision-making becomes shorter, unit-record equipment continues to offer economies and advantages, but not at the same rate of improvement in time-saving and dollar-saving. The next available step in the mechanization process takes the businessman into intermediate and large-scale data processing."

More than 50 years later, the only outdated idea in this paragraph is "unit-record equipment". Business and academic analysts are still using automated methods to extract knowledge and insights from large collections of low-level records.

Now coined "big data", the activity of "large-scale data processing" is still ongoing, and at larger and larger scales. A characteristic feature of the big data movement is to record any information passing by, which quickly generates millions or billions of records, or more. This is pushed by historically low storage costs, currently counted in *cents* per gigabyte, and the availability of various parallel systems that solve the size, throughput and reliability issues. Data is not labelled "big" because of its absolute size — as our reference to the IBM 1401 manual suggests, unit scales evolve quickly. We can instead consider that "big data" refers to "data whose size forces us to look beyond the tried-and-true methods that are prevalent at that time" [28]. That leads us to the present work.

1.1 Association rules mining in the big data era

Among the various ways to automatically extract knowledge from data [57], we focus on association rules, as introduced by Agrawal and Srikant [3] for supermarket tickets analysis. Association rules allow to discover which sets of products are frequently associated together in customers' baskets. Before formalizing association rules, we illustrate their principle on the following tickets:

Ticket 1: bread, butter, noodles, pesto sauce, frozen shrimps

Ticket 2: eggs, cheese, noodles

Ticket 3: bread, butter, strawberry jam, eggs, shampoo

Ticket 4: bread, butter, wine

Ticket 5: bread, cereals, milk

Ticket 6: bread, butter, strawberry jam, orange juice

Ticket 7: bread, butter, maple syrup, flour

We can observe that *bread* is almost always bought with *butter*. This is represented by the association rule $bread \rightarrow butter$. In two tickets, *bread* and *butter* also appear with *strawberry jam*. These yield another association rule: $bread \rightarrow \{butter, strawberry\ jam\}$.

Following the model of Agrawal and Srikant, we call a ticket a *transaction*, and a product an *item*. A transaction is a set of items. Many kinds of data can fit into this model. For instance, we also use data crawled from *LastFM*, a music streaming website. We can read on each user's public profile the list of her favorite artists. By considering that an item represents an artist, we create a transaction from each user profile. Thus, each transaction represents a user's favorite artists:

User 1: The Beatles, Red Hot Chili Pepper, Linkin Park

User 2: Red Hot Chili Pepper, Radiohead, The White Stripes

User 3: The Beatles, Red Hot Chili Pepper, Muse

User 4: Daft Punk, Air

User 5: The Beatles, Red Hot Chili Pepper, Britney Spears

User 6: The Beatles, Muse, Coldplay

User 7: The Beatles, Red Hot Chili Pepper, Muse, Led Zeppelin

From these transactions we discover that users listening to *The Beatles* also listen to *Red Hot Chili Pepper*, or to *Muse* and *Red Hot Chili Pepper*. Those are represented respectively by the association rules *The Beatles* $\rightarrow$ *Red Hot Chili Pepper* and *The Beatles* $\rightarrow$ *{Muse, Red Hot Chili Pepper}*. An association rule is

derived from two *itemsets*. The first one is the left side of the association, the other is the union of all items in the association. For example, our last example comes from the discovery of transactions containing *{The Beatles}* and *{The Beatles, Muse, Red Hot Chili Pepper}*. For brevity we do not mention here all the rules that can be generated from our examples.

Our example associations result from the mining of *frequent itemsets* — *ie.* sets of items which frequently appear together in the input transactions. They intuitively represent how items are usually associated. Over the past twenty years, various frequent itemset mining algorithms have been proposed and applied successfully on such datasets to uncover associations [7].

The number of potential combinations is a well-known combinatorial problem. Indeed, given n items we have 2^n possible itemsets. With a million artists or products, this combinatorial explosion may overwhelm our computers' capacity. When analyzing nation-wide supermarket purchases, or world-wide playlists, existing methods thus show two important limitations.

Firstly, state-of-the-art algorithms cannot grasp so many items and their combinations. Usually most items are filtered out, whether beforehand by the analyst or during the computation according to a parameter. Hence existing algorithms provide insights, but only about a minority of items. In order to explore less frequent items' associations, the analyst has to guide manually the mining — a tedious task, at large scale.

The second limitation is the readability of results. Even if some algorithms are able to mine data at this scale, the resulting associations are too numerous. Filtering these results has the same drawback as filtering before mining: the analyst firstly has to guess what to filter, and will likely miss interesting associations. Another solution is to sort the resulting associations: many quality measures have been proposed and can be used to sort itemsets or association rules [17, 36]. However these measures were not designed for datasets containing millions of transactions, and existing studies cannot help the analyst to choose a measure that will reliably rank interesting associations among top results.

1.2 Scope of this work

In this thesis we are interested in the extraction of an easy-to-browse collection of interesting association rules from large-scale transactional datasets, that is millions of transactions over hundreds of thousands or millions items. Our notion of interest does not rely on external data or knowledge: interest should be based on the combined items' distributions alone.

We address two use cases, which respectively overcome the limitations mentioned in Section 1.1. The first one corresponds to the early analysis on an itemset. In this case we should facilitate the navigation in all parts of the data, *ie.* be able to provide associations about all items. The second one is more targeted. When the analyst is able to specify which items she is interested in, we should compute and select only the most interesting associations. In both cases our results are presented directly to the analyst; we assume she will not consult more than a hundred associations at once.

We present sets of products found in supermarket receipts, demographic attributes associated to various product categories, and sets of artists found in music playlists. But the same method can also find which pages are consulted consecutively on a website, which set of words usually appear in a text, etc.

To tackle the computational difficulties raised by the number of possible associations, we leverage the parallelism of the systems storing the large datasets we are interested in. We parallelize both on shared-memory systems and on distributed systems.

1.3 Technology transfer to Datalyse

This thesis is funded by the DATALYSE project[1], a french consortium involving 4 computer science laboratories and 3 companies. Those partners cover a wide scope of data-intensive applications: collection, storage, analysis, or presentation. This work belongs to the user data analytics branch of the project.

Our main partner in this project is Groupement des Mousquetaires, a major retailer funded in 1969 in France and now developed across Europe and various markets, from hardware stores to car centers. We collaborate more particularly with the business intelligence and marketing departments of Intermarché, which is their main and first brand, specialized in generalist retail stores. Intermarché is a franchise-based network of almost 2000 stores across France.

This collaboration allow us to run experiments with real data, but also to benefit from business use cases and deploy our algorithms on a production system. Our contributions have been provided early on to Intermarché, allowing us to take into account their feedback in different iterations of our work.

1.4 Overview of this thesis and contributions

The rest of this manuscript is organized as follows:

- In **Chapter 2** we start by detailing the data model we use for association rules mining. Then we review existing algorithms, and motivate our implementation of a state-of-the-art itemsets miner as an open source library, *j*LCM[2]. This chapter also presents related work on selecting high-quality associations and parallelism for large-scale mining.

- **Chapter 3** depicts our experimental and production platforms. We firstly show the complete analytics workflow of the DATALYSE project, in which our contributions are integrated. This system creates 3 of the 5 datasets used in our experiments. We conclude by discussing which characteristics make these datasets relevant to our study.

[1] `http://datalyse.fr/`

[2] `https://github.com/slide-lig/jlcm`

- **Chapter 4** presents our first contribution, TopPI. It implements the item-centric mining semantics, whose goal is to find the k most frequent itemsets containing each item in the input dataset. We show that TopPI provides interesting results in reasonable time, which can be reduced further when parallelized on a high-end server or a commodity cluster.

- **Chapter 5** presents our second contribution: a framework comparing quality measures for association rules, CAPA (**C**omparative **A**nalysis of **PA**tterns). A first quantitative comparison allows us to group the 39 measures we implemented in 5 families based on the similarity of the rule rankings they produce. Then we involve marketing experts in a user study, in order to point out which measure or family of measures highlights the most interesting association rules for the retail domain.

- We conclude in **Chapter 6** by showing how our two contributions can be combined to create an organized and concise selection of association rules. We finally describe open perspectives for future work in association rules mining.

CHAPTER 2

Related Work

Contents

Association rules mining is widely covered by existing work. Before presenting our datasets of interest and detailing why state-of-the-art solutions are not sufficient for such data, we start by reviewing in this chapter related algorithms and methods. As association rules are generated from itemsets, this chapter mostly covers itemset mining.

Our core heritage from existing work are notions and models for itemset and association rules mining. These are presented in Section 2.1 along LCM, our algorithm of choice for closed itemsets enumeration. We review more completely algorithms for itemsets mining in Section 2.2. Some literature proposes to mine itemsets in parallel, in order to speed up the computation or to tackle larger datasets. This precisely echoes our needs, hence we discuss these propositions in Section 2.3.

Most of this work focuses on frequent itemsets, yet frequency is not always a good indicator of interest. Therefore we also discuss, in Section 2.4, which quality measure can provide alternatives and how they can be integrated in the analytics process. Reviewing existing algorithms also shows the importance of the underlying structures and their implementation. In practice, however, fast implementations are not published or difficult to re-use. We illustrate this in Section 2.5, and show that our concerns on this topic led us to maintain jLCM, our implementation of LCM, as an open-source project. The following chapters re-use jLCM itself, or some of its components.

We conclude in Section 2.6 by distinguishing what we can borrow to existing work and what should be improved.

TID	Transaction
t_0	$\{0, 1, 2\}$
t_1	$\{0, 1, 2\}$
t_2	$\{0, 1\}$
t_3	$\{2, 3\}$
t_4	$\{0, 3\}$

Table 2.1: An example input $\mathcal{D}$. Transaction identifiers (first column) are indexes necessary to the storage system. In this dataset, the itemset $\{1, 2\}$ has a support equal to 2 and $closure(\{1, 2\}) = \{0, 1, 2\}$. The itemset $\{3\}$ is closed and has a support equal to 2. The projected dataset $\mathcal{D}[\{3\}]$ contains transactions t_3 and t_4.

2.1 Mining itemsets and association rules

We will see in the next chapter that receipt records can be transformed in various ways, depending on the desired associations' semantics. Though, in all cases data has to fit a common model to be processed by our itemset mining algorithms. We use Agrawal's model for association rules mining in transactional databases [3], with the restriction to closed itemsets [51].

In this section we start by defining this model and notions, which will be used through the rest of the manuscript. Then we focus on LCM, an efficient and parallelizable closed frequent itemsets mining algorithm [64]. CAPA uses LCM at the mining step, and TopPI inherited from LCM its basic enumeration principles.

2.1.1 A model for itemset and association rules mining

Mining itemsets requires the definition of a ground set of items, $\mathcal{I}$. A *transaction*, usually denoted T, is a subset of $\mathcal{I}$. Our mining algorithms' input is a transactional dataset $\mathcal{D}$, *i.e.* a collection of transactions. An *itemset*, denoted I, is also subset of $\mathcal{I}$; we introduce another term in order to distinguish the input and the output of mining algorithms. In mining programs and itemsets collections, items are represented as integers, as the example in Table 2.1.

Abstracting the notion of item allows us to apply the same algorithms on datasets carrying various semantics. For example, if items are products, then transactions may represent the receipt given to the customer after his purchase, like $\{bread, butter, gratedcheese, noodles, cola\}$. In this case frequent itemsets (defined below) represent sets of products that are frequently purchased together, *e.g.* $\{bread, butter\}$. In the following we will also consider a case where $\mathcal{I}$ is the union of demographic attributes' values and product categories. Then, each receipt record may be converted into a transaction, like $\{< 35, Male, $ Ile-de-France, $Paris, cola\}$. In that case itemsets of interest, for instance $\{< 35, Sodas\}$, would represent how a customer segment is akin to buy products from one of our taxonomy's categories.

Given an itemset I, a transaction t is an *occurrence* of I if $I \subseteq t$. The number of occurrences of an itemset in $\mathcal{D}$ is called its *support*, denoted $support_{\mathcal{D}}(I)$. When it's clear from the context we may omit $\mathcal{D}$ and note the support as $support(I)$. An itemset I is said to be *closed* if there exists no itemset $I' \supseteq I$ such that $support(I) =$

$support(I')$. The number of closed itemsets, further abbreviated as *CIS*, is less important than the number of itemsets. Though, CIS provide the same amount of information on $\mathcal{D}$ [51]. For example, if $support(\{2,5,8\}) = support(\{2,5\})$, then the latter can be dismissed. Several algorithms therefore extract closed itemsets only, in order to improve performance and avoid redundant results [52, 65].

Definition 1 (Closure)**.** *The greatest itemset $I' \supseteq I$ having the same support as I is called the* closure *of I, further denoted as $closure(I)$.*

The *projected dataset* for an itemset I on a dataset $\mathcal{D}$ is the collection of occurrences of I: $\mathcal{D}[I] = \langle T \in \mathcal{D} \mid I \subseteq T \rangle$. To further reduce its size, we always remove all items of I, yielding the *reduced dataset*: $\mathcal{D}_I = \langle T \setminus I \mid T \in \mathcal{D} \wedge I \subseteq T \rangle$. Note that $support_{\mathcal{D}}(I) = support_{\mathcal{D}[I]}(I) = |\mathcal{D}_I|$. Table 2.1 shows a dataset and some closure examples.

In this work, we will mine closed itemsets in transactional datasets. That is, we will find all closed itemsets I matching a given criteria in a dataset $\mathcal{D}$, along with each $support_{\mathcal{D}}(I)$. The motivation of this computation is the generation of association rules [1]:

Definition 2 (Association rule)**.** *An association rule is an implication of the form $A \to B$, where $A \subseteq \mathcal{I}$, $B \subseteq \mathcal{I}$ and $A \cap B = \emptyset$. A is the rule's* antecedent *and B its* consequent*.*

The computation of $support_{\mathcal{D}}(A)$ and $support_{\mathcal{D}}(A \cup B)$ gives to the analyst a quick intuition of how the purchase of product(s) A leads to the purchase of product(s) B, for example. In another dataset (*LastFM*, presented in Section 3.3), a transaction is a set of artists listened to by a single user. Then an association $\{a, b\} \to \{c, d\}$ represents how many listeners of artists a and b are also listeners of c and d.

2.1.2 Enumerating frequent closed itemsets with LCM

Given a frequency threshold ε, an itemset I is said to be *frequent* in a transactions set $\mathcal{D}$ if $support_{\mathcal{D}}(I) \geq \varepsilon$. In this manuscript, we generally use absolute values for our frequency threshold, whereas most of the literature on frequent CIS mining uses relative thresholds. Indeed, marketing analysts involved in the Datalyse project are used to absolute measures of their customer base. For example, they may state they are interested in trends involving at least 1000 customers. Such number is much more intuitive to them than its equivalent relative threshold in our tickets dataset: 0.0003%.

Among existing frequent CIS mining algorithms, we firstly present LCMv2 [64], awarded the best implementation of the second workshop on Frequent Itemset Mining Implementations [30]. It was also shown by Négrevergne *et al.* that LCM can be efficiently parallelized on multi-core machines [50]. As we are counting transactions and items in millions and targeting a distributed platform, this potential for parallelization was a strong motivation when choosing LCM as our initial itemset miner.

LCM is a backtracking algorithm relying on two fundamental properties: the *closure extension*, that generates new closed itemsets from previously computed

ones, and the *first parent* that avoids redundant computation. We define these principles below.

Definition 3. *An itemset $J \subseteq \mathcal{I}$ is a closure extension of a closed itemset $I \subseteq \mathcal{I}$ if $\exists e \notin I$, called an* extension item*, such that $J = closure(\{e\} \cup I)$.*

LCM enumerates CIS by recursively performing closure extensions, starting from the empty set.

In Table 2.1 (p.8), $\{0, 1, 2\}$ is a closure extension of both $\{0, 1\}$ and $\{2\}$. This example shows that a new itemset can be generated by two different closure extensions. Uno *et al.* [64] introduced two principles which guarantee that each closed itemset is traversed only once in the exploration. First, closure extensions are restricted to *prefix extensions*: only items smaller than the previous extension are allowed. Furthermore, we prune extensions that do not satisfy the *first-parent* criterion:

Definition 4. *Given a closed itemset I and an item $e \notin I$, $\langle e, I\rangle$ is the first parent of $J = closure(\{e\} \cup I)$ iff. $max(J \setminus I) = e$.*

Algorithm 1: LCM

```
Data: dataset D, minimum support threshold ε
Result: Outputs all frequent closed itemsets in 𝒟
1  begin
2      ⊥_closed ← ⋂_{T∈𝒟} T
3      output ⊥_closed
4      foreach i ∈ ℐ | i ∉ ⊥_closed do
5          expand(⊥_closed, i, 𝒟, ε)
6  Function expand(I, i, 𝒟_I, ε)
       Data: Closed frequent itemset I, extension item i, reduced dataset D_I,
             minimum support threshold ε
       Result: Outputs all closed itemsets containing {i} ∪ I
7      begin
8          if support_{𝒟_I}({i}) ≥ ε then                    // Frequency test
9              I_ext ← ⋂_{T∈𝒟_I[{i}]} T                  // Closure computation
10             if maxItem(I_ext) = i then                   // 1st parent test
11                 J ← I ∪ I_ext
12                 output (J, support_{𝒟_I}({i}))
13                 D_J = {T \ J | T ∈ 𝒟_I[{i}]}
14                 foreach j ∈ ℐ \ J | j < i do              // Augmentations
15                     expand(J, j, 𝒟_J, ε)
```

Algorithm 1 summarizes the main function of LCM. The extension enumeration order and the first parent test shapes the closure extensions lattice as a tree. This is illustrated in Figure 2.1, which shows the itemsets tree for the dataset in Table 2.1. $\langle 1, \{2\}\rangle$ is the first parent of $\{0, 1, 2\}$, but $\langle 0, \{2\}\rangle$ is not. Therefore the branch led by $\langle 0, \{2\}\rangle$ is pruned.

These enumeration principles also lead to the following property:

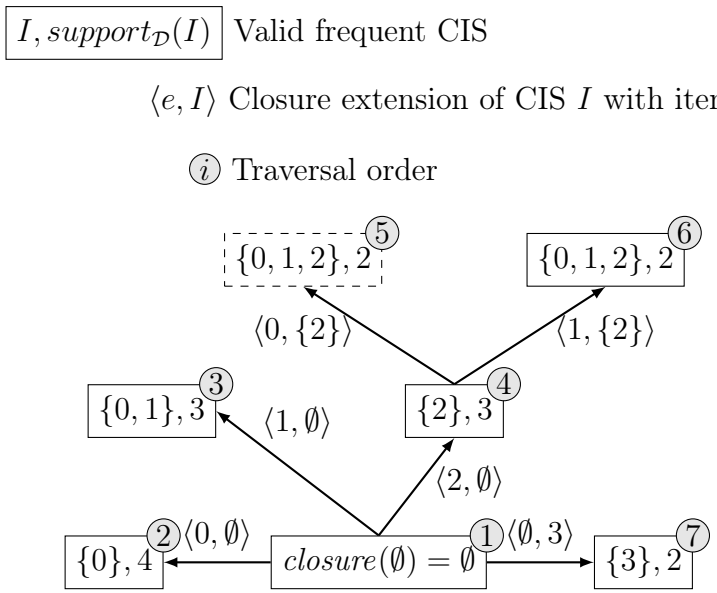


Figure 2.1: Frequent CIS enumeration tree on our example dataset (Table 2.1), with $\varepsilon = 2$. $\langle e, I \rangle$ denotes the closure extension operation.

Property 1. *By extending I with e, LCM can only recursively generate itemsets J such that $max(J) = e$.*

This property allows us, for any CIS, to know where it will be outputted in the enumeration. This is fundamental, both for the parallelization of the enumeration, and to prune the CIS tree as in TopPI (see Chapter 4).

Figure 2.1 also mentions in which order CIS are enumerated in LCM. We now unroll this example enumeration:

1. The algorithm starts by checking if any item in $\mathcal{D}$ appears in all transactions. Such items belong to the empty itemset's closure, $closure(\emptyset)$. As in most cases, here $closure(\emptyset)$ is empty. The empty set is therefore used as the initial itemset. It is extended by each frequent item in $\mathcal{D}$, called *starter items*.

2. The first starter is 0, and $\{0\}$ is a closed itemset so the algorithm outputs it. As 0 is the smallest item, no recursion happens.

3. The next starter is 1, which generates by closure the itemset $\{0, 1\}$. The closure satisfies the first parent test thus $\{0, 1\}$ is outputted. The only remaining item in $\mathcal{D}_{\{0,1\}}$ is 2, but it's greater than the previous extension item so no recursion happens.

4. The following starter is 2, leading to a closed singleton, $\{2\}$. Two frequent items, smaller than 2, remain in $\mathcal{D}_{\{2\}}$ so the enumerator performs recursive extensions of $\{2\}$.

5. The algorithm first extends $\{2\}$ with 0 which, by closure, generates the pattern $\{0, 1, 2\}$. However, $max(\{0, 1, 2\} \setminus \{2\}) = 1 > 0$, so $\langle 0, \{2\} \rangle$ is not the first parent: this exploration branch is aborted.

6. The next extension of $\{2\}$ is 1, which also closes to $\{0, 1, 2\}$. But this one satisfies the first-parent test so this CIS is outputted at this step. No further extension can happen because $\mathcal{D}_{\{0,1,2\}}$ only contains empty transactions. $\{2\}$ has no more extensions, so the algorithm backtracks to the starters.

7. The last starter item is 3. $\{3\}$ is closed and valid, but no recursion happens because no frequent item remains in $\mathcal{D}_{\{3\}}$.

2.2 Itemset mining algorithms

LCM is not the first frequent CIS mining algorithm proposed in the literature; we now review other itemsets mining algorithms and discuss if they are relevant to mine large-scale datasets or retail data analysis. We classify frequent itemsets mining algorithms into two families: those defining frequency with a minimum support threshold, like LCM, and those limiting the results set's size. We start by presenting these two families. We finally mention other definitions of the results set, showing that these do not satisfy the requirements of marketing studies.

2.2.1 Threshold-based frequent itemset mining algorithms

The first frequent itemset mining algorithm is APriori [1, 3]. Given a minimum frequency threshold ε and a transactional dataset $\mathcal{D}$, it computes all itemsets I such that $support_{\mathcal{D}}(I) \geq \varepsilon$.

APriori introduced the generate-and-test approach, which consists in iteratively generating a set of candidate itemsets, then computing their support in the dataset. The test phase of each iteration requires a pass on the complete dataset, and candidates satisfying the frequency test are outputted progressively. In its simplest variant, APriori generates and tests itemsets of length k during the k-th iteration. The authors prove that the number of frequent itemsets decreases with their length k, therefore APriori terminates when no more frequent itemsets are found. This is a corollary of the *anti-monotony property*, also known as the *downward closure property*:

Property 2 (Anti-monotony)**.** *Given two itemsets I and J, if $I \supset J$ then $support_{\mathcal{D}}(I) \leq support_{\mathcal{D}}(J)$.*

This property is leveraged by all frequent itemsets mining algorithms. The generate-and-test approach is however less popular, firstly because each test phase requires a complete scan of the dataset. The second limitation is the number of candidates generated: in some cases most of them will not be frequent, and are therefore useless. Both issues are particularly acute on our datasets, where the number of items combinations can exhaust our machines' capacity, and finding sequentially the support of many itemsets in gigabytes of transactions takes minutes, or even hours.

The first algorithm both reading the dataset once and enumerating only frequent itemsets is Eclat [78, 74]. Another algorithm having the same properties

is FP-Growth [23, 22], who gained more popularity due to its internal representation of datasets. Both algorithms avoid the generation of candidate itemsets by traversing the itemsets lattice, similarly to our example of Figure 2.1. They also rely on succinct representations of the dataset, which inspired the reduced datasets used in LCM and defined in Section 2.1.

FP-Growth starts by reading the dataset, and stores all transactions in a prefix-tree. Coupled with an indexing of items by decreasing frequency, this results in a compact representation because the prefixes containing very frequent items are merged in a few nodes of the tree. FP-Growth then traverse the itemset lattice, and generates a projected tree for each. Prefix-trees are also efficient when used as itemsets indexes [41].

Pasquier et al. identified *closed* itemsets [51], which are typically one order of magnitude less numerous, while conveying the same information. This definition paved the way for algorithms which, as LCM, enumerate only closed itemsets to improve their efficiency. The corresponding variants of FP-Growth are CLOSET [52] and its optimized version, CLOSET+ [71].

On our datasets, however, the instantiation of prefix trees is not amortized, because between 90% and 99% of transactions are unique (we completely present our datasets in Section 3.3). This is close to prefix trees' worst case. This phenomena worsens when the tree's nodes also store references to their supporting transactions, as in the *Itemset-Tidset Search Tree* at the heart of the CHARM algorithm [76, 75].

The inefficiency of prefix-trees over our datasets motivated our choice of LCM as our initial CIS miner. Its structures' efficiency is even more relevant in a multi-threaded setting, as discussed in Section 2.3.1. We however do not implement the most complex variants of datasets in LCM, namely the complete prefix tree of LCMv3 [66] and zero-suppressed binary decision diagrams [26]. These are more relevant with dense datasets.

Overall this evolution of CIS mining algorithms shows the importance of the underlying data structures, and their implementation. For example, switching from node and pointers representations to array-based tries can speedup FP-Growth by a factor of 10 [54].

2.2.2 Top-k frequent itemset mining algorithms

When confronted to a new dataset, guessing a relevant minimum frequency threshold is not easy and usually requires a few attempts. To overcome this usability problem, some algorithms propose to instead ask the analyst how many itemsets she wants, thus replacing the threshold ε with a parameter k. The algorithm then computes the k most frequent itemsets in the dataset. We refer to this approach as *global* top-k, because a maximum of k itemsets are mined for the whole dataset, as opposed to the *per-item* top-k itemsets we propose in Chapter 4.

Top-k processing is a well studied research area in the domain of databases [27]. A common case is the problem of top-k query processing, in which the goal is to identify the k highest ranking items according to a query and a ranking function. Efficient top-k ranking algorithms are designed to identify the k highest scoring results without having to compute an exact score for every single item. For in-

stance, the TA algorithm [14] compares the lowest score in the current version of the top-k with an upper bound on the score of items that have not been processed yet. The algorithm stops when it is sure that no more unseen item will have a score higher than the k-th score so far. It does so by comparing the dynamically maintained upper bound with the k-th score. This early termination significantly reduces processing time while guaranteeing the same result as when reading all solutions. The two algorithms we present in this section, and the TopPI algorithm presented in Chapter 4, rely on an analogous early termination strategy.

The computation of top-k-frequent itemsets was firstly solved by Han *et al.* with the TFP algorithm [24, 70]. Many of the most frequent itemsets in a dataset are actually singleton itemsets, which are not revealing associations between items. Hence TFP introduces another parameter, the minimum itemset length l_{min}. Internally TFP relies on a frequent itemsets enumeration analogous to FP-Growth, but TFP dynamically adjusts its frequency threshold depending on the support of the k best current results. Our preliminary experiments show that TFP is memory-consuming, in particular when we release minimal length constraint, *i.e.* when $l_{min} = 0$. Indeed, in this case the exploration strategy is almost greedy and TFP cannot prune prefix trees.

This was also observed by Chuang *et al.*, who proposed the MTK algorithm to circumvent this problem [9]. MTK does not implement the minimum itemsets length constraint, nor it relies on an FP-Growth-like enumeration of itemsets. Instead MTK requires the user to define a memory usage limit and uses a generate-and-test strategy. The memory limit is converted into an upper bound on the number of candidate itemsets to be generated and tested per database scan. Because the number of candidates of length i may not exhaust the memory capacity, the authors also propose the δ-stair search. It consists in evaluating candidates of δ different lengths at each generate-and-test pass. The δ-stair search is more efficient in the MTK_Close variant of the algorithm, which mines the top-k-frequent CIS. Indeed a non-closed itemset of length i may have a closure of length $i+1$ or $i+2$. As TFP, MTK finds during the exploration the minimum frequency ensuring the correctness of its results.

In Section 4.4.3 we need a global top-k miner to build one of TopPI's baselines; as the generate-and-test approach of MTK is inefficient on our datasets, we instead implemented TFP with an additional optimization which compensates our release of the minimal length constraint.

2.2.3 Limitations of frequent CIS miners

Although it is more convenient to define the output by its size rather than with a frequency threshold, it does not change the order in which itemsets are considered by the algorithm. Global top-k algorithms are unable to compute frequent itemsets for rare items, as this would require generating billions itemsets containing common items first. In their experiments, the authors of the MTK algorithm set k in $[100, 100000]$. TFP is also tested on such large values of k, when $l_{min} = 0$. Overall, whether they are using a threshold or top-k approach, frequent itemset mining algorithms return results containing a negligible proportion of the available items. Typically, results appearing less than 1000 or 10,000 times are filtered out.

Hence frequent itemset mining hides the variety of large-scale datasets.

But frequency is not the only statistic able to characterize an itemset or an association rule. We will see in Section 2.4.1 that many quality measures have been proposed. Although some of them can be integrated in an itemset mining algorithm, on large-scale datasets this is prohibitive because it requires the enumeration of too many itemsets. For the same reason, restricting the results set to itemsets that satisfy user-specified constraints, as CAP [49], is not feasible in our setting.

To overcome the overwhelming amount of results typically returned by itemset miners, recent work defined highly-specialized classes of itemsets and proposed algorithms to mine them. These classes of itemsets are often referred to as *concise representations*. Closed itemsets were the first of such concise representations, and is lossless. Surprisingly, this can be a downside: CIS are often too detailed for large-scale datasets. Among the proposed classes we notice the *most informative* itemsets [44, 45] and the *non-derivable itemsets* [6]. Similarly, the KRIMP algorithm proposes to mine the itemsets that are the most efficient to compress the dataset [69].

Overall these methods are not relevant in the retail domain, because analysts may be interested in subtle variations of associations. For example, if *{bread, butter, salt}*, *{bread, butter, salt, noodles}* and *{bread, butter, salt, rice}* have very similar supports, then they might be considered as redundant by one of the previous approaches, who will output at most one of them. But their supports' similarity is itself an important information. This motivated our focus on *frequent* itemset mining. Although the vast majority of items are not covered by the results, existing work provide efficient and reasonably complex routines and structures for frequent CIS enumeration. We leverage these in the next chapters in order to solve this detail-concision dilemma.

2.3 Parallel approaches

Because of the important computation times incurred by itemset mining, parallelization has been considered early on by the community. We start by mentioning existing work on shared-memory systems, which are similar to the sequential algorithms described previously. Then we present distributed mining algorithms, focusing only on clusters of servers. We do not review work on mining data distributed geographically, across organizations or systems [32], as it is not relevant in our application context.

2.3.1 On shared-memory systems

All algorithms relying on a traversal of the CIS space can assign distinct portions of this space to different execution threads, using properties analogous to our Property 1 (p.11). Therefore most algorithms mentioned in Section 2.2 can leverage shared-memory systems to speed up the enumeration [77, 42, 73].

Gothing *et al.* studied the memory accesses behavior and performance of prefix-tree-based frequent itemset mining algorithms in [18]. Their experiments

show the cache-inefficiency of these structures, even though they use the fastest known public implementations at that time, obtained thanks to the FIMI repository [21]. For example they measure L3 hit rates below 40%, and less than 10% of CPU utilization. They overcome these bottlenecks by developing a *tile-able cache-conscious prefix tree*, in which sub-trees are organized as tiles that fit in cache pages. This is particularly compliant with cache pre-fetching strategies implemented by modern processors. Their resulting implementation is up to five times faster, and more importantly shows excellent speedups on multi-core and multi-processor systems. The importance of cache-efficiency for multi-threaded itemset mining was also identified by PLCM's authors [50].

PLCM shows almost ideal speedups using simpler data structures (transactions concatenated in an array). These are straightforward and CPU cache-friendly, thus we used PLCM as a reference when implementing jLCM, as shown in Section 2.5.2.

2.3.2 On clusters of servers or commodity machines

The first distributed itemset mining algorithms follow APriori [8, 2]. Both distribute the generate-and-test approach by distinguishing *local* candidates sets from the *global* set of frequent itemsets — algorithms differ on whether candidates or transactions (or both) are exchanged across nodes. As we discussed in Section 2.2.1, in our case the generate-and-test approach would hit the combinatorial explosion of itemsets. This would be aggravated by shuffling candidate itemsets across the network.

At the time of writing the MapReduce framework is widely available for distributed computing over commodity computers [12]. It has been popularized by the open-source implementation Hadoop [59], and the more recent Spark [72]. Both ensure reliably the distributed execution of user-defined functions, and manage internally the data locality. In MapReduce, data is considered as a collection of key-value pairs. A *job* is the distributed execution of two functions: `map` and `reduce`. At first, `map`:$(K, V) \to \langle K', V' \rangle$ is executed on each couple of the input data. It can return an arbitrary number of couples, which are not necessarily of the same type as the input's. The system then groups the resulting couples by intermediate key. For each distinct intermediate key (and its corresponding values), the system executes `reduce`:$(K', \langle V' \rangle) \to \langle K'', V'' \rangle$, whose results are written to the job's output file. The execution of `reduce` is also distributed. The execution system is coupled with a distributed storage, hence the `map` function is (ideally) executed locally, on a machine storing a portion of the input couples. This strategy allows a theoretically perfect parallelization of the `map` execution, and accounts for the important scalability of the framework.

The Spark framework provides a wider array of transformations of the dataset, and does not constrain the programmer to a succession of `map` and `reduce` executions. It can implement the MapReduce framework but, as opposed to Hadoop, in Spark intermediate data is not necessarily written to disk. Its extensive use of in-memory structures (particularly beneficial to iterative algorithms) and a simpler, flexible API account for the recent popularity of Spark. This however has a negligible impact on itemset mining programs, who often need a fixed number of jobs. Whether with Spark or Hadoop, the major time-consumers remain mining tasks,

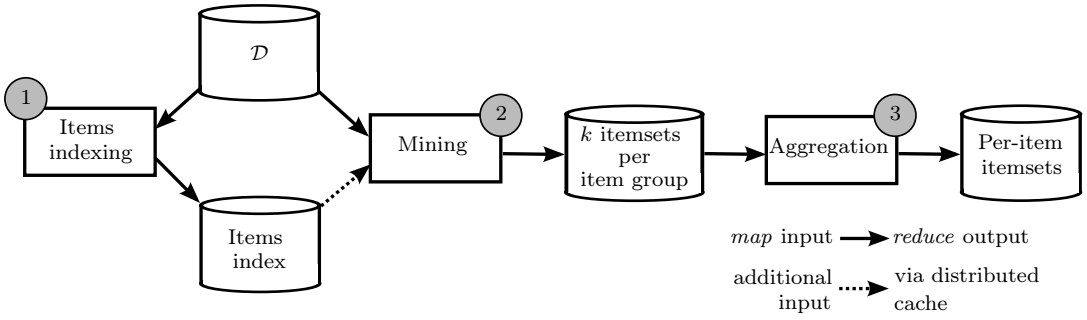


Figure 2.2: PFP's data flow

therefore our distributed algorithms and experiments have not been migrated to Spark.

The academic and industrial success of these two systems motivated the distribution of itemsets mining algorithms over the MapReduce framework. Lin *et al.* developed a MapReduce implementation of Apriori [38]. Moens *et al.* also proposed two algorithms inspired by Eclat [48]. Although these parallel algorithms can handle large-scale datasets, they inherit the initial drawback of frequent itemset mining: most items do not appear in results. For example, even items occurring 10,000 times are usually filtered out in [48].

PFP, the first itemset mining algorithm proposed for MapReduce overcomes the problem of items coverage; we now detail it more particularly because we use it as a baseline for TopPI's Hadoop version (in Section 4.4.4, p.58).

2.3.3 PFP: Parallel FP-Growth

Parallel FP-Growth [37] (PFP) proposes a slightly different computation: for each frequent item in the dataset, PFP mines at most k itemsets containing this item. Their notion of frequent item is unusually large, as they show examples supported by only 6 transactions out of 16 millions. Thus PFP can provide itemsets covering the majority of the available items.

Along the dataset $\mathcal{D}$, PFP has 3 parameters: a minimum frequency threshold ε, a maximum number of itemsets to return per item, k, and a number of groups $\mathcal{G}$. Each group corresponds to an independent mining task, and these tasks are distributed in the cluster. As depicted by Figure 2.2, the execution of PFP is divided in 3 MapReduce jobs. The complete algorithm unfolds as follows:

Job ① PFP counts the support of individual items in the dataset. Then it orders them by decreasing frequency, and assigns each one to a group in a round-robin fashion. Thus the most frequent item is assigned to group 1, the second most frequent to group 2, the $\mathcal{G}+1$ most frequent to group 1, etc. This ensures that the $\mathcal{G}$ most frequent items are dispatched to distinct groups, ensuring the balancing of the next job.

