Universiteit Antwerpen

Faculteit Wetenschappen

Informatica

An Analysis of Mining Algorithms in Databases and Streams

Een analyse van mining algoritmes voor databanken en gegevensstromen

Proefschrift voorgelegd tot het behalen van de graad van
Doctor in de Wetenschappen, richting Informatica
te verdedigen door

Nele DEXTERS

Promotor: prof. dr. Jan Paredaens

Antwerpen, 12 September 2007

Distribution:
University of Antwerp

Available in .pdf format.
Just send an email to nele.dexters@gmail.com

ISBN

To my family
Antwerp, September 2007

Acknowledgments

I would like to thank many people that contributed to the realization of this dissertation.

First of all, I would like to thank my adviser Jan Paredaens, who gave me the opportunity to work at the University of Antwerp. As a mathematician, he challenged me with a job at SCOB, together with Frans Buelens and Hans Willems, and databases started to dominate my life. Jan believed in me and motivated me to visit Bloomington. Thanks to him, I was in the States for three times.
I would like to thank Dirk Van Gucht and his family for all the support. I'll never forget the way they accepted me, and treated me as a part of their family. These visits to Indiana will stay in my heart forever. Also thanks to Paul Purdom, for the nice cooperation when I was in Bloomington. As a short break, he was telling me how to do *lock picking* and he explained me what a *street sweeper brush bristle* is and how these two combine. Collaboration with Dirk and Paul resulted in the work covered in Chapters 3 and 4.
Thanks to my colleagues here at the University of Antwerp, for creating this nice and stimulating place to work. Special thanks to Toon Calders and Bart Goethals and the rest of ADReM. I really enjoyed your company and the discussions we had, although we sometimes have a different interpretation of the word deadline :-). The many hours we spent together resulted in the material presented in Chapter 5. Also thanks to so many other colleagues/friends I worked with. I really liked "het Wetenschapsfeest", "de Kinderuniversiteit", "de Mathdropping" and all other projects in which we succeeded together. Thanks for all the creativity, the joy and the smiles.
I an very much in dept of my parents. Thank you, mama and papa, for offering me the opportunity to start studying at the university. For believing in me, and helping me with the practical organization. For being patient and for offering me healthy food. For grabbing me from my desk and forcing me to take a break with our dog Max.
Finally, special thanks to David, for supporting me in so many ways ...

Antwerpen, 2007

Abstract

In this dissertation, we study the *frequent set mining problem* in both *databases* and *streams*. In particular, we focus on algorithms that find frequent sets in both static datasets and dynamic data streams.

In the context of static databases, we perform a theoretical analysis for a selection of well-known frequent set mining algorithms. The considered algorithms are all typed by the characteristic of generating candidates and testing, i.e. counting, in the database if these candidates are frequent. The concepts candidate, success and failure are analyzed from a probabilistic point of view, for the selection of candidate-based frequent set mining algorithms and different distributions of the dataset. This results in a visualization of the probability of candidacy, success and failure. We also theoretically study datasets that contain large maximal frequent sets. The goal here is jumping, to discover the maximal frequent sets early, and exploit this knowledge to save time. A family of theoretical jumping algorithms is presented, for a very basic and simple data model. The analysis shows that we are very close to Max-Miner.

In the context of dynamic data streams, we start with a completely new frequency measure, max-frequency. Based on the new frequency measure, we present an incremental algorithm that maintains a small summary of the stream to reproduce the exact frequency of a target set immediately at any time. Every time a new transaction enters the stream, the summary has to be updated. Obviously, the summary has to remain very small for the algorithm to be efficient. Theoretical results for the worst case and experiments for real-life data show that this new method is indeed space efficient.

Contents

List of Figures

1

Introduction

This dissertation investigates topics within the wide area of **data mining**, a relatively young research discipline that combines elements of databases, statistics, machine learning, artificial intelligence and probabilistic modeling.

With the development of relational databases, companies had the opportunity to efficiently store huge amounts of data. Not only this huge amount of data itself, but also the possibility to research and question it, makes it so precious. For a company, it is very important to extract useful knowledge from the data they collected. The existence of such data, and the necessity and challenge to analyze it, is the main motivation for data mining: it is the analysis of huge amounts of observed data to discover interesting, unexpected relationships, patterns or regularities, to summarize the data in novel ways that are understandable and useful to the data owner [34]. The idea is to describe the data with the help of the information found, so it can be interpreted and used practically.

Discovering frequent patterns is a fundamental problem in data mining. It is by no means completely solved and it remains a major challenge, in particular for extremely large databases and for streaming data. Data mining literature discusses lots of algorithms to find frequent patterns experimentally and empirically, but only few papers are devoted to their *theoretical* analysis. This work tries to fill this gap.

This chapter gives an introduction to the world of finding frequent patterns, while focusing on one particular kind of patterns: *frequent sets*. Section 1.1 presents the traditional setting of mining frequent sets in static databases, while Section 1.2 talks about mining frequent sets in dynamic streaming data. In Section 1.3, an overview of the contents of this dissertation is given.

1.1 Frequent Set Mining

In recent years, data mining (also known as knowledge discovery in databases - abbreviated KDD) has attracted a lot of interest in the database community, caused by the large amounts of computerized data that organizations have about their business. Data mining tries to find interesting patterns within databases, such as association rules, correlations, sequences, episodes, classifiers, clusters and many more [37, 52]. The discovery of *frequent sets*, and based on these sets, *association rules* is probably the most popular discipline within the field of data mining [11]. The motivation for this research originates with the need to analyze large amounts of *supermarket basket data*, that is, to examine customer behavior in terms of the purchased products. Association rules describe how often products are bought together. For instance "bread ⇒ butter (87%)" states that 87% of the customers that bought bread, also bought butter. Such rules can be useful to improve the quality of decisions concerning product pricing, promotion or store layout: what to put on sale, how to design coupons, how to place merchandise on shelves in order to maximize the profit, etc. A key step in the process of generating association rules is frequent set mining; once the frequent sets are found, it is very straightforward to find association rules.

In market basket analysis, the goal is to discover associations between products, mostly called *items*, that are often bought together. If $\mathcal{I} = \{i_1, i_2, \ldots, i_n\}$ represents the set of all items that are sold in the supermarket, we can model a shoppers' basket as a collection of these items from the universe of all items. For example, the basket of customer x is denoted by set I_x consisting of items from $\mathcal{I}$. If we do so for each customer, we obtain a collection of sets that describe the products that are bought in the supermarket. Together with an *identifier*, each *set* is stored in the *database*. The combination of the identifier and the corresponding set of items is called a *transaction*.

Example 1.1.1. *If John buys bread, butter and milk, and Lisa buys chips, beer and bread too, these two sets of items can be saved in database $\mathcal{D}$.*

$\mathcal{D}$ =

TID	***Items***	
1	*bread, butter, milk*	← *transaction*
2	*chips, beer, bread*	

↑ *transaction identifier*

Most of the time, the information is stored in the database as *binary* vectors. The occurrence of a particular item is reflected by using 1, and the absence by 0.

Example 1.1.2. *If John buys bread, butter and milk, and Lisa buys chips, beer and bread too, these two sets of items can be saved binary in database $\mathcal{D}$.*

$\mathcal{D}$ =

TID	***bread***	***butter***	***milk***	***chips***	***beer***
1	*1*	*1*	*1*	*0*	*0*
2	*1*	*0*	*0*	*1*	*1*

Definition 1.1.1. *The size of database $\mathcal{D}$ is the amount of transactions in $\mathcal{D}$, denoted $|\mathcal{D}| = b$. It is perfectly possible that some transactions occur more than once in the database.*

This setting is formalized in the *Association Rule Mining problem*, that partly covers the *Frequent Set Mining problem*. Before we can formally specify both problems, we need to define some important concepts.

Definition 1.1.2. *To denote a particular set X, we assume a* natural *order on the items. For example, set $X = \{a, b, c, d\}$ is denoted* abcd. *The* length *of set X is the amount of items contained in the set and is denoted by $|X|$. For set $X = \{a, b, c, d\}$, we thus have $|X| = 4$. Thanks to this ordering, we can pick the* i*-th item from set X, with $1 \leq i \leq |X|$. In our example, the 3rd item in X, denoted as $X[3]$, equals c.*

Definition 1.1.3. *Given a set $\mathcal{I} = \{i_1, i_2, \ldots, i_n\}$ and a database $\mathcal{D}$, the* support *of a particular set $X \subseteq \mathcal{I}$ in $\mathcal{D}$, denoted $support(X, \mathcal{D})$, is the number of transactions in $\mathcal{D}$ that contain all the items of X. We omit $\mathcal{D}$ when this is clear from the context; in this case we use the notation $support(X)$.*

Definition 1.1.4. *Given a set $\mathcal{I} = \{i_1, i_2, \ldots, i_n\}$, a database $\mathcal{D}$ of size b and a threshold σ, a set $X \subseteq \mathcal{I}$ is called* frequent *if $support(X) \geq \sigma$. If $\sigma \leq b$, it is clear that $\emptyset$ is frequent.*

This means that the set X is called frequent if at least σ transactions in database $\mathcal{D}$ contain all the items from X. In the shopping context, this illustrates that at least σ shoppers, collected in database $\mathcal{D}$, bought all the items from the set X. For the rest of the dissertation, we assume that the user-defined support threshold $\sigma \leq b$.

Definition 1.1.5. *The* Frequent Set Mining problem *is, given a set of items $\mathcal{I} = \{i_1, i_2, \ldots, i_n\}$, a database $\mathcal{D}$ and a threshold σ, to determine which subsets of $\mathcal{I}$ are frequent, so occur in at least σ of the transactions in $\mathcal{D}$. These sets are therefore called* frequent sets.