Job ② In this job, group identifiers are used as intermediate keys. The `map` function copies each transaction to each group its items belong to. Therefore

the `reduce` function processing an items group g has enough transactions to mine all the itemsets containing items from g, and more particularly the frequent itemsets I such that $max(I)$ belongs to g [1]. As item groups are dispatched among workers, this ensures a distribution of the itemsets space in the cluster. Mining relies on FP-Growth [23], presented in Section 2.2.1, but each `reduce` function only mines the k most frequent itemsets in its group's transactions.

Job ③ Items are the intermediate key of this MapReduce job, which processes the results of the mining job. For each itemset I, the `map` function outputs a couple $(i, I), \forall i \in I$. Thus each `reduce` functions holds, for the given item i, all itemsets containing i discovered by the previous job. Each execution of `reduce` only outputs the k most frequent itemsets, so the output file contains at most k itemsets for each item i.

PFP's mining phase has a single top-k heap per group of items, thus it generates at most k itemsets per items group. It is therefore unlikely to obtain the required k itemsets for an item with a low frequency because, during the mining, itemsets corresponding to more frequent items of its group will fill the heap. PFP also does not ensure that itemsets found for each item are the most frequent, nor that they are closed. We discuss in Section 4.4.4 how these factors impact the results for rare items, and show experimentally that its coverage of items is sparser than expected.

2.4 Selecting itemsets and association rules

So far we mostly mentioned frequent itemsets mining algorithms, because these are efficient and unveil prominent associations in datasets. But this is not the case when a dataset contains more than a few thousands items. Frequent itemsets usually combine frequent items with each others and, because of the many possible combinations, frequency is often not sufficient to distinguish surprising associations among the numerous frequent itemsets. This is even more pronounced with a million items.

Two approaches can circumvent this problem: we can either rank the discovered association rules using a quality measure borrowed to statistics, or we can restrict the itemsets mining according to such measure. We now review how these two approaches have already been experimented in existing work.

2.4.1 Rule ranking

In order to select less than a hundred association of particular interest, and without additional knowledge like prices or users' ratings, we can only evaluate an association $A \rightarrow B$ by comparing $support(A)$, $support(B)$, $support(A \cup B)$ and the size of the dataset, $|\mathcal{D}|$. This is proposed by extensive work in statistics and

[1] Because of this specification, PFP does not need to copy transactions entirely in the `map` phase. See Section 2.4.1 in [37].

data mining, as summarized in [17]. In this survey, Geng *et al.* review as many as 38 measures for association rules. They also discuss 4 sets of properties like symmetry or monotony, and how each of them highlights different meanings of "rule quality", such as novelty and generality. But this still leaves too many choices for an analyst trying to find a ranking measure.

These 38 measures are also compared in [33]. Authors consider the case of extracting and ranking temporal rules (*event* $A \rightarrow$ *event* B) from the execution traces of Java programs. Each measure is evaluated by its ability to rank highly rules known from a ground truth: the Java library specification. This experimental evaluation allow the authors to recommend one quality measure, the *Odds Ratio* [46], as it is the best at highlighting associations expected according to the ground truth. However we cannot state if this choice is also relevant on retail or Web data.

Another question raised by the number of quality measures available is: do they rank really differently? This is all the more relevant as most of them are designed to filter according to a user-defined threshold (as frequent itemset miners do with ε). This is observed by HERBS [36, 67], which relies on a different and smaller set of measures than [33]. In their experiment, authors generate association rules, rank them according to each measure, and compare each pair of rankings using Kendall's τ correlation measure [31]. They finally use these pairwise comparisons to cluster the ranking measures. This clustering distinguishes 4 families of analogous rankings among the 20 measures tested. One of these families contains almost a dozen measures which are very similar to ranking by *confidence*; *i.e.* $support(A \cup B)/support(A)$. They also study the analytical properties of each measure, and observe that measures belonging to the same experimental family often verify the same properties. Though, HERBS' experiments rely on smaller datasets than ours. The datasets used are from the health and astronomy domains, and each of them contains at most 1728 transactions and leads to the extraction of 49 to 6312 rules. As the size of the dataset has an impact on most of the quality measures, it may also impact the similarity of their rankings.

Overall, existing work provides many quality measures for association rules, but does not provide enough material to reliably choose one to rank large-scale retail data. This motivated the development of CAPA, presented in Chapter 5.

2.4.2 Mining statistically significant itemsets

Ranking a collection of association rules supposes, of course, to firstly mine these associations. Those usually result from frequent itemset mining. In order to avoid this costly computation, existing literature proposes another approach: guiding the itemsets space traversal according to a statistical quality measure.

Le Bras *et al.* define in [34] the *General Universal Existential Upward Closure* (GUEUC) which is analogous to the frequency's anti-monotony, and also allows to prune the itemsets lattice. Authors show that 13 quality measures out of the 32 they study verify this property, and integrate these in a generate-and-test mining algorithm. The candidate generation uses the GUEUC to limit the candidate itemsets to those potentially satisfying the score limit defined by the user. As discussed in Section 2.2.1, the generate-and-test approach is however not suitable

to large-scale datasets.

In [40], Liu *et al.* propose to use the p-value (via *Pearson's* χ^2 *test*) to select statistically significant association rules. A low p-value shows a correlation between a rule's antecedent and consequent (left and right terms). Authors also propose an exploration framework where rules are grouped by consequent, then traversed by progressively adding items to the antecedent. The framework provides hints to help the user to guess how each additional item would make a difference. On large-scale datasets, this method raises resource issues because the system should hold the whole dataset in memory during the exploration, and each step may take too much time to meet the requirements of interactive applications. This interactive process is also not suitable to our application case (presented in Chapter 3), as the user would have too much choices among the available items.

Another integration of the p-value in the mining algorithm is proposed by LAMP [47]. It mines a transactional dataset $\mathcal{D}$ where each transaction is annotated positively or negatively. In biology, this annotation may distinguish which persons in the dataset suffer from a disease, for example. Given a maximal p-value threshold α, LAMP will find all itemsets in $\mathcal{D}$ which are significantly correlated to positive annotations, *i.e.* having a p-value smaller than α (using Fisher's exact test). Minato *et al.* show that, given the number of frequent itemsets according to a threshold, we can compute a lower bound on the p-value of itemsets whose support is equal to the threshold. This lower bound is increasing as the threshold decreases, and may therefore exceed α for some value; in this case we hit the greatest threshold guaranteeing that we are not missing any highly-correlated itemset. This however requires to find the number of frequent itemsets in the dataset for each thresholds. LAMP iteratively adjusts a threshold and mines the corresponding frequent CIS, until the threshold converges to a value ensuring the results' correctness.

Both of these works aim at finding highly-correlated itemsets, which requires the analyst to set a threshold on the p-value. This is common practice in biology, but less meaningful in the retail industry where the challenge is instead on ranking association rules. We also observe that these algorithms do not significantly reduce the number of solutions traversed. LAMP is even running LCM multiple times before producing its final results. Thus we consider that closed itemsets should be firstly mined by frequency, then ranked using a user-defined criteria.

2.5 Implementations availability

So far we only discussed algorithms. But we remark that the quality of their implementation has a strong impact on their performance. As we experiment on unusually large transactional datasets, in our case this performance is a condition of usability. In this section we therefore discuss the issue of implementations availability, starting with the case which raised our concerns, PFP. We also present our LCM implementation, jLCM, and explain why we released it as an open-source library.

2.5.1 The need for open source implementations

PFP was firstly available in Mahout, an open-source collection of machine learning algorithms for Hadoop [58]. This implementation has not been maintained in the project, and even deprecated in version 0.9. Later on, it has been re-implemented from scratch in MLlib [61]. This implementation does not match the algorithm described by Li *et al.* [37] because it lacks a k parameter[2]. MLlib thus implements a complete distributed frequent itemset miner, which relies on an original, concise and elegant implementation of FP-Growth in Scala. Our experiments suggest that this is not memory-efficient: mining frequent itemsets from *LastFM* using a frequency threshold of 0.01% fails on our production cluster (see specifications in Sections 3.2.2 and 3.3, p.32). The same task (resulting in 7513 CIS) takes 20 seconds with jLCM, on the author's laptop with 2 threads and a Java heap of 1GB. With MLlib's PFP mining tasks run out of memory, or raise "`GC overhead limit exceeded`" exceptions, typically caused by algorithms instantiating too many small objects. This is coherent with the observations on tree-based mining in [54, 18].

Though, being part of a high-visibility project should allow the community to progressively enhance the implementation, and provides baselines for further research, as ours. Unfortunately such publication is not systematic (yet). For example, for one of our experiments in Section 4.4.3 we had to re-implement TFP [24]. We use the prefix-tree implementation from SPMF [15], another popular collection of mining algorithms.

Goethals and Zaki actively advocated for the publication of datasets and algorithms' source code, by initiating two workshops on Frequent Itemset Mining Implementations [21, 30] and a workshop on Open Source Data Mining. When concluding their report on the latter workshop, Goethals *et al.* express their "hope that the open source implementations of the presented algorithms may help many researchers in the development of their own frequent pattern mining algorithms and implementations." [20].

We share their need for re-usable and efficient building bricks for data mining. Hence we released jLCM, presented below, as a free software. jLCM is easily embeddable as a Java library via the Maven central repository. Because of our regular use, whether for preliminary experiments or in CAPA, we released 10 versions of jLCM[3] including bug-fixing and feature releases. TopPI will be released similarly upon its publication.

2.5.2 *j*LCM: our implementation of LCM

We implemented LCM from scratch and in Java, hence its name: jLCM[4]. The choice of the Java language eases the Hadoop integration (in Chapter 4) and overall clarifies its API, in particular when adding constraints to the exploration (as we do in Chapter 5). Thus jLCM provides an important code base when implementing the following chapters' algorithms. jLCM is actually an implementation of the

[2]Hence we stick to Mahout's implementation when comparing to PFP in Section 4.4.4.

[3]`https://repo1.maven.org/maven2/fr/liglab/jlcm/jLCM/`

[4]`https://github.com/slide-lig/jlcm`

multi-threaded version presented by Négrevergne *et al.*, PLCM [50], with two implementation optimizations adapted to large-scale datasets.

In PLCM, invocations of $expand(\perp_{closed}, i, \mathcal{D}, \varepsilon)$ (Algorithm 1, p.10, line 5) are dispatched to different threads for each item i. Therefore each thread explores a different CIS branch: Property 1 ensures that the threads' explorations are not overlapping. Thus we can speed up the computation on large datasets, by using multi-core CPUs.

LCM was originally designed for mining smaller and denser datasets than ours, with high support thresholds. Two internal operations (also performed by PLCM) have a completely different amortization on our sparse datasets. These are the "fast ppc" (first parent test without closure computation) and "anytime dataset reduction", performed by LCM at each step of the enumeration [65]. jLCM does not always perform these operations.

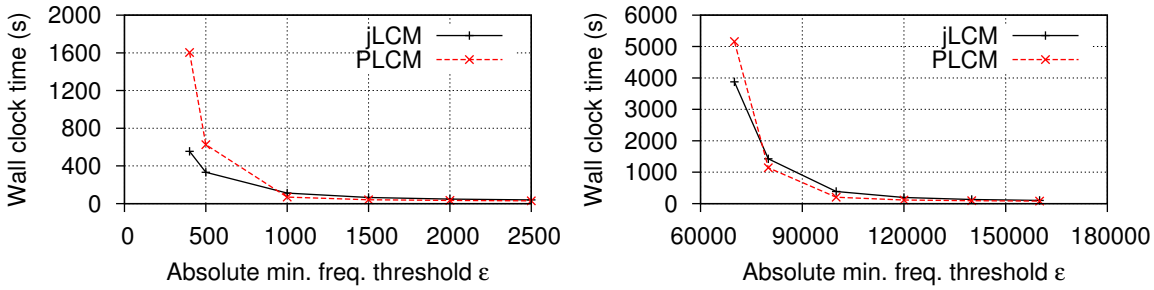
Formally, the items set $\mathcal{I}$ is an ordered set of identifiers. In practice, as jLCM targets sparse datasets, it uses a sparse representation of transactions where a transaction of length n only requires n integers. Our transactions representations leverage two additional optimizations.

- The first one is Dynamic Element Reordering [75]. It is based on the intuition that, when we perform closure extensions with smaller items, it is more memory-efficient to index items from the most frequent to the least frequent, *i.e.* 0 is the most frequent item. Indeed no smaller item exists so it will not lead to a recursive call, thus avoiding the construction of a huge projected dataset (line 13 in Algorithm 1). Item indexes are updated (locally) in projected datasets, thus in $expand(I, i, \mathcal{D}_I, \varepsilon)$ items smaller than i have a greater support (in $\mathcal{D}_I$) than i, and greater items have a smaller support. This heuristic reduces the chances to fail the first-parent test.

- Another important optimization is a switch from Java's `int` (32 bits) to `short` integers (16 bits) when a projected dataset contains less than 2^{15} (32,768) items. As LCM, jLCM progressively reduces datasets by filtering out items which are no longer frequent. This is why recursive calls use the reduced dataset, $\mathcal{D}_I$ (Algorithm 1, line 13). Thus intermediate datasets tend to contain less and less items. Thanks to this phenomena the switch to `short` integers is frequent.

Not only these techniques reduce the global memory consumption, but because they reduce the size of the memory structures in deepest branches, they also improve the CPU's cache hits. This is important in a multi-threaded setting [50].

The indexing by frequency however interferes with the "natural" items indexing from the system in which our algorithms are integrated, and imposes both a conversion of the input dataset and of resulting itemsets. For example, supermarket tickets refer to items with their International Article Number (*i.e.* bar code), a 13 digit number requiring a `long` integer (64 bits), although most extracted datasets would need only a `short`. For a complete and streamlined integration of our work, the transactions store should adopt jLCM's indexation strategy, or keep dataset views which are re-indexed by decreasing frequency.



(a) *LastFM* (b) *WebDocs*

Figure 2.3: jLCM and PLCM run-times w.r.t. frequency threshold ε.

2.5.3 Experimental validation of our implementation

To assess the quality of our LCM implementation, we now compare jLCM against the reference C++ implementation of PLCM [50]. It is, to the best of our knowledge, the current fastest parallel CIS mining algorithm. This benchmark is done on a machine containing 4 Intel Xeon X7560 8-cores CPUs, for a total of 32 cores. This machine has a NUMA architecture: each CPU has a faster access to its closest 16GB memory block, resulting in 64 GB of available RAM.

Running times of both algorithms are displayed on Figures 2.3a and 2.3b, for *LastFM* and *WebDocs* (both presented in Section 3.3). Both algorithms are executed using 32 threads. jLCM's performance is comparable to PLCM for high thresholds (ε) and jLCM tends to be faster as ε decreases. For low values of ε, the dataset reduction can eliminate fewer items, which means that the amount of memory transfers increases. jLCM has better memory management than PLCM, with more compact representations of datasets and NUMA awareness. This increases the benefits of the CPU cache and mitigates the problem of memory bandwidth. Given that jLCM was designed to operate at low support values, this result is encouraging. Nevertheless, as ε decreases, the number of closed frequent itemsets increases exponentially, and so does the execution time.

This experiment validates our implementation: jLCM is comparable to PLCM in terms of run-time. These results also show that traditional itemset mining approaches are unable to generate itemsets for low support items in a reasonable amount of time. With the lowest value of ε used in this experiment, the generated itemsets cover less than 1% of the available items. This phenomena, further discussed in Section 3.3, does not only happen with Web data: we also observe it with our retail datasets.

2.6 Conclusion

In this thesis our main data representation is a transactional dataset, usually denoted $\mathcal{D}$. It is a collection of transactions, which are sets of items belonging to a ground set $\mathcal{I}$. Many kinds of data can fit into this model: a transaction can represent a supermarket ticket, an application user or a photo, where items respectively represent products in a ticket, a user's favorite artists, or tags assigned

to a photo. We detail the different semantics of our datasets in the next chapter.

20 years of literature in the frequent itemsets mining field provides much details and experience on efficient algorithms for itemsets enumeration. Existing work also discusses the underlying datasets representations and algorithms implementations, including on parallel systems. Reviewing this literature led us to choose PLCM for preliminary experiments. We re-implemented it in *j*LCM, released as an open source project (the first release happened on February 2014). As *j*LCM is object-oriented, it provides us with useful and efficient building blocks when implementing our contributions. *j*LCM also backs our first experiments, showing that modern datasets (counting both items and transactions in millions) are not efficiently analyzed with existing itemset mining techniques.

The first problem is that existing algorithms cannot provide the analyst with an overview of the dataset, that is itemsets about all items, especially without requiring cryptic parameters. To the best of our knowledge, PFP is the itemset mining algorithm achieving the best coverage of items. PFP is a MapReduce algorithm, designed for large-scale datasets. Our experiments, however, show that available implementations cannot mine our largest datasets. As clusters of commodity computers relying on Hadoop are increasingly the architecture of choice in academia and in industry, we also consider to port our mining algorithm, TopPI, to this distributed platform.

Another problem is the selection of association rules. Even when filtered by item (answering for example "which sets of products are often associated with bread?"), the associations are often too numerous, or redundant when ranked by decreasing frequency. Many quality measures have been proposed as alternatives to frequency, but most measures are designed to filter associations having a score above a user-defined threshold. Overall we cannot know from existing work how to rank interesting associations in the retail domain, for example.

These two difficulties are respectively addressed by TopPI and CAPA, presented in Chapter 4 and 5. Beforehand, Chapter 3 describes the industrial setting in which TopPI and CAPA are integrated.

CHAPTER 3

Industrial setting

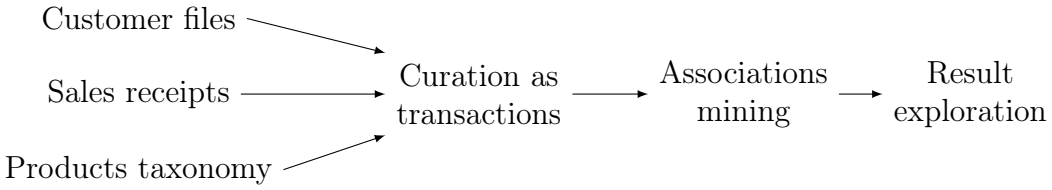
Contents

Our contributions are integrated in the DATALYSE project, more precisely in the sub-project on business intelligence for the retail domain. This part of the project involves 3 companies and our laboratory, collaborating to find precise and relevant insights on customer behavior for marketing experts. These companies are:

- Intermarché, that provides anonymized data and evaluates the results,
- Business & Décision, assisting Intermarché for data acquisition, and
- Eolas, that is hosting the system we present in this chapter.

Our own work aims at solving the challenges raised by the analysis of customer data from a nation-wide retailer like Intermarché. To this end, we propose innovative algorithms and methods in the following chapters. As a preliminary, in this chapter we describe the analytics workflow in which our contributions fit.

In Section 3.1 we give an overview of the data prepared by our partners: customer files, product information and, chiefly, receipts. Our complete analytics pipeline can be summarized as follows:



As we are considering millions of receipts generated by millions of customers, each step requires specific systems to process this data reliably and in a reasonable

amount of time. We specify these in Section 3.2, and illustrate how itemsets can represent various patterns of user behavior by defining 3 mining scenarios representative of marketing analysts' work.

In order to validate the performance of our mining algorithm, TopPI, we include two Web datasets (curated independently) in our collection. We conclude in Section 3.3 by presenting all our datasets of interest, and highlighting their common characteristics that serve as a basis to our work.

Figure 3.1 gives a schematic representation of the system's components. The first module, **acquisition and storage**, performs the classic data warehousing tasks. The **curation** module is used to build transactions, which are then processed by one of our **mining** programs. Finally, the **exploitation** application allows the analyst to interactively explore results.

The complete data mining process relies on 5 steps:

1. Sales records are produced locally at each store, and imported daily into Intermarché's data center.

2. Receipts are stored in a *sales* table, where they can be joined with customer segments coming from the *customers* table. The same database holds the *taxonomy* table.

3. The analyst selects one of our 3 analysis scenario, sets the required parameters (detailed in the corresponding chapters) and optionally defines "targets", which are items of interest. The collection of transactions $\mathcal{D}$ is generated accordingly.

4. The required itemset mining algorithm is executed on $\mathcal{D}$.

5. Resulting itemsets are converted to association rules and loaded in a relational database dedicated to the exploration application.

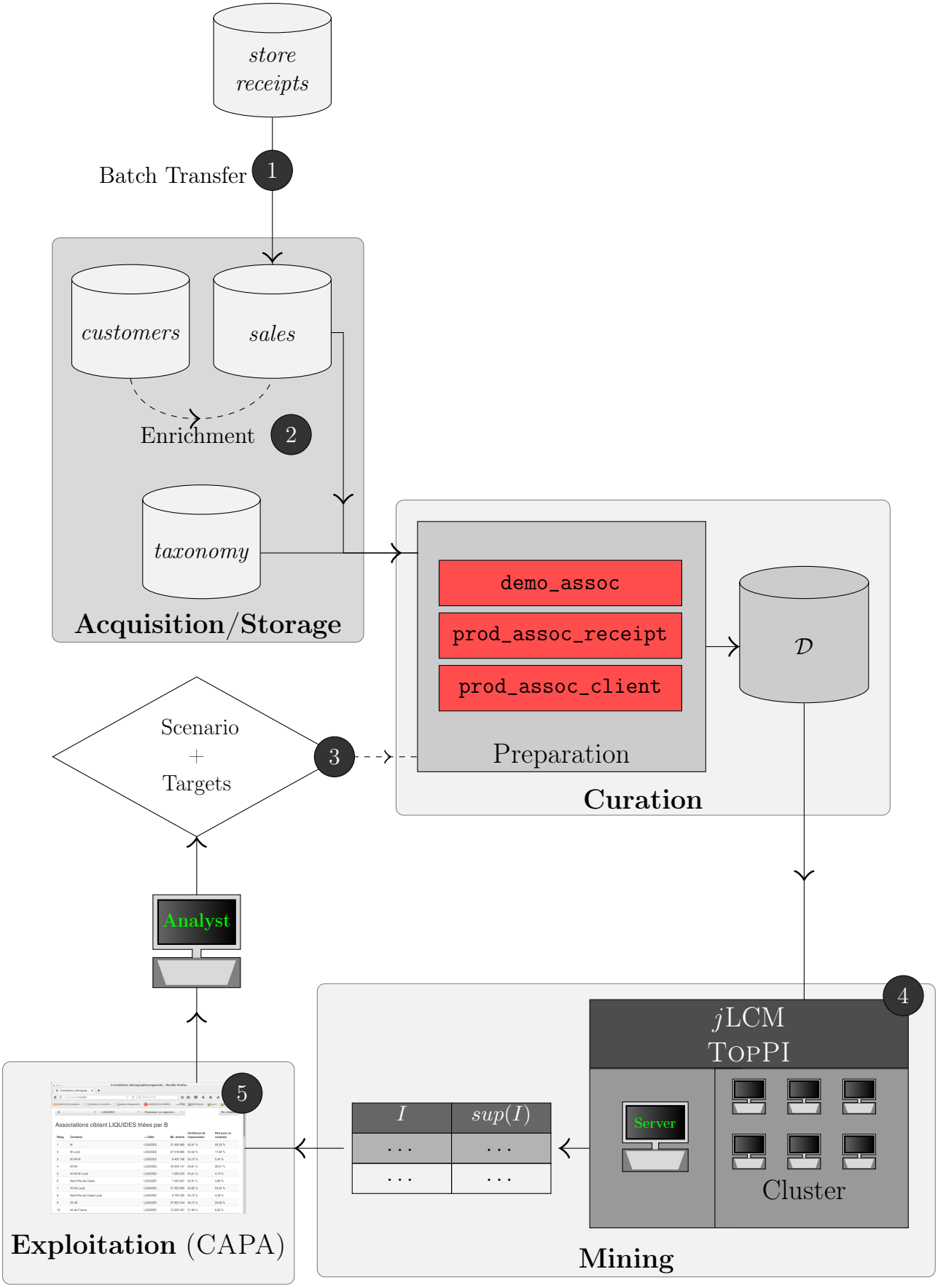


Figure 3.1: Overview of our complete system

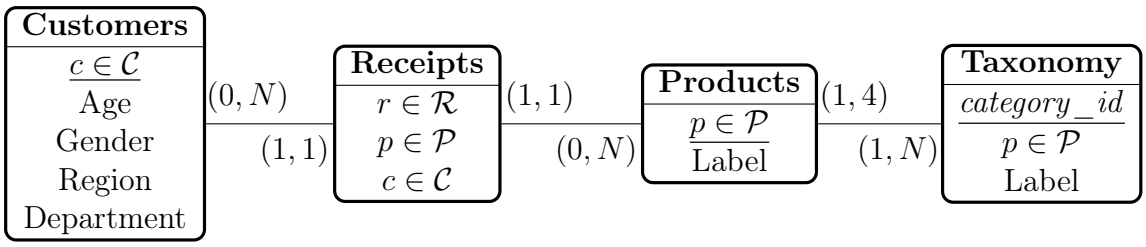


Figure 3.2: Entity-relationship diagram of our input data. Primary keys are underlined.

3.1 Data provided by Datalyse partners

3.1.1 Raw data on receipts, customers and products

The initial collection is a set of records of the form $\langle r, p, c \rangle$, where r is a unique receipt identifier (generated at each purchase) and p is a product purchased by a customer c. When a customer c purchases multiple products at the same time, several records with the same r and c are generated. Figure 3.2 summarizes our input's schema.

The set of receipt identifiers, $\mathcal{R}$, contains over 290 million entries spanning 3.5 billion records, generated by a retail chain consisting of 1884 stores over the whole year 2013. The set of customers, $\mathcal{C}$, contains over 9 million identifiers. Each customer has 4 demographic attributes: *age, gender, region* and *department*. The attribute *age* takes values in *{<35, 35-49, 50-65, >65}*, while *region* and *department* refer to the customer's location in France's administrative divisions (18 regions, each divided in a few departments).

Demographic attributes are used to form customer segments: each segment is described by a set of attribute values interpreted in the usual conjunctive manner. For example, the segment $\{< 35, Paris\}$ refers to young Parisian customers. Given a customer identifier $c \in \mathcal{C}$, the function $demo(c)$ provides its complete record from our *Customers* table. For example, if *Mary* is a 48 years old *female* from the *Calvados* department, then $demo(Mary)$ = *{35-49, female, Normandie, Calvados}*.

The set of products $\mathcal{P}$ contains over 200,000 entries, organized in a taxonomy with 19,557 nodes over 4 levels. Figure 3.3 shows a sample from our taxonomy. Products are leaf nodes, and belong to all their ancestor categories. Given a product p, the set of categories it belongs to is denoted as $cat(p)$. For example, *chocolate cream* belongs to the categories *{Fresh food, Dairy, Ultra fresh, Desserts}*.

3.1.2 Acquisition and storage

Each of the 1884 stores logs locally all customer transactions completed during the day. Whenever a customer checks out, a receipt is generated, indicating the list of products purchased, their price, as well as potential discounts. If the customer has and shows a loyalty card, the card identifier is recorded along its receipt. Otherwise, a unique identifier is generated using a combination of the store identifier

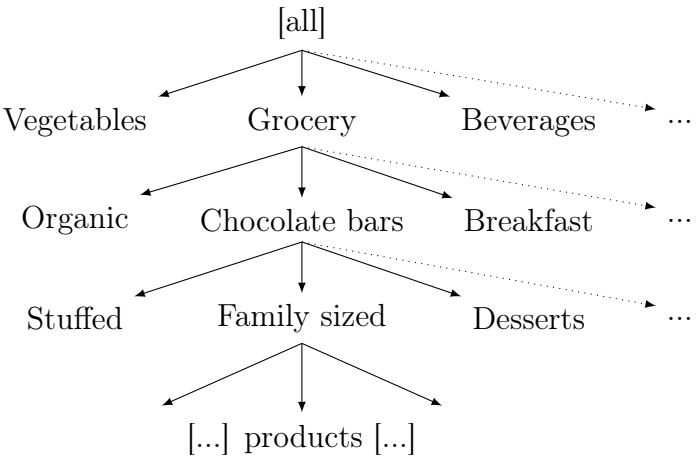


Figure 3.3: Extract from our products taxonomy. While the 11 top categories are intuitive, the 19,546 others are much more specialized, like the "family-sized chocolate bars" shown here. Lower-level categories also include "liquid soups for children", "men sport socks" or "A4 technical sheets", for example.

and a timestamp. In our three mining scenarios, we only consider receipts having a genuine card identifier. Once a day, during each store's closing time, its log is transmitted to the main data center that centralizes all sales records.

Our experimental platform holds an anonymized copy of these records (tuples $\langle r, p, c \rangle$ mentioned in Section 3.1.1) and the *customers* and *taxonomy* tables. We rely on Hadoop YARN [68] to administrate this dedicated cluster. All data is stored in an HBase database [60]. Enrichment and extractions are performed using the Hadoop MapReduce framework [12].

To avoid redundancy and ease data processing, records are grouped by receipt before being stored in our *sales* table. Thus, each receipt is a line in the table and its products list is stored in the *articles* column family. We leverage HBase's flexibility on columns by recording each product identifier as a column qualifier, while the corresponding value holds information such as the cardinality or the unit-price. The row key is written as *storeId-date-customerId-ticketId*, and we set-up an HBase region per store to ensure a good balancing in our cluster. This allows operations such as extracting the sales of a given store to be efficiently performed in a single scan, while selecting a specific time period can also be done by a single key scan. This data layout is optimized to perform these selections efficiently, without incurring unnecessary reads. That allows to store large amounts of data without increasing the cost of analyzing a fixed number of records.

When registering for loyalty cards, customers provide demographic information which can be leveraged by marketing analysts to better understand customer behavior. Each customer constitutes an entry in the *customers* table, which records the segments she belongs to as demographic attributes (for example, *{>65, male, Paris}*). After loading the sales records into the database, we enrich the *sales* table using a MapReduce job. Each receipt's row is augmented with the user segment by querying the *customers* table and copying the segment's attributes in the *meta* column family, as column qualifiers. Hence, each receipt is assigned a snapshot of the customer's demographic attributes.

3.2 Use cases and systems supporting our work

This section presents how the last steps of the DATALYSE workflow are implemented in our use case, starting from the instanciation of meaningful transactional datasets. Thus we reach the point in the workflow where our contributions to the project appear: mining the dataset, and exploring the mining's result. These two steps are fully detailed in the following chapters; in this section we only describe how and in which systems they are integrated.

3.2.1 Curation: 3 mining scenarios that fit in our itemset mining model

In order to construct a transactional dataset $\mathcal{D}$, as described in Section 2.1.1, our raw records $\langle r, p, c \rangle$ can be

- joined with the customer's demographic attributes, $demo(c)$;
- joined with the products taxonomy, to augment each record with the products' categories, $cat(p)$;
- then, grouped by any of these attributes. We usually group records at least by receipt identifiers.

This manuscript follows three mining scenarios designed by experienced analysts from the marketing studies department of Intermarché. They are interested in studying two kinds of buying patterns: those representing associations between customer segments and a product category (e.g. *young people in the north of France consume soda*), and those associating a set of products to a single product (e.g. *people who purchase pork sausage and mustard also buy dry Riesling*).

In the first scenario, `demo_assoc`, the analyst expects rules of the form *customer segment* $\rightarrow$ *category*. Such a rule quantifies how customers belonging to the described segment purchase products in the given category. In that case, $\mathcal{I}$ is the union of the demographic attributes set and the products categories' set. We perform the two joins mentioned above, and group records by customer identifier (additional groupings can be done, as discussed below).

In the two other scenarios, the analyst expects rules of the form *set of products* $\rightarrow \{p\}$, where p is a single product. $\mathcal{I}$ is therefore the set of products. In the second scenario, `prod_assoc_receipt`, raw records are grouped by receipt identifier r. The resulting transactions are equivalent to the actual receipt given to the customer, except we ignore multiple purchases of a single product. In the third scenario, `prod_assoc_client`, records are joined with customer identifiers and grouped by these. Its goal is to study how products are purchased by customers over time.

The following page gives, for each scenarios, a few examples of transactions and desired association rules.

MOULIN. MODULE PETIT DEJEUNE, BALAI PVC 60CM SM

BRANDADE MORUE 300G, MOISSON 6 OEUFS, HC240X220+2TO OCEAN, DH 140X190 OCEAN, TT 90X185 OCEAN

CROQUET.ADULTE 10KG

HARICOT VERT EF4/4 440G, BOXER X2 3D FLEX, LIQ. TAPIS MOQUETTE 600ML, MOUCH.ETUIS 15X10, SALADE MELANGE 250G

CAMEMBERT NOIR 250G, KNACKS X20 700 GR, COMTE 6M PORTION 350G, CHEVRE LONG 180G, SALADE MELANGE 250G

CIDRE RGE CELLIER, CIDRE CELLIER 6*75, BUCHE FONDANTE250G, BOURGUIGNON+PDT PERSILLEES 1KG, GALET ROIS POMME 400G, BEURRE DOUX PQ 250G, PUREES AV.OIGN/CIBOULE.375G, BEURRE 1/2 SEL 500G+100G

Table 3.1: Extract from our `prod_assoc_receipt` dataset (items are separated by commas, brand names removed). `prod_assoc_client` is similar, except transactions usually contain ten times more products.

$A \to B$	$support(A)$	$support(A \cup B)$
LIEGEOIS CHOCO 4X100G → CR.DESS.CHOCO 4X115G	477,710	98,693
{P.FEUILLT ROULE, PREP.FRANGIPANE} → SUCRE POUDRE 1/2KG	9743	522
{ALGUES NORI 17.50G, WASABI 43G, SAUCE SOJA JAP 200ML} → RIZ SUSHI 450G	212	133

Table 3.2: Example association rules extracted from `prod_assoc_receipt` (we search similar rules in `prod_assoc_client`).

<35, Male, Auvergne, Allier, SURGELES, CRUSTACES, BISCUITS FRUITES, BALLADEUR AUDIO, PETITE PUERICULTURE, D.P.H., MOUCHOIR & PAPIERS HYGIENIQUES, ...

<35, Female, Rhône-Alpes, Isère, FRUITS ET LEGUMES, F/L FRAIS EMBALLE, F/L RAYON REFRIGERE DLC COURTE, EPICERIE SALEE, FARINES, FARINE PREPA PAINS, ...

>65, Male, Poitou-Charentes, Vienne, SAUCISSERIE, SURIMI, BOUL PAT TRAD, VIENNOISERIE TRAD, TRAITEUR TRAD, LIQUIDES, APERITIFS SANS ALCOOL, EAUX MINERALES, ...

50-65, Female, Franche-Comté, Haute Saône, D.P.H., MAQUILLAGE, MAQUILLAGE COFFRETS, CREMERIE LS, BEURRE LS, OEUFS, PRODUITS CULTURELS, JEUX CONSOLES, FROMAGE TRAD, FROMAGE COUPE, SPECIALITES REGIONALES, ...

50-65, Female, Franche-Comté, Haute Saône, CHARCUTERIE TRAITEUR, CRUDITES, SANDWICH, FOIES GRAS, PAIN PAT LS INDUS, PAIN DE MIE ET ASSIMILES, PAINS PRECUITS, ...

Table 3.3: Extract from our `demo_assoc` dataset, with shortened transactions as customers are usually buying products from dozens of categories, because each product contributes its 4 containing categories here. Demographic attributes are italicized.

$A \to B$	$support(A)$	$support(A \cup B)$
Alsace → PAINS DE CAMPAGNE	2,843,532	28,889
35-49, Female → PRODUITS SOLAIRES	49,677,175	196,612
>65, Nord, Nord-Pas-de-Calais → CHAMPAGNES MILLESIMES	2,957,290	1187

Table 3.4: Example association rules extracted from `demo_assoc`.

For the 3 mining scenarios, the corresponding dataset is created using MapReduce jobs on the *sales* table, and stored on HDFS as a text file with one line per transaction.

In the `prod_assoc_receipt` case, the set of product identifiers, from each line of our *sales* table, generates one transaction. As records are already grouped by receipt, this is a `map`-only job, resulting in a 23.4GB file.

For `prod_assoc_client`, the products bought by each customer are grouped using a `reduce` operation, where the customer identifier is the key. For each customer, the resulting products set is outputted as a transaction, yielding a 13.3GB file.

Generating transactions for the `demo_assoc` scenario also requires a single MapReduce job, but is less straightforward. In this case the analyst searches for rules of the form *customer segment* $\rightarrow$ *category*, hence we need transactions of the form *{category, demographic attributes [...]}*. But writing such transactions would generate many duplicates, because thousands of customers are usually represented by a single segment (*ie.* a given attributes set). Moreover, in this case we search for associations leading to a *single* category. Categories are never combined in an itemset, so we can further group the transactions in order to write a line per customer segment. Each line stores how many times customers from this segment bought a product from each available category.

Therefore the dataset curation for `demo_assoc` starts with a `map`-side join. All mappers load the products taxonomy in memory (through the distributed cache) and, for each product in each row, generate as many $\langle segment, category \rangle$ records as the product has categories in the taxonomy. Customer segments are directly available in the *meta* column family, thanks to the enrichment phase. Then, for each segment (*ie.* for each distinct demographic attributes set), the `reduce` phase can count how many times each category has been purchased. The counters are outputted in a single record, along their corresponding segment. This double grouping gives a very compact representation of the receipts: for the whole year 2013, the resulting file is only 39MB large.