Example 1.1.3. *If we set $\sigma = 2$ in Example 1.1.1, we have $support(bread) = 2 \geq \sigma$, so bread is frequent. We also have $support(chips) = 1 < \sigma$, so chips is not frequent.*

You can see that the supports in Example 1.1.3 are expressed *absolute*, in terms of the amount of transactions in the database. Other approaches in data mining work *relative*, in percent. In this case, the support is expressed as a percentage of the total amount of transactions in the database. Unless mentioned otherwise, the rest of this dissertation will use absolute counts.

Definition 1.1.6. *Given a set $\mathcal{I} = \{i_1, i_2, \ldots, i_n\}$, a database $\mathcal{D}$ of size b and a threshold σ, the* search space *for frequent sets in $\mathcal{D}$ w.r.t. σ is exponential in the number of items, n. More concrete, the size of the search space is of order $\mathcal{O}(2^n)$, which corresponds with $\mathcal{O}(b\ 2^n)$ work.*

The amount of work related to the frequent set mining problem therefore is dependent on both the size of the database, b, and the amount of items in the database, n.

Definition 1.1.7. *Given a set of items $\mathcal{I} = \{i_1, i_2, \ldots, i_n\}$ and a database $\mathcal{D}$, an* association rule *is an expression $X \Rightarrow Y$, with $X, Y \subseteq \mathcal{I}$ and $X \cap Y = \emptyset$. X is called the* body *or* antecedent *and Y is called the* head *or* consequent *of the rule.*

An association rule is used to denote that when all the items from set X are in a transaction, also the items from Y tend to be in the transaction; stated otherwise: when X occurs, so will probably Y. The degree of uncertainty can be expressed using the following definitions.

Definition 1.1.8. *Given a set of items $\mathcal{I} = \{i_1, i_2, \ldots, i_n\}$ and a database $\mathcal{D}$, the* support of an association rule $X \Rightarrow Y$ *is defined as*

$$support(X \Rightarrow Y) := support(X \cup Y) \ .$$

We are only interested in rules with support above some minimal threshold. If the support is not large enough, it means that the rule is not worth consideration, or that it is less preferred.

Definition 1.1.9. *Given a set of items $\mathcal{I} = \{i_1, i_2, \ldots, i_n\}$, a database $\mathcal{D}$, and threshold σ, an association rule $X \Rightarrow Y$ is called* frequent *if its support exceeds the given minimal support threshold σ:*

$$support(X \Rightarrow Y) \geq \sigma \ .$$

Definition 1.1.10. *Given a set of items $\mathcal{I} = \{i_1, i_2, \ldots, i_n\}$, and a database $\mathcal{D}$, the strength of an association rule $X \Rightarrow Y$ is called the* confidence*, and is defined as the support of the rule divided by the support of the antecedent of the rule:*

$$confidence(X \Rightarrow Y) := \frac{support(X \cup Y)}{support(X)} \ .$$

The confidence of an association rule is in fact the conditional probability of having Y in a transaction, given that X is contained in that transaction: $P(Y|X)$. Remark that there is a fundamental difference between support and confidence of an association rule. While confidence is a measure of the rule's strength, support corresponds to statistical significance.

Definition 1.1.11. *Given a set of items $\mathcal{I} = \{i_1, i_2, \ldots, i_n\}$, a database $\mathcal{D}$, and threshold c, the association rule $X \Rightarrow Y$ is called* confident *if its confidence exceeds the given minimal confidence threshold c:*

$$confidence(X \Rightarrow Y) \geq c \ .$$

The user-defined minimal confidence threshold c is used to exclude rules that are not strong enough to be interesting.

Example 1.1.4. *In our example, the confidence of rule "$bread \Rightarrow butter$" is*

$$\frac{support(\{bread, butter\})}{support(bread)} = \frac{1}{2} = 50\% \ .$$

Definition 1.1.12. *Given a set of items $\mathcal{I} = \{i_1, i_2, \ldots, i_n\}$, a database $\mathcal{D}$, support threshold σ and confidence threshold c, the* Association Rule Mining problem *is to generate all association rules $X \Rightarrow Y$ with $support(X \Rightarrow Y) \geq \sigma$ and $confidence(X \Rightarrow Y) \geq c$.*

To achieve this goal, the problem of finding association rules is divided into *two subproblems*:

1. Find all the *frequent sets* $X \subseteq \mathcal{I}$.
2. Find all the *association rules*: For each frequent set X, test for all non-empty subsets $Y \subset X$ if $X \setminus Y \Rightarrow Y$ holds with sufficient confidence.

The second part of the problem can be solved in a straightforward manner, once the frequent sets and their corresponding frequencies are known. The first part, the discovery of all frequent sets is the hardest part of the problem. This division into two subproblems defines the link between the frequent set mining problem and the association rule mining problem. For the rest of the dissertation, we will focus on frequent set mining.

The frequent set mining problem [11, 12] is a basic problem at the core of *many* data mining problems [11, 12, 13, 14, 22, 38, 41], from which association rule mining is just one topic. To illustrate, frequent sets are not only used for finding association rules [11, 12, 38], but also for sequential patterns [13], classification [14], etc. Since the introduction of the frequent set mining problem, several different algorithms for solving it have been proposed [11, 12, 15, 18, 30, 35, 38, 41, 51, 57, 58, 59, 60]. Goethals presented a survey of the best known frequent set mining algorithms [29].

The Apriori Algorithm (see Section 2.2) is probably the best-known algorithm that solves the frequent set mining problem [11, 18]. It is based on the following important principle.

Definition 1.1.13. *The* Monotonicity Principle *or* Downward Closure Property *or* Apriori Property *[48]. Given a set of items $\mathcal{I} = \{i_1, i_2, \ldots, i_n\}$, a transaction database $\mathcal{D}$, and two sets $I_1 \subseteq \mathcal{I}$ and $I_2 \subseteq \mathcal{I}$ such that $I_1 \subseteq I_2$. In database $\mathcal{D}$, the support of I_2 will be at most as high as the support of I_1:*

$$support(I_2) \leq support(I_1) \ .$$

Corollary 1.1.1. *According to the Monotonicity Principle, the confidence of association rule $X \Rightarrow Y$, defined in Definition 1.1.10, is always less than or equal to* 1.

A lot of other algorithms were proposed after the introduction of Apriori [49]. A selection of these algorithms will be considered in Chapter 2.

The task of discovering all frequent sets of items is quite challenging, because the search space is exponential w.r.t. the number of items in the database. If there are $|\mathcal{I}|$ items, there are $2^{|\mathcal{I}|}$ sets that possibly can be frequent. Most approaches to find frequent sets are iterative in nature, requiring multiple database scans, which of course is very expensive. The size of support threshold σ can help limiting the output to a reasonable subspace. An efficient frequent set mining algorithm needs to find which sets are frequent according to σ, without testing all the $2^{|\mathcal{I}|}$ possibilities. Therefore, most algorithms do some *pretesting* to eliminate sets that are not interesting. These pretests usually depend on the fact that support is monotone.

In this dissertation, we analytically study an important class of algorithms for finding frequent sets in databases: *the candidate-based frequent set mining algorithms.* These algorithms follow a generate-and-test strategy, in the sense that they repeatedly create ever more specific candidate patterns, and verify (test) them against the database, until it is no longer possible to generate new patterns. Very important w.r.t. this iterative generation-and-test process is the fact that, based on the results from earlier iterations, often the number of candidates can be reduced significantly. Depending on the candidacy definition and the strategy followed by an algorithm, more or less patterns can be pruned. A successful strategy will generate many candidates that will turn out to be frequent, and few candidates that will turn out infrequent.

Another important class of algorithms that we study in this dissertation are *maximal frequent set mining algorithms*. This means that in this case, we are mining for *large* frequent sets. The idea is that we want to discover these sets in an early phase of the mining process and exploit this knowledge to save time. This research is inspired by the Max-Miner algorithm and specifically the question why this algorithm works so well [15].

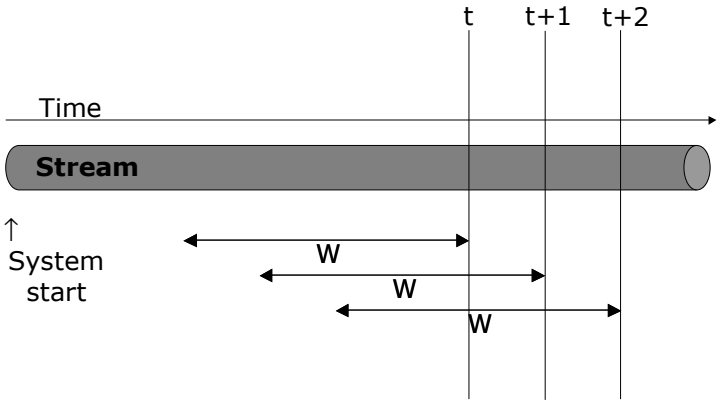
1.2 Stream Mining

In the previous section, we focused mainly on a *static* view in mining frequent sets: *offline* detection of frequent sets in databases. Another important subfield in data mining considers *online* mining. It is called *stream mining* and the goal here is to manage and query *dynamic* streaming data, hoping to find interesting patterns. Compared to mining from a static transaction dataset (a database), the streaming case has more information to track and greater complexity to manage. Therefore, it is a very challenging new topic with lots of applications, from computational biology to network monitoring. Power consumption measuring, sensor network data analysis, dynamic tracing of stock fluctuation, etc: they all call for a study of streaming data. Even research about shopping behavior can be done with streams. For these applications, we can distinguish between streams of singleton items, like those found in IP-network monitoring, and streams of variable sized sets of items, as illustrated by a sequence of market basket transactions at a retail store.