Grouping identical transactions is surprisingly inefficient for the receipts datasets. Only 35,394 transactions (0.4%) can be merged in `prod_assoc_client`, and 26,232,544 (9%) in `prod_assoc_receipt`- not enough to compensate the cost of storing an additional weight with each transaction, especially as only short transactions are merged. This suggests that a products set can be enough to identify a customer with a decent accuracy. But in this work we leave aside privacy considerations (access to this data is restricted) and send datasets to the mining programs.

3.2.2 Experimental and production platforms

The mining step is the execution of one of the two algorithms we implemented, *j*LCM and TopPI, which are respectively presented in Section 2.5.2 and Chapter 4. Thorough our experiments they may run on two hardware settings, which will be referred to as *server* and *cluster*.

Hadoop programs are deployed on the *cluster* configuration, which consists of up to 65 machines running Hadoop 1.2.1 (without speculative execution), one of them acting as the master node. Each machine contains 2 Intel Xeon E5520

4-cores CPUs and 24 GB of RAM. We use the default configuration, except for resource allocation. The number of tasks per machine and the memory allocated to each is detailed for each experiment.

Multi-threaded implementations run on the *server* configuration: a single machine containing 128GB of RAM and 2 Intel Xeon E5-2650 8-cores CPUs with Hyper Threading, allowing up to 32 threads in parallel.

It is worth mentioning that the production architecture, set up by our industrial partners in DATALYSE, is logically equivalent but deployed differently. The only common point is the curation platform. The production platform is a YARN cluster made of 4 worker nodes. Each one is a virtual machine which has been assigned 32GB of RAM and 4 cores out of the 8 of an Intel Xeon E5-2650L CPU. In production, this YARN cluster is also the analytics platform, running the mining step. This sets the bar for our resource requirements: on this cluster, our algorithms should be able to run the mining step of our 3 scenarios.

3.2.3 Result exploration and exploitation

Finally, the user needs a tool to explore, filter and sort the discovered association rules. The target audience of this tool is not only computer scientists or data analysts from Intermarché, but also marketing experts. Hence an interactive and highly intuitive tool is required at this step.

We therefore developed a Web application which present results stored in a PostgreSQL database [62]. The amount of association rules we generate (usually in the millions) is easily handled by a classic setup on a dedicated virtual machine.

Dataset	#Transactions	#Items	File size	Average trans--action length
LastFM	1,218,831	1,206,195	277 MB	47
WebDocs	1,692,082	5,267,656	1.4 GB	177
`demo_assoc`	2398	241,921	39 MB	1617
`prod_assoc_receipt`	290,734,163	222,228	23.4 GB	12
`prod_assoc_client`	9,267,961	222,228	13.3 GB	213

Table 3.5: Our datasets' cardinalities. `demo_assoc` is actually a compressed representation of our transactions of interest (see Section 3.2.1, p.30); during the mining, the algorithm interprete these as transactions of at most 5 items (see Section 5.1.2, p.69).

3.3 Datasets summary: common characteristics

In order to validate the performance of our mining algorithms beyond our industrial setting, we also run experiments on Web data. Although our transactional datasets represent very different entities, they share some properties that raise new challenges for existing closed itemsets (CIS) mining algorithms. Before developing and answering the different challenges in the following chapters, we present our 5 datasets of choice and highlight these common properties. Table 3.5 summarizes our datasets' details.

Our first two are Web datasets. The first one *LastFM*, represents user activity on a music recommendation website. We crawled 1.2 million public profile pages, each resulting in a transaction containing the 50 favorite artists of a user. Hence in this dataset we mine associations between artists. The other, *WebDocs*, is a dataset frequently used in the itemset mining community [43]. Each transaction contains the words used on a Web page.

Our three other datasets represent supermarket data, from which we want to extract insights on consumer behavior. As detailed in Section 3.2.1, these studies are formalized as three mining scenarios: `prod_assoc_receipt`, `prod-_assoc_client` and `demo_assoc`. The first step of the DATALYSE workflow is the transformation of this data into a transactional dataset; depending on the scenario, transactions may carry different semantics. In `prod_assoc_receipt` and `prod-_assoc_client` a transaction is a set of products chosen by a single customer, in a single purchase or over the year 2013, respectively. Hence transactions are longer in `prod_assoc_client`. In `demo_assoc` a transaction represents product categories chosen by a customer in a demographic segment.

Using a generic model for closed itemsets mining (detailed in Section 2.1.1, p.8) allows us to experiment with various data. Whether originating from our industrial partner, from the itemset mining community or from the Internet, our transactional datasets are representative data that motivates our adaptations of itemset mining: millions of short transactions over a few hundred thousands items (or more), with a long tail distribution. This is illustrated in Figure 3.4 where we can observe that the majority of items occur in less than a hundred transactions.

This is even more striking for the two Web datasets, *LastFM* and *WebDocs*,

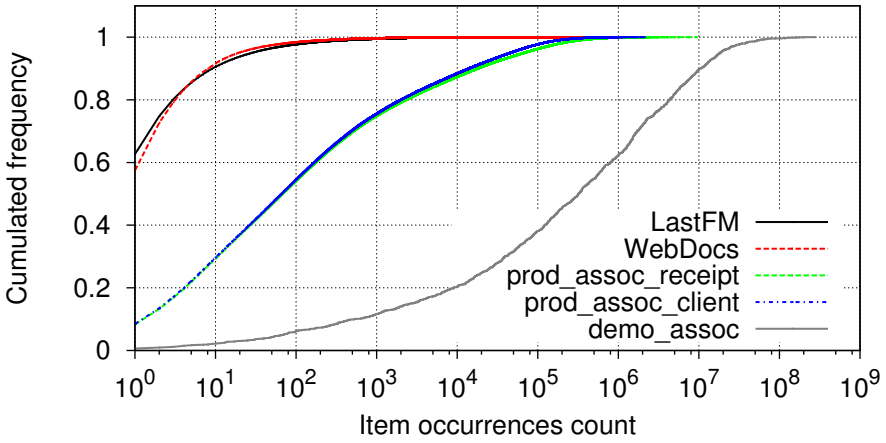


Figure 3.4: Cumulative distribution of items' supports in our datasets. They form three groups, from top to bottom: Web datasets (*LastFM* and *WebDocs*), have the most pronounced long tail distribution. Receipts sets also follow this trend. The last one, `demo_assoc`, is an outlier here because it's mixing hierarchical items (*e.g.* narrow or wide categories).

as 90% of their items occur in at most 10 transactions. For example, in *LastFM* this means that 90% of artists have 10 listeners or less. In *WebDocs*, 90% of words appear in 10 documents or less. These are typical examples of a "long tail" distribution [19]. The sparsity of market baskets is more pronounced on non-singleton itemsets: the support of a pair of items is usually two orders of magnitude lower than the support of each item alone. `demo_assoc` does not show this long-tail distribution, and its compact representation does not raise any computational challenge. However the variety of its items makes it relevant to our work on exploring association rules. The 5 datasets also have in common a low itemset density: items are spread irregularly in the dataset, and rarely combine with each other.

Each dataset carries its own patterns semantics, but the techniques presented in the following chapters can be applied to any of them.

CHAPTER 4

Mining item-centric top-k closed itemsets with TopPI

Contents

Given a transactional dataset, without additional knowledge it is tempting to search for frequent items associations. An attempt to apply such algorithms directly raises a question: how many occurrences make an itemset frequent? A thousand? Ten? One percent? Indeed, the algorithm needs this threshold to be fixed before starting the computation. A quick study of the items' individual frequencies may give a first hint of a relevant threshold. In practice, though, the analyst often does a more iterative guess. Starting from a high frequency threshold (10%, for example), she progressively lowers the threshold until she finds interesting results. This makes the discovery process more and more difficult, as lowering the frequency threshold yields more and more results (even with the restriction to closed itemsets). Indexing the resulting CIS by item is quite intuitive. But the analyst would expect such an index to include any item, because she can be interested in any item's association. This is particularly true at the first encounter with a dataset, when analysts need to explore easily all its aspects.

Applying itemset mining on millions of transactions therefore raises the following problems:

- A single mining attempt should provide results about all items appearing in at least two transactions. But, as seen from existing algorithms, this is close to mining *all* CIS.

- The input dataset is the only knowledge we have, hence our parameter(s) should only express the desired quantity of output.
- The processing should be parallelized, in order to fit in the systems which are usually storing datasets as big as ours.

These challenges lead us to propose an *item-centric* problem statement and an algorithm to solve it, TopPI. The present chapter details our contributions :

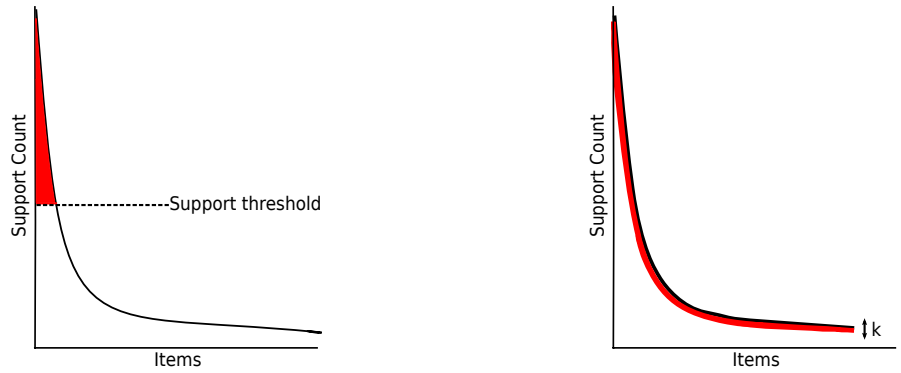
- We formalize item-centric mining, and highlight its afferent challenges with an experiment and an example, in Section 4.1.
- We detail our algorithm, its heuristics and pruning strategies in Section 4.2.
- We show how to leverage additional CPUs or machines for item-centric mining in Section 4.3.
- Section 4.4 shows example results and evaluates experimentally TopPI. We show that it outperforms similar or straightforward solutions, and measure its speedup on both our *server* and *cluster* platforms.
- Thanks to the technology transfer to the Datalyse project, we provide industrial feedback on the run-time performance and results quality of TopPI in Section 4.5.

4.1 Item-centric itemset mining: goal and challenges

We start by further discussing how item-centric mining differs from frequent CIS mining. Section 4.1.1 formalizes TopPI's problem statement. In Section 4.1.2 we show experimentally that a frequent itemset mining algorithm like jLCM fails to provide results that satisfy our problem statement. Finally Section 4.1.3 gives a first intuition of TopPI's underlying principles, by detailing a CIS enumeration. This example allows us to precise the challenges inherent to item-centric itemset mining.

4.1.1 Relevant semantics for long-tailed datasets

As the number of transactions increases, searching iteratively for the right frequency threshold becomes less and less convenient — not because of the computation time, but because of the number of results. In our *server* setting, once the dataset is loaded in memory and using a frequency threshold of 32,000 (0.01%), it takes less than 2 minutes to extract 50,464 closed itemsets (CIS) out of `prod_assoc_receipt` with jLCM. But browsing such results set is too cumbersome for the analyst. Looking back at Figure 3.4 (p.35) we also observe that 92% of the articles are filtered out with this threshold. In other words, our results show



(a) Frequent Itemset Mining (b) Our approach, TopPI

Figure 4.1: Schematic distinction between frequent itemsets mining and TopPI. The area in red represent each method's output.

associations between only 8% of the available articles. We could include more articles by lowering the frequency threshold, but this would produce an overwhelming amount of results. In `prod_assoc_receipt` again, after 4 minutes of computation we can find the 376,495 itemsets appearing more than 10,000 times. But they only cover 13% of the articles. This observation also applies to Web datasets, which often follow a "long tail" distribution [19].

A common request in the retail industry is the ability to access a product's sales trends and associations with other products. This allows managers to obtain feedback on customer behavior and to propose relevant product bundles. Whether items are products, categories or segments, the analyst will spontaneously search for an item she already knows, discover new items associated to the first one, continue to one of the new items' associations, and so on. Hence our item-centric approach and our introduction of a unique parameter: k, the number of itemsets to be returned per item.

Figure 4.1 illustrates our distinction from frequent itemset mining algorithms. TopPI extracts itemsets for all items, therefore providing the analyst with an overview of the dataset. As illustrated in the previous paragraph, this is essential when discovering a transactional dataset.

TopPI is also beneficial to other applications such as query recommendation [37], where we usually should give a dozen recommendations related to a requested item. TopPI ensures the ability to answer recommendation queries, even for rare terms. We therefore formalize the objective of TopPI as follows:

Definition 5. *Given a dataset $\mathcal{D}$ and an integer k,* TopPI *returns, for each item in $\mathcal{D}$, the k most frequent closed itemsets containing this item.*

Table 4.1b shows the solution to this problem applied to the dataset in Table 4.1a, with $k = 2$. Note that we purposely ignore itemsets that occur only once, as they do not show a behavioral pattern. The restriction to closed itemsets avoid redundant results (as discussed in Sections 2.1.1 and 2.1.2). The number of itemsets returned for each item may be tuned depending on the application. If the itemsets are directly presented to an analyst, $k = 10$ can be sufficient, while $k \geq 100$ may be used when those itemsets are analyzed automatically.

TID	Transaction
t_0	$\{0, 1, 2\}$
t_1	$\{0, 1, 2\}$
t_2	$\{0, 1\}$
t_3	$\{2, 3\}$
t_4	$\{0, 3\}$

(a) Dataset $\mathcal{D}$

item	$top(i)$: P, $support(P)$	
i	1^{st}	2^{nd}
0	$\{0\}, 4$	$\{0, 1\}, 3$
1	$\{0, 1\}, 3$	$\{0, 1, 2\}, 2$
2	$\{2\}, 3$	$\{0, 1, 2\}, 2$
3	$\{3\}, 2$	

(b) Results for $k = 2$

Table 4.1: Example TopPI input and output

TopPI relies on frequency to rank each k itemsets associated with an item. Another possibility would have been to rank the itemsets associated to an item by statistical correlation with this item, using for example the p-value as a measure. However, ranking by p-value implies a much wider exploration of the itemsets space, because the strongest correlations are sometimes found with rare itemsets [47]. This also applies to other quality measures, which are not affordable at our target scale. We instead propose, in Section 6.2 (p.85), to re-rank TopPI's results according to a finer quality measure.

4.1.2 First challenge: the need for a specialized algorithm

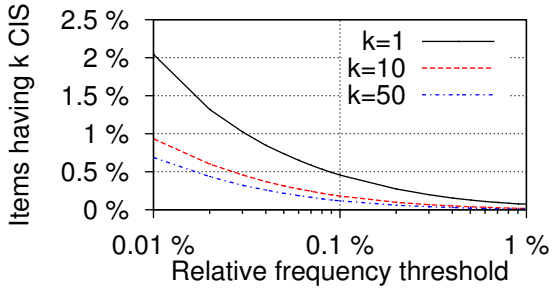
We now present and discuss preliminary experiments showing that we cannot perform item-centric mining with existing algorithms.

The most obvious solution is to use a global top-k-frequent CIS miner like TFP [24] on each projected dataset $\mathcal{D}[i]$, for each item i. This solution runs out of memory on our datasets. Even if we let it benefit from one of our optimizations, it is not as robust as our dedicated algorithm — this is further discussed in the Section 4.4.3 of our experiments.

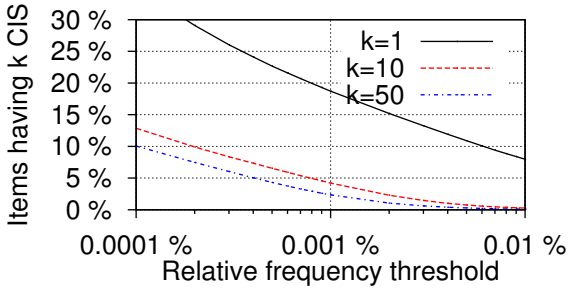
In this section we present another experiment, where we simulate item-centric mining with our frequent CIS mining algorithm, jLCM (fully presented in Section 2.1.2, p.9). The top-k CIS of each item can be computed as a post-processing of these results, thus producing the same output as TopPI. We evaluate this approach for different frequency thresholds ε and present the results in Figure 4.2. This post-processing approach does not take the number of desired CIS per item k as a parameter, so instead we measure for each value of ε the number of items appearing in at least k CIS, for 3 values of k.

Each item is present in at least one CIS, whose support is equal to the frequency of the item in the dataset. This CIS, a singleton in most cases, is the only result sufficient for $k = 1$. We already notice that, because of the long-tailed distribution of the data, it is necessary to reach very low values of ε to select a large number of items. In the case of *LastFM*, only 2% of items appear in more than 0.01% of the transactions. Furthermore, for the same value of ε, only 1% of items are included in more than 10 frequent CIS. But it takes 150 hours of CPU time to extract all CIS at this support threshold: the algorithm mines over 2 billion CIS, and only 24,536 items out of 1.2 million appear in these results.

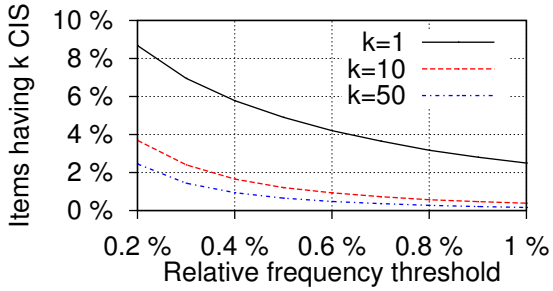
The behavior of this approach on `prod_assoc_receipt`, although slightly bet-



(a) *LastFM*



(b) `prod_assoc_receipt`



(c) `prod_assoc_client`

Figure 4.2: Proportion of available items appearing in at least k itemsets (for $k \in \{1, 10, 50\}$), using frequent CIS mining, with respect to the frequency threshold.

ter, highlights the same limitations. Using a frequency threshold of $10^{-5}\%$, mining frequent CIS takes one hour. This may appear to be a reasonable run-time, but the results are not very useful for the analysts: the 34,151,639 CIS only contain a minority of the available products, and 90% of the items are in less than 10 CIS. We obtain similar figures over `prod_assoc_client`: when $\varepsilon = 0.2\%$, it takes 45 hours of CPU time to obtain 13,612,569 CIS covering less than 10% of the available items.

As this experiment shows, standard CIS mining algorithms are perfectly capable of efficiently enumerating billions of CIS. The problem is that these itemsets only contain the most frequent items in the dataset, and fail to cover items from the long tail. Hence, the post-processing approach spends most of the mining time generating an overwhelming amount of CIS which are of no interest to the analyst. This observation motivated the development of TopPI, that efficiently computes item-centric CIS using an integrated approach. We leverage the efficient enumeration principles from LCM, but use different heuristics and pruning to perform item-centric mining.

4.1.3 Second challenge: pruning while garanteeing correctness

We now discuss how we can optimize and prune jLCM's enumeration for item-centric mining, over the example of Figure 4.3, where $k = 2$. In this example we follow the *expand* function presented in Algorithm 1, p.10. The only modification is that we now assume that the enumerated CIS are progressively kept in a collection of heaps, denoted $top(i)$ for each item i. Items are already indexed by decreasing frequency. Candidate extensions of steps ③ and ⑨ are not collected as they fail the first-parent test (their closure is $\{0, 1, 2, 3\}$).

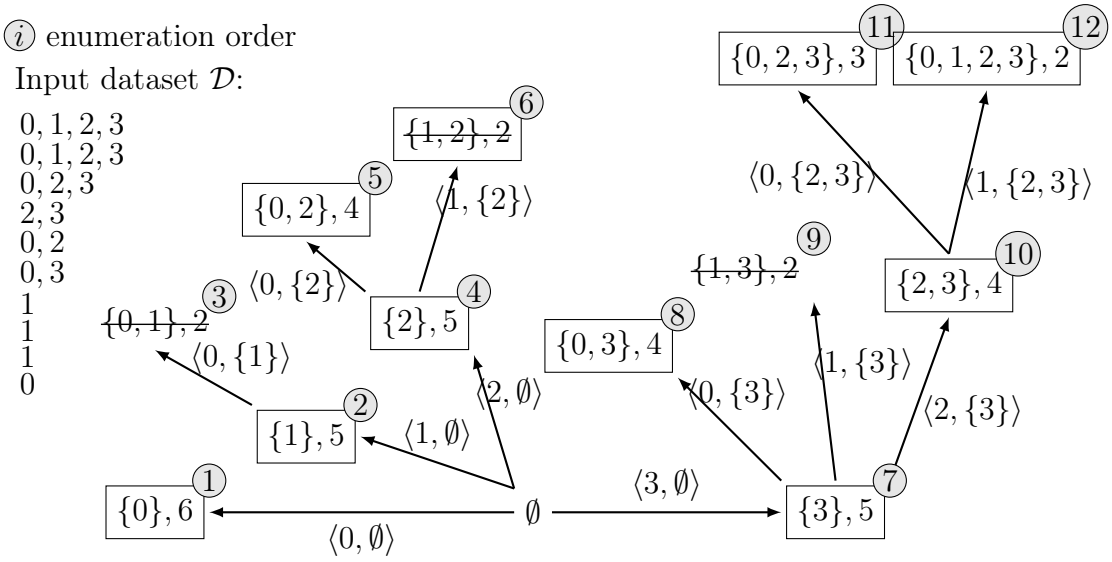


Figure 4.3: An example dataset and its corresponding CIS enumeration tree in jLCM. Each node is an itemset and its support. $\langle i, P\rangle$ denotes the closure extension operation. Striked out itemsets are candidates failing the first-parent test (Algorithm 1, line 10, p.10).

In frequent CIS mining algorithms, the frequency threshold allows the program to lighten the intermediate datasets ($\mathcal{D}_I$) involved in the enumeration. In TopPI we have an implicit minimum support ε equal to 2, because we are not interested in CIS occurring only once. Though, increasing ε above 2, when exploring some parts of the CIS space, could speed up the exploration as in a frequent CIS miner. This is the goal of the dynamic threshold adjustment proposed in TopPI. In our example, before step ④ we can compute items' supports in $\mathcal{D}[2]$ — these supports are re-used in $expand(\emptyset, 2, \mathcal{D}, \varepsilon)$ — and observe that the two most frequent items in $\mathcal{D}[2]$ are 2 and 0, with respective supports of 5 and 4. These will yield two CIS of supports 5 and 4 in $top(2)$. The intuition of dynamic threshold adjustment is that 4 might therefore be used as a frequency threshold in this branch. It is not possible in this case because a future extension, 1, does not have its k itemsets at step ④. This is also the case at step ⑦. The dynamic threshold adjustment done in TopPI takes this into account, in order to ensure its results' correctness.

After step ⑧, $top(0)$, $top(2)$ and $top(3)$ already contain two CIS, as required, all having a support of 4 or more. Hence it is tempting to prune the extension $2, \langle\{3\}\rangle$ (step ⑩), as it cannot enhance $top(2)$ nor $top(3)$. However, at this step, $top(1)$ only contains a single CIS and 1 is a future extension. Hence ⑩ cannot be pruned: although it yields an useless CIS, one of its extensions leads to a useful one (step ⑫). In this tree we can only prune the recursion towards step ⑪.

This example's distribution is unbalanced in order to show TopPI's corner cases with only 4 items; but in real datasets, with hundreds of thousands of items, such cases regularly occur. This shows that an item-centric mining algorithm requires rigorous strategies for both pruning the search space and filtering the datasets.

Rather than exhaustively extracting the most frequent itemsets, TopPI en-

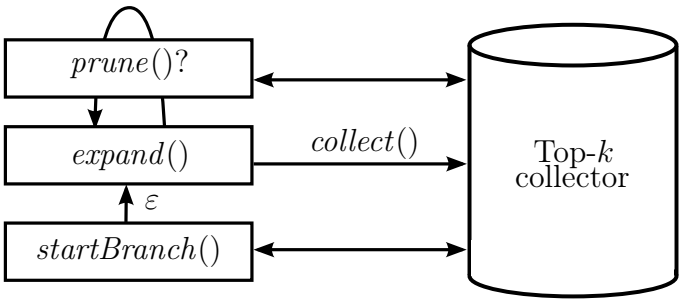


Figure 4.4: TopPI's architecture.

sures that all items are covered by the results and restricts its search space according to the k parameter. Our datasets' size raises an additional challenge: the exploration and pruning strategy should remain efficient when parallelized, in order to handle millions of transactions in a reasonable amount of time.

4.2 TopPI algorithm

We now present our sequential solution of item-centric CIS mining, the TopPI algorithm.

Dataset notations reminder $\mathcal{D}$ is our input dataset, a collection of transactions. Given an itemset I, the projected dataset of I is $\mathcal{D}[I] = \langle t \in \mathcal{D} \mid I \subseteq t \rangle$. Its reduced dataset is $\mathcal{D}_I = \langle t \setminus I \mid t \in \mathcal{D}[I] \rangle$. When I is a singleton $\{i\}$, we may omit the braces and write $\mathcal{D}[i]$ and $\mathcal{D}_i$.

4.2.1 TopPI architecture

Figure 4.4 gives an overview of TopPI. The CIS enumeration at the core of TopPI follows principles from jLCM (presented in Section 2.1.2, p.9). In TopPI it is done by a variant of the *expand* function, presented in Section 4.2.2. Similarly to traditional top-k processing approaches [14], TopPI relies on heap structures to progressively collect its top-k results, and outputs them once the execution is complete. More precisely, TopPI stores traversed itemsets in a *top-k collector* which maintains, for each item $i \in \mathcal{I}$, $top(i)$, a heap of size k containing the current version of the k most frequent CIS containing i. As soon as a CIS and its support is discovered by *expand*(), it is kept in the top-k-collector via the *collect*() function. We can query the top-k-collector through $lowestSupportIn(top(i))$, which is the k^{th} support value in $top(i)$, or 2 if $|top(i)| < k$. We mine all the k-lists simultaneously to maximize the amortization of each itemset's computation. Indeed an itemset is a candidate for insertion in the heap of all items it contains. Moreover, as shown in Chapter 3, the analysis is not performed on the live transaction database but on dedicated machines. Hence we choose to create an itemset database for the associations explorer application.

TopPI introduces an adequate pruning of the solutions space. For example, we should be able to prune an itemset $\{a, b, c\}$ once we know it is not a top-k frequent

for a, b nor c. However, as highlighted in the previous example (Section 4.1.3), we cannot prune $\{a, b, c\}$ if it precedes interesting CIS in the enumeration. TopPI's *prune* function, presented in Section 4.2.3, tightly cuts the CIS space while ensuring results' completeness. Section 4.2.4 how we can simplify its computation with *prefix short-cutting*.

The CIS exploration is initiated by the *startBranch* function, presented in Section 4.2.5, which enumerates itemsets P such that $max(P) = i$. TopPI does not require the user to define a minimum frequency, but we observe that the support range in each item's top-k CIS varies by orders of magnitude from an item to another. Because filtering out less frequent items can speed up the CIS enumeration in some branches, *startBranch* implements a *dynamic threshold adjustment*: it tries to find the greatest frequency threshold ε that garantees our results' completeness in each branch. ε defaults to 2, because we are not interested in itemsets occurring once.

Algorithm 2: TopPI's main function

```
   Data: dataset 𝒟, integer k
   Result: Output top-k CIS for all items of 𝒟
1  begin
2  |  foreach i ∈ ℐ do                          // Collector instantiation
3  |  |  initialize top(i), heap of max size k
4  |  foreach i ∈ ℐ do                          // In increasing item order
5  |  |  startBranch(i, 𝒟, k)
```

The main program, presented in Algorithm 2, initializes the collector in lines 2 and 3. Then it invokes, for each item i, $startBranch(i, \mathcal{D}, k)$. In our examples, as in TopPI, items are represented by integers. While loading $\mathcal{D}$, TopPI indexes items by decreasing frequency, hence 0 is the most frequent item. Items are enumerated in their natural order in line 4, thus items of greatest support are considered first.

4.2.2 CIS enumeration in TopPI

TopPI traverses the CIS space with the *expand* function, detailed in Algorithm 3. *expand* performs a depth-first exploration of the CIS tree, and backtracks when no frequent extension items remain in $\mathcal{D}_J$ (line 6). Additionally, in line 7 the *prune* function (presented in Section 4.2.3) determines if each recursive call may enhance results held in the top-k-collector, or if it can be avoided.

Upon validating a closure extension J, TopPI updates $top(i)$, $\forall\ i \in J$, via the *collect* function (line 5). The support computation exploits the fact that $support_{\mathcal{D}}(J) = support_{\mathcal{D}_I}(e)$, because $J = closure(\{e\} \cup I)$. The last parameter of *collect* is set to `true` to point out that J is a closed itemset (we show in Section 4.2.5 that it is not always the case).

When enumerating items in line 6, TopPI relies on the items' indexing by decreasing frequency. As extensions are only done with smaller items this ensures that, for any item $i \in \mathcal{I}$, the first CIS containing i enumerated by TopPI combine i with some of the most frequent items. This heuristic increases their probability

Algorithm 3: TopPI's CIS exploration function

```
1 Function expand(I, e, D_I, ε)
    Data: CIS I, extension item e, reduced dataset D_I, frequency threshold ε
    Result: If ⟨e, I⟩ is a relevant closure extension, collects CIS containing {e} ∪ I and items smaller than e
2   begin
3     J ← closure({e} ∪ I)
4     if max(J \ I) = e then
5       collect(J, support_D(J), true)        // Update top-k-collector
6       foreach i < e | support_{D_J}[i] ≥ ε do    // In increasing item order
7         if ¬prune(J, i, D_J, ε) then          // Additional pruning
8           expand(J, i, D_J, ε)
```

of having a high support, and overall raises the support of itemsets in the top-k-collector.

For clarity, the algorithms presented in this chapter do not show the complete implementation of datasets and their attached structures. Since the instantiation of projected datasets is the major CPU time consumer, these are extensively re-used in TopPI:

- As our algorithms regularly perform projections on singleton items, the input dataset $\mathcal{D}$ and reduced datasets $\mathcal{D}_I$ hold occurrences lists. These are indexes providing instantly $\mathcal{D}[j]$ or $\mathcal{D}_I[j]$ for any item j. Those projected datasets are therefore accessed as views over $\mathcal{D}$ or $\mathcal{D}_I$, respectively, a technique called "occurrence deliver" in LCM [63].

- In TopPI, as in jLCM, computing $closure(\{e\} \cup I)$ is done by counting items' frequency in $\mathcal{D}_I[e]$. Any item having a 100% frequency belongs to $J = closure(\{e\} \cup I)$ (Algorithm 3, line 3). The resulting item frequency counters are re-used when instantiating the reduced dataset $\mathcal{D}_J$ or the set of potential extensions items (Algorithm 3, line 6).

4.2.3 Pruning the CIS space

As shown in the example of Section 4.1.3, TopPI cannot prune a sub-tree rooted at I by observing I alone. We also have to consider itemsets that could be enumerated from I through first-parent closure extensions. This is done by the *prune* function presented in Algorithm 4. It queries the collector to determine whether $expand(I, e, \mathcal{D}_I, \varepsilon)$ and its recursions may impact the top-k results of an item. If it is not the case then *prune* returns `true`, thus pruning the sub-tree rooted at $closure(\{e\} \cup I)$.

The anti-monotonicity property [3] ensures that the support of all CIS enumerated from the closure extension $\langle e, I\rangle$ is smaller than $support_{\mathcal{D}_I}(\{e\})$. It also follows from Property 1 (p.11) that the only items potentially impacted by $\langle e, I\rangle$ are:

Algorithm 4: TopPI's pruning function

```
1  Function prune(I, e, D_I, ε)
       Data: itemset I, extension item e, reduced dataset D_I, minimum
             support threshold ε
       Result: true if expand(I, e, 𝒟_I, ε) will not provide new results to the
               top-k-collector, false otherwise
2      begin
3          if support_{𝒟_I}({e})) ≥ lowestSupportIn(top(e)) then        // Case 1
4              return false
5          foreach i ∈ I do                                             // Case 2
6              if support_{𝒟_I}({e})) ≥ lowestSupportIn(top(i)) then
7                  return false
8          foreach i < e | support_{𝒟_I}[i] ≥ ε do                      // Case 3
9              bound ← min(support_{𝒟_I}({e}), support_{𝒟_I}({i}))
10             if bound ≥ lowestSupportIn(top(i)) then
11                 return false
12         bound_min(I) ← min(bound, bound_min(I))
13         return true
```

1. e, the extension item;
2. the items of the extended CIS, I;
3. potentially any item inferior to e, to be added by closure or recursive extensions.

The two first cases, considered in lines 3 and 5, check $top(e)$ and $top(i), \forall i \in I$. Smaller items, which may be included in future extensions of $\{e\} \cup I$, are considered in lines 8–11. It is not possible to know the exact support of these CIS, as they are not yet explored. However we can compute, as in line 9, an upper bound such that $bound \geq support(closure(\{i, e\} \cup I))$. If this bound is smaller than $lowestSupportIn(top(i))$, then extending $\{e\} \cup I$ with i cannot provide a new CIS to $top(i)$. Otherwise, as tested in line 10, we should let the exploration deepen by returning `false`. If this test fails for all items i, then it is safe to prune because all $top(i)$ already contain k itemsets of greater support.

The inequalities of lines 3, 6 and 10 are not strict to ensure that no partial itemset (inserted by the *startBranch* function, see Section 4.2.5) remains at the end of the exploration. We also remark that the loop of lines 8–11 may iterate on up to $|\mathcal{I}|$ items, and thus may take a significant amount of time to complete. Hence our implementation of the *prune* function includes an important optimization.

4.2.4 Fast pruning with prefix short-cutting

We can leverage the fact that TopPI enumerates extensions by increasing item order. Let e and f be two items successively enumerated as extensions of a CIS P (Algorithm 3 line 6). As $e < f$, in the execution of $prune(I, f, \mathcal{D}_I, \varepsilon)$ the loop of lines 8–11 can be divided into:

- iterations on items $i \leq e \wedge i \notin P$,
- and the last iteration where $i = f$.

We observe that the first iterations were also performed by $prune(I, e, \mathcal{D}_I, \varepsilon)$, which can therefore be considered as a prefix of the execution of $prune(I, f, \mathcal{D}_I, \varepsilon)$.

To take full advantage of this property, TopPI stores the smallest *bound* (computed line 9) such that $prune(I, *, D_I, \varepsilon)$ returned `true`, denoted $bound_{min}(I)$. This represents the lowest known bound on the support required to enter $top(i)$, for items $i \in \mathcal{D}_I$ ever enumerated by line 8. When evaluating a new extension f by invoking $prune(I, f, \mathcal{D}_I, \varepsilon)$, if $support_{\mathcal{D}_I}(f) \leq bound_{min}(I)$ then f cannot satisfy any test of lines 6 and 10. In this case it is safe to skip the loop of lines 5–7, and more importantly the prefix of the loop of lines 8–11, therefore reducing this latter loop to a single iteration. As items are sorted by decreasing frequency, this simplification happens very frequently.

Thanks to prefix short-cutting, most evaluations of the pruning function are reduced to a few invocations of $lowestSupportIn(top(i))$. This allows TopPI to guide the itemsets exploration with a negligible overhead.

4.2.5 Filtering intermediate datasets with dynamic threshold adjustment

If we initiate each CIS exploration branch by invoking $expand(\emptyset, i, \mathcal{D}, 2), \forall i \in \mathcal{I}$, then *prune* would be inefficient during the k first recursions — that is, until $top(i)$ contains k CIS. For frequent items, which yield the biggest projected datasets, letting the exploration deepen with a negligible frequency threshold is particularly expensive. It is also crucial to diminish the size of the dataset as often as possible, by filtering out less frequent items that do not contribute to our results. Hence we propose the dynamic threshold adjustment technique, to filter projected datasets when possible and avoid the cold start situation. It is implemented by the *startBranch* function, presented in Algorithm 5.

Algorithm 5: TopPI's CIS enumeration branch preparation

```
1 Function startBranch(i, D, k)
      Data: root item i, dataset D, integer k
      Result: Enumerates CIS I such that max(I) = i
2     begin
3         foreach j ∈ topDistinctSupports(D[i], k) | i ≠ j do // Pre-filling
4             collect({i, j}, support_{D[i]}(j), false)
5         ε_i ← min_{j≤i}(lowestSupportIn(top(j)))          // Dynamic th.
              adjustment
6         expand(∅, i, D, ε_i)
```

Given a CIS $\{i\}$ and an extension item $e < i$, computing $Q = closure(\{e\} \cup \{i\})$ is a costly operation that requires counting items in $\mathcal{D}_{\{i\}}[e]$. However we observe that $support(Q) = support_{\mathcal{D}}(\{e\} \cup \{i\}) = support_{\mathcal{D}[i]}(e)$, and the latter value is computed by the items counting preceding the instantiation of $\mathcal{D}_i$. Therefore,

when starting the branch of the enumeration tree rooted at i, we can already know the supports of some of the upcoming extensions.

The function *topDistinctSupports* counts items' frequencies in $\mathcal{D}[i]$ — resulting counts are re-used in *expand* for the instantiation of $\mathcal{D}_i$. Then, in lines 3–4, TopPI considers items j whose support in $\mathcal{D}[i]$ is one of the k greatest, and stores the partial itemset $\{i, j\}$ in the top-k collector (this usually includes $\{i\}$ alone). We call these itemsets partial because their closure has not been evaluated yet, so the top-k collector marks them with a dedicated flag: the third argument of *collect* is `false` (line 4). Later in the exploration, these partial itemsets are either ejected from $top(i)$ by more frequent CIS, or replaced by their closure upon its computation (Algorithm 3, line 5). Thus $top(i)$ contains k itemsets at the end of the loop of lines 3–4.

The CIS recursively generated by the *expand* invocation of line 6 may only contain items lower than i. Therefore the smallest $lowestSupportIn(top(j)), \forall j \leq i$, can be used as a frequency threshold in this branch. TopPI computes this value, ε_i, on line 5, in order to filter the biggest projected datasets as a frequent CIS miner would. This combines particularly well with the frequency-based iteration order, because $lowestSupportIn(top(i))$ is relatively high for more frequent items.

Note that two partial itemsets $\{i, j\}$ and $\{i, l\}$ of equal support may in fact have the same closure $\{i, j, l\}$. Inserting both into $top(i)$ could lead to an overestimation of the frequency threshold, which would let the enumeration miss legitimate top-k CIS of i. To avoid this, TopPI only selects partial itemsets with *distinct* supports.

4.3 Scaling TopPI

In Section 4.2 we presented a sequential version of TopPI. We now describe how TopPI can be deployed on parallel and distributed processing platforms to process large scale datasets. The goal is to divide the mining process into independent sub-tasks executed on workers while ensuring the output completeness, avoiding redundant computation, and maintaining pruning performance. We first focus on shared-memory parallel architectures in Section 4.3.1. Then we consider the case of distributed systems in Section 4.3.2, and present its implementation over the MapReduce framework [12]. We finally discuss the problem of transactions repartition in a distributed environment in Section 4.3.3.

4.3.1 Shared-memory parallel systems

Thanks to our optimizations, TopPI spends most of its execution time enumerating CIS. Négrevergne *et al.* proposed PLCM, a parallel version of LCM [50], that speeds up significantly this task on multi-core systems. PLCM dispatches each exploration branch among the available cores. In a similar fashion, TopPI instantiates multiple threads and dispatches *startBranch* invocations. The collector is shared by all threads, but this does not cause any congestion because most accesses are read operations from the *prune* function.

However this could interfere with the dynamic threshold computation, presented in Section 4.2.5, because raising the threshold in Algorithm 5, line 5 re-

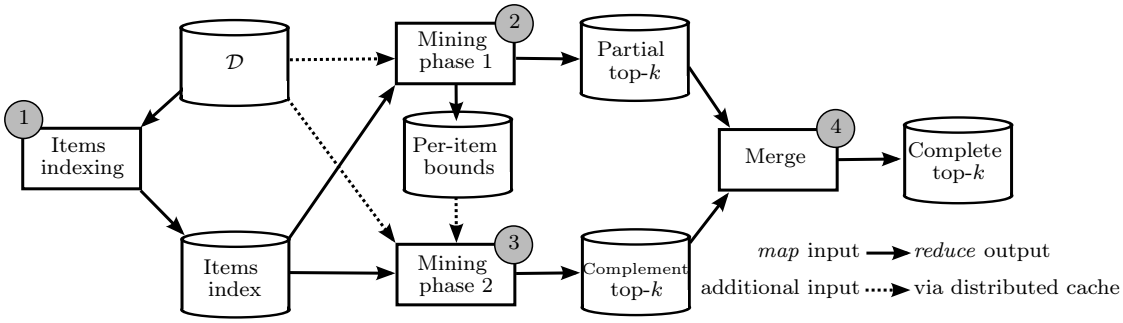


Figure 4.5: Hadoop implementation of TopPI

quires the top-k of lower items to provide high lower-bounds. This could also impact the verifications performed by the pruning function in Algorithm 4, line 8. For these reasons, we divide *startBranch* in two before line 5, and use a producer-consumer structure to ensure that the ε_i computation (Algorithm 5, line 5) can only be executed for an item i once all items lower than i are done with the top-k collector pre-filling (Algorithm 5, lines 3-4). As we experiment in Section 4.4.4, this design achieves excellent scalability.

4.3.2 Distributed systems and Hadoop implementation

In a distributed setting, we dispatch *startBranch* invocations among workers. Figure 4.5 gives an overview of our solution, when implemented over the Hadoop framework. We start by detailing how the CIS exploration is distributed, then discuss how we dispatch the top-k collector to ensure its completeness while maintaining its pruning power. This latter aspect accounts for the separation of item-centric mining in two phases.

Partitioning the CIS enumeration tree Each worker is assigned a partition of items $G \subseteq \mathcal{I}$, which will be referred to as its *group* of items. A worker restricts its exploration to the branches starting with an element of G (Algorithm 2, line 4, p.44). Following the example of Table 4.2, a worker that is assigned the partition $G = \{0, 2\}$ collects the itemsets $\{0\}$,$\{2\}$ and $\{0, 1, 2\}$. It follows from Property 1 (p.11) that each worker therefore generates a distinct set of itemsets, without overlap nor need for a synchronization among them.

Another benefit of this dispatching is that a worker only needs the transactions of $\mathcal{D}$ that contain items of its partition G. We call this set of transactions the sub-dataset of the group G, denoted $\mathcal{D}[G]$. Our choice to send $\mathcal{D}$ via the distributed cache for these projections (jobs ② and ③ of Figure 4.5) relies on implementation considerations discussed in Section 4.3.3. Items are indexed by decreasing frequency in a single MapReduce job (Figure 4.5, job ①). Then, items are assigned to groups in a round robin fashion. This ensures that the most frequent items, which form the largest sub-datasets, are assigned to different groups. This balances the load of workers.

Note that the CIS enumeration can be parallelized in a worker, as presented in the previous section. Using all CPUs of a single machine to mine a single

TID	Transaction
t_0	$\{0, 1, 2\}$
t_1	$\{0, 1, 2\}$
t_2	$\{0, 1\}$
t_3	$\{2, 3\}$
t_4	$\{0, 3\}$

Table 4.2: Sample dataset $\mathcal{D}$

Partition	Partial top-k (phase 1)	Bounds	Complement top-k (phase 2)
G_0	$top(0) \to \{0\}, 4; \{0, 1, 2\}, 2$	$0 \to 2$	$top(1) \to \{0, 1, 2\}, 2$
$\{0, 2\}$	$top(2) \to \{2\}, 3; \{0, 1, 2\}, 2$	$2 \to 2$	$top(3) \to \emptyset$
G_1	$top(1) \to \{0, 1\}, 3$	$1 \to 0$	$top(0) \to \{0, 1\}, 3$
$\{1, 3\}$	$top(3) \to \{3\}, 2$	$3 \to 0$	$top(2) \to \emptyset$

Table 4.3: 2-phase mining over the sample database (Table 4.2) with 2 workers, $k = 2$.

sub-dataset $\mathcal{D}[G]$ provides an excellent speedup and amortizes both the memory consumed by $\mathcal{D}[G]$ and the disk/network cost of its instantiation.

Partitioning the collector Our partitioning of the enumeration tree introduces the drawback that the top-k CIS of an item may be generated by any worker, without the possibility of predicting which ones. A naive solution would make each worker compute a local top-k for all items, then merge all local top-k into the exact top-k. This would require enumerating up to $k \times |\mathcal{I}|$ CIS per worker, thus exploring a much larger fraction of the CIS space than the centralized TopPI and significantly limit the scalability.

Instead, we rely on the following idea: if the worker responsible for a group G collects only $top(i), \forall i \in G$, overall we would miss some results, but only a few. Indeed, as for $i \in \mathcal{I}$ a worker generates CIS combining i with smaller items of the dataset (*i.e.* more frequent items, thanks to the pre-processing), those are likely to have high support. But a few CIS in the actual $top(i)$ may be produced by another group's worker.

Consequently, we run distributed TopPI as a two-phase mining process. The CIS enumeration is distributed identically in both phases. We illustrate this process with the example in Table 4.3. In the first phase (Figure 4.5, job ②), the worker only collects itemsets for items in G (as described by the previous paragraph). This is done by restricting the iterated set at line 8 in Algorithm 4. This phase produces a first, partial version of each item's top-k. It also writes a side output file which lists $lowestSupportIn(top(i)), \forall i \in \mathcal{I}$, denoted "Bounds" in Table 4.3 In the second phase (Figure 4.5, job ③), this list is shared and loaded by each worker, allowing them to pre-fill their local top-k collector as if it benefited from the global results of the first phase. This, in turn, allows the computation of relatively high dynamic thresholds. Each worker only collects itemsets for items $i \notin G$ to generate the complement top-k. In this case, the insertion test (line 10, Algorithm 4) is "$i \notin G$ and $bound \geq lowestSupportIn(top(i))$". A final MapRe-

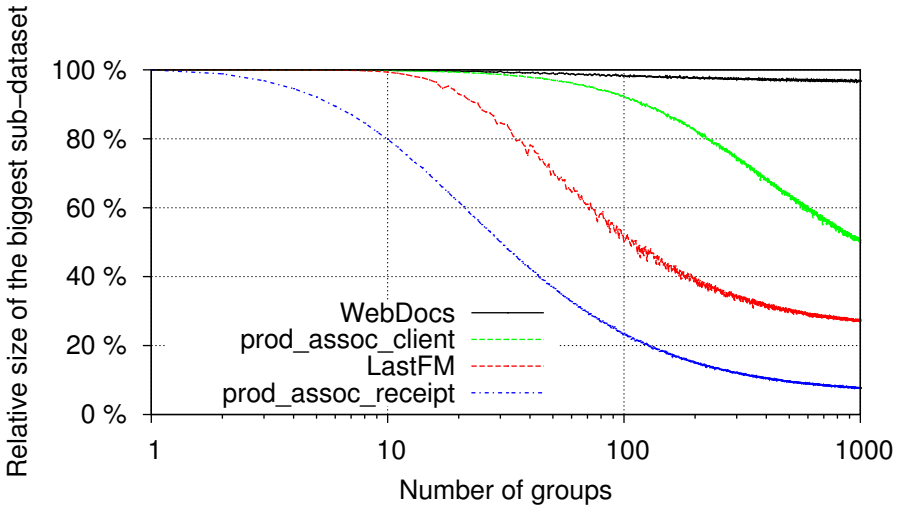


Figure 4.6: Size of the biggest sub-dataset relatively to the original dataset's size, with respect to the number of groups

duce job is executed to merge each item's partial top-k and complement top-k (Figure 4.5, job ④).

As we show in Section 4.4.4, in practice this two-phase process is scalable. The CIS enumeration is always well divided among workers. We showed theoretically that the second phase is the exact complement of phase one (which prunes too much) to overall achieve the same task as the centralized version. In practice, the collectors' pre-filling in the second phase allows each worker to benefit from the others' work in the first phase, therefore mining is much shorter and accurately targeted to complete the results of phase one.

4.3.3 Creating sub-datasets in a distributed environment

In a distributed environment, we partition the items set $\mathcal{I}$ in item groups G and assign each group to a worker. This worker needs to load in memory the sub-dataset of G, denoted $\mathcal{D}[G] = \langle t \in \mathcal{D} \mid t \cap G \neq \emptyset \rangle$. As most transactions contain items belonging to different groups, this does not result in a partition of $\mathcal{D}$. We now study how transactions are distributed in the cluster, and motivate our implementation choices.

Figure 4.6 shows how the size of the biggest sub-dataset evolves with respect to the number of groups. The biggest sub-dataset is the first one, because it includes the most frequent item, after our re-indexing by decreasing frequency. We firstly observe that the result is not consistent among datasets. In the worst case, *WebDocs*, the biggest sub-dataset is always almost equal to the complete one, even when the corresponding worker has to explore a thousand times smaller CIS space. The length of transactions in *WebDocs* accounts for this originality: its transactions contain 177 items, on average. But even in the best case (`prod_assoc_receipt`, in which transactions are much shorter), when $\mathcal{I}$ is divided in 100 partitions this sub-dataset weights much more than 1% of the original one.

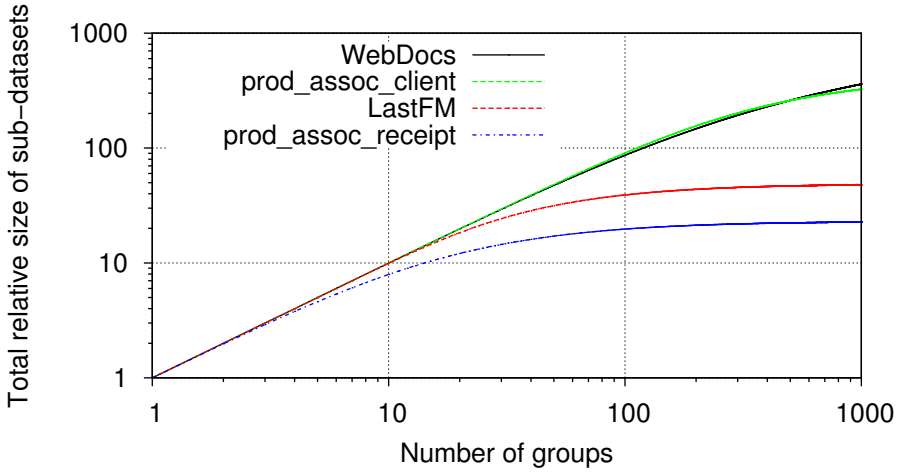


Figure 4.7: Total size of sub-datasets relatively to the original dataset's size, with respect to the number of groups

We obtain similar results for the other sub-datasets, hence the total size of sub-datasets increases in some cases linearly with the number of groups, as shown by Figure 4.7.

These measures show that the distributed variant of TopPI is duplicating a non-negligible amount of data. Indeed the first priority of this version is to distribute and balance the CPU load. This was motivated by the *WebDocs* dataset, with which TopPI takes more than 9 hours to terminate, even for $k = 10$. As shown by our experiments, our single server can mine the other datasets in minutes.

Ideally, in order to minimize this data duplication, we should create as many groups as the number of workers available in the cluster. However, in our setting a dataset like `prod_assoc_receipt` exhausts workers' memory if we only create 4 groups (one for each worker of the production cluster). According to Figure 4.6, we can fit sub-datasets in memory by setting the number of groups to 100, making each machine analyze sequentially 20 different sub-datasets. But, according to Figure 4.7, in this case the cluster have to generate 500GB of sub-datasets.

This raises a new question: what is the most effective way to instantiate sub-datasets? Using the Hadoop framework 3 methods are possible :

1. Use a MapReduce job over $\mathcal{D}$, where for each transaction t the `map` function writes a tuple $\langle t, gid \rangle, \forall gid \in \{group(i) \mid i \in t\}$. Hence each `reduce` function has an instance of $D[G]$ as input. This is close to the method adopted by PFP [37]. While it is intuitive, in practice it outputs much data in MapReduce's shuffle'n'sort phase, which is highly unusual for a MapReduce job. Such jobs often require a particular tuning of the system.

2. Copy $\mathcal{D}$ in the distributed cache, such that each worker machine holds a complete copy on its hard drive. Then, each task has a to instantiate its sub-dataset by scanning this local copy. Although this complete scan is not negligible when the dataset's size is over a gigabyte, this method is the fastest in terms of network transfers (especially with two-phases mining).

3. Use the HDFS API to scan $\mathcal{D}$ at the beginning of each mining task. This

is the less space consuming for workers but incurs much network traffic, especially if more than one mining task is running on a single machine.

Preliminary experiments on *LastFM* and *WebDocs* show that these methods are roughly as fast as each other, but also suggest that such results may be highly platform-dependant. The projection via the distributed cache has been chosen as it is the most robust (it does not require Hadoop tuning) and 5 to 50% faster.

4.4 Evaluation

We now assess experimentally that TopPI outperforms naive or existing solutions, and validate our parallelization strategy on a shared-memory system and over Hadoop. The following run-times and speedups are averaged over three attempts. We start with Section 4.4.1 by showing example results from *LastFM* and our tickets. Then we study our optimizations' impact on TopPI's run-time, in Section 4.4.2. Experiments are then separated by platform.

On the *server* configuration (Section 4.4.3), we compare TopPI with a baseline implementing the most straightforward solution, *i.e.* running a top-k frequent CIS miner over each item's projected dataset. We also confirm that TopPI shows a very good speedup when allocating additional work threads, especially on our biggest datasets.

Experiments on the *cluster* configuration, presented in Section 4.4.4, follow the same methodology. We first compare TopPI's run-time with PFP, the closest available algorithm, then confirm that TopPI's distribution strategy divides the workload among workers.

TopPI is able to mine all our datasets on all our platforms for k in our target parameter range, *ie.* $k \in [1, 100]$. However, due to our baselines' limitations or to the limited availability of our experimental platforms, some datasets will not appear in some figures. In such cases our experiments with TopPI on the missing datasets will be mentioned in comments. We do not evaluate TopPI on `demo_assoc` because all its items are not comparable: it contains both wide and narrow product categories, along demographic attributes. Moreover the resulting items' frequency distribution (see Figure 3.4, p.35) does not show the long tail pattern.

4.4.1 Example itemsets

From `prod_assoc_receipt`: Itemsets with high support can be found for very common products, such as milk[1]: "milk, puff pastry" (152,991 occurrences), "milk, eggs" (122,181) and "milk, chocolate spread" (98,982). Although this particular milk product was bought 5,402,063 times (i.e. in 1.85% of the transactions), some of its top-50 associated patterns would already be out of reach of traditional CIS algorithms: "milk, chocolate spread" appears in 0.034% transactions.

Interesting itemsets can also be found for less frequent (tail) products. For example, "frangipane, puff pastry, sugar" (522), shows the basic ingredients for french king cake. We also found evidence of some sushi parties, with itemsets

[1]Brands and packaging details were removed.

Dataset	Complete TopPI	Without dyn. th. adjustment	Without prefix short-cutting	Without both
LastFM	116	177 (+53%)	150 (+29%)	243 (×2)
`prod_assoc_receipt`	222	1136 (×5)	230 (+4%)	13863 (×62)
`prod_assoc_client`	661	Out of memory	4177 (×6)	Out of mem.

Table 4.4: TopPI run-times (in seconds) on our datasets, using 32 threads and $k = 50$, when we disable the operations proposed in Section 4.2.

such as "nori seaweed, wasabi, sushi rice, soy sauce" (133). We observe similar patterns in `prod_assoc_client`.

From LastFM: When executed on the *LastFM* dataset, TopPI finds itemsets grouping artists of the same music genre. For example, the itemset "Tryo, La Rue Ketanou, Louise Attaque" (789 occurrences), represents 3 french alternative bands. Among the top-10 CIS that contain "Vardoger" (a black-metal band from Norway which only occurs 10 times), we get the itemset "Vardoger, Antestor, Slechtvalk, Pantokrator, Crimson Moonlight" (6 occurrences). TopPI often finds such itemsets, which, in the case of unknown artists, are particularly interesting to discover similar bands.

4.4.2 Contributions impact

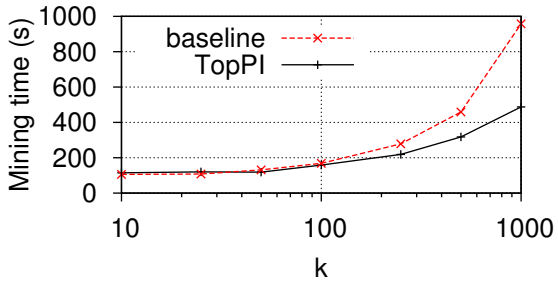
We now validate the individual impact of our contributions: the dynamic threshold adjustment in *startBranch*, described in Section 4.2.5, and the prefix short-cutting in *prune*, presented in Section 4.2.4. To do so, we make these features optional in TopPI's implementation and evaluate their impact on the execution time. Finally we disable both features, to observe how TopPI would perform as a simple pruning by top-k-collector polling in LCM.

Table 4.4 compares the run-time measured for these variants against the fully optimized version of TopPI's, on all our datasets when using the full capacity of our server. We use $k = 50$, which is sufficient to provide interesting results. We cannot experiment on *WebDocs*, as the complete TopPI already takes 9 hours to complete.

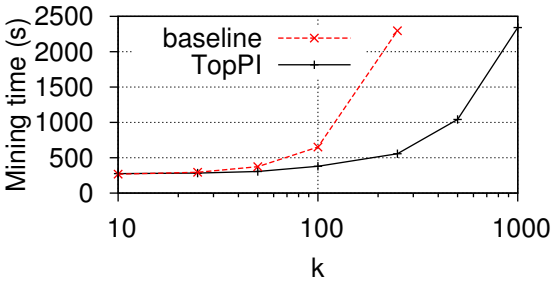
Disabling dynamic threshold adjustment implies that all projected datasets created during the CIS exploration carry more items. Hence intermediate datasets are bigger. This slows down the exploration but also increase the memory consumption. Therefore, without dynamic threshold adjustment, TopPI runs out of memory on `prod_assoc_client`.

When we disable prefix short-cutting in the *prune* function, it has to evaluate more extensions in Algorithm 4, lines 8–11. Hence we observe greater slowdowns on datasets having longer transactions: on average, transactions contain 12 items in `prod_assoc_receipt`, 50 items in *LastFM* , and 213 in `prod_assoc_client`.

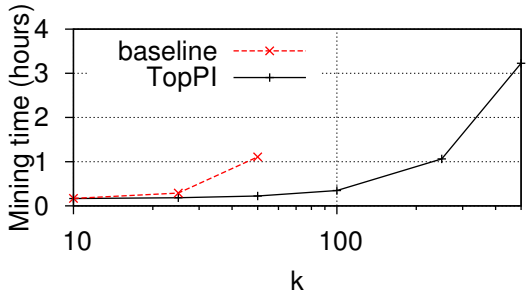
The *prune* function is even more expensive if we disable both optimizations, as potential extensions are not only all evaluated but also more numerous. This experiment shows that it's the combined usage of dynamic threshold adjustment and prefix short-cutting that allows TopPI to mine large datasets efficiently.



(a) *LastFM*



(b) `prod_assoc_receipt`



(c) `prod_assoc_client`
(11 hours at $k = 1000$ for TopPI)

Figure 4.8: TopPI and baseline run-times (in seconds), using 16 threads

4.4.3 TopPI on shared-memory systems

Comparison to a global top-k CIS miner

On a single server, we start by comparing TopPI to its baseline. It is the most straightforward solution to our problem statement: given the parameter k, the baseline applies a *global* top-k CIS mining algorithm on the projected dataset $\mathcal{D}[i]$, for each item i in $\mathcal{D}$ occurring at least twice.

We implemented TFP[24, 70] to serve as the top-k miner. It has an additional parameter l_{min}, which is the minimal itemset size. In our case l_{min} is always equal to 1, but this is not the normal use case of TFP. For a fair comparison, we added a major pre-filtering: $\forall i$, when projecting i on $\mathcal{D}$, we keep only the items having one of the k highest supports in $\mathcal{D}[i]$. Hence each invocation of TFP is done on a reduced dataset, containing at best k items only — in other words, the baseline also benefits from a dynamic threshold computation. The baseline cannot handle our datasets without this optimization.

The baseline also uses the occurrence delivery from our input dataset implementation (*ie.* instant access to $\mathcal{D}[i]$). Its parallelization is obvious, hence both solutions use 16 threads, the number of physical cores on *server*.

Figure 4.8 shows the run-times on our datasets when varying k, not including the time necessary to load each dataset in memory. Both solutions are equally fast for $k = 10$, but as k increases TopPI shows better performance. The baseline even fails to terminate in some cases, either taking over 8 hours to complete, or running out of memory. Instead TopPI can extract 50 CIS per item from `prod_assoc_client` in 13 minutes, or even 500 CIS per item out of the 320 million receipts of `prod_assoc_receipt` in less than 20 minutes.

On `prod_assoc_receipt` and `prod_assoc_client`, the exponential increase

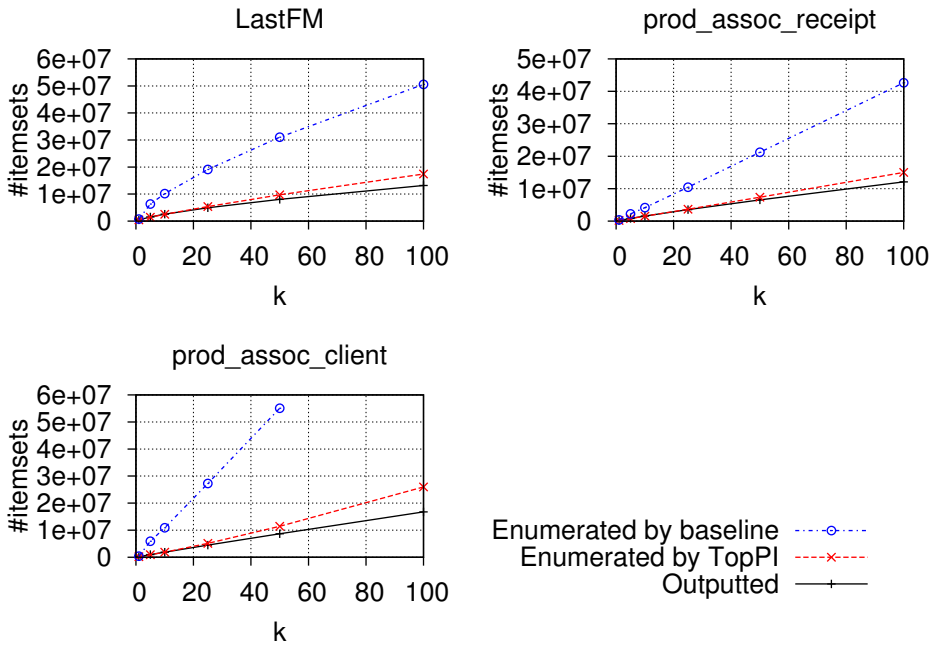


Figure 4.9: Comparison of the number of itemsets explored by each algorithm with the number of itemsets actually outputted.

of run-time for $k \geq 200$ is explained by the increasing number of items having less than k CIS. In such cases, TopPI's mining becomes equivalent to frequent CIS mining with $\varepsilon = 2$. In a discovery setting we only need 10 to 50 CIS per item so such complete exploration of CIS branches only occur for rare items (having a support below 100 or 10, as shown p. 60), where even a complete exploration is extremely fast.

WebDocs is missing in Figure 4.8: even with 31 threads and $k = 10$, TopPI takes 9.5 hours to complete. In the same setting, the baseline completes the top-k mining of only 3% of the available items in 24 hours. This is surprising, as in the frequent itemsets mining community *WebDocs* is always classified as a sparse dataset, *ie.* it does not contain many frequent CIS. But when running item-centric CIS mining, this is not the case: 2,242,989 items occur in two transactions or more in *WebDocs*. Some items also yield unusually large projected datasets, typically a thousand transactions, each containing 20,000 items (on average). Such projected datasets are a worst case for both TopPI and its baseline, which are optimized for datasets where transactions are much more numerous than items.

To better understand the performance difference between TopPI and the baseline, we trace the execution of each algorithm to evaluate the number of itemsets enumerated. Figure 4.9 reports this result and compares it to the number of itemsets actually present in the output. Ideally, the algorithm should only enumerate outputted solutions. As explained in Section 4.2.3, the problem of item-centric CIS mining is not anti-monotone, so both solutions have to enumerate a few additional itemsets to reach some solutions. Thanks to the the use of appropriate heuristics to guide the exploration, TopPI only enumerates a small fraction of discarded itemsets, which explains its good performance.

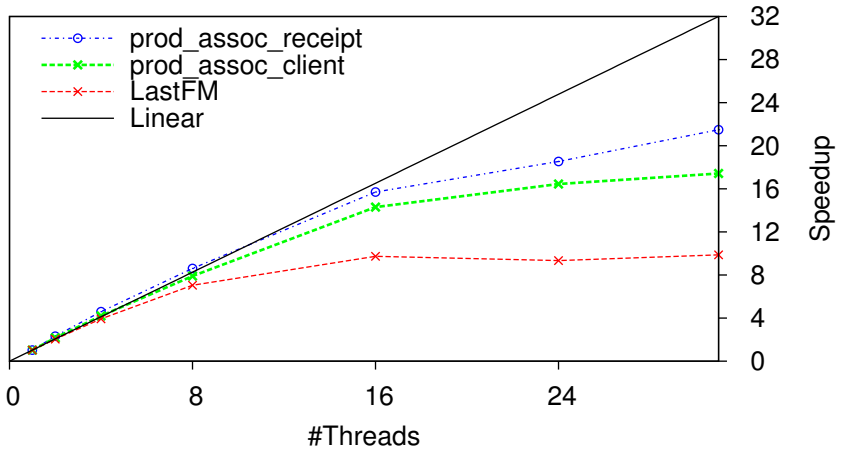


Figure 4.10: TopPI speedup on *server* (with 16 physical cores and $k = 50$)

Conversely, TFP enumerates a significant amount of itemsets that do not contribute to the output. While it is highly efficient when mining the projected datasets of the most frequent items, for the others TFP struggles to find the frequency threshold which guarantees the correctness of its top-k. When supports are counted in thousands, many more CIS have a similar support. It also does not perform closure extensions, but merges enumerated itemsets into their closure. Hence its deeper traversal of the CIS space.

As each $\mathcal{D}[i]$ is mined independently for all items i, the baseline cannot amortize the thresholds adaptations from a branch to another, so this result would likely be also observed with another top-k CIS mining algorithm. This highlights the need for a specific algorithm for item-centric CIS mining.

Scalability

We now evaluate how the centralized parallel version of TopPI leverages additional threads for the CIS exploration. We report the mining time speedup with respect to the number of mining threads in Figure 4.10. As this machine has 16 physical cores, we cannot expect a perfect speedup with more than 16 threads. We use up to 31 threads only; because we only have 16 CPU cores (with hyper threading) this leaves the last virtual thread to the system or the Java Virtual Machine's garbage collector.

Overall, TopPI achieves excellent scalability. Depending on the size of the dataset, the limiting factor is either the CPU, in which case adding more cores provides linear scalability (`prod_assoc_receipt` and `prod_assoc_client`), or the memory bus, in which case the scalability is sub-linear above 8 threads (*LastFM*).

When 31 threads are used in parallel, 31 branches of the CIS tree are explored simultaneously and therefore held in memory, with the corresponding stack of reduced datasets. Though, even with 31 threads, the peak memory usage of TopPI remains reasonable. This is confirmed by Figure 4.11, where we report TopPI's peak RAM consumption for *LastFM*, `prod_assoc_receipt` and `prod_assoc_client`. As expected, this peak increases with k. Not only the top-

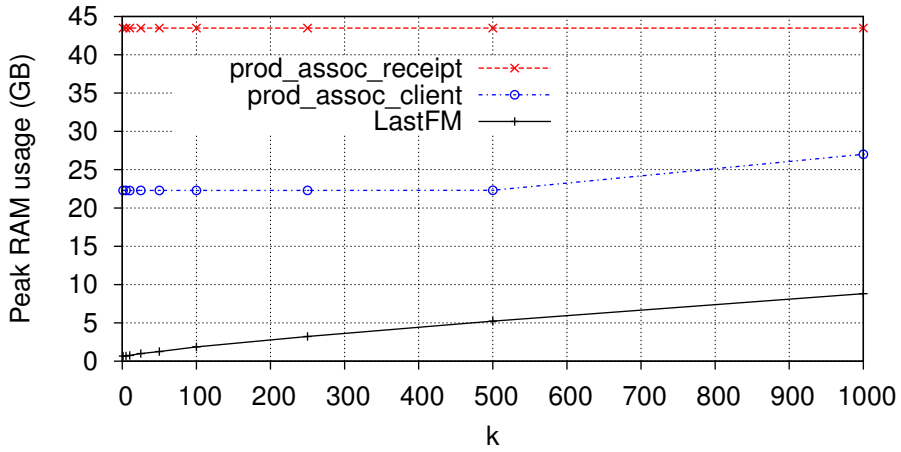


Figure 4.11: TopPI peak memory usage, using 31 threads

k collector is expanded accordingly, but the *prune* function also guides TopPI towards deeper tree explorations to generate additional itemsets, thus keeping more reduced datasets in memory. However this is less visible for `prod_assoc_receipt` and `prod_assoc_client`, where the size of these objects is negligible compared to the initial dataset and its indexes.

WebDocs is missing again in this study. As TopPI takes 9.5 hours to complete with 31 threads and $k = 10$, we could not afford the memory study for $k \in [1, 1000]$, nor the run-time measurement with a single thread.

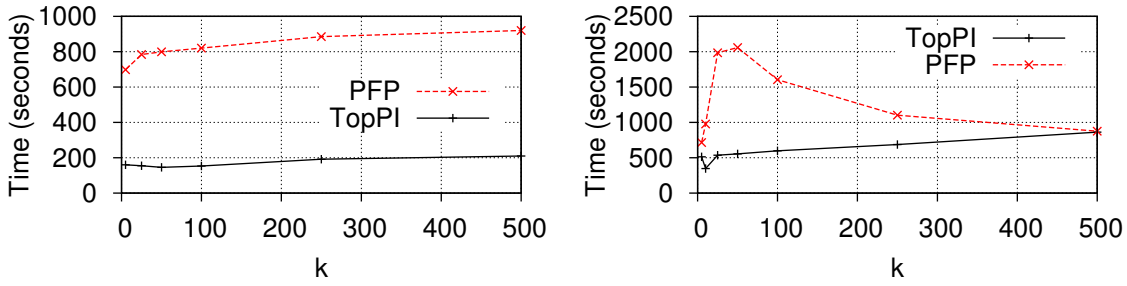
Overall, these results confirm that TopPI is fast and scalable. Even on common hardware: TopPI is able to mine *LastFM* with $k = 50$ on a laptop with 4 threads (Intel Core i7-3687U) and 6 GB of RAM in 16 minutes, while this operation takes 13 minutes for PFP on our *cluster* configuration with 200 cores, as presented in the next section.

4.4.4 TopPI over Hadoop

Although the multi-threaded version of TopPI is able to mine millions of transactions on a single server or even a laptop, mining long tail itemsets from a more challenging dataset like *WebDocs* requires more computational power. A 25GB file like `prod_assoc_receipt` and its memory requirement may also be out of reach of the analyst's machine and, as in the Datalyse project, she may have a Hadoop cluster at hand instead of a single high-end server. These examples motivated the development of the Hadoop variant of TopPI, presented in Section 4.3.2 and evaluated hereafter.

Performance comparison to PFP

In the following experiment we compare our Hadoop variant to PFP [37], the closest algorithm to TopPI: it returns, for each item, at most k frequent item-



(a) *LastFM* (b) *Supermarket*

Figure 4.12: TopPI and PFP run-times on a 200-tasks Hadoop cluster

sets containing that item. We use the implementation[2] available in the Apache Mahout Library [58]. We use 50 worker machines and launch 4 tasks per machine, allocating 5GB of memory to each. Hence, up to 200 MapReduce tasks are launched in parallel. Even with this computing power at hand, PFP is not able to mine `prod_assoc_receipt`, `prod_assoc_client`, nor *WebDocs* — even if we allocate more memory to mining tasks. As only *LastFM* remains, for this comparison we introduce a new dataset, *Supermarket*, which is a preliminary version of `prod_assoc_receipt`. This 2.8GB file contains 54.8 million receipts over 389,372 items[3], from 87 supermarkets over 27 months. Its items distribution is very similar to `prod_assoc_receipt`.

As PFP is not multi-threaded, to ensure the fairness of the comparison TopPI uses a single thread per task. Mahout's documentation recommends to create one task for every 20 to 30 items, thus PFP runs 20,000 tasks on *LastFM* and 17,000 on *Supermarket*. TopPI launches 200 tasks. The measured run-times are presented in Figure 4.12.

On *LastFM*, increasing the value of k has only a moderate influence on TopPI's run-time because, when distributed among 200 tasks, the mining phases are very short so the reported execution time mostly consists of Hadoop I/O. TopPI remains 3 to 4 times faster that PFP.