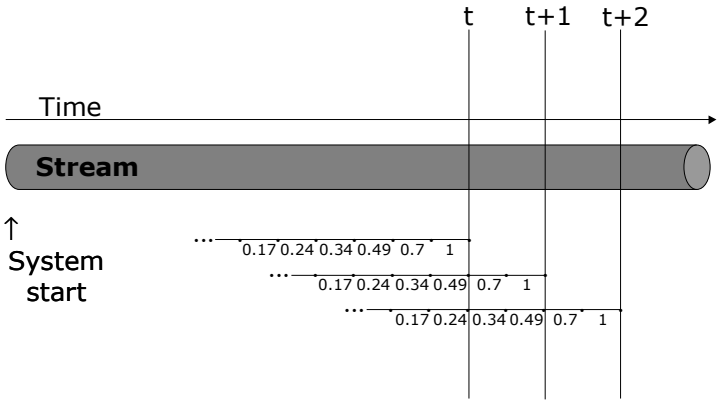
The setting for mining streams is completely different compared to that of mining databases, and this in many ways. A *data stream* is formed by transactions arriving in series. Each transaction corresponds to a set of items and is associated with a time stamp. These sets arrive continuously at a high rate and the amount of information passing by is typically huge, so there is no possibility to store it. Because of this, streaming data can only be read once. Hence, if an item has passed, it can not be revisited. Streaming data is also unbounded, and therefore time-dependent. All these features show that it is not possible to simply reuse the vast knowledge from databases and apply it to streams. For example, infrequent sets can become frequent later in the stream, and hence, cannot be ignored. Therefore, the solution is to remember only crucial information from the stream and to dynamically adjust the storage structure for this info to reflect the evolution of frequencies over time. All of these points illustrate that it is quite challenging to extend and adapt the traditional mining techniques from static databases to the dynamic environment of streams.

Because of the wide applicability of research in this area, we are of course not the first to be interested in mining streams. Most previous work on mining frequently occurring items or sets from data streams either focuses on (1) the sliding window model, (2) the time-fading model, or (3) the landmark model. The tilted-time window combines basic ideas from all three. Each of these models requires a *fixed* window length or decay factor given by the user.

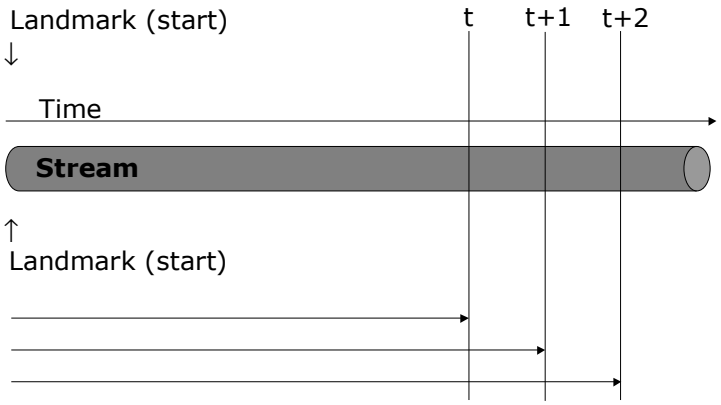
In the *sliding window* model, the importance of the data contained in the sliding window of fixed length is emphasized and the goal is to discover *recent usage trends* [26, 31, 40, 42, 45, 46, 55]. Thus, under this model, the focus is on the most recent items in a window of predefined fixed length. The size of the window is fixed at the start of the analysis and the content of the window changes when the stream (and time) continues. Two different points



(a) The *sliding window* model



(b) The *time-fading* model, with decay factor $d = 0.7$



(c) The *landmark* model

Figure 1.1: Illustration of the different models

of view are considered: the *count-based* view, in which the number of transactions in the window is fixed despite the generating speed of the transactions, and the *time-based* windows, in which the number of time units in the window is fixed, causing the number of transactions to be different. When a transaction leaves the sliding window, it is eliminated and ceases to contribute to the frequency measure. Figure 1.1(a) illustrates the sliding window model when we mine from the latest W transactions.

In the *time-fading* model, the entire stream is taken into account to compute the frequency of a set, but the sensitivity of time is emphasized in the sense that the recent data are weighted higher than the earlier data. This means that more *recent* transactions *contribute more* to the frequency, compared to older ones. This is achieved by introducing a decay mechanism, based on a decay factor d [23, 44]. Usually, this results in an exponentially decreasing weight for transactions that occurred longer ago; e.g. a transaction that appeared n time stamps in the past is weighted d^n. In general, the closer to 1 the decay factor d is, the more the history is taken into account. An illustration of the time-fading model can be found in Figure 1.1(b). Each time stamp n corresponds to a certain transaction that is associated with a weight d^n and the weights are decreasing when time goes by. For this figure, d is chosen to be 0.7.