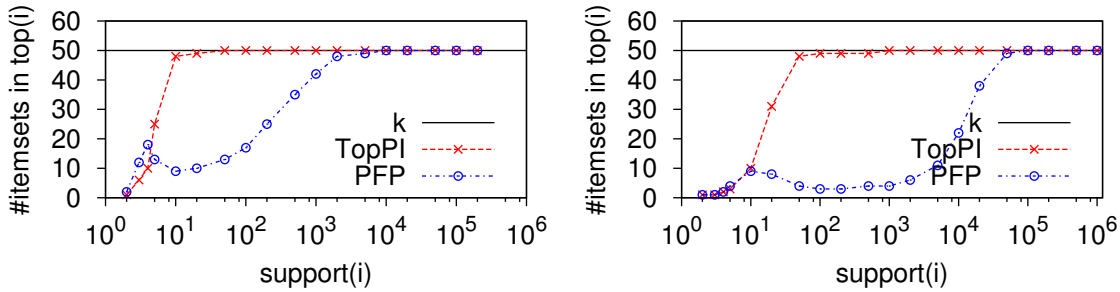
On *Supermarket*, PFP's run time is surprisingly uncorrelated to k. PFP can also not terminate correctly above $k \geq 1000$ (TopPI can still run with $k \geq 2000$). This may be caused by an implementation corner case, or by the difference between the two algorithms' problem statements — TopPI provides stronger guarantees on long tail itemsets.

Comparing PFP and TopPI's output

TopPI and PFP both rely on an item-centric approach and return, for each frequent item $i \in \mathcal{I}$, at most k frequent itemsets containing i. But TopPI adds the guarantee that these k itemsets are closed and the most frequent ones. We now

[2]Mahout 0.7 provides two implementations of PFP. The alternative one is more memory-consuming, so for these experiments we use the default implementation.

[3]That is almost twice as many products as in the final version. A majority of products in this preliminary set are actually rare and unlabeled products, which makes the qualitative study impractical. They have been removed in the final version.



(a) *LastFM* (b) *Supermarket*

Figure 4.13: Average $|top(i)|$ per item w.r.t. $support_{\mathcal{D}}(i)$ ($k = 50$)

compare the output of TopPI and PFP, using $k = 50$, to understand the impact of these different problem statements on the results returned for long tail items.

Figure 4.13 shows the average number of itemsets outputted for each item, with respect to its frequency. In order to make these measures readable, we average them over item frequency intervals. For the least frequent items, neither TopPI nor PFP manages to output the requested k itemsets. This is expected as an item of frequency 3, for example, can be part of at most 4 CIS. For these items PFP returns more itemsets than TopPI on *LastFM*, but most of them are redundant — both algorithms give the same number of *closed* itemsets. As item frequency increases, TopPI returns more itemsets, with almost filled top-k heaps for all items appearing at least 10 times in the dataset. PFP, however, is unable to find these itemsets and only returns close to k itemsets for items whose frequency is above 1000. The difference is even more striking on *Supermarket*, where PFP provides the required k itemsets per item only for items occurring more than 10,000 times.

This comes from PFP's division of the mining workload: its mining phase fills a single top-k heap per group of items. Thus, an item with a low frequency is unlikely to obtain k itemsets as, during the mining, itemsets corresponding to more frequent items of its group will fill the heap. To mitigate this problem, Mahout's documentation recommends to raise the number of item groups, such that each group only contains 20 to 25 items, but this does not guarantees to obtain a complete top-k per item.

These results show that, unlike PFP, TopPI performs a complete top-k CIS mining for all items, even long tail ones.

Scalability over Hadoop

Figure 4.14 shows the speedup gained by the addition of worker nodes, on *Supermarket*, when $k = 1000$. Here we execute a single mining task per machine, which fully exploits the available resources by running 8 threads in all the available memory, 24GB.

TopPI shows a perfect speedup from 1 to 8 machines (64 cores), and steadily gets faster with the addition of workers. Overall, the total CPU time (summed over all machines) spent in the mining phases (*expand* function) remains stable: from 35,000 seconds on average from 1 to 8 machines, it only raises to 38,500 seconds with 48 machines (using their 384 cores). Above, the speedup remains linear but

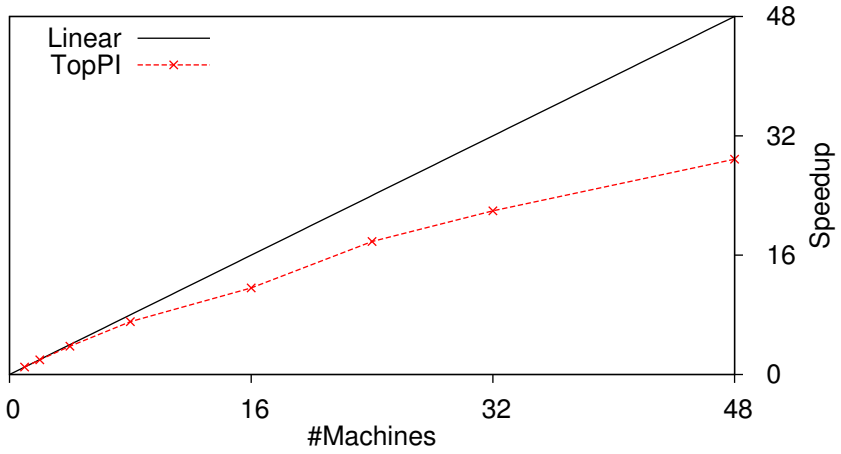


Figure 4.14: TopPI speedup when mining *Supermarket* on Hadoop, using $k = 1000$. Each machine is assigned a single items group and uses 8 threads.

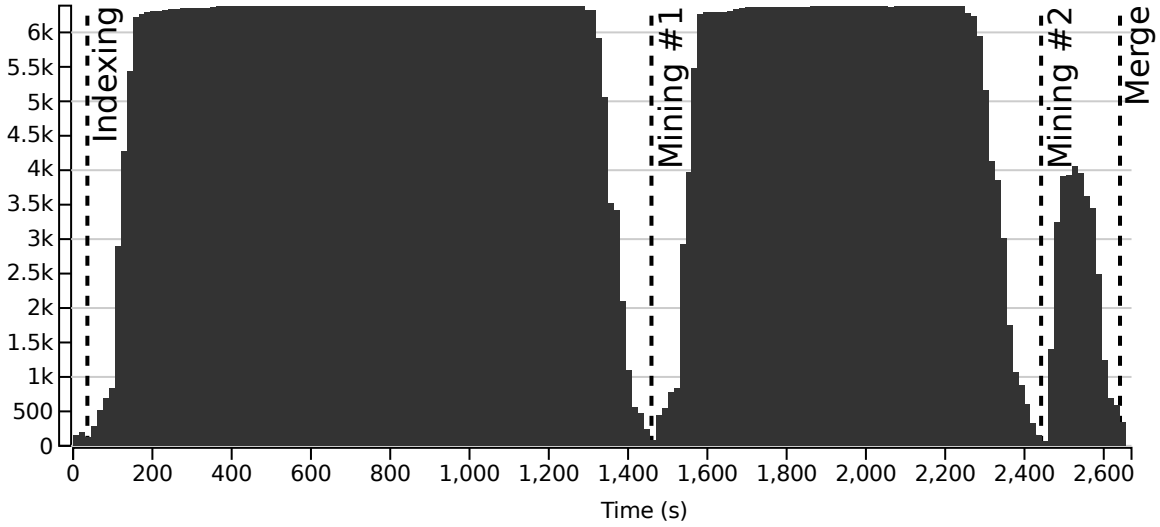


Figure 4.15: TopPI aggregated CPU usage on a 64-machines cluster on *WebDocs* ($k = 10$, 64 tasks running 8 threads each)

below the ideal parallelization. In this configuration, the sweet spot is around 8 tasks. Should the workload increase, TopPI would achieve optimal speedup on larger clusters, as the overall mining time increases and compensates the I/O costs. This is easily understandable, as we selected parameters such that the experiment could also run on a single machine. Above 16 tasks, the I/O overhead represents a more significant fraction of the execution time and the efficiency decreases.

We further confirm TopPI's scalability by mining *WebDocs* with $k = 10$. This computation takes 4570 seconds with 32 machines and 2641 with 64. This represents a 1.7 speedup when doubling the tasks count. This validates the distribution strategy used for the mining phases: the load is well partitioned and does not increase significantly with the number of tasks.

We confirm these observations by measuring the aggregated CPU usage of the whole cluster when using 64 workers. The results are depicted on Figure 4.15. The mining phases correspond to high CPU usage, while I/O operations have low CPU

usage. As we can see, the amount of I/O is not negligible, which explains the lower speedup gain with 64 machines. We also observe that the second mining phase is shorter than the first one. Indeed, as described in Section 4.3.2 (p.49), it leverages the per-item bounds discovered during the first stage. While the first phase returns 4,037,769 itemsets, the second one returns significantly less (999,511), hence its shorter run-time.

The CPU usage distribution shows that both mining tasks complete on all servers in comparable amounts of time. We do not observe straggler tasks significantly delaying TopPI. Hence, TopPI balances well the load among the machines.

4.5 Technology transfer

As TopPI has been delivered early in the Datalyse project, it benefited from the feedback of our industrial partners who could run the program on the production cluster. In this section, we summarize this feedback and its impact on our work.

4.5.1 Run-time impact of sub-datasets instantiation

The first results from industrial tests confirmed that TopPI is able to provide insights on all the available products of a retailer as big as Intermaché. However, sometimes the mining takes hours to complete. Although the analytics cluster supporting these tests is small (4 workers with 4 CPU cores) for datasets counting their size in gigabytes, such run-times are unexpectedly long. Those are caused by the instantiation of sub-datasets, which create many transactions duplicates. This is discussed in Section 4.3.3.

We should also note that this transactions duplication is imposed by our search of *closed* itemsets. Indeed we need complete transactions to perform the first-parent test (see Definition 4, p.10). Another way to run efficiently TopPI over MapReduce would be to relax this constraint, thus allowing the transmission of truncated transactions as does PFP [37]. Relaxing the constraint on closures would however include duplicate information and decrease the results' quality. Hence it can only be considered on extremely sparse datasets (like *Supermarket* or `prod_assoc_receipt`). We also remark that these considerations also apply to distributed CIS mining: our distribution of the exploration could as well be used with jLCM. It would even be simpler for jLCM, which would only require the items indexation and a single mining job (*cf.* Section 4.3).

Overall, the Hadoop variant of TopPI is able to tackle all our datasets, but the perfect speedup quest of distributed CIS mining would require a lot of ad-hoc development. As analyzing a year worth of receipts is not done on a daily basis, considering this run-time as acceptable is up to the analyst's patience and resources. But the evolution of hardware suggests to keep such computations local: mining the top-50 CIS per items from `prod_assoc_receipt` takes 20 minutes on our experimental server (including 15 minutes of dataset loading).

4.5.2 The noise of star products

The other important feedback about TopPI is qualitative. Some of the most frequent products are orders of magnitude more frequent than others. Marketing experts pay special attention to them, and nickname them "star products". These include, for example, cola, shopping bags or grated cheese. In TopPI's use case (dataset discovery) they are considered as noise: because they are hyper-frequent, TopPI finds them combined with almost every other product, hence showing one or more useless result(s) in those products' top-k. For example, the association of most products with shopping bags does not provide any insight on the product's associations — it only shows that customers often forget their shopping bag, and have to buy the ones sold at the store. We also observe this phenomenon in our *LastFM* dataset. The most popular artists, like *The Beatles* or *Radiohead*, appear among the most frequent associations of almost all other artists (especially for large values of k). Such results are artifacts of "pop stars".

As a workaround, for retail data our industrial partners filtered out star products when curating transaction sets (*cf.* Section 3.2.1 p.30). However, the list of star products is likely to evolve and will be difficult to maintain over the long-term in the analytics system. Also, there may be products having interesting associations with star products, or on the other hand some products may appear as noise unexpectedly.

This shows the limits of ranking by frequency and motivated our first experiments with the p-value, and the development of CAPA (Chapter 5). The second version of TopPI delivered to our industrial partners includes a final re-ranking of itemsets in $top(i), \forall i \in \mathcal{I}$, by decreasing correlation with the corresponding i. We further discuss this method in Section 6.2 (p.85), after studying alternative ranking functions in Chapter 5.

4.6 Conclusion

To the best of our knowledge, TopPI is the first algorithm to formalize and solve at scale the problem of mining item-centric top-k closed itemsets, a semantic more appropriate to early explorations of long-tailed datasets. TopPI is able to operate efficiently on long tail content, which is out of reach of standard mining and global top-k algorithms. Instead of generating millions of itemsets only containing the very few frequent items, TopPI spreads its exploration evenly to mine a fixed number of k itemsets for each item in the dataset, including rare ones. This is particularly important in the context of Web datasets, in which the long tail contains most of the information [19], and in the retail industry, where it can account for a large fraction of the revenue [4].

TopPI scales from multi-cores to Hadoop clusters, and is able to analyze a year of activity of 1884 Intermarché stores in minutes on a single server, allowing data analysts to easily obtain key associations for any product, including very infrequent ones. We demonstrate that TopPI's results are interesting by themselves, but the following chapter suggests that TopPI can also be used as a building block to obtain alternative results based on refined interestingness measures.

CHAPTER 5

Sorting association rules with CAPA

Contents

Although item-centric mining with TopPI provides an intuitive organization of each item's associations, its ranking by decreasing frequency does not always select the most interesting ones — see the discussion in Section 4.5.2. Existing work proposes many functions to measure the quality of association rules without additional knowledge [17, 36]. However, we cannot know from existing studies which measure is able to assign top rankings to significant associations in the retail domain.

In this chapter, we want to perform targeted analytics over the 3 Datalyse mining scenarios, described in Section 3.2.1 (p.30). The analyst provides input items or product categories of interest, and expects the system to directly highlight a few dozen results. This is very common in the marketing domain, when checking the impact of advertisement operations or evaluating their opportunity. As the analyst not only provides a non-negligible frequency threshold, but also one or more conditions on the itemsets to be generated, the computation is tractable on our datasets. Still, it generates a very high number of rules, typically in the order of millions.

Building a system able to highlight automatically interesting associations therefore raises two questions:

- when sorting association rules, how different are the top results proposed by quality measures proposed in the literature?
- which quality measure(s) highlights association rules of the greatest interest to retail analysts?

In this chapter we address these questions with the following contributions:

- We present in Section 5.1 CAPA (Comparative Analysis of PAtterns), a framework to compare rule rankings produces by 39 interestingness measures [17, 36].
- We automatically compare the 39 rankings in Section 5.2, and distinguish 5 groups of similar measures.
- For this automatic evaluation we introduce a new ranking similarity measure: Normalized Discounted Correlation Coefficient (*NDCC*).
- We conduct a user study with two experienced domain experts from Intermarché, in Section 5.3, in order to evaluate which of our 5 groups of interestingness measures are meaningful in their domain.

We conclude in Section 5.4.

5.1 The CAPA framework

Looking back at our complete workflow (depicted p.27) CAPA belongs to the mining and exploitation steps. We start by describing and motivating how each step is implemented and integrated in our system.

Our goal is to enable analysts to test and compare different interestingness measures on a set of transactions $\mathcal{D}$ produced by our curation module. Thus an analyst firstly specifies one of our 3 mining scenarios (`demo_assoc`, `prod_assoc_receipt` or `prod_assoc_client`) and a minimum support threshold ε. When using CAPA analysts focus on one or several targets: categories in the case of `demo_assoc`, products in the case of the other two. Thus, in addition to the common model for itemset mining presented in Section 2.1.1 (p.8), CAPA also requires the definition of a *target items* set $\mathcal{B} \subset \mathcal{I}$. Once these settings are determined, the system generates a list of association rules ranked using different interestingness measures.

In Section 5.1.1 we show that different quality measures from the literature can highlight very different association rules. We also detail the 39 measures implemented in CAPA and the data needed for our rankings comparison. CAPA mines association rules whose consequent side is a single target item from $\mathcal{B}$: Section 5.1.2 shows how we modify jLCM to mine this finer set of association rules, and the post-processing step which provides enough data to compute each quality measure for each rule. Section 5.1.3 then describes the exploitation step of CAPA, *ie.* how associations are ranked and presented to the analyst.

5.1.1 Different measures highlight different rules

As discussed in Section 2.1.2 (p.9), given a frequency threshold ε we can find all frequent CIS with jLCM and use them to derive association rules. Association rules were originally selected using thresholds for support and confidence [1]. However, support and confidence do not always coincide with the interest of analysts. Setting thresholds is also not easy and may not be sufficient to filter results.

by confidence	
$\{>65, F$, Aube$\}$	$\rightarrow$ *Dairy*
$\{>65, F$, Aveyron$\}$	$\rightarrow$ *Dairy*
$\{>65, F$, Val de Marne$\}$	$\rightarrow$ *Dairy*
$\{>65, F$, Seine S^t Denis$\}$	$\rightarrow$ *Dairy*
$\{>65, F$, Haute Saone$\}$	$\rightarrow$ *Dairy*
$\{>65, F$, Meuse$\}$	$\rightarrow$ *Dairy*
$\{>65, *$, Aube$\}$	$\rightarrow$ *Dairy*
$\{>65, F$, Haute Vienne$\}$	$\rightarrow$ *Dairy*
$\{>65, F$, Maine et Loire$\}$	$\rightarrow$ *Dairy*
$\{>65, *$, Val de Marne$\}$	$\rightarrow$ *Dairy*

by Piatetsky-Shapiro [53]	
$\{*, *$, Nord$\}$	$\rightarrow$ *Liquids*
$\{*, *$, Nord$\}$	$\rightarrow$ *Soft drinks*
$\{*, *$, Nord$\}$	$\rightarrow$ *Beers*
$\{*, *$, Nord$\}$	$\rightarrow$ *Spreads*
$\{*, F$, Nord$\}$	$\rightarrow$ *Soft drinks*
$\{*, *$, Nord$\}$	$\rightarrow$ *Imported beers*
$\{*, F$, Nord$\}$	$\rightarrow$ *Liquids*
$\{*, F$, Nord$\}$	$\rightarrow$ *Beers*
$\{*, *$, Finistere$\}$	$\rightarrow$ *Butters*
$\{*, F$, Garonne$\}$	$\rightarrow$ *Drugstore*

by Pearson's χ^2	
$\{*, *$, Somme$\}$	$\rightarrow$ *Cut cheese*
$\{*, F$, Somme$\}$	$\rightarrow$ *Cut cheese*
$\{>65, *$, Morbihan$\}$	$\rightarrow$ *Fresh milk*
$\{>65, *$, Somme$\}$	$\rightarrow$ *Cut cheese*
$\{*, *$, Finistere$\}$	$\rightarrow$ *Canned pork*
$\{*, *$, Cotes d'Armor$\}$	$\rightarrow$ *Canned pork*
$\{>65, F$, Morbihan$\}$	$\rightarrow$ *Fresh milk*
$\{*, *$, Nord$\}$	$\rightarrow$ *Beer*
$\{*, *$, Nord$\}$	$\rightarrow$ *Sparkling liquors*
$\{*, *$, Vienne$\}$	$\rightarrow$ *Breakfast biscuits*

Table 5.1: Top-10 demographics association rules, according to different interestingness measures. Rules are denoted $\{$*age*, *gender*, *department*$\}$ $\rightarrow$ *product category*. Product categories were translated to English for clarity. French departments were left unchanged.

For example, in the `demo_assoc` case, using a minimum support of 1000 *j*LCM mines 2,746,418 frequent rules of the form *customer segment* $\rightarrow$ *product category*. Out of these, 15,063 have a confidence of 50% or higher. Hence a number of interestingness measures that serve different analyses needs were proposed in the literature [17, 36].

Table 5.1 shows a ranking of the top-10 rules according to 3 different interestingness measures proposed in [17]. If we denote rules as $A \rightarrow B$, *confidence* is the probability to observe B given that we observed A, i.e., $P(B|A)$. *Piatetsky-Shapiro* [53] measures the difference between the probability to observe A and B together and the expected probability assuming they are independent, i.e., $P(AB) - P(A)P(B)$. *Pearson's* χ^2, measures how unlikely observations of A and B are independent. Clearly, these measures highlight very different associations.

Measure	Formula	Group
One-Way Support	$P(B\|A) \times log_2 \frac{P(AB)}{P(A)P(B)}$	G_1^a
Relative Risk	$P(B\|A)/P(B\|\neg A)$	
Odd Multiplier	$\frac{P(AB)P(\neg B)}{P(B)P(A\neg B)}$	
Zhang	$\frac{P(AB)-P(A)P(B)}{max(P(AB)P(\neg B),P(B)P(A\neg B))}$	
Yule's Q $\diamond$	$\frac{P(AB)P(\neg A\neg B)-P(A\neg B)P(B\neg A)}{P(AB)P(\neg A\neg B)+P(A\neg B)P(B\neg A)}$	
Yule's Y $\diamond$	$\frac{\sqrt{P(AB)P(\neg A\neg B)}-\sqrt{P(A\neg B)P(B\neg A)}}{\sqrt{P(AB)P(\neg A\neg B)}+\sqrt{P(A\neg B)P(B\neg A)}}$	
Odds Ratio $\diamond$	$\frac{P(AB)P(\neg A\neg B)}{P(A\neg B)P(B\neg A)}$	
Information Gain $*\ominus$	$log(P(AB)/(P(A)P(B)))$	
Lift $*\ominus$	$P(AB)/(P(A)P(B))$	
Bayes Factor $*$	$P(A\|B)/P(A\|\neg B)$	
Added Value $*$	$P(B\|A) - P(B)$	G_1^b
Certainty Factor $*$	$(P(B\|A) - P(B))/(1 - P(B))$	
Confidence $*\otimes$	$P(B\|A)$	
Centered Confidence $*$	$P(B\|A) - P(B)$	
Loevinger †	$1 - \frac{P(A)P(\neg B)}{P(A\neg B)}$	
Conviction †	$\frac{P(A)P(\neg B)}{P(A\neg B)}$	
Examples and Counter-examples Rate	$1 - \frac{P(A\neg B)}{P(AB)}$	
Sebag-Schoenauer	$\frac{P(AB)}{P(A\neg B)}$	
Leverage	$P(B\|A) - P(A)P(B)$	
Laplace Correction $*\otimes$	$\frac{support(AB)+1}{support(A)+2}$	
Least Contradiction	$\frac{P(AB)-P(A\neg B)}{P(B)}$	G_2
Accuracy	$P(AB) + P(\neg A\neg B)$	
Pearson's χ^2 $\triangleright$	$\|\mathcal{T}\| \times \left(\frac{(P(AB)-P(A)P(B))^2}{P(A)P(B)} + \frac{(P(\neg AB)-P(\neg A)P(B))^2}{P(\neg A)P(B)}\right)$ $+\|\mathcal{T}\| \times \left(\frac{(P(A\neg B)-P(A)P(\neg B))^2}{P(A)P(B)} + \frac{(P(\neg A\neg B)-P(\neg A)P(\neg B))^2}{P(\neg A)P(\neg B)}\right)$	G_3
Gini Index $\triangleright$	$P(A) \times (P(B\|A)^2 + P(\neg B\|A)^2) + P(\neg A) \times (P(B\|\neg A)^2 +$ $P(\neg B\|\neg A)^2) - P(B)^2 - P(\neg B)^2$	
J-measure	$P(AB)log(\frac{P(B\|A)}{P(B)}) + P(A\neg B)log(\frac{P(\neg B\|A)}{P(\neg B)})$	
Φ Linear Correlation Coefficient	$\frac{P(AB)-P(A)P(B)}{\sqrt{P(A)P(B)P(\neg A)P(\neg B)}}$	
Two-Way Support Variation	$P(AB) \times log_2 \frac{P(AB)}{P(A)P(B)} + P(A\neg B) \times log_2 \frac{P(A\neg B)}{P(A)P(\neg B)} +$ $P(\neg AB) \times log_2 \frac{P(\neg AB)}{P(\neg A)P(B)} + P(\neg A\neg B) \times log_2 \frac{P(\neg A\neg B)}{P(\neg A)P(\neg B)}$	
Fisher's exact test	$\frac{\binom{\|\mathcal{T}\|\times P(B)}{\|\mathcal{T}\|\times P(AB)}\binom{\|\mathcal{T}\|\times P(\neg B)}{\|\mathcal{T}\|\times P(A\neg B)}}{\binom{\|\mathcal{T}\|}{\|\mathcal{T}\|\times P(A)}}$	
Jaccard	$P(AB)/(P(A) + P(B) - P(AB))$	
Cosine	$\frac{P(AB)}{\sqrt{P(A)P(B)}}$	
Implication Index	$\frac{support(A)support(B)-\|\mathcal{T}\|support(AB)}{\sqrt{\|\mathcal{T}\|support(A)support(\neg B)}}$	
Kappa Coefficient	$2\frac{\|\mathcal{T}\|support(AB)-support(A)support(B)}{\|\mathcal{T}\|support(A)+\|\mathcal{T}\|support(B)-2support(A)support(B)}$	
Two-Way Support	$P(AB) \times log_2 \frac{P(AB)}{P(A)P(B)}$	G_4
Piatetsky-Shapiro	$P(AB) - P(A)P(B)$	
Klosgen	$\sqrt{P(AB)}max(P(B\|A) - P(B), P(A\|B) - P(A))$	
Specificity	$P(\neg B\|\neg A)$	
Recall	$P(A\|B)$	G_5
Collective Strength	$\frac{P(AB)+P(\neg B\|\neg A)}{P(A)P(B)+P(\neg A)P(\neg B)} \times \frac{1-P(A)P(B)-P(\neg A)P(\neg B)}{1-P(AB)-P(\neg B\|\neg A)}$	

Table 5.2: Interestingness measures of a rule $A \rightarrow B$. $*$, $\triangleright$ indicate measures that produce the same rule ranking when a single target is selected. $\diamond$, †, $\ominus$, $\otimes$ indicate measures that always produce the same rule ranking. $|\mathcal{T}|$ is the number of transactions. $P(A) = support(A)/|\mathcal{T}|$.

Algorithm 6: Itemsets extraction

Data: dataset $\mathcal{D}$, minimum support threshold ε, target items $\mathcal{B}$
Result: Output all frequent closed itemsets in $\mathcal{D}$ that contain an item from $\mathcal{B}$

```
1 begin
2     foreach e ∈ B do
3         if demo_assoc then
4             D' = {T \ B | T ∈ D[e]}
5         else
6             D' = D[e]
7         jLCM (∅, {e}, D', ε)
```

Table 5.2 summarizes the interestingness measures we use in this work. The first column contains the name of the measure, the second its expression. The last column will be referred to later. Our implementation of *Fisher's exact test* is more detailed in Annex A (p.99). The quality measures we selected require at most $P(A)$, $P(B)$ and $P(A \cup B)$ because, given $|\mathcal{D}|$, other probabilities like $P(B|A)$ or $P(A\neg B)$ can be derived from them.

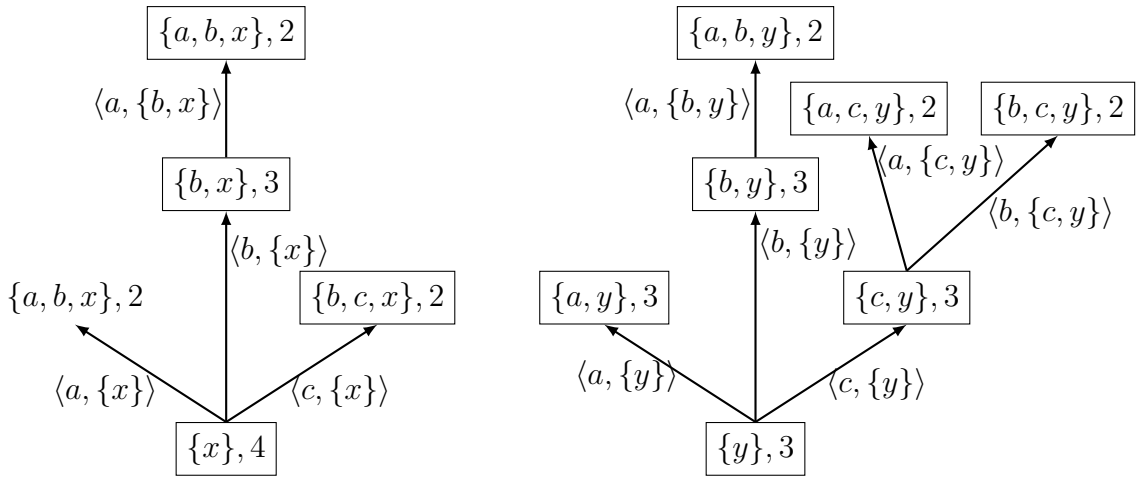
5.1.2 Mining step: constraining *j*LCM to targeted itemsets

The goal of the mining step in CAPA is to find all association rules $A \rightarrow b$ such that $b \notin A$, $A \subset \mathcal{I}$ and $b \in \mathcal{B}$. A mining scenario may also require that $A \cap \mathcal{B} = \emptyset$ — this is the case for `demo_assoc`, where $\mathcal{B}$ are products categories that cannot appear in rules' antecedents, which are demographic attributes. Evaluating the interestingness of an association rule also requires computing $support(A)$, $support(\{b\})$ and $support(A \cup \{b\})$. We do not use TopPI, because in this context we cannot predict which value is appropriate for its k parameter. Instead we have a minimum threshold which allows us to use jLCM (fully presented in Section 2.1.2, 9).

jLCM is integrated in CAPA as presented in Algorithm 6. We use a dataset $\mathcal{D}$, prepared by our curation module (see Section 3.2.1 p.30). For each target item e, we launch jLCM over transactions containing e. In `demo_assoc`, we also have to remove all remaining target items once one has been selected (see $T \setminus \mathcal{B}$ line 4). In other cases this filtering is not necessary, because both the antecedent and the consequent are products. Algorithm 6 is easily parallelized: we start a jLCM threads pool and launch each thread on a different target item e. We set the frequency threshold to different values for each scenario:

- 1000 for `demo_assoc`
- 1000 for `prod_assoc_receipt`
- 10,000 for `prod_assoc_client`

These thresholds are set depending on the cardinalities of each scenario's transactions. These values have also been validated by our associated marketing experts, who even consider that a thousand purchases is an extreme minimum when studying nation-wide trends.



$$\mathcal{D} = \{\{a, b, c, x, y\}, \{a, c, y\}, \{a, b, x, y\}, \{b, c, x, y\}\}$$

Figure 5.1: jLCM enumeration trees over an example dataset $\mathcal{D}$, with $\varepsilon = 2$, in the `demo_assoc` case. Hence items in $\mathcal{B} = \{x, y\}$ are the only ones allowed as rules' consequent. An edge $\langle i, I\rangle$ represents an recursive invocation of jLCM (I, i). Only boxed nodes (closed itemsets satisfying the first-parent test) are returned.

Figure 5.1 gives an example of the resulting enumeration. It illustrates that Algorithm 6 allows CAPA to obtain itemsets that can be converted to the desired association rules $A \to b$. It also computes $\mathit{support}(b)$ and $\mathit{support}(A \cup b)$, but we still need $\mathit{support}(A)$, Thus, while CIS are outputted by jLCM we also materialize the set of all antecedent itemsets whose support needs to be evaluated as a prefix tree. In a post-processing step, we read $\mathcal{D}$ once to compute their support. This two-step approach avoids the computation of many itemsets that never appear as a rule antecedent.

5.1.3 Exploitation step: ranking and displaying associations

The last step in CAPA is to show ranked lists of association rules to the analyst. She should be able to switch easily from a ranking to another, hence we precompute all rankings just after the mining step.

The quality measures we selected require $P(A)$, $P(B)$ and $P(A \cup B)$. We denormalize the results of the mining phase in order to store these 3 probabilities along each A and B. Thus each row represents an association rule and has enough information to compute its score. This table is then augmented with 39 columns, one for each measure implemented. The complete table is stored in a relational database in order to ease the exploration by the final application.

The final component of CAPA is a Web application allowing the analyst to explore this augmented table. In any scenario (`demo_assoc`, `prod_assoc_receipt` or `prod_assoc_client`), the analyst picks a measure and selects a target product or category, or a set of target products or categories. This selection is made by navigating the product taxonomy in a top-down fashion. Finally, association rules are returned in a table and sorted according to the selected measure, as shown in Figure 5.2.

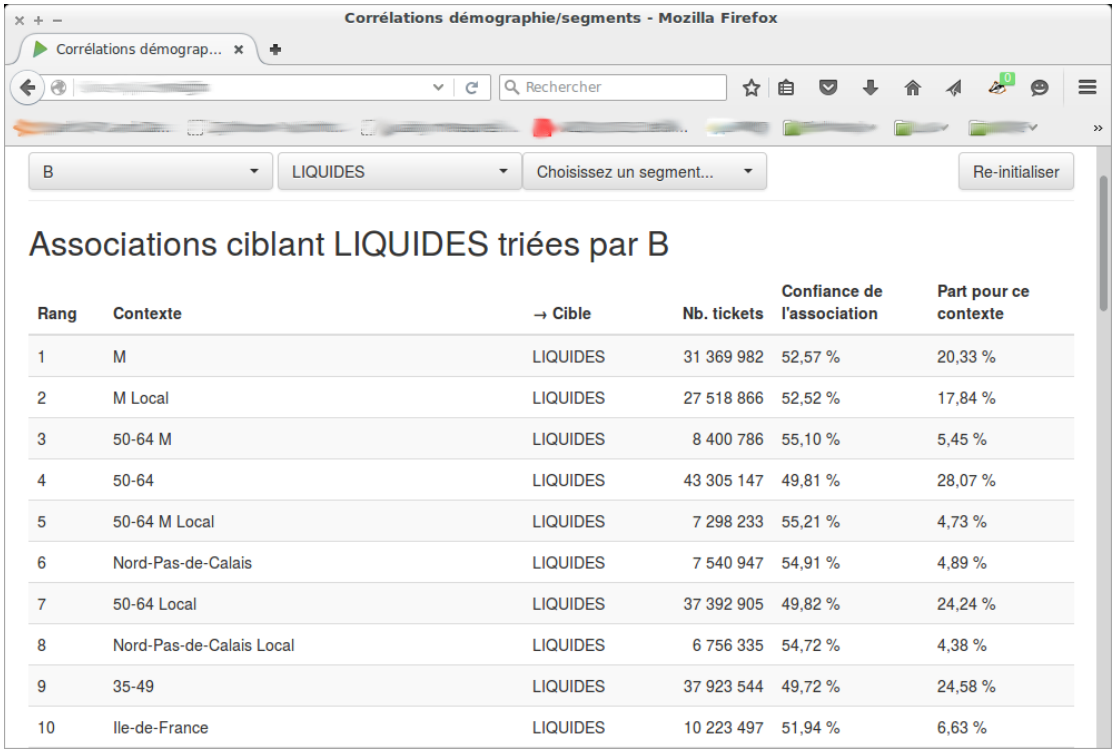


Figure 5.2: Screenshot of the exploration application presented to analysts.

A rule like *yogurt* → *cheese* is displayed with 3 values: *support* (number of customers who bought both cheese and yogurt), *confidence* (fraction of yogurt buyers who also bought cheese) and *recall* (fraction of cheese buyers who also bought yogurt). During the user study these figures help the analyst to quickly judge the volume of sales of each association.

To simplify the analysts' work in our user study (Section 5.3), the application does not propose 39 measures. Instead it proposes only 5 measures, each one representative of each measures group identified by our quantitative study.

5.2 Quantitative study

The goal of the following evaluation is to automatically detect similarities between interestingness measures and reduce the number of candidate measures to present to analysts in the user study of Section 5.3. To this end, we generate association rules for our 3 mining scenarios and evaluate the pairwise similarity of the 39 rankings produced by the measures we selected. We then use these similarities to classify ranking measures into *groups*, and annotate these groups based on common properties.

We first present in Section 5.2.1 the similarity measures used to compare our ranked lists. These include a new one, Normalized Discounted Correlation Coefficient (*NDCC*), whose score is more influenced by top results. Then, we compare quality measures in two different cases: rules having the same target (Section 5.2.2) and rules having different targets (Section 5.2.3). We conclude this evaluation with

the selection of representative measures in Section 5.2.4.

5.2.1 Ranking similarity measures

We use four methods to compare the ranked lists obtained in our 3 mining scenarios. The first three methods are taken from the literature. We then introduce *NDCC*, a new parameter-free ranking similarity designed to emphasize differences at the top of the ranking.

In the following $\mathcal{R}$ denotes the set of association rules obtained from the mining phase. We interpret each quality measure m as a function that receives a rule and generates a score, $m : \mathcal{R} \rightarrow \mathbb{R}$. We use $L^m_{\mathcal{R}}$ to denote the ordered list of rules in $\mathcal{R}$ sorted by decreasing score according to m. Thus, $L^m_{\mathcal{R}} =< r_1, r_2, \ldots >$ s.t. $\forall i > i'\ m(r_i) < m(r_{i'})$. The rank of a rule r in $L^m_{\mathcal{R}}$ is denoted r^m. We generate 39 lists, one for each measure m, from the same set of rules $\mathcal{R}$. To assess the dissimilarity between two measures m and m', we compute the dissimilarity between their ranked lists, $L^m_{\mathcal{R}}$ and $L^{m'}_{\mathcal{R}}$.