In the *landmark* model, a *particular time period* is fixed, from the landmark designating the start of the system up till the current time [40, 42, 56]. Usually, a selection of landmarks are fixed. The analysis of the stream is performed only for the part of the stream between the landmark and the current time instance, as illustrated in Figure 1.1(c). A major disadvantage of this method is that the size of the window varies; it starts with size zero and grows until the next occurrence of the landmark, at which point it is reset to zero.

The *tilted-time* window was introduced as a *combination* of these methods [20, 21]. The technique is inspired by the time-fading philosophy, reflected in the alteration of the time scales of the windows as time goes by. Recent data is mined at a fine granularity while long-term data is mined at a coarse granularity. For a set, frequencies are computed for the most recent windows of length w, $2w$, $4w$, ..., etc. By doing so, it is possible to efficiently answer queries over the history of the stream, while taking into account that more recent parts of the stream are more important.

All these methods illustrate that most of the previous work on mining frequently occurring items or sets over data streams use models that work with a window length or a decay factor which is *fixed* in advance. In many applications, however, it is *not* possible to fix a window length or a decay factor that is most optimal for every item at every time point in an evolving stream. Therefore, in this dissertation, we introduce a *new frequency measure*, based on a flexible window length. This new frequency measure for finding frequent sets in a stream is quite different from the ones typically found in the literature on stream mining. To the best of our knowledge, we are the first to present an approach in which no fixed window length or decay factor is used. Even more, in almost all related work, data stream processing sacrifices the correctness of its analysis by allowing some error. Our work differs, since our approach produces *exact* information, as compared to the many approximations in other works.

We also present an *algorithm* based on this new measure that maintains a small *summary* of relevant information of the history of the stream, which allows to produce the current

frequency of a specific set immediately at any time. Critical for the usefulness of the technique are the memory requirements of the summary that needs to be maintained in memory. A *theoretical upper bound* on the size of the summary based on a specific type of streams, called Farey streams, is computed and *experiments* show that, even though in worst case the summary depends on the length of the stream, for realistic data distributions its size is extremely small. Obviously, this property of having a small summary is highly desirable as it allows for an efficient and effective computation of our new measure.

1.3 Outline of the Dissertation

The first chapters of the dissertation focus on **Frequent Set Mining** in static databases. In Chapter 2, an important class of algorithms for finding frequent sets in databases is studied analytically: the *candidate-based frequent set mining algorithms*. In Chapter 3, the reader can find a theoretical, probabilistic analysis of the events candidate, success and failure, with different data distributions, in the offline context of static databases. Chapter 4 continues with studying the behavior of maximal frequent set mining algorithms. A collection of level-by-level jumping frequent set mining algorithms is presented and Max-Miner, a well-known and famous algorithm in frequent set mining, is rediscovered. At the same time, it is explained why Max-Miner performs well.

With Chapter 5, we change the focus of the dissertation to **Stream Mining**. Because it is not straightforward to fix a window length or a decay factor which is most appropriate for every set at every point in time for an evolving stream, we propose to consider for each set the window in which it has the highest frequency. We give all the details about this new definition of frequency, yielding a completely new view on streams, together with some important properties. An incremental algorithm that allows to produce the current frequency of a set immediately at any time is proposed and interesting results about stream mining follow. A worst-case analysis is done by computing a theoretical upper bound on the size of the summary based on a specific type of streams, called Farey streams. Experiments show that, even though in worst case the summary depends on the length of the stream, for realistic data distributions its size is extremely small.

After the bibliography, the dissertation concludes with a summary of the covered topics in Dutch.

2

Frequent Set Mining Algorithms

From the early beginning of data mining as a research field, with the frequent set mining problem at its core, researchers are mostly interested in the development of good *algorithms* to quickly find frequent sets. This process of algorithm development and optimization started in the early 90's and dozens of new algorithms and adaptations or improvements of already existing algorithms were presented. Within the past decade, hundreds of research papers have been published to solve the problem more efficiently. The presented algorithms all follow the same strategy: *a candidate-generating and testing process.*

The search space for the frequent set mining problem is exponential w.r.t. the number of items in the database. If there are $|\mathcal{I}|$ items, there are $2^{|\mathcal{I}|}$ sets of items that possibly can be frequent. The support threshold already limits the output to a hopefully reasonable subspace. To test the minimum amount of sets that are not frequent, a more directed search through the search space is developed. Therefore, the concept of *candidates* is introduced. A set is called a candidate if an algorithm evaluates whether the set is frequent or not. Most frequent set mining algorithms first try to *generate candidate* sets that then are *tested* against the database to physically check, i.e. count, if they are indeed frequent, as they would expect. The process of generating and testing candidate sets continues until no new frequent sets can be found.

In this chapter, a *selection* of candidate-based frequent set mining algorithms is presented, that will be used for the rest of this dissertation. It is a collection of some well-known and influential algorithms that made significant contributions to the frequent set mining problem. All algorithms considered are *complete*, in the sense that they are all able to find all frequent sets. For each algorithm, we focus on the *candidate*-test: the requirement that is used by the algorithm for a set to be a candidate. In the following chapters of this dissertation, we present an in-dept probabilistic analysis of this small selection of basic algorithms.

2.1 The AIS Algorithm

The Agrawal-Imielinski-Swami (AIS) Algorithm is the first algorithm introduced to solve the frequent set mining problem [11]. To start, the level-wise, bottom-up AIS Algorithm assumes a fixed ordering on the items. When the algorithm is doing its level k processing, a set I of size k is only considered when the algorithm comes across a transaction in the database that contains all the items of I. Moreover, it considers set I if and only if it is a 1-extension of a frequent set of size $(k-1)$.

Definition 2.1.1. *Given a database $\mathcal{D}$, based on items from $\mathcal{I}$, on which a certain order ρ is imposed. A* 1-extension *of a set X of size $(k-1)$ is a set X' of size k, that is created by extending X by exactly one item, that follows in the ordering. More concrete, $X' = X \cup \{i\}$, with $i \in \mathcal{I}$, such that i is greater than every item from X, according to ρ.*

When candidate I is found to be frequent, it is used to generate candidates for the next pass.

To summarize, a set I of size k is treated as a candidate if (1) it is a 1-extension of a frequent set of size $(k-1)$, and (2) I has at least support 1 in the database: $support(I) \geq 1$. Thus, if set I is a candidate for AIS, there exists *a* set of size one less that is frequent.

Example 2.1.1. *The set* abcd *is considered as a candidate by the AIS Algorithm, if it occurs in the database and* abc *is frequent.*

2.2 The Apriori Algorithm

Apriori is probably the best-known algorithm that solves the frequent set mining problem [11, 18]. It was developed from AIS by strengthening the candidacy test: a set I consisting of k items is a candidate if and only if *all* of its $(k-1)$-size subsets are frequent.

The Apriori Apriori is a breadth-first, level-wise, bottom-up algorithm, that for each level consists of *two* parts: a *pruning* part and a *generation* part. It considers sets in order of their size, determines for each size candidate sets and checks whether they are frequent or not. The algorithm starts with the singleton sets, finds out which of these are frequent and uses these 1-element successes to generate candidates for level 2. These candidate 2-element sets are tested (by counting in the database) to determine the frequent ones that will be used as candidates for the next level (level 3). Apriori continues by increasing level by level, each time (at level k, starting with $k = 1$) *checking* the frequencies of the candidates C_k of size k and *generating* candidates C_{k+1} of size $(k+1)$. The algorithm stops when no more frequent sets can be found. An outline is given in Algorithm 1. The nature of Apriori is that it, on each level, knows the results from the previous level (and therefore, from all previous levels). With this knowledge from pretesting and monotonicity (Definition 1.1.13), candidates are generated that are checked in the next step and sets that are not interesting are eliminated *a priori*.

When processing level k, the support of all candidate k-sets from C_k, created on the previous level $(k-1)$, is counted by scanning the database, one transaction at a time. The supports of all candidate sets that are included in that transaction are incremented (Algorithm 1 - lines 5–11). All sets that turn out to be frequent are grouped into $\mathcal{F}_k$ (Algorithm 1 - lines 12–13).

Algorithm 1 The Apriori Algorithm

Require: Items $\mathcal{I} = \{i_1, i_2, \ldots, i_n\}$, database $\mathcal{D}$, user-defined support threshold σ
Ensure: $\mathcal{F}(\mathcal{D}, \sigma)$:= Frequent sets from $\mathcal{D}$ w.r.t. that particular threshold σ

1: $C_1 := \{\{i\} |\ i \in \mathcal{I}\}$ *//Start with singleton sets*
2: $k := 1$
3: **while** $C_k \neq \{\}$ **do**
4: *//Pruning Part*
5: **for all** transactions $(tid, I) \in \mathcal{D}$ **do**
6: **for all** candidate sets $X \in C_k$ **do**
7: **if** $X \subseteq I$ **then**
8: $support(X) + +$
9: **end if**
10: **end for**
11: **end for** *//Computes the supports of all candidate sets*
12: $\mathcal{F}_k := \{X | support(X) \geq \sigma\}$ *//Extracts all frequent sets*
13: *//Generating Part*
14: **for all** $X, Y \in \mathcal{F}_k, X[j] = Y[j]$ for $1 \leq j \leq k-1$, and $X[k] < Y[k]$ **do**
15: $I = X \cup \{Y[k]\}$ *//Join step*
16: **if** $\forall J \subset I, |J| = k : J \in \mathcal{F}_k$ **then**
17: $C_{k+1} := C_{k+1} \cup I$ *//Prune step*
18: **end if**
19: **end for**
20: $k + +$
21: **end while**