Spearman's rank correlation coefficient

Given two ranked lists $L^m_{\mathcal{R}}$ and $L^{m'}_{\mathcal{R}}$, *Spearman's rank correlation* [11] computes a linear correlation coefficient that varies between 1 (identical lists) and -1 (opposite rankings), as shown below.

$$Spearman(L^m_{\mathcal{R}}, L^{m'}_{\mathcal{R}}) = 1 - \frac{6 \sum\limits_{r \in \mathcal{R}} (r^m - r^{m'})^2}{|\mathcal{R}|(|\mathcal{R}|^2 - 1)}$$

This coefficient depends only on the difference between each rule's rank in the two lists, and not on the ranks themselves. Hence the penalization is the same for differences occurring at the beginning or at the end of the lists.

Kendall's τ rank correlation coefficient

Kendall's τ rank correlation coefficient [31] is based on the idea of agreement among element (rule) pairs. A rule pair is said to be *concordant* if their order is the same in $L^m_{\mathcal{R}}$ and $L^{m'}_{\mathcal{R}}$, and *discordant* otherwise. τ computes the difference between the number of concordant and discordant pairs and divides by the total number of pairs as shown below.

$$\begin{aligned}
\tau(L^m_{\mathcal{R}}, L^{m'}_{\mathcal{R}}) =& \frac{|C| - |D|}{\frac{1}{2}|\mathcal{R}|(|\mathcal{R}| - 1)}, \text{ where:} \\
C = \{(r_i, r_j) | & r_i, r_j \in \mathcal{R} \wedge i < j \wedge \\
& \operatorname{sgn}(r^m_i - r^m_j) = \operatorname{sgn}(r^{m'}_i - r^{m'}_j)\} \\
D = \{(r_i, r_j) | & r_i, r_j \in \mathcal{R} \wedge i < j \wedge \\
& \operatorname{sgn}(r^m_i - r^m_j) \neq \operatorname{sgn}(r^{m'}_i - r^{m'}_j)\}
\end{aligned}$$

As *Spearman*, τ varies between 1 and -1 and penalizes uniformly across all ranks.

Overlap@k

Overlap@k is a ranked lists similarity measure widely used in Information Retrieval. It is based on the premise that, in large ranked lists, the analyst is only expected to look at the top few results, that are highly ranked. While *Spearman* and τ account for all elements uniformly, Overlap@k compares two rankings by computing the overlap between their top-k elements only. We use $k \in \{10, 20, 50, 100, 500, 1000\}$.

$$Overlap@k(L^m_{\mathcal{R}}, L^{m'}_{\mathcal{R}}) = \frac{|\{r \in \mathcal{R} | r^m \leq k\} \cap \{r \in \mathcal{R} | r^{m'} \leq k\}|}{k}$$

Normalized Discounted Correlation Coefficient

Overlap@k, *Spearman* and τ sit at two different extremes. The first takes only into account the top k elements of each list, whereas the latter two consider all parts of the lists uniformly. In our use case, we aim for a good trade-off between these extremes.

To bridge this gap, we propose a new ranking correlation measure coined *Normalized Discounted Correlation Coefficient* or *NDCC*. *NDCC* draws inspiration from *NDCG, Normalized Discounted Cumulative Gain* [29], a ranking measure commonly used in Information Retrieval. The core idea in *NDCG* is to reward a ranked list $L^m_{\mathcal{R}}$ for placing an element r of relevance rel_r by $\frac{rel_r}{\log r^m}$.

The logarithmic part acts as a smoothing discount rate representing the fact that as the rank increases, the analyst is less likely to observe r. In our setting there is no ground truth to properly assess rel_r. Instead, we use the ranking assigned by m' as a relevance measure for r, with an identical logarithmic discount. When summing over all of $\mathcal{R}$, we obtain DCC, which presents the advantage of being a symmetric correlation measure between two rankings $L^m_{\mathcal{R}}$ and $L^{m'}_{\mathcal{R}}$.

$$DCC(L^m_{\mathcal{R}}, L^{m'}_{\mathcal{R}}) = \sum_{r \in \mathcal{R}} \frac{1}{\log(1 + r^{m'}) \log(1 + r^m)}$$

We compute *NDCC* by normalizing DCC between 1 (identical rankings) and -1 (reversed rankings).

$$\begin{aligned} NDCC(L^m_{\mathcal{R}}, L^{m'}_{\mathcal{R}}) &= \frac{dcc - avg}{max - avg} \\ \text{where } dcc &= DCC(L^m_{\mathcal{R}}, L^{m'}_{\mathcal{R}}) \\ max &= DCC(L^{m'}_{\mathcal{R}}, L^{m'}_{\mathcal{R}}) \\ min &= DCC(L*, L^{m'}_{\mathcal{R}}),\ L* = rev(L^{m'}_{\mathcal{R}}) \\ avg &= (max + min)/2 \end{aligned}$$

Ranking	Content
L^1	r_1, r_2, r_3, r_4
L^2	r_2, r_1, r_3, r_4
L^3	r_1, r_2, r_4, r_3
L^4	r_2, r_3, r_1, r_4

	Spearman	τ	*Overlap*@2	*NDCC*
L^2	0.80	0.67	1	0.20
L^3	0.80	0.67	1	0.97
L^4	0.40	0.33	0.5	−0.18

Table 5.3: Example rankings, and correlations between L^1 and the three other rankings.

Ranking comparison by example

We illustrate the difference between all ranking correlation measures with an example in Table 5.3. This shows correlation of a ranking L^1 with 3 others, according to each measure. *NDCC* does indeed penalize differences at higher ranks, and is less sensitive at lower ranks.

5.2.2 Ranking rules with identical targets

We first consider the case of ranking association rules $A \rightarrow B$ where B is a single product or category, *i.e.* all rules have the same $B = \{b\}$. We perform a comparative analysis of ranking measures on our 3 mining scenarios, which lead to similar conclusions. We use as targets for this comparison 64 products or categories previously studied by analysts. We compute one rule ranking per interestingness measure.

While all measures are computed differently, we notice that some of them always return the same ranking for association rules of a given target. We identify them in Table 5.2 (p.68) using symbols. We also observe important similarities. For example, in `prod_assoc_client`, the 64 targets provide 1,651,024 association rules. 89% of rankings are equal with *Sebag-Schoenauer* and *lift*, or 87% with *Loevinger* and *lift*. This difference between the number of interestingness measures considered (39) and the number of different rankings obtained (31) can easily be explained analytically in the case of a fixed target. Indeed, for a given ranking, $P(B)$ is constant, which eliminates some of the differences between interestingness measures. In addition, some measures only have subtle differences which only appear when selecting extreme values for $P(A)$, $P(B)$ and $P(AB)$, which do not occur in practice in our retail datasets.

Comparative analysis

We now evaluate similarity between interestingness measures that do not return the same rankings. We compute a 39×39 correlation matrix of all rankings according to each correlation measure described in Section 5.2.1, and average them over the 64 target products. This gives us a ranking similarity between all pairs of measures. We then rely on hierarchical clustering with average linkage [56] to obtain a dendrogram of interestingness measures and analyze their similarities. Figure 5.3 shows the clustering by *NDCC* in the `demo_assoc` scenario; the dendrograms for *NDCC* and τ in our three scenarios are presented in Annex B.1,

p.106. For better readability, we merge sub-trees when correlation is above 0.9. To describe the results more easily, we partition interestingness measures into 5 groups, as indicated in the third column in Table 5.2.

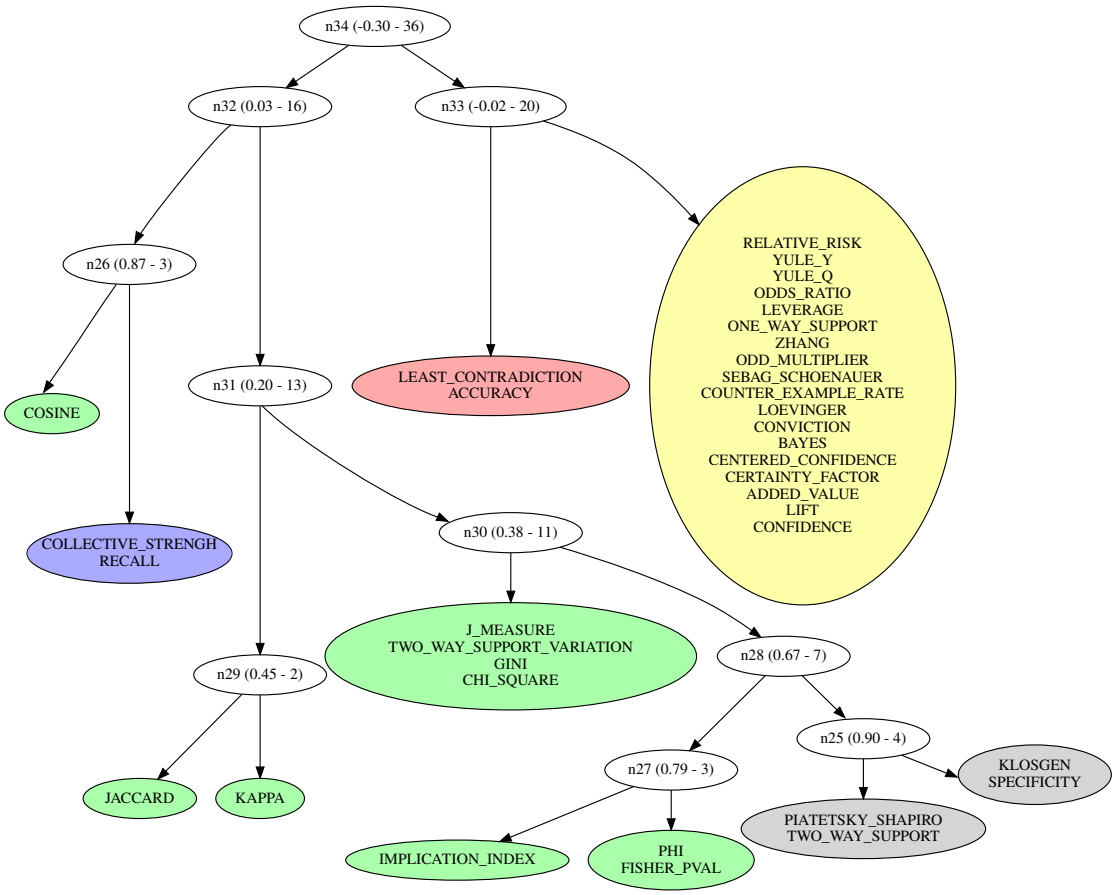


Figure 5.3: Clustering of quality measures, by *NDCC* in the `demo_assoc` scenario for association rules having identical targets.

G_1 is by far the largest group: in addition to 4 measures that always generate the same rankings, 14 other measures output very similar results. A second group, G_2, comprising 2 measures, is quite similar to G_1 according to *NDCC*, especially for `prod_assoc_client`. τ also discovers this similarity, but considers it lower, which shows that it is mostly caused by high ranks. The most dissimilar group is G_5. We note that G_1, G_2 and G_5 are extremely consistent across our 3 scenarios. The two other groups, G_3 and G_4, each have a constant "core" of measures. But *Klosgen*, *Jaccard*, *Two-way support* and *Cosine* are not always clustered with the same group. Their rankings differ more or less at each extreme of the lists.

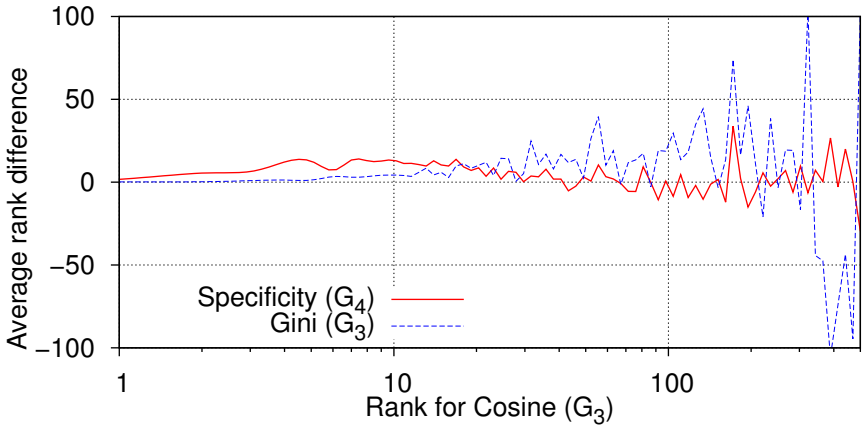


Figure 5.4: Rank differences between *Cosine* and *Specificity*, and *Cosine* and *Gini*.

We illustrate this behavior in Figure 5.4 by displaying correlation between rankings obtained with different interestingness measures. This experiment clearly shows that overall, *cosine* (G_3) is closer to *specificity* (G_4) than *Gini* (G_3), as the observed rank difference is overall smaller. However, when focusing on the top-10 results of *cosine*, *Gini* assigns closer ranks than *specificity*. This explains the difference in clustering between *NDCC* and τ, and motivates our choice to assign such outlier measures to G_3 or G_4. Our group assignment for these measures also takes into account clusterings according to *Spearman* and *overlap*@50.

Annotating groups

While using hierarchical clustering on interestingness measures allows the discovery of groups of measures and their relative similarity, it does not fully explain which types of results are favored by each of them. We propose to compare their outputs according to the two most basic and intuitive interestingness measures employed in data mining: *recall* and *confidence*. *recall* represents the proportion of target items that can be retrieved by a rule, that is, $P(A|B)$. Its counterpart, *confidence*, represents how often the consequent is present when the antecedent is, that is, $P(B|A)$. We present, in Figure 5.5, the average *recall* and *confidence* of the top-20 rules ranked according to each interestingness measure. G_1 contains *confidence*, so it is expected to score the highest on this dimension. G_2 is extremely close to G_1, but obtains slightly lower *confidence* and *recall*. We then have, in order of increasing *recall* and decreasing *confidence* G_3 and G_4. Finally, G_5, which contains *recall*, obtains the highest *recall* but the lowest *confidence*. Figure 5.5 also shows that executing a Euclidean distance-based clustering, such as k-means, with recall/confidence coordinates would lead to groups similar to the ones obtained with hierarchical clustering. Hence, this analysis is consistent with the hierarchical grouping and the correlation with *NDCC*.

While we believe that *NDCC* reflects better the interpretation of analysts browsing rules, it is important to note that the grouping of interestingness measures created through this evaluation is stable across all 4 correlation measures and for all 3 scenarios. Correlation between different groups of measures may vary,

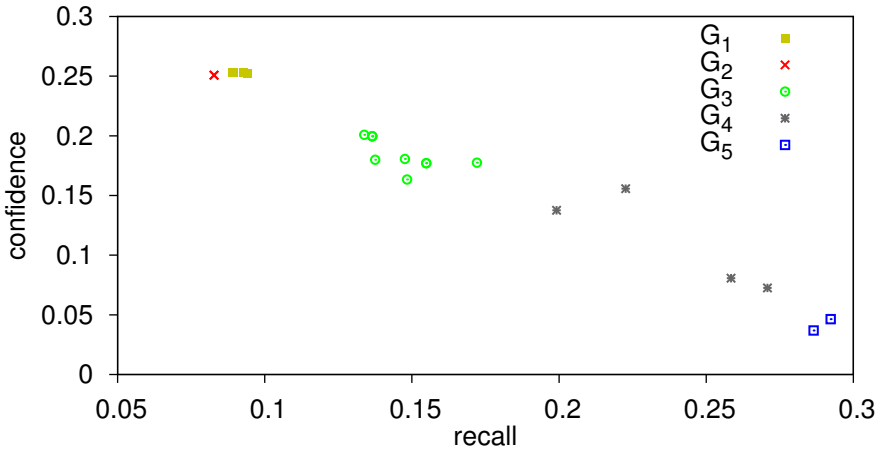


Figure 5.5: Average recall/confidence of the top-20 results of interestingness measures

but measures within a single group always have a high similarity. Thus, we state that the obtained results are true in the general case of food retailers and we can rely on these groups to reduce the number of options presented to analysts.

5.2.3 Ranking rules with different targets

We now consider the problem of ranking association rules when many targets are provided as input, i.e. association rules $A \rightarrow B$ can have different targets $b \in \mathcal{B}$. Compared to having a single target, this setting introduces one more degree of freedom in the quality measure of the association rules, as $P(B)$ varies. We rely on the same set of (64) products as in the identical target experiment, but instead of generating rankings for each target, we rank all association rules together. The dendrograms of quality measures obtained for *NDCC* and τ are presented in Annex B.2, p.112. Only Yule's Q, Yule's Y and Odds Ratio produced identical rankings.

We observe a much wider variety in rankings, which is expected given the presence of one more parameter. The group G_1, observed previously, splits into two different sub-groups, G_1^a and G_1^b. A large fraction of G_3 remains similar, but the two measures that constitute G_2 and G_5 are not correlated in all scenarios. G_2 and G_4 are not preserved when ranking different targets simultaneously. We observe a stronger agreement between *NDCC* and τ.

Measures that prioritize high values of $P(B)$, i.e. favor targets that are more frequent, are G_1^a, *Piatetsky-Shapiro*, *Klosgen* and *Gini*. Indeed, in the case of *confidence* (G_1^a), an association rule $A \rightarrow B$ that has a very frequent B can easily score highly by selecting a very specific A. Conversely, *specificity*, *collective strength*, *accuracy*, *recall* and measures of G_1^b tend to rank less frequent targets highly. A similar explanation applies to *recall*, as a low frequency of B makes it easier to find association rules that capture most of its appearances in the data.

Group Representative	Description
G_1^a Lift	Highest precision Very low recall Favors frequent targets
G_1^b Added value	Highest precision Very low recall Favors rare targets
G_2 Accuracy	Very high precision Very low recall
G_3 Fisher's exact test	High precision Low recall Low sensitivity to target frequency
G_4 Piatetsky-Shapiro	Low precision High recall
G_5 Collective strength	Lowest precision Highest recall Favors rare targets

Table 5.4: Summary of quality measure groups

5.2.4 Selecting representative measures

We summarize the findings of the comparative evaluation in Table 5.4. When the analyst selects a single target, we identify 5 groups of measures that behave similarly. When multiple targets are selected, G_1 splits into 2 sub-groups. Each group offers a different trade-off in terms of confidence and recall, and thus ranks association rules differently. When ranking rules with different targets, some groups are sensitive to the frequency of the target.

We select the quality measure that most represents each group of measures (i.e. with highest average similarity) in order to confront the results of this analysis with the opinion of domain experts in our user study. Taking a general data mining perspective leads us to considering G_3 and G_4 as the most promising groups for finding interesting association rules. Indeed, it is important to achieve a good trade-off between *recall* and *confidence* in order to find reliable association rules that can be applied in a significant number of cases. For example the *F1* score, that combines *recall* and *confidence*, would prefer G_3 and G_4 to others. The goal of our user study is to confront this intuition to marketing experts' opinion.

5.3 User study

We now report the results of a user study with domain experts from Intermarché. The goal of this study is to assess the ability of interestingness measures to rank association rules according to the needs of an analyst. In the previous section we identified 5 groups of measures, and selected a representative of each group for the user study (Table 5.4). We rely on the expertise of our industrial partner to determine, for each analysis scenario, which group produces the most interesting results. This experiment involved 2 experienced analysts from the marketing department of Intermarché. We let them interact with CAPA without any time restriction, and collect their feedback in a free text form.

Each analyst firstly has to pick a mining scenario among `demo_assoc`, `prod_assoc_receipt`, or `prod_assoc_client`. Then she picks a target category or a target product in the taxonomy. In `prod_assoc_receipt` and `prod_assoc_client`, she also has the option to filter out rules whose antecedent products are not from the same category as the target. Finally, she chooses one of our 5 ranking measures to sort association rules. Neither the name of the measure nor its computed values for association rules are revealed, because we wanted analysts to evaluate rankings without knowing how they were produced. Resulting association rules are ranked according to a selected measure. Each rule is displayed with its support, confidence and recall, such that analysts can evaluate it at a glance (see Figure 5.2, p.71). For each scenario, our analysts are asked which representative measure highlights the most interesting results. As detailed below, in all cases a few of them were chosen.

5.3.1 Scrolling behavior

Once the analyst selects a target, *all* matching rules are returned. The initial motivation of this choice was to determine how many results are worth displaying and are actually examined by the analysts. According to the follow-up interview with the analysts, they carefully considered the first ten results, and screened up to a hundred more. Interestingly, analysts mentioned that they also scrolled down to the bottom of the list in order to see which customer segments are not akin to buying the selected category. For example, when browsing demographic association rules, they expected to find $\{$*50-64*$\} \rightarrow$ *pet food* among top results, but also expected $\{$*<35, Paris*$\} \rightarrow$ *pet food* among bottom results. This confirms that all rules should remain accessible. This also indicates that while interestingness measures favor strong associations, it would also be interesting to highlight *anti*-rules, as those can also convey useful information.

5.3.2 Feedback on ranking measures

We let marketing experts explore all 3 scenarios and express their preference towards groups of measures.

In the `demo_assoc` case, G_1 and G_3 were both highly appreciated. G_1 favors rules such as $\{< 35, M, \text{Oise}\} \rightarrow$ *Flat and Carbonated drinks*. These rules are very

specific and thus have a very high confidence (31,58 % in this particular case). However, this comes at the cost of recall (0,08 %). Experts value *confidence* much more than *recall*, as their priority is finding rules that they consider reliable. A low support is not necessarily an issue, and can lead to the discovery of surprising niche rules that can be exploited nonetheless. As discussed in Section 5.2.2, G_3 offers a more balanced trade-off between confidence and recall, and prioritizes rules such as $\{< 35, *, *\} \rightarrow$ *Baby food* (confidence 8,57 %, recall 37,61%). These rules are interesting because they capture a large fraction of the sales of a given category, but are less reliable and generally less surprising. G_2 and G_4 were considered as less interesting than G_1 and G_3 respectively. Their results offer similar trade-offs, but with lower confidence each time. G_5 was considered unusable because of its very low confidence.

When experimenting with `prod_assoc_client` or `prod_assoc_receipt`, we observed a slightly different behavior. By default, the analysts favored G_1 and G_2 because of the confidence of their results. Then, we offered the analysts the possibility of filtering the rules to only keep the ones in which the antecedent contains products from the same category as the target. This led to analysts favoring G_3 and G_4. This difference is caused by an important but implicit criterion: the ability of a measure to filter out very popular products. For example, the rule {*vanilla cream, emmental*}$\rightarrow$ *chocolate cream* usually appears just above its shorter version {*vanilla cream*}$\rightarrow$ *chocolate cream*, because the first one has a confidence of 32% and the second 31%. However, experts prefer the second one, because *emmental* (cheese) is one of the star products we mentioned in Section 4.5.2. Hence its addition to the rule is considered insignificant. This "noise" generally increases with *recall*. Hence, when no filtering is available, G_1 is selected, but analysts prefer the *recall* and *confidence* trade-off provided by G_3 and G_4. Again, G_4 suffered from its proximity to G_3 with lower confidence, while G_5's confidence was too low.

5.4 Conclusion

From a dataset of transactions, created by our curation module according to one of our 3 mining scenarios, CAPA extracts association rules containing targets pre-selected by the analyst. It then allows to compare how 39 interestingness measures [17, 36] rank these associations. Although the quality measures yield very different absolute values, when ranking association rules many of them are similar, or even identical.

Our first comparison is quantitative: we generate and rank association rules, and compare the rankings obtained by the measures we selected. We show that interestingness measures can be automatically grouped into 5 groups of similar rankings, regardless of the mining scenario. These groups are consistent across our 3 mining scenarios and 4 similarity measures. Our choice of similarity measures include a novel one: Normalized Discounted Correlation Coefficient *(NDCC)*. Our motivation to propose *NDCC* is to give more importance to top rules in the similarity evaluation. Hence it fits our use case, where analysts are interested in the first few dozen associations.

Our second comparison is qualitative, and involves two experienced domain

experts from Intermarché. The goal of this user study is to find which of the 5 groups of interestingness measures are meaningful in the food retail domain. Results suggest that no measure fits the general case: their preference depends on whether the initial constraints are highly selective or not. In all cases, analysts mentioned G_5 (see Table 5.2, p.68) as uninteresting overall because it selects rules of low *confidence*. Thus, sorting by decreasing *lift* (which is close to sorting by decreasing *confidence*) is the most preferred choice. Combined with the minimum support threshold used in the mining phase, this ranking promotes rules that are considered reliable. However, the preference of the analysts changes when filters are available to narrow down the set of rules to specific product categories. In this case, they favor the compromise between *confidence* and *support* offered, for instance, by the *Piatetsky-Shapiro*'s measure [53]. That is because filtering allows them to get rid of noisy products.

To the best of our knowledge, CAPA targets datasets which are orders of magnitude bigger (and sparser) than those tested in existing work on ranking association rules. The present work is also the first to complement a quantitative analysis with a user study involving domain experts. Therefore the groups of measures and recommendations we present here differ from the studies mentioned in Section 2.4.1 (p.18).

The highest scoring measures in [33] all belong to our groups G_1 and G_3, which were also favored by our analysts. But a few measures from our G_1 are scoring poorly in [33], hence our measures of choice do not match theirs, overall. The largest group identified by HERBS [36, 67] is quite similar to G_1 and also includes *confidence*. But there are also significant differences. For instance, we find G_2 and G_5 to be very different, while [36] considers the measures of these two groups as similar. The authors observe a weak resemblance between the theoretical and experimental analysis of the measures. The main similarity between HERBS and CAPA is the reliance on a pairwise correlation measure followed by a hierarchical clustering to detect groups of measures. But CAPA is entirely focused on retail data, which has different properties and contains millions of transactions and rules. CAPA is also more exhaustive in the analysis of measures: we consider more interestingness measures, and 4 different ranking correlation measures, instead of Kendall's τ only. Such differences in the results highlight the importance of performing domain-specific studies, as the properties of data and the expectations of analysts vary significantly.

Overall, the quality measures we selected with CAPA allow the analyst to pick a few dozen associations of great interest out of the thousands available. We observe this in our 3 mining scenarios, both for highly-targeted analyzes (when the analyst specifies a narrow set of target items) and in an exploratory setting. In the latter case, these quality measures can re-rank the item-centric itemsets lists produced by TOPPI. This chapter therefore allows us to display interesting associations in all variants of the DATALYSE project.

CHAPTER 6

Conclusion

Contents

Our concluding chapter summarizes our contributions, shows their complementarity and discusses their possible evolutions.

Section 6.1 summarizes our two contributions: TopPI, an algorithm to mine efficiently association rules for any item in the dataset, and CAPA, a framework to study which quality measures can highlight the most interesting association rules.

Section 6.2 shows that item-centric results mined by TopPI contain many associations of high interest according to Fisher's exact test. This confirms the advantage of re-ranking each item's top itemsets according to a quality measure selected via CAPA.

Section 6.3 proposes three research directions that result from our work.

6.1 Contributions summary

6.1.1 Finding association rules about any item with TopPI

The recent evolution of database systems allow the generation and storage of transactional datasets containing hundreds of millions of transactions over millions of items. Existing frequent itemsets mining algorithms cannot extract interesting results from such datasets, even with the restriction to closed itemsets. Indeed, by definition frequent itemsets only contain frequent items: in our datasets of interest, this implies that results only cover the few frequent items. The analyst may choose to increase the number of items considered frequent, but this triggers a combinatorial explosion of results.

We therefore propose a new mining semantics, where the analyst is only required to define the size of the desired output using a single parameter, k. Given k we find, for each item in the dataset, the k most frequent closed itemsets (CIS) containing this item.

Mining these top-k CIS for all items at once provides the best amortization of each itemset's mining. Our algorithm, TopPI, combines a dynamic enumeration of the frequent CIS space with an efficient and fast pruning function. We show that the combination of these two features is essential to find per-item CIS in reasonable time. We also show that mining item-centric CIS can be efficiently parallelized, both on a multi-core server or on a Hadoop cluster, in order to analyze large-scale datasets.

Not only TopPI is able to provide results which would be out of reach to existing methods, but its item-centric approach also fits the way analysts usually browse association rules. When the data concerns so many items, she usually starts by picking an item she's familiar with, and search its associations. She repeats this operation for other items, as she progressively encounters new ones. Thus TopPI provides the analyst with exhaustive and intuitively organized association rules.

6.1.2 Choosing a quality measure to rank association rules with CAPA

When the distribution of items is highly unbalanced, as in our datasets of interest, frequent associations may sometimes reflect artifacts. Typically, a very frequent product like a shopping bag appears in almost all items' frequent associations. But in most cases it does not really represent a link between the two products: it only shows that everybody needs to carry groceries. We can circumvent this phenomena by changing the sorting function of our associations. But too many functions have been proposed in the literature, from which we cannot judge which function is adequate to sorting associations in the retail domain. Such artifacts occur in TopPI's results, but are also observed in more targeted studies, for example when the analyst specifies a set of interesting items. We therefore develop CAPA, a framework to compare how 39 quality measures rank association rules extracted from 3 retail datasets.

We first perform an automatic comparison, by randomly picking 64 items, mining their association rules and generating their 39 ranked lists of associations. Then we perform a hierarchical clustering, by comparing the rankings they produce. Ranking similarity is measured using *Spearman's rank correlation*, Kendall's τ, *Overlap@k*, and a novel one we propose: *NDCC* (Normalized Discounted Correlation Coefficient). This study allows us to distinguish 5 groups of measures.

In a second phase, we ask marketing experts to evaluate a representative measure of each group. We give them access to an association rules exploration application where they can choose to rank rules according to one among 5 anonymized methods. In all cases, analysts mentioned that ranking by decreasing *recall* is uninteresting because it selects rules of too low *confidence*. In general, sorting by decreasing *lift* (which is close to sorting by decreasing *confidence*) is the preferred choice. Combined with a minimum support threshold used in the mining

phase, this ranking promotes rules that are considered reliable. However, the preference of the analysts changes when filters are available to narrow down the set of rules to specific product categories. In this case, they favor the compromise between *confidence* and *support* offered, for instance, by the *Piatetsky-Shapiro*'s measure [53].

6.1.3 Industrial impact

This work has been conducted in the context of Datalyse, an industrial project. Our industrial partners provide the infrastructure for the joined analysis of three data sources: supermarket receipts, customer information and a taxonomy of products. End-users are marketing analysts who need to study the associations between products, or between customers and products.

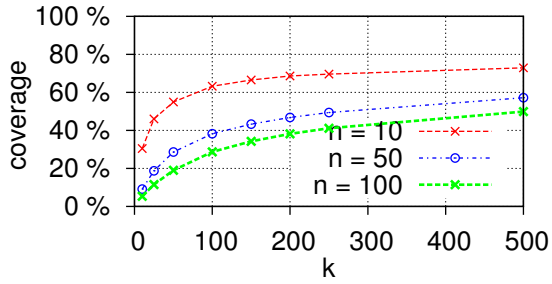
Thanks to our collaboration with Intermarché we could work on real data, following realistic use cases of analytics in the retail domain. We also benefited from the feedback of marketing experts when evaluating the results produced both by TopPI and CAPA.

Our contributions were also progressively integrated in the production workflow of Intermarché. This started with our earliest work, *j*LCM, our implementation of PLCM [50] in Java. TopPI was also delivered, which led Intermarché to propose the improvements detailed in Section 4.5 (p.62). This motivated the development of CAPA, which involved experts from the marketing studies department of Intermarché. The final version of TopPI, deployed in production, includes a post-processing: instead of being ranked by decreasing frequency, each item's itemsets are ranked by decreasing correlation with the item, according to the p-value. The following experiment shows that this post-processing allows our miner to return smaller lists of more relevant itemsets per item.

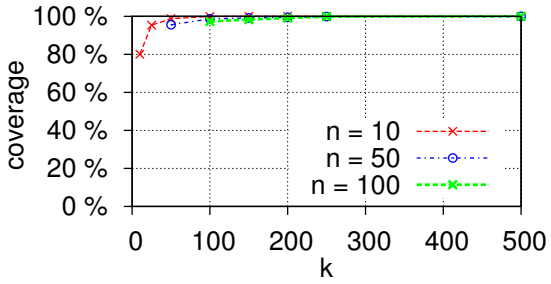
6.2 Improving TopPI by re-ranking per-item closed itemsets

TopPI relies only on support to select the top-k CIS containing an item. In this section we show that it is sufficient to find the top results per-item according to a finer quality measure: the p-value, computed with Fisher's exact test as described in Annex A (p.99).

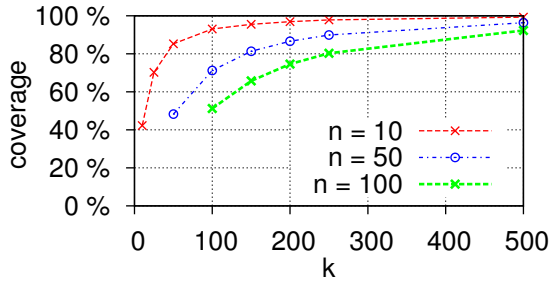
Ranking by decreasing frequency allows TopPI to efficiently traverse a small portion of the CIS space, as shown in Chapter 4. Although it provides valuable results, TopPI's output also contains a few uninteresting associations: the most frequent items appear too often with long-tail items. Our experiments with CAPA suggest that some scoring functions (from G_3, see Table 5.2 p.68) may allow the automatic extraction of more interesting itemsets for each item. The traditional approach (and the most efficient in our case, see Section 2.4.2, p.19) to mine association rules with both high *support* and *confidence* is to first extract frequent CIS, then filter them using a *confidence* threshold [3]. In a similar fashion, we



(a) `prod_assoc_receipt`



(b) *Supermarket*



(c) *LastFM*

Figure 6.1: TopPI's output coverage of the top-n by p-value

propose to re-rank TopPI's results (for each item) to obtain high quality item-centric association rules.

From here we note top-k when itemsets are sorted by decreasing frequency (as in TopPI), and top-n when itemsets are sorted according to a finer quality measure (as in CAPA). To assess the relevance of our re-ranking strategy, we measure in the following experiment how TopPI's results contain the top-n itemsets per-item according to the representative measure of G_3: Fisher's exact test. The bound described by Minato *et al.* [47] allows us to find a minimum frequency threshold ensuring that the candidate CIS mined to generate each top-n do have a better p-value than any other.

This may still require the generation of millions of itemsets per item. Hence we perform this experiment on a sample of the items from our datasets. We generate 10 buckets of items according to their support, and randomly select 50 items in each, for a total of 500 items. For each selected item i, we mine frequent CIS in $\mathcal{D}_i = \langle t \setminus \{i\} \mid t \in \mathcal{D}, i \in t \rangle$. Then we measure the correlation of occurrences of these CIS with occurrences of i in $\mathcal{D}$. Finally we rank the CIS by increasing p-value, thus obtaining the item's top-n. Figure 6.1 shows TopPI's output coverage of the ground truth we generated, for various k and n.

For supermarket tickets, this experiment shows that the majority of the sampled items' top-n CIS are contained in TopPI's top-k lists. However, results are extremely different if the experiment is done on the complete dataset, `prod_assoc_receipt`, or on its preliminary version, *Supermarket*. This is all the more surprising as these two datasets have very similar items distributions. But, at this scale (`prod_assoc_receipt` is 10 times larger than *Supermarket*), some extremely rare associations may have an artifically strong p-value and are not mined by TopPI.

For *LastFM*, at coordinates $k = 100$ we can see that the top-k of an item contains on average 93% of the itemsets of its top-n (by p-value) for $n = 10$, 71%

for $n = 50$, and 51% for $n = 100$. With this dataset, TopPI's coverage of the top-by-p-value is above 90% for $k \geq 5n$. This means that, when $k \geq 5n$, for the sampled items 90% of the top-n (by p-value) CIS appear in TopPI's top-k (by frequency). Therefore, if we re-rank TopPI's results by p-value, the final lists would be quite close to the exact top-n. This is interesting, because computing the exact top-n is much more costly.

For this reason, we cannot show results for `prod_assoc_client`. In this dataset, computing the sampled items' top-n by p-value is too complex and would require several months of CPU time. Because its transactions are around 10 times longer than in `prod_assoc_receipt`, the number of frequent CIS explodes quickly. Therefore, guaranteeing the result's correctness sometimes requires the mining and storage of billions of itemsets. Similarly, computing the ground truth for only 500 items of *LastFM* takes 8.6 hours of CPU time. On the same dataset, TopPI consumes 3.1 hours of CPU for all the 450,000 items and the highest value of k.

Overall, this confirms that TopPI can be used as part of a two-step approach to accurately emulate refined, but computationally costly, interestingness measures on large-scale datasets. It is currently in use at Intermarché. Thanks to this approach, extracting association rules from a large collection of transactions is both affordable and interesting for a nation-wide retailer, that would otherwise have to ask experts to filter manually the input or the results [16].

6.3 Towards an interactive associations explorer

We conclude by describing potential evolutions of our contributions. These would improve the system's response time, the quality of its results and its reactivity to the continuous updates usually performed on modern datasets.

6.3.1 Shortening computation batches

Both jLCM, CAPA and TopPI are implemented using batch processing and on-disk storage. While mining is already fast, I/O operations introduce some latency (sometimes measured in hours) between the definition of a mining scenario and the display of results. Thanks to the evolution of memory technologies, at the time of writing our datasets can fit in a single server. The curation step of our workflow could be executed efficiently in minutes or less using an in-memory database [39, 35].

For an even faster processing, or larger datasets, we could also rely on an in-memory distributed dataset representation. For example, Spark proposes the *Resilient Distributed Datasets* [72], which have been massively adopted by industry and academia, and improved since [5]. Spark would also speed up the distributed variant of TopPI, but not significantly because the local mining done by each worker remains the major time bottleneck.

The ability of large-scale in-memory systems to obtain small selections or samples of a dataset almost instantly also makes them necessary to a more interactive mining process, as proposed in the following sections.

6.3.2 Adaptive mining and ranking

In [40], Liu *et al.* propose an association rules exploration framework where rules are grouped by consequent, then traversed by progressively adding items to the antecedent. The framework provides hints on how each additional item would make a difference. Such a framework is suitable to the scenarios we consider, and would improve our association rules exploration. It would also drastically restrict the CIS enumeration, because at each step the system would only have to traverse direct extensions of current associations.

This iterative approach would also allow the system to learn the user's preference for ranking associations [13]. Each interaction would allow the system to refine its understanding of the analyst's needs, and in particular its notion of interesting association.

6.3.3 Mining as we explore

A dynamic definition of interesting associations would shorten mining time, because we would only have to traverse the small portion of the CIS space pointed out by the analyst. But with large datasets this may not be enough to reach sub-second response times, in particular when most transactions contain more than a thousand items. In our work, the greatest mining times are observed with *WebDocs*, which has the longest transactions (177 items, on average). Should they contain ten or a hundred times more items, the datasets would be out of reach of TopPI. In these cases, mining closed itemsets is too precise to be efficient: we have to count precisely the occurrences of each item or itemset in each transaction, which is too costly when transactions are too long.

To avoid this, the iterative process described in Section 6.3.2 may include an intermediate step. The system would firstly approximate the itemsets or their support, and mine precisely only once the analyst has shown interest for the result. For instance, PARMA [55] shows that we can get a reliable approximation of frequent itemsets, in distributed datasets, using distributed sampling.

Thanks to the improvements proposed in this section, the system would be able to provide insightful results for large scale datasets using much less CPU time. As large-scale datasets are evolving continuously, it will be more and more relevant to only explore the portion of the CIS space which is interesting to the analyst at the time she's using it.

Bibliography

[1] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining Association Rules between Sets of Items in Large Databases. In *Proc. SIGMOD '93*, pages 207–216, 1993.

[2] Rakesh Agrawal and John C Shafer. Parallel mining of association rules. *Knowledge and Data Engineering, IEEE Transactions on*, (6):962–969, 1996.

[3] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB)*, pages 487–499, 1994.

[4] Chris Anderson. *The Long Tail: Why the Future of Business Is Selling Less of More*. Hyperion, 2006.

[5] Michael Armbrust, Reynold S Xin, Cheng Lian, Yin Huai, Davies Liu, Joseph K Bradley, Xiangrui Meng, Tomer Kaftan, Michael J Franklin, Ali Ghodsi, et al. Spark sql: Relational data processing in spark. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 1383–1394. ACM, 2015.

[6] Toon Calders and Bart Goethals. Non-derivable itemset mining. *Data Mining and Knowledge Discovery*, 14(1):171–206, 2007.

[7] Aaron Ceglar and John F. Roddick. Association mining. *ACM Comput. Surv.*, 38(2), 2006.

[8] David W Cheung, Jiawei Han, Vincent T Ng, Ada W Fu, and Yongjian Fu. A fast distributed algorithm for mining association rules. In *Parallel and Distributed Information Systems, 1996., Fourth International Conference on*, pages 31–42. IEEE, 1996.

[9] Kun-Ta Chuang, Jiun-Long Huang, and Ming-Syan Chen. Mining top-k frequent patterns in the presence of the memory constraint. *The VLDB Journal*, 17(5):1321–1344, 2007.

[10] William James Cody and Kenneth E. Hillstrom. Chebyshev approximations for the natural logarithm of the gamma function. *Mathematics of Computation*, 21(98):198–203, 1967.

[11] W.W. Daniel. *Applied Nonparametric Statistics*. Houghton Mifflin, 1978.

[12] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. In *Proceedings of the 6th Symposium on Operating System Design and Implementation (OSDI)*, 2004.

[13] Vladimir Dzyuba, Matthijs van Leeuwen, Siegfried Nijssen, and Luc De Raedt. Interactive learning of pattern rankings. *International Journal on Artificial Intelligence Tools*, 23(06):1460026, 2014.

[14] Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. In *Proceedings of the Twentieth Symposium on Principles of Database Systems (PODS)*, pages 102–113, 2001.

[15] Philippe Fournier-Viger, Antonio Gomariz, Ted Gueniche, Azadeh Soltani, Cheng-Wei Wu, and Vincent S Tseng. Spmf: a java open-source pattern mining library. *The Journal of Machine Learning Research*, 15(1):3389–3393, 2014.

[16] Paul Suganthan G.C., Chong Sun, Krishna Gayatri K., Haojun Zhang, Frank Yang, Narasimhan Rampalli, Shishir Prasad, Esteban Arcaute, Ganesh Krishnan, Rohit Deep, Vijay Raghavendra, and AnHai Doan. Why Big Data Industrial Systems Need Rules and What We Can Do About It. In *Proc. SIGMOD '15*, pages 265–276, 2015.

[17] Liqiang Geng and Howard J. Hamilton. Interestingness Measures for Data Mining: A Survey. *ACM Comput. Surv.*, 38(3), 2006.

[18] Amol Ghoting, Gregory Buehrer, Srinivasan Parthasarathy, Daehyun Kim, Anthony Nguyen, Yen-Kuang Chen, and Pradeep Dubey. Cache-conscious frequent pattern mining on modern and emerging processors. *The VLDB Journal*, 16(1):77–96, 2007.

[19] Sharad Goel, Andrei Broder, Evgeniy Gabrilovich, and Bo Pang. Anatomy of the long tail: ordinary people with extraordinary tastes. In *Proceedings of the Third International Conference on Web Search and Data Mining (WSDM)*, pages 201–210, 2010.

[20] Bart Goethals, Siegfried Nijssen, and Mohammed J Zaki. Open source data mining: Workshop report. *SIGKDD Explor. Newsl.*, 7(2):143–144, 2005.

[21] Bart Goethals and Mohammed J Zaki, editors. *FIMI '03, Frequent Itemset Mining Implementations, Proceedings of the ICDM 2003 Workshop on Frequent Itemset Mining Implementations, 19 December 2003, Melbourne, Florida, USA*, volume 90 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2003.

[22] Jiawei Han, Jian PeGoeli, Yiwen Yin, Runying Mao, and Runying Mao. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data Mining and Knowledge Discovery*, pages 53–87, 2004.

[23] Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. In *ACM Sigmod Record*, volume 29, pages 1–12. ACM, 2000.

[24] Jiawei Han, Jianyong Wang, Ying Lu, and Petre Tzvetkov. Mining top-k frequent closed patterns without minimum support. In *Proceedings of the International Conference on Data Mining (ICDM)*, pages 211–218. IEEE, 2002.

[25] IBM. *1401 Data Processing System - General Information Manual*. International Business Machines Corporation, 02 1960.

[26] Shin ichi Minato, Takeaki Uno, and Hiroki Arimura. Lcm over zbdds: Fast generation of very large-scale frequent itemsets using a compact graph-based representation. *Advances in Knowledge Discovery and Data Mining*, pages 234–246, 2008.

[27] Ihab F. Ilyas, George Beskales, and Mohamed A. Soliman. A survey of top-k query processing techniques in relational database systems. *ACM Computing Surveys*, 40(4):11:1–11:58, 2008.

[28] Adam Jacobs. The pathologies of big data. *Communications of the ACM*, 52(8):36–44, 2009.

[29] Kalervo Järvelin and Jaana Kekäläinen. Cumulated Gain-based Evaluation of IR Techniques. *ACM Trans. Inf. Syst.*, 20(4):422–446, 2002.

[30] Roberto J. Bayardo Jr., Bart Goethals, and Mohammed J Zaki, editors. *FIMI '04, Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations, Brighton, UK, November 1, 2004*, volume 126 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2005.

[31] M. G. Kendall. A New Measure of Rank Correlation. *Biometrika*, 30(1/2):81–93, 1938.

[32] Anup Kumar, Mehmed Kantardzic, and Samuel Madden. Guest editors' introduction: Distributed data mining–framework and implementations. *IEEE Internet Computing*, 10(4):15, 2006.

[33] Tien-Duy Le and David Lo. Beyond Support and Confidence: Exploring Interestingness Measures for Rule-Based Specification Mining. In *Proc. SANER '15*, pages 331–340, 2015.

[34] Yannick Le Bras, Philippe Lenca, and Stéphane Lallich. *Mining interesting rules without support requirement: a general universal existential upward closure property*, volume 8 of *Annals of information systems*, chapter Data Mining, pages 75–98. Springer, 2010.

[35] Juchang Lee, Yong Sik Kwon, Franz Farber, Michael Muehle, Chulwon Lee, Christian Bensberg, Joo Yeon Lee, Arthur H Lee, and Wolfgang Lehner. Sap hana distributed in-memory database system: Transaction, session, and

metadata management. In *Data Engineering (ICDE), 2013 IEEE 29th International Conference on*, pages 1165–1173. IEEE, 2013.

[36] Philippe Lenca, Benoît Vaillant, Patrick Meyer, and Stéphane Lallich. Association Rule Interestingness Measures: Experimental and Theoretical Studies. In *Quality Measures in Data Mining*, pages 51–76. Springer, 2007.

[37] Haoyuan Li, Yi Wang, Dong Zhang, Ming Zhang, and Edward Y. Chang. Pfp: parallel fp-growth for query recommendation. In *Proceedings of the second conference on Recommender systems (RecSys)*, pages 107–114, 2008.

[38] Ming-Yen Lin, Pei-Yu Lee, and Sue-Chen Hsueh. Apriori-based frequent itemset mining algorithms on mapreduce. In *Proceedings of the 6th International Conference on Ubiquitous Information Management and Communication (ICUIMC)*, page 76, 2012.

[39] Jan Lindström, Vilho Raatikka, Jarmo Ruuth, Petri Soini, and Katriina Vakkila. Ibm soliddb: In-memory database optimized for extreme speed and availability. *IEEE Data Engineering Bulletin*, 36(2):14–20, 2013.

[40] Guimei Liu, Mengling Feng, Yue Wang, Limsoon Wong, See-Kiong Ng, Tzia Liang Mah, and Edmund Jon Deoon Lee. Towards exploratory hypothesis testing and analysis. In *ICDE*, pages 745–756, 2011.

[41] Guimei Liu, Andre Suchitra, and Limsoon Wong. A performance study of three disk-based structures for indexing and querying frequent itemsets. *PVLDB*, 6(7):505–516, 2013.

[42] Claudio Lucchese, Salvatore Orlando, and Raffaele Perego. Parallel mining of frequent closed patterns: Harnessing modern computer architectures. In *Data Mining, 2007. ICDM 2007. Seventh IEEE International Conference on*, pages 242–251. IEEE, 2007.

[43] Claudio Lucchese, Salvatore Orlando, Raffaele Perego, and Fabrizio Silvestri. Webdocs: a real-life huge transactional dataset. In *Proceedings of the Workshop on Frequent Itemset Mining Implementations (FIMI)*, 2004.

[44] Michael Mampaey, Nikolaj Tatti, and Jilles Vreeken. Tell me what i need to know: succinctly summarizing data with itemsets. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 573–581. ACM, 2011.

[45] Michael Mampaey, Jilles Vreeken, and Nikolaj Tatti. Summarizing data succinctly with the most informative itemsets. *ACM Trans. Knowl. Discov. Data*, 6(4):16:1–16:42, 2012.

[46] Mary L McHugh. The odds ratio: calculation, usage, and interpretation. *Biochemia Medica*, 19(2):120–126, 2009.

[47] Shin-Ichi Minato, Takeaki Uno, Koji Tsuda, Aika Terada, and Jun Sese. A fast method of statistical assessment for combinatorial hypotheses based on frequent itemset enumeration. In *Machine Learning and Knowledge Discovery in Databases*, volume 8725, pages 422–436. Springer, 2014.

[48] Sandy Moens, Emin Aksehirli, and Bart Goethals. Frequent itemset mining for big data. In *SML: BigData 2013 Workshop on Scalable Machine Learning*. IEEE, 2013.

[49] Raymond T Ng, Laks VS Lakshmanan, Jiawei Han, and Alex Pang. Exploratory mining and pruning optimizations of constrained associations rules. In *ACM SIGMOD Record*, volume 27, pages 13–24. ACM, 1998.

[50] Benjamin Négrevergne, Alexandre Termier, Jean-Francois Méhaut, and Takeaki Uno. Discovering closed frequent itemsets on multicore: Parallelizing computations and optimizing memory accesses. In *Proceedings of the International Conference on High Performance Computing and Simulation (HPCS)*, pages 521–528, 2010.

[51] Nicolas Pasquier, Yves Bastide, Rafik Taouil, and Lotfi Lakhal. Discovering frequent closed itemsets for association rules. In *Proceedings of the 7th International Conference on Database Theory (ICDT)*, pages 398–416, 1999.

[52] Jian Pei, Jiawei Han, and Runying Mao. Closet: An Efficient Algorithm for Mining Frequent Closed Itemsets. In *Proc. SIGMOD '00*, pages 21–30, 2000.

[53] Gregory Piatetsky-Shapiro. Discovery, analysis, and presentation of strong rules. *Knowledge discovery in databases*, pages 229–238, 1991.

[54] Balázs Rácz, Ferenc Bodon, and Lars Schmidt-Thieme. On benchmarking frequent itemset mining algorithms: from measurement to analysis. In *Proceedings of the 1st international workshop on open source data mining: frequent pattern mining implementations*, pages 36–45. ACM, 2005.

[55] Matteo Riondato, Justin A. DeBrabant, Rodrigo Fonseca, and Eli Upfal. Parma: a parallel randomized algorithm for approximate association rules mining in mapreduce. In *Proceedings of the 21st International Conference on Information and Knowledge Management (CIKM)*, pages 85–94, 2012.

[56] R. R. Sokal and C. D. Michener. A Statistical Method for Evaluating Systematic Relationships. *Univ. Kans. Sci. Bull.*, 38:1409–1438, 1958.

[57] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to Data Mining, (First Edition)*. W. W. Norton & Company, 2007.

[58] The Apache Software Foundation. Apache mahout library. `http://mahout.apache.org/`. [Online; accessed 01-May-2016].

[59] The Apache Software Foundation. Hadoop. `http://hadoop.apache.org`. [Online; accessed 01-May-2016].

[60] The Apache Software Foundation. HBase. `http://hbase.apache.org`. [Online; accessed 01-May-2016].

[61] The Apache Software Foundation. Mllib. `https://spark.apache.org/mllib/`. [Online; accessed 01-May-2016].

[62] The PostgreSQL Global Development Group. PostgreSQL. `http://www.postgresql.org/`. [Online; accessed 26-Jan-2016].

[63] Takeaki Uno, Tatsuya Asai, Yuzo Uchida, and Hiroki Arimura. Lcm: An efficient algorithm for enumerating frequent closed item sets. In *FIMI*, 2003.

[64] Takeaki Uno, Tatsuya Asai, Yuzo Uchida, and Hiroki Arimura. An efficient algorithm for enumerating closed patterns in transaction databases. In *Discovery Science*, pages 16–31, 2004.

[65] Takeaki Uno, Masashi Kiyomi, and Hiroki Arimura. Lcm ver. 2: Efficient mining algorithms for frequent/closed/maximal itemsets. In *Proceedings of the Workshop on Frequent Itemset Mining Implementations (FIMI)*, 2004.

[66] Takeaki Uno, Masashi Kiyomi, and Hiroki Arimura. Lcm ver. 3: Collaboration of array, bitmap and prefix tree for frequent itemset mining. In *Proceedings of the 1st international workshop on open source data mining: frequent pattern mining implementations*, pages 77–86. ACM, 2005.

[67] Benoît Vaillant, Philippe Lenca, and Stéphane Lallich. A clustering of interestingness measures. In *Discovery Science*, pages 290–297. Springer, 2004.

[68] Vinod Kumar Vavilapalli, Arun C. Murthy, Chris Douglas, Sharad Agarwal, Mahadev Konar, Robert Evans, Thomas Graves, Jason Lowe, Hitesh Shah, Siddharth Seth, Bikas Saha, Carlo Curino, Owen O'Malley, Sanjay Radia, Benjamin Reed, and Eric Baldeschwieler. Apache Hadoop YARN: Yet Another Resource Negotiator. In *Proc. SOCC '13*, pages 5:1–5:16, 2013.

[69] Jilles Vreeken, Matthijs Van Leeuwen, and Arno Siebes. Krimp: mining itemsets that compress. *Data Mining and Knowledge Discovery*, 23(1):169–214, 2011.

[70] Jianyong Wang, Jiawei Han, Ying Lu, and Petre Tzvetkov. Tfp: An efficient algorithm for mining top-k frequent closed itemsets. *Knowledge and Data Engineering, IEEE Transactions on*, 17(5):652–663, 2005.

[71] Jianyong Wang, Jiawei Han, and Jian Pei. Closet+: Searching for the best strategies for mining frequent closed itemsets. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 236–245. ACM, 2003.

[72] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. *HotCloud*, 10:10–10, 2010.

[73] Osmar R Zaïane, Mohammad El-Hajj, and Paul Lu. Fast parallel association rule mining without candidacy generation. In *Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on*, pages 665–668. IEEE, 2001.

[74] Mohammed J Zaki. Scalable algorithms for association mining. *Knowledge and Data Engineering, IEEE Transactions on*, 12(3):372–390, 2000.

[75] Mohammed J Zaki and Ching-Jiu Hsiao. Charm: An efficient algorithm for closed itemset mining. In *SDM*, volume 2, pages 457–473, 2002.

[76] Mohammed J Zaki, Ching-Jui Hsiao, et al. Charm: An efficient algorithm for closed association rule mining. Technical report, Technical Report 99, 1999.

[77] Mohammed J Zaki, Srinivasan Parthasarathy, Mitsunori Ogihara, and Wei Li. Parallel algorithms for discovery of association rules. *Data mining and knowledge discovery*, 1(4):343–373, 1997.

[78] Mohammed J Zaki, Srinivasan Parthasarathy, Mitsunori Ogihara, Wei Li, et al. New algorithms for fast discovery of association rules. In *KDD*, volume 97, pages 283–286, 1997.

Discography

Because there's not only Science behind this manuscript.

Mixtape available at `http://mkir.ch/phdtape`.

1. Chilly Gonzales - *Shut Up And Play The Piano*
2. Depeche Mode - *Everything Counts*
3. Felix Da Housecat - *Madame Hollywood (Tiga's Mister Hollywood Version)*
4. Todd Terje - *Inspector Norse*
5. Siriusmo - *Einmal In Der Woche Schreien*
6. Sebastien Leger - *Snow Flakes*
7. Noir Désir - *The Holy Economic War*
8. NoFX - *Freedom Lika Shopping Cart*
9. Beat Connection - *Sunburn*
10. Alborosie - *Kingston Town*
11. Easy Star All-Stars feat. Sugar Minott - *When I'm Sixty-Four (Extended Dub Mix)*
12. Yael Naim - *Coward (Rone Remix)*
13. Casseurs Flowters - *06h16 - Des Histoires à Raconter*
14. The Doors - *People Are Strange*
15. Clara Moto - *Joy Departed*
16. LV feat. Tigran Hamasyan - *Ruiselede*
17. Stupeflip - *Stupeflip Vite !*
18. Guts - *As The World Turns*
19. Talpa - *Mathematical Existance*
20. Anja Schneider - *Dubmission*
21. Rone - *Bora Vocal*
22. Radiohead - *Everything In Its Right Place (Andi Müllers Mix)*
23. Jamie xx feat. Romy - *Seesaw*

APPENDIX A

Ranking with Fisher's exact test

The introduction of Fisher's exact test among CAPA's quality measures (page 68) or as a post-processing of TopPI's results (when ranking by p-value, p.85) is inspired by the LAMP algorithm, proposed by Minato *et al.* [47] and detailed in page 19. LAMP analyses transactional datasets where each transaction is annotated to show if it satisfies or not a target factor. The algorithm computes all closed itemsets having a p-value below a given threshold, *ie.* itemsets whose occurrences are positively correlated to the target factor. Its p-value computation relies on Fisher's exact test. Thus LAMP shows the interest and feasibility of Fisher's exact test to find strong correlation between some sub-populations and a target factor. However their experiments are made on small datasets (compared to ours) which contain a few thousands transactions over hundreds of frequent items.

Analogously, our target factor can be the presence of an item i in transactions, who can be correlated to the occurrence of itemsets I such that $i \notin closure(I)$. This would allow us to assess a statistical correlation between an itemset and a product, or between a population and a products category. But, as we are regularly observing supports in millions, Fisher's exact test raises a computational problem because it relies on binomial coefficients. We circumvent this problem by observing that we only need to rank itemsets or association rules: we are not interested in the absolute p-value. Hence we design a computable ranking measure, equivalent to Fisher's exact test.

We start in Section A.1 by recalling how a p-value is computed with Fisher's exact test. Then Section A.2 demonstrates a property of the p-value's factors, which is leveraged in Section A.3 to define a ranking measure equivalent to the p-value.

For brevity, in the following $support(I)$ is denoted σ_I and $support(\{i\})$ as σ_i.

	i	$\overline{i}$	
I	$\sigma_{I\cup i}$	$\sigma_I - \sigma_{I\cup i}$	σ_I
$\overline{I}$	$\sigma_i - \sigma_{I\cup i}$	$\lvert\mathcal{D}\rvert - (\sigma_I + \sigma_i - \sigma_{I\cup i})$	$\lvert\mathcal{D}\rvert - \sigma_I$
	σ_i	$\lvert\mathcal{D}\rvert - \sigma_i$	$\lvert\mathcal{D}\rvert$

Table A.1: Contingency table between the occurrence of an item i and those of an itemset I ($i \notin closure(I)$).

A.1 Computing p-values with Fisher's exact test

In our setting, computing the correlation between occurrences of an item i and those of an itemset I relies on the contingency table shown in Table A.1. Following Fisher's exact test, we know that the probability to obtain the observed values, under the null hypothesis, is defined by:

$$P_{i,I,\mathcal{D}}(z) = \frac{\binom{\sigma_I}{z}\binom{|\mathcal{D}|-\sigma_I}{\sigma_i - z}}{\binom{|\mathcal{D}|}{\sigma_i}}, \text{ where } z = \sigma_{I\cup i}$$

The p-value is obtained by summing this probability with the probabilities to obtain more biased distributions under the same marginal totals:

$$p(i, I, \mathcal{D}) = \sum_{z=\sigma_{I\cup i}}^{min(\sigma_I,\sigma_i)} P_{i,I,\mathcal{D}}(z)$$

Measuring a low p-value (typically, below 0.05) invalidates the null hypothesis, allowing one to claim that "buying all items in I is correlated to buying item i", for example. The binomial coefficients involved in $P_{i,I,\mathcal{D}}(z)$ can quickly overflow the capacities of our usual representations (whether as integer or floating-point numbers). Hence the computation of p relies on another, equivalent definition:

$$p(i, I, \mathcal{D}) = \sum_{z=\sigma_{I\cup i}}^{min(\sigma_I,\sigma_i)} e^{ln(P_{i,I,\mathcal{D}}(z))} \tag{A.1}$$

$ln(P_{i,I,\mathcal{D}}(z))$ is computed by developing binomial coefficients as factorials, then distributing the logarithm and introducing Gamma functions using $\Gamma(n) = (n-1)!$. The logarithm of the Gamma function is a well known computational problem [10]; we re-use the implementation available in WordHoard[1].

Thus each $ln(P_{i,I,\mathcal{D}}(z))$ is quickly computed, without overflow. But it reaches very low (negative) values, especially for our itemsets of interest: when applying the exponential, all sum's factors are rounded to zero. Because of this rounding error, we may compute $p(i, I, \mathcal{D}) = 0$ for thousands of itemsets which are correlated to i. Such results are false and prevent us from ranking the corresponding itemsets precisely.

These computational difficulties explain why Fisher's exact test is usually invoked on small datasets, counting their samples in thousands. With millions of

[1] `http://wordhoard.northwestern.edu/`

transactions, we are pushing the computation to its limits. But we are not interested in the p-value itself: we only need to compare the $p(i, I, \mathcal{D})$ obtained by various itemsets I, for a fixed i and $\mathcal{D}$. By precising which factors have the greatest contribution to the p-value we found a new, computable, ranking function which performs a similar ordering.

A.2 Comparing each factor's impact on the p-value

We now precise how to assess if I and i are positively or negatively correlated. Then we demonstrate that, when the correlation is positive, the most important terms in the sum of equation A.1 are the first ones, starting from $z = \sigma_{I \cup i}$.

If we assume that I and i occur in transactions independently, then the probability to observe both in a transaction is:

$$P(i)P(I) = \frac{\sigma_i}{|\mathcal{D}|}\frac{\sigma_I}{|\mathcal{D}|}$$

Hence the expected support of $I \cup \{i\}$, under the independence assumption, is

$$\eta(I, i, \mathcal{D}) = \frac{\sigma_I \sigma i}{|\mathcal{D}|}$$

If the observed support $\sigma_{I \cup i}$ is lower than $\eta(I, i, \mathcal{D})$, then I and i are negatively correlated. In this case the p-value $p(i, I, \mathcal{D})$ will be closer to 1 than to 0, and I will not appear among i's most correlated itemsets. Therefore we tolerate the lack of precision when computing $p(i, I, \mathcal{D})$. Otherwise, we make the following observation:

Property 3. *For $z \geq \eta(I, i, \mathcal{D})$, $P_{i,I,\mathcal{D}}(z)$ is monotonically decreasing.*

Proof. Assuming that $z \geq \frac{\sigma_I \sigma i}{|\mathcal{D}|}$, we prove that $P_{i,I,\mathcal{D}}(z+1) \leq P_{i,I,\mathcal{D}}(z)$.

We start by decomposing $P_{i,I,\mathcal{D}}(z+1)$:

$$\begin{aligned}
P_{i,I,\mathcal{D}}(z+1) =& \frac{\binom{\sigma_I}{z+1}\binom{|\mathcal{D}|-\sigma_I}{\sigma_i-z-1}}{\binom{|\mathcal{D}|}{\sigma_i}} \\
=& \frac{\sigma_I!}{(z+1)!(\sigma_I-z-1)!} \cdot \frac{(|\mathcal{D}|-\sigma_I)!}{(\sigma_i-z-1)!(|\mathcal{D}|-\sigma_I-\sigma_i+z+1)!} \cdot \frac{\sigma_i!(|\mathcal{D}|-\sigma_i)!}{|\mathcal{D}|!} \\
=& \frac{\sigma_I - z}{z+1} \cdot \frac{\sigma_I!}{z!(\sigma_I-z)!} \cdot \frac{\sigma_i - z}{|\mathcal{D}|-\sigma_I-\sigma_i+z+1} \cdot \frac{(|\mathcal{D}|-\sigma_I)!}{(\sigma_i-z)!(|\mathcal{D}|-\sigma_I-\sigma_i+z)!} \cdot \\
& \frac{\sigma_i!(|\mathcal{D}|-\sigma_i)!}{|\mathcal{D}|!} \\
=& \frac{1}{z+1} \cdot (\sigma_I - z) \cdot (\sigma_i - z) \cdot \frac{1}{|\mathcal{D}|-\sigma_I-\sigma_i+z+1} \cdot P_{i,I,\mathcal{D}}(z)
\end{aligned}$$

From our assumption on z, we can derive an upper bound for the three first factors:

1. $z+1 > z \geq \frac{\sigma_I \sigma_i}{|\mathcal{D}|}$, hence $\frac{1}{z+1} < \frac{|\mathcal{D}|}{\sigma_I \sigma_i}$
2. $\sigma_I - z \leq \sigma_I - \frac{\sigma_I \sigma_i}{|\mathcal{D}|} = \frac{|\mathcal{D}|\sigma_I - \sigma_I \sigma_i}{|\mathcal{D}|} = \frac{\sigma_I(|\mathcal{D}| - \sigma_i)}{|\mathcal{D}|}$
3. likewise, $\sigma_i - z \leq \frac{\sigma_i(|\mathcal{D}| - \sigma_I)}{|\mathcal{D}|}$

Therefore:

$$\begin{aligned} P_{i,I,\mathcal{D}}(z+1) \leq & \frac{|\mathcal{D}|}{\sigma_I \sigma_i} \cdot \frac{\sigma_I(|\mathcal{D}| - \sigma_i)}{|\mathcal{D}|} \cdot \frac{\sigma_i(|\mathcal{D}| - \sigma_I)}{|\mathcal{D}|} \cdot \frac{1}{|\mathcal{D}| - \sigma_I - \sigma_i + z + 1} \cdot P_{i,I,\mathcal{D}}(z) \\ \leq & \frac{1}{|\mathcal{D}|} \cdot (|\mathcal{D}| - \sigma_i) \cdot (|\mathcal{D}| - \sigma_I) \cdot \frac{1}{|\mathcal{D}| - \sigma_I - \sigma_i + z + 1} \cdot P_{i,I,\mathcal{D}}(z) \end{aligned}$$

The product of the first four terms in the latter inequality is smaller than 1. Indeed, from our assumption that $z+1 \geq \frac{\sigma_I \sigma_i}{|\mathcal{D}|}$, by adding $|\mathcal{D}| - \sigma_I - \sigma_i$ we obtain:

$$\begin{aligned} |\mathcal{D}| - \sigma_I - \sigma_i + z + 1 \geq |\mathcal{D}| - \sigma_I - \sigma_i + \frac{\sigma_I \sigma_i}{|\mathcal{D}|} = & \frac{1}{|\mathcal{D}|} \cdot (|\mathcal{D}|^2 - |\mathcal{D}|\sigma_I - |\mathcal{D}|\sigma_i + \sigma_I \sigma_i) \\ = & \frac{1}{|\mathcal{D}|} \cdot (|\mathcal{D}| - \sigma_I) \cdot (|\mathcal{D}| - \sigma_i) \end{aligned}$$

Thus $P_{i,I,\mathcal{D}}(z+1) \leq P_{i,I,\mathcal{D}}(z)$.

□

Property 3 implies that, if $\sigma_{I \cup i} > \eta(I, i, \mathcal{D})$, then the first term in the p-value's sum (see Equation A.1) is the greatest. In order to compute a useful approximation of $p(i, I, \mathcal{D})$, we therefore have to ensure that $P_{i,I,\mathcal{D}}(\sigma_{I \cup i})$ gets the best precision.

A.3 An equivalent ranking measure

Following the previous observation, when I and i are positively correlated we introduce an adjustment factor $a(i, I, \mathcal{D})$, which is the only integer such that:

$$1000 * a(i, I, \mathcal{D}) + ln(P_{i,I,\mathcal{D}}(\sigma_{I \cup i})) \in [-500, 500] \tag{A.2}$$

We choose $[-500, 500]$ because, using 64-bits floating point numbers, e^x never overflows for $x \in [-500, 500]$. If $\sigma_{I \cup i} \leq \eta(I, i, \mathcal{D})$ then $a(i, I, \mathcal{D}) = 0$.

Once this adjustment factor has been set we can compute:

$$r(i, I, \mathcal{D}) = e^{1000 * a(i,I,\mathcal{D})} * p(i, I, \mathcal{D}) = \sum_{z=\sigma_{I \cup i}}^{min(\sigma_I, \sigma_i)} e^{1000 * a(i,I,\mathcal{D}) + ln(P_{i,I,\mathcal{D}}(z))} \tag{A.3}$$

Our ranking function sorts itemsets descendently regarding $a(i, I, \mathcal{D})$, then ascendently regarding $r(i, I, \mathcal{D})$. This closely approximates a ranking by increasing $p(i, I, \mathcal{D})$.

In practice, $r(i, I, \mathcal{D})$ is never null because the first term of the above sum fits in a 64-bits floating point number, thanks to the adjustment factor. Some of the following terms may still overflow and get rounded to 0, in which case the monotony allows an early termination of the sum's computation. The computed value is a satisfying approximation of $r(i, I, \mathcal{D})$ and, usually, the most correlated itemsets differ in the ranking by a alone.

APPENDIX B

Clustering results

This appendix contains the raw results of the clustering done for our empirical evaluation, presented in Section 5.2, p.71. Clusters have been coloured to show our families, following Table 5.2 (p.68) but ambiguous clusters are left blank. Note that our final families also take into account similar clusterings, according to *Spearman's rank correlation* and Overlap@50.

B.1 With identical targets

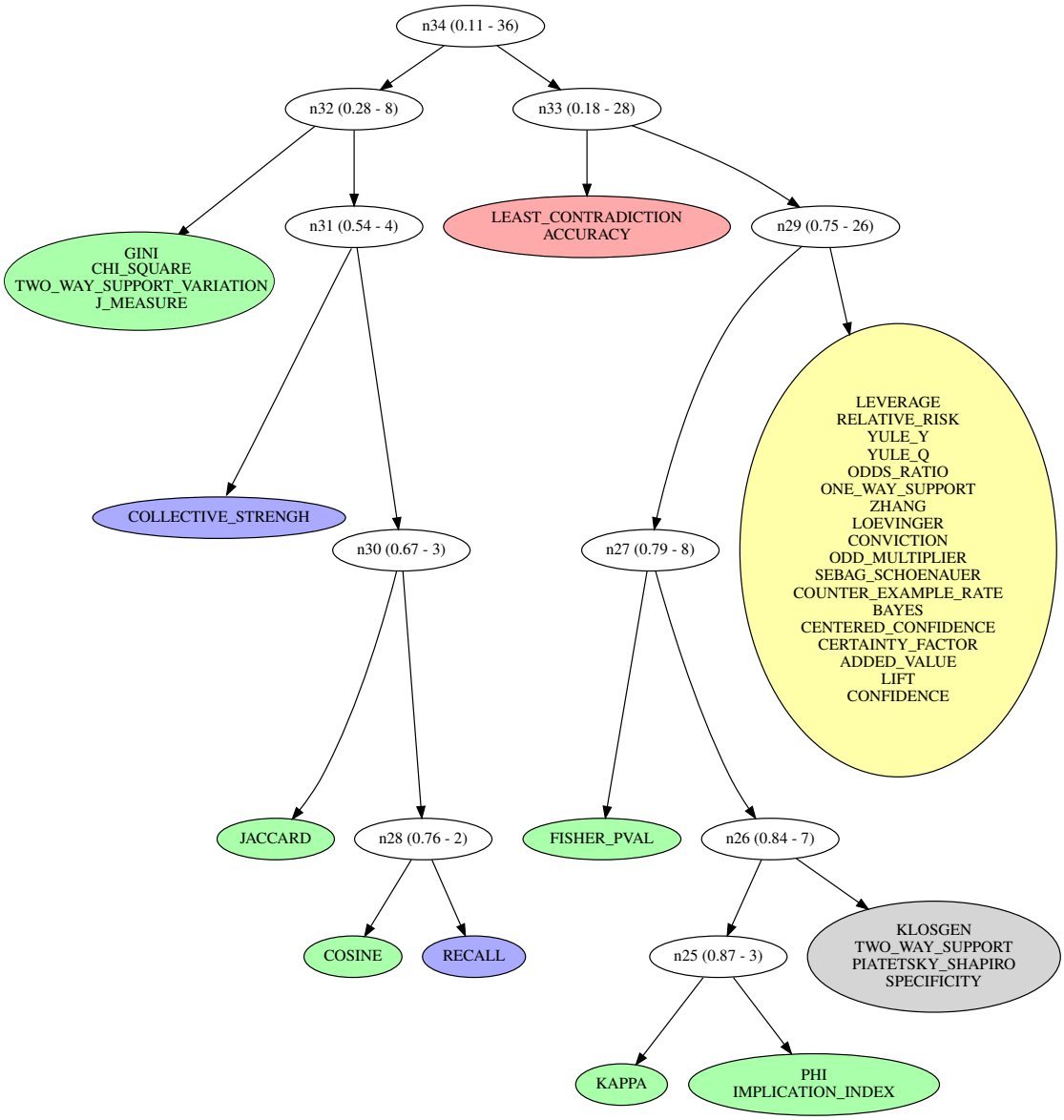


Figure B.1: `demo_assoc`, clustered by τ

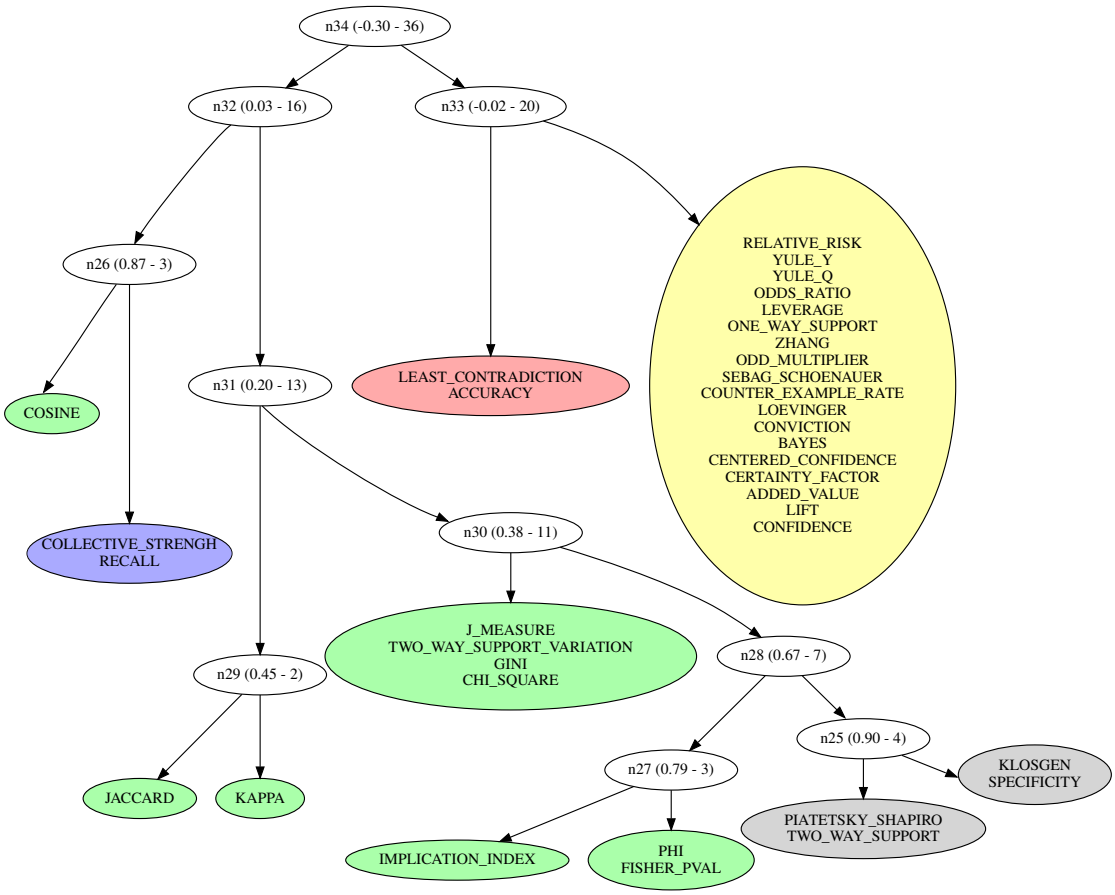


Figure B.2: `demo_assoc`, clustered by *NDCC*

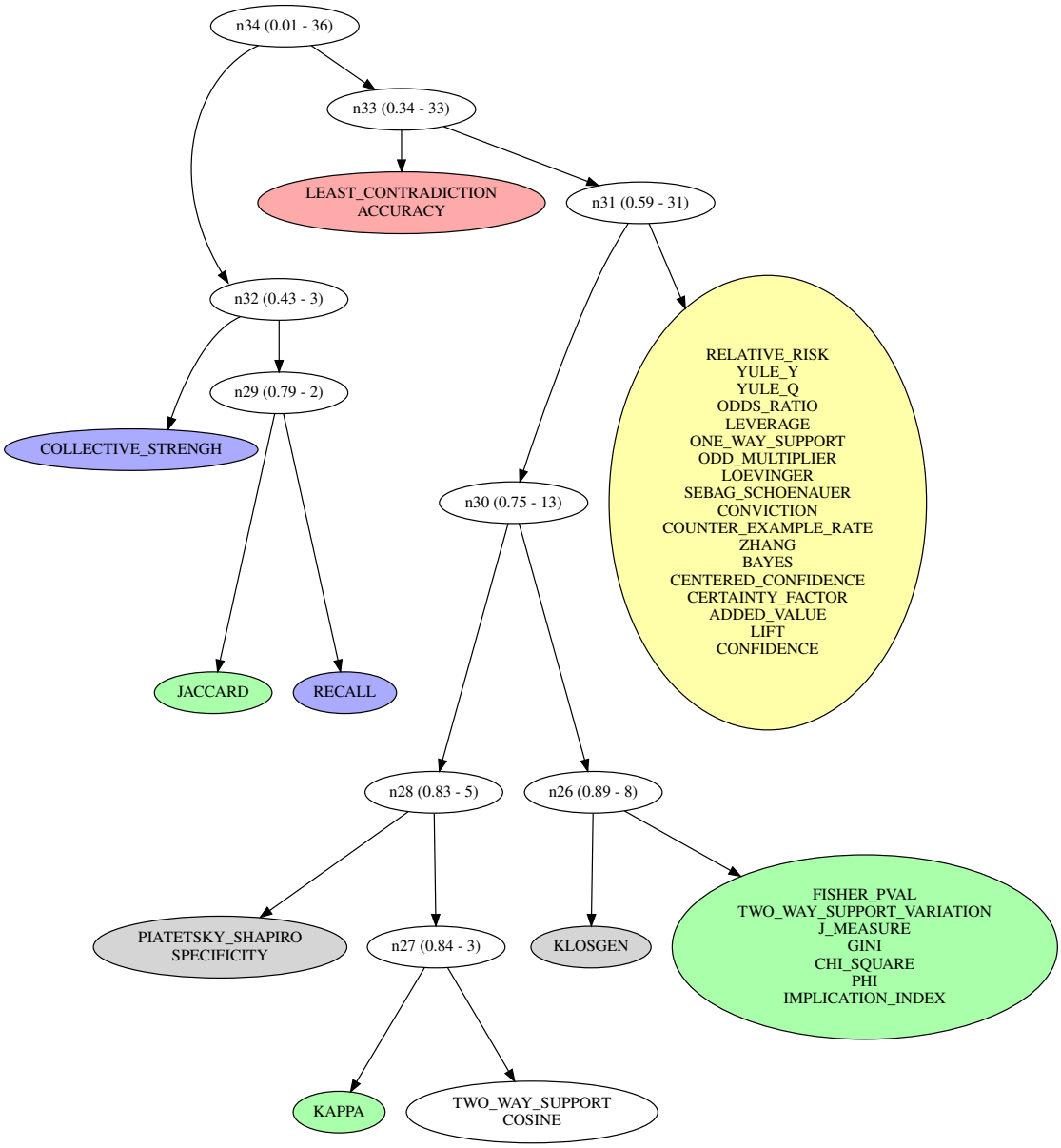


Figure B.3: `prod_assoc_receipt`, clustered by τ

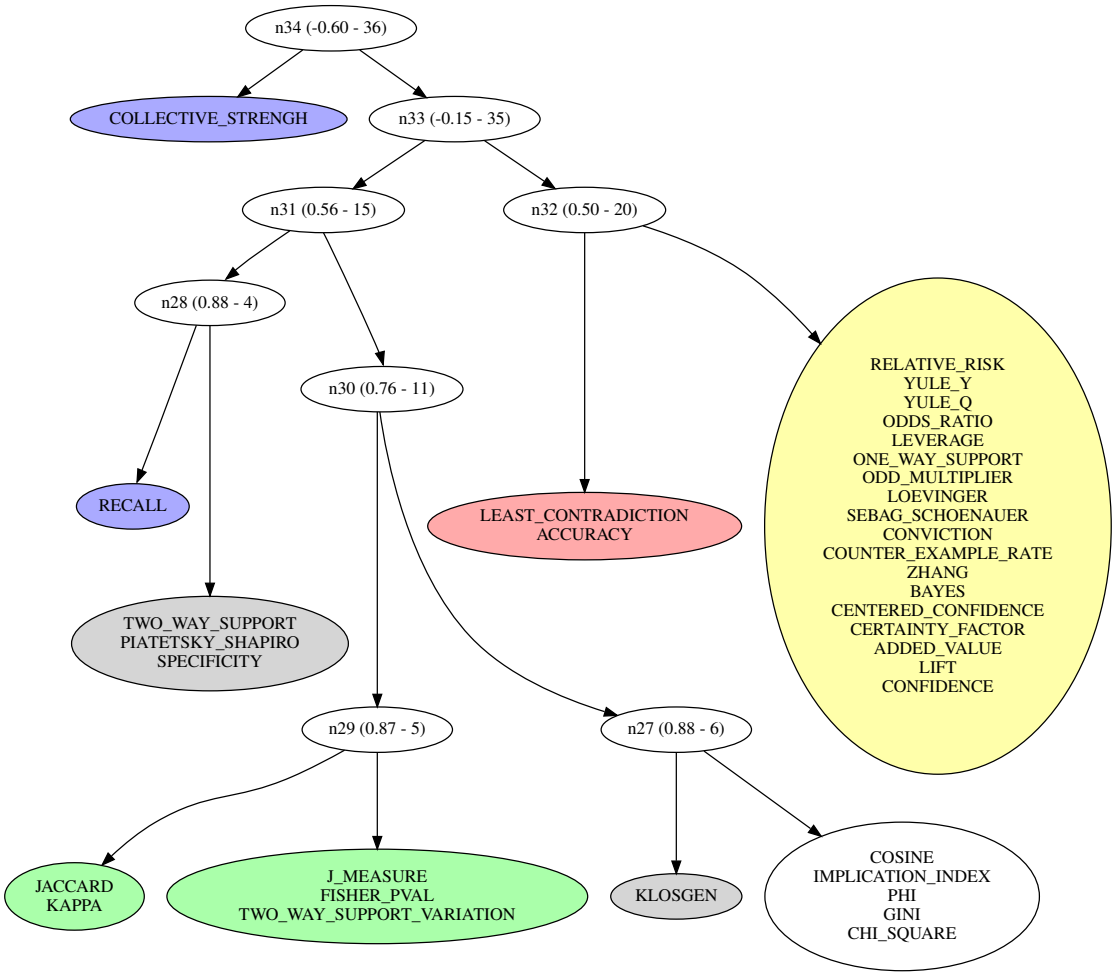


Figure B.4: `prod_assoc_receipt`, clustered by *NDCC*

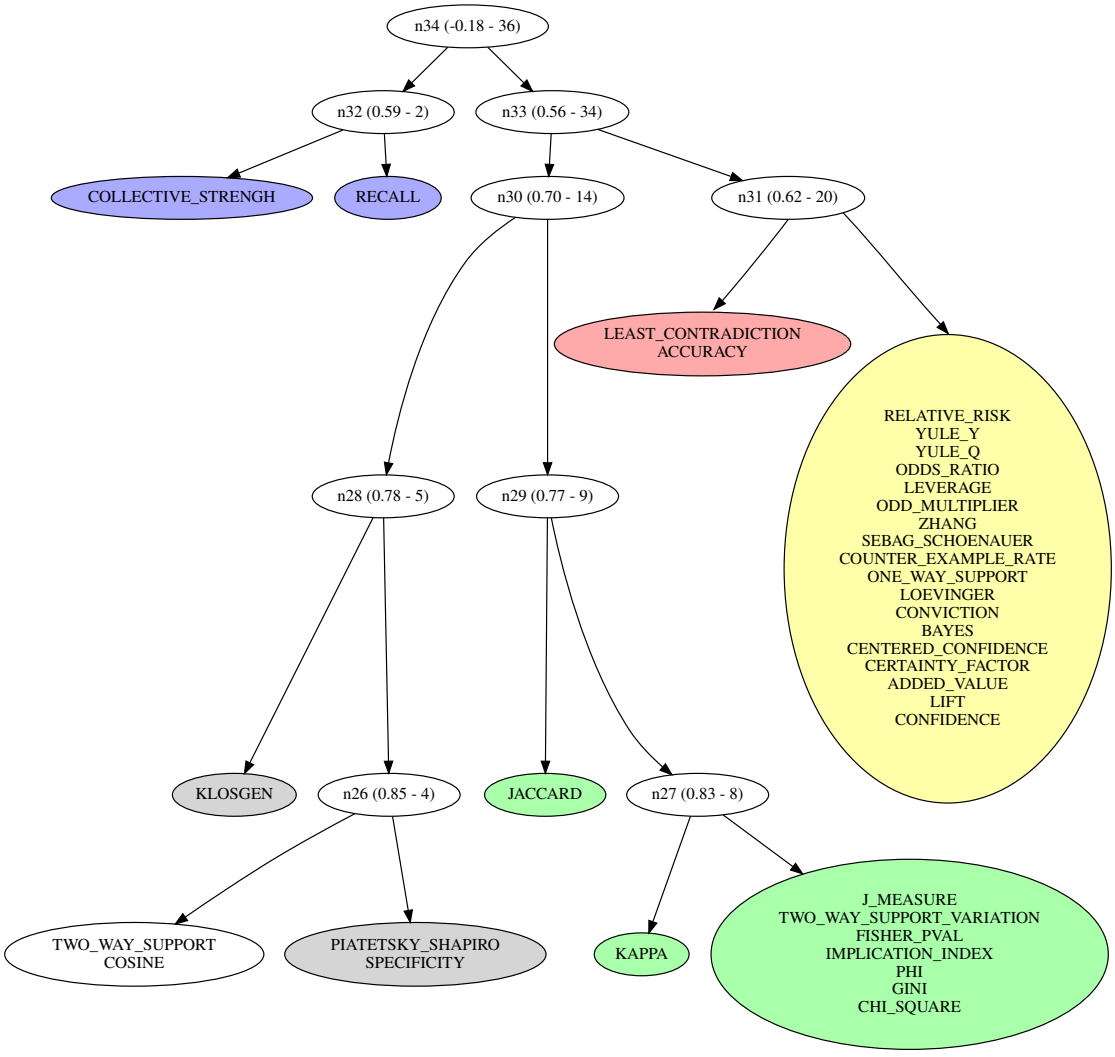


Figure B.5: `prod_assoc_client`, clustered by τ

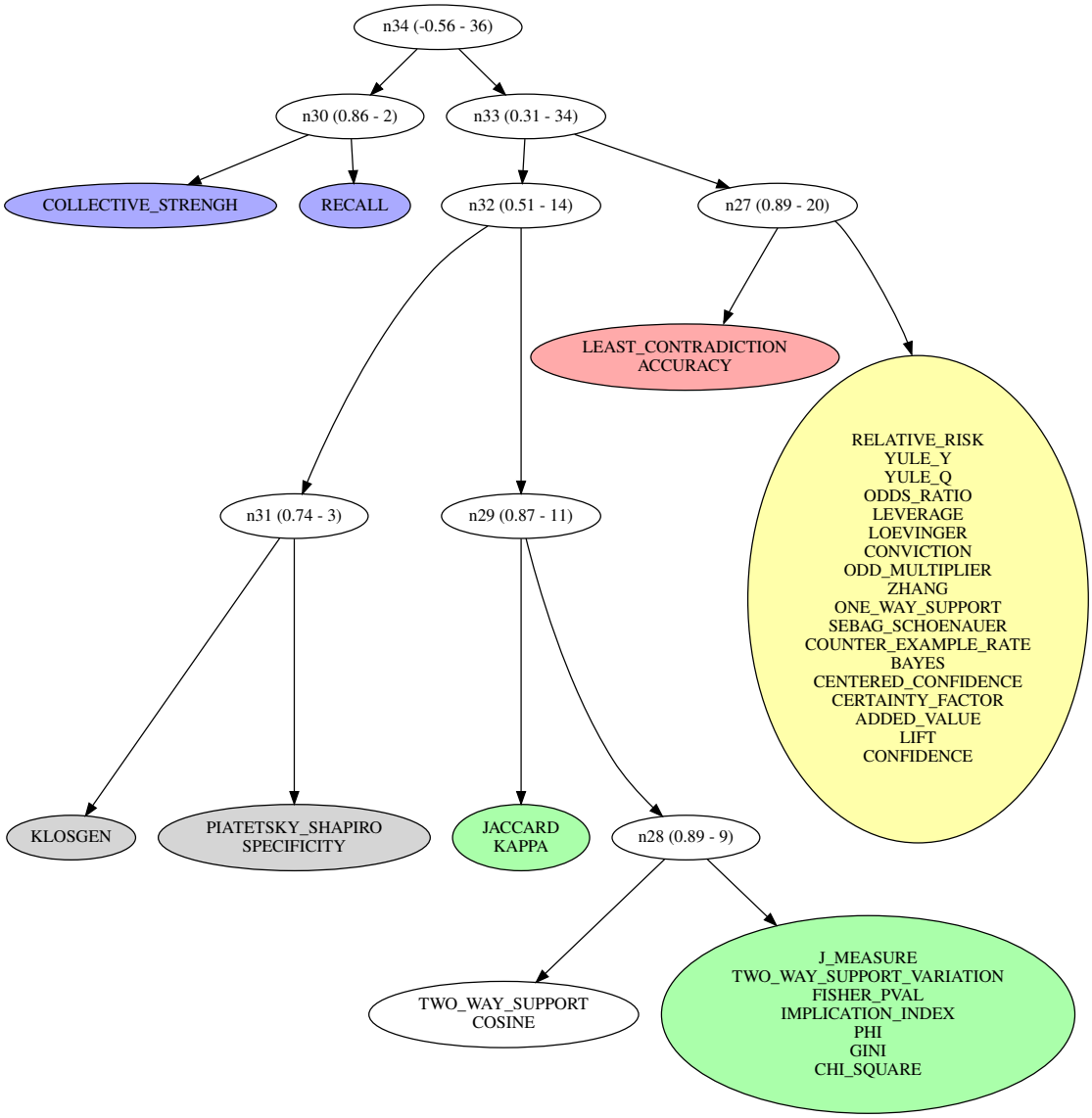


Figure B.6: `prod_assoc_client`, clustered by *NDCC*

B.2 With different targets

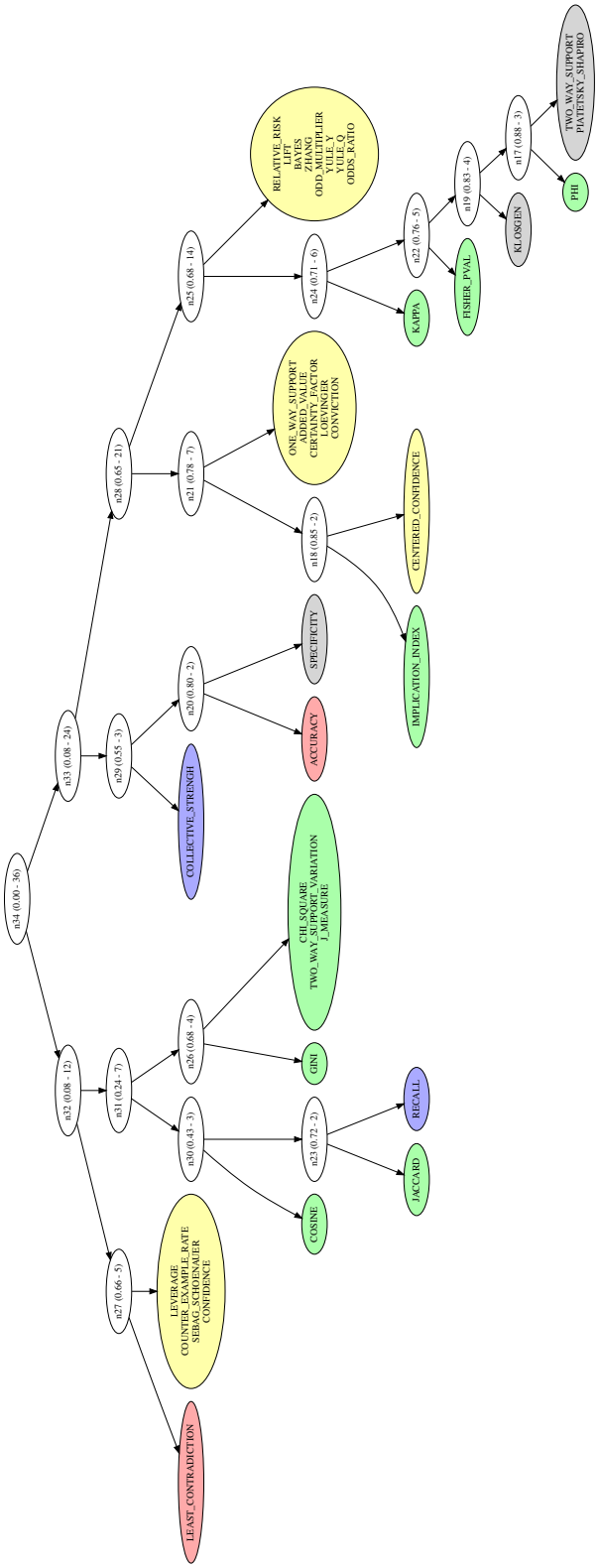


Figure B.7: `demo_assoc`, clustered by τ

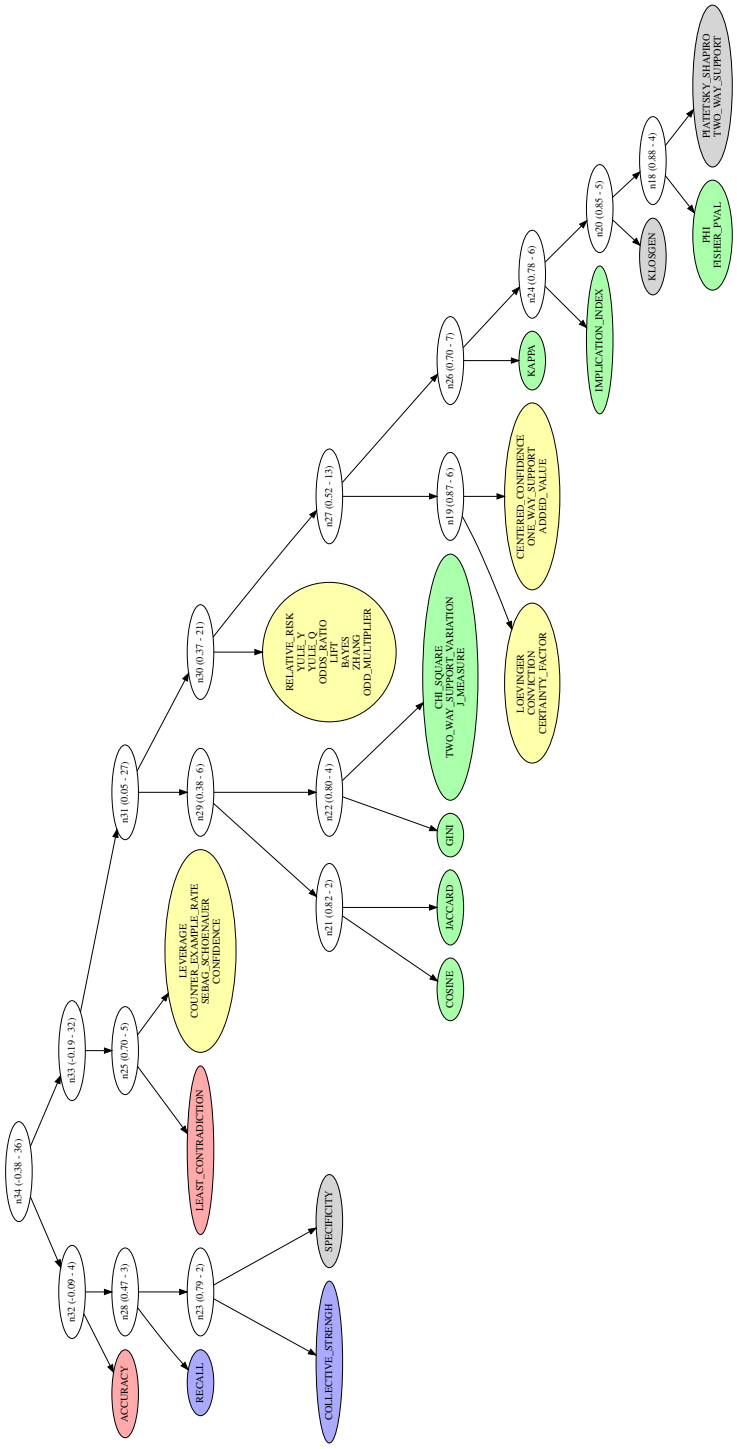


Figure B.8: `demo_assoc`, clustered by *NDCC*

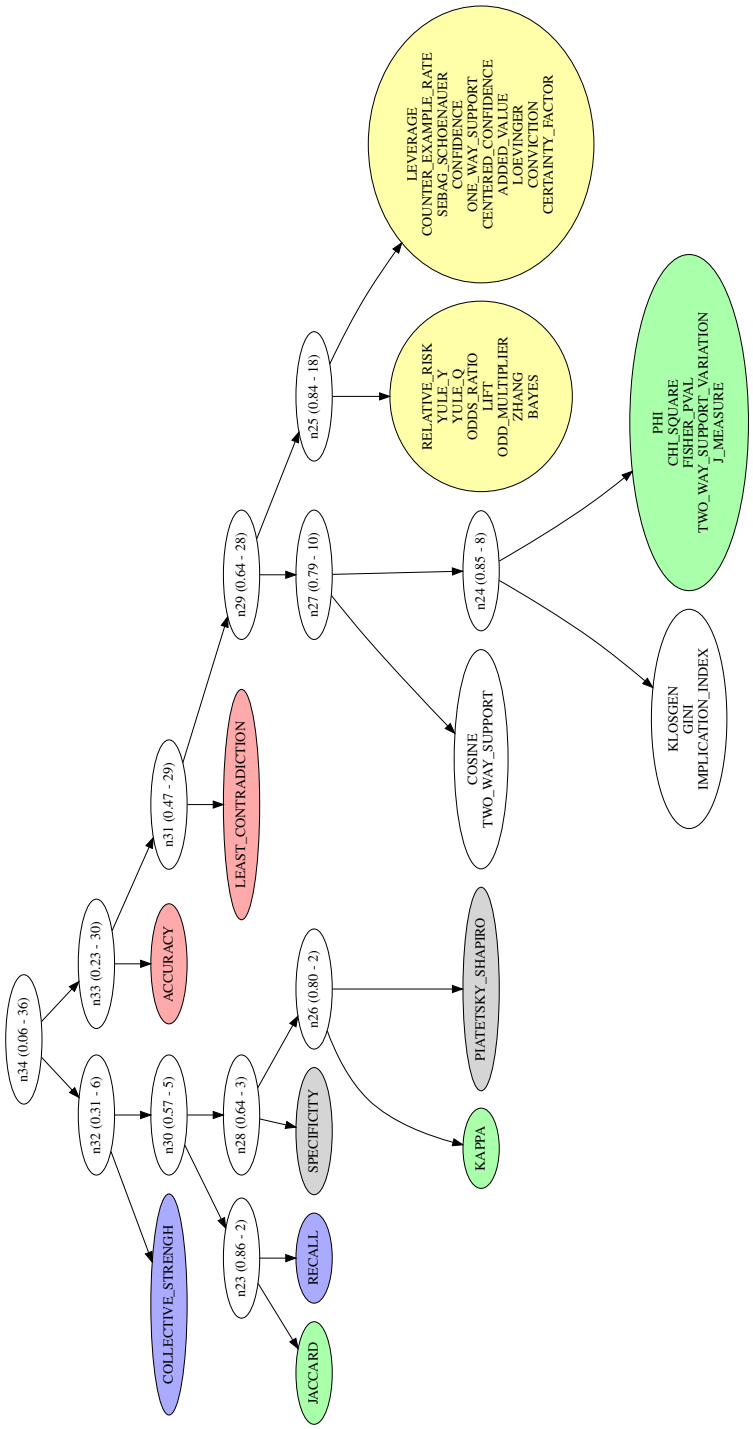


Figure B.9: `prod_assoc_receipt`, clustered by τ

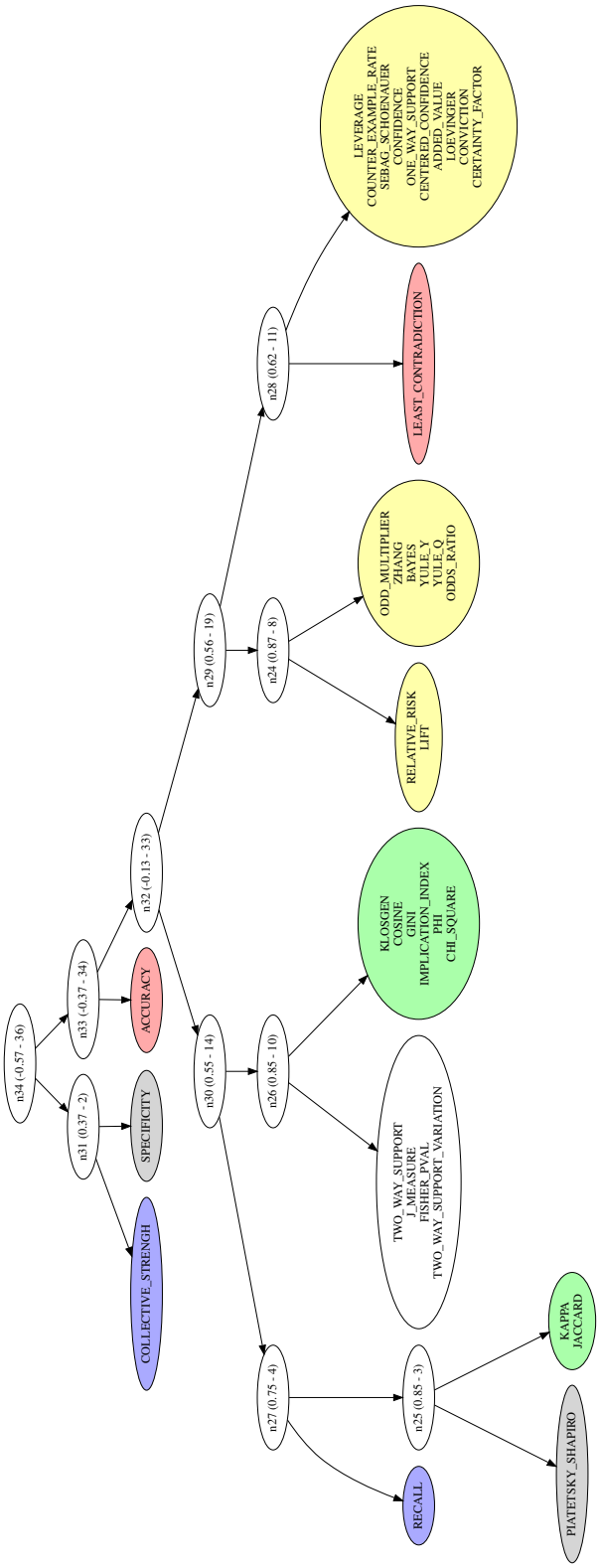


Figure B.10: `prod_assoc_receipt`, clustered by *NDCC*

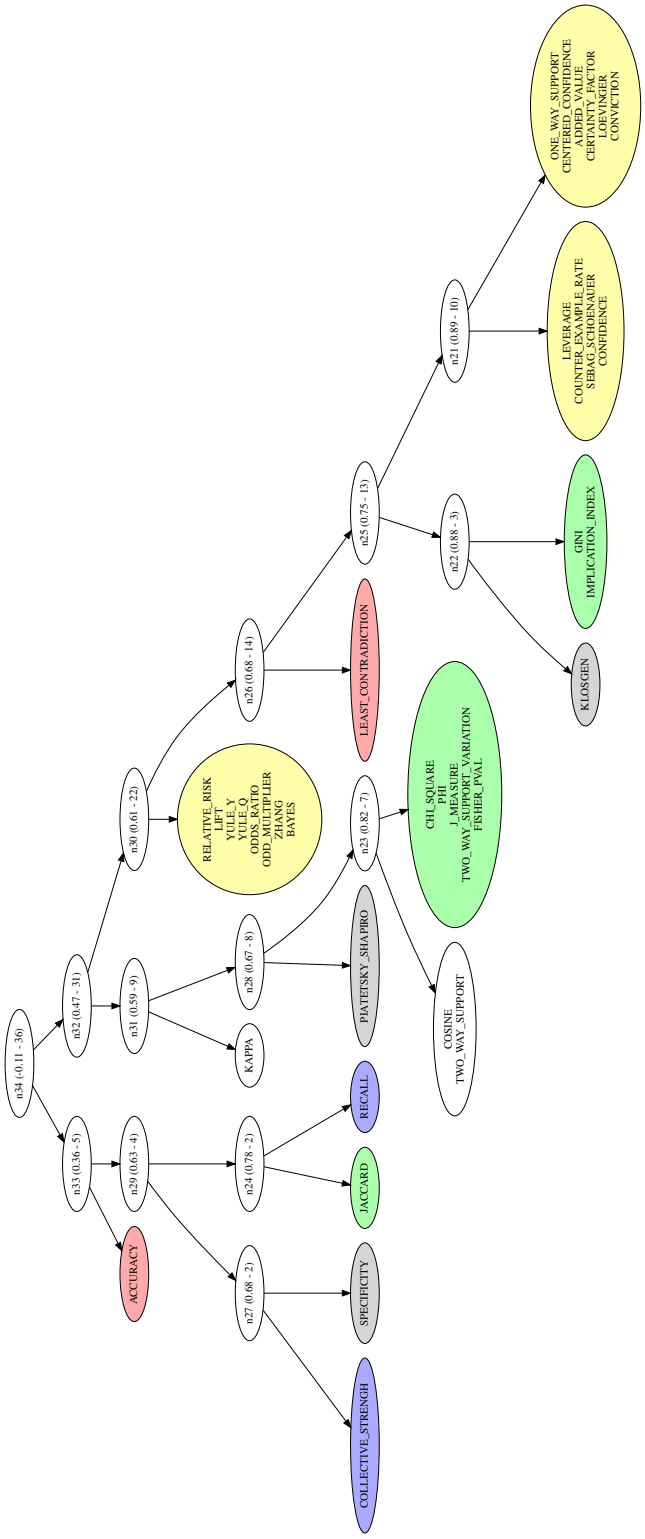


Figure B.11: `prod_assoc_client`, clustered by τ

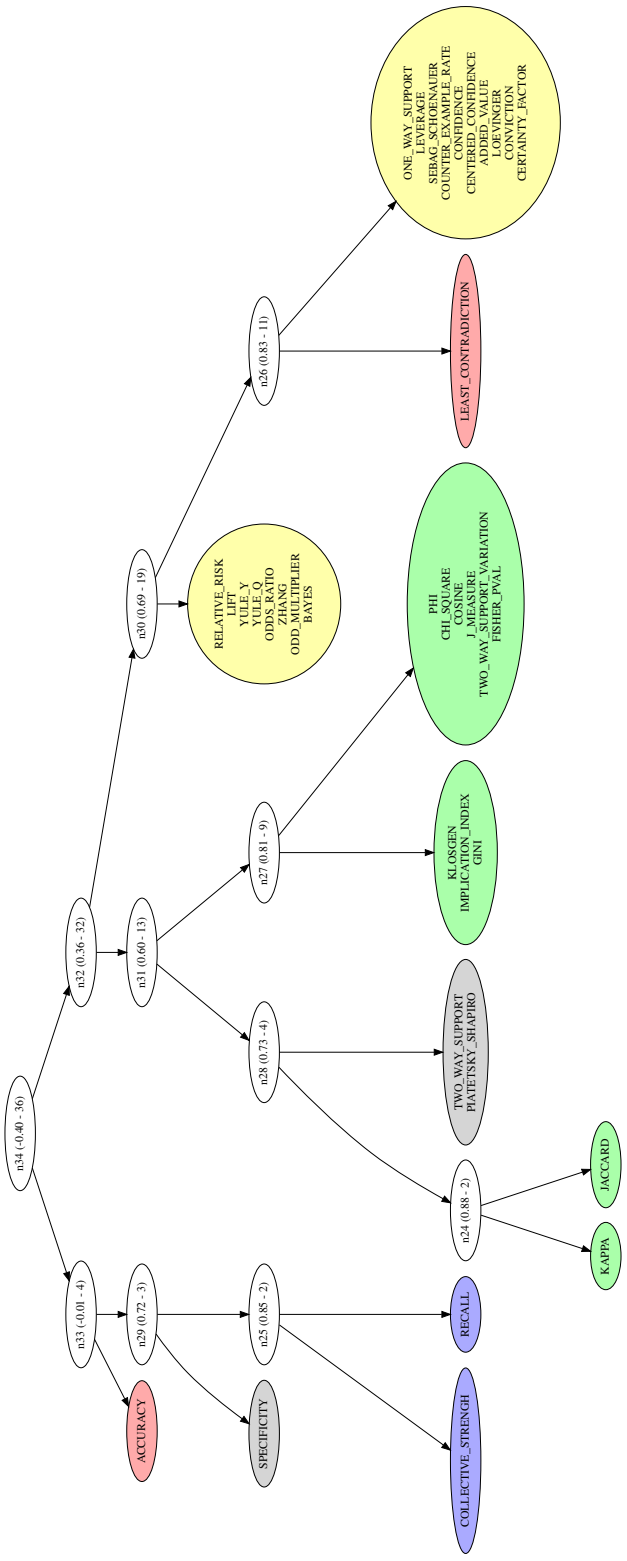


Figure B.12: `prod_assoc_client`, clustered by *NDCC*