Apriori now continues with generating candidates for the next level $(k+1)$. This candidate generation process consists of two steps: a *join* step and a *prune* step. When continuing processing level k, in the *join* step, all frequent sets of size k are joined according to a special technique. The union of frequent k-sets X and Y is only done if they have the same $(k-1)$-prefix (Algorithm 1 - lines 15–16). With this step, Apriori in fact generates a collection of potential candidates. In the *prune* step, a potential candidate is only saved as a new final candidate of size $(k+1)$ in C_{k+1} if *all* of its k-subsets are frequent (Algorithm 1 - lines 17–19).

Example 2.2.1. *We assume that we are in the processing of Apriori on level* 3*.*

- Pruning *part: assume that two sets of size* 3*,* abc *and* abd*, were found frequent*
- Generation *part:*
  - Join *step (to construct potential candidates of size* 4*): The common prefix of size* 2 *of sets* abc *and* abd *is* ab*, and the set* abcd *is generated as the result from the join from both sets*
  - Prune *step (to find final candidates of size* 4*, that are saved in* C_4*): This set* abcd *is a candidate if all subsets of size* 3*,* abc*,* abd*,* acd *and* bcd *are frequent.*

The time needed by the Apriori Algorithm to find all frequent sets in a certain dataset is determined by the number of candidates: the sets that are effectively in the output of the

algorithm, so are frequent — these sets will be called *successes* in the rest of the dissertation — and the number of sets that are counted by the algorithm but that are not in the output of the algorithm, so turn out to be infrequent - these sets will be called *failures*.

2.3 The Fast Completion Apriori Algorithm

The Fast Completion Apriori Algorithm (FCA) starts out like the Apriori Algorithm, proceeding in a level-wise manner, generating and testing candidates as it goes, but as soon as it determines that the number of candidates for all remaining levels is not too large, it generates candidates for the remaining levels based on the currently available information [12]. So, in fact, it runs the Apriori Algorithm but stops at a crucial level k. It is very important that this level k is determined carefully. If k is chosen too small, it is possible that the algorithm will do too much work. If k is too large, the final jump is probably not helpful anymore to save work and time. From that level k on, the algorithm uses the frequent sets of size k to generate candidates for all remaining higher levels. Therefore, a set I of size $k+h$ is a candidate if *all* of its subsets of size k are frequent. This generalization of the traditional Apriori algorithm reduces the number of passes, but may result in some sets being unnecessarily counted.

2.4 The Eclat and FP-growth Algorithm

Instead of a breadth-first search approach, like Apriori, there are also algorithms that use a depth-first search approach. The first algorithm proposed to generate all frequent sets in a depth-first, bottom-up manner is Eclat, the Equivalence Class Transformation Algorithm [57, 60]. Later on, several others were introduced, from which the algorithm FP-growth, the Frequent Pattern-growth Algorithm, is probably the best known [35, 36].

Although the authors of the FP-growth algorithm claim that their algorithm does not generate any candidate sets, Goethals shows that the algorithm actually generates a lot of candidate sets since it essentially uses the same candidate generation technique as used in Apriori, but without the prune step [29]. It is also shown that Eclat and FP-growth, as far as it concerns the candidacy test, are essentially the same algorithm, even though they have important differences in the details of how they organize the data for processing. When we assume that all items that occur in the database are frequent[1], the only real difference between Eclat and FP-growth is the way they count the support of each candidate set. Therefore, in the rest of this dissertation, we will consider both algorithms as a single technique.

In Algorithm 2, the pseudo code for Eclat is given. When we have database $\mathcal{D}$ and user-defined support threshold σ, we denote the set of all frequent sets of size k with the same $(k-1)$ prefix $I \subseteq \mathcal{I}$ by $\mathcal{F}[I](\mathcal{D},\sigma)$. It is clear that $\mathcal{F}[\{\}](\mathcal{D},\sigma) = \mathcal{F}(\mathcal{D},\sigma)$, the collection of all frequent sets from $\mathcal{D}$ w.r.t. threshold σ. Eclat recursively generates for every item $i \in \mathcal{I}$ the set $\mathcal{F}[\{i\}](\mathcal{D},\sigma)$. The notation $cover(I)$ is used for the set of all transactions in which I is contained. In this pseudo code, a candidate set is represented by $I \cup \{i,j\}$.

[1] In practice, all frequent items can be computed during an initial scan over the database, after which all infrequent items will be ignored.

Algorithm 2 The Eclat Algorithm

Require: $\mathcal{I} = \{i_1, i_2, \ldots, i_n\}$, database $\mathcal{D}$, user-defined support threshold σ, prefix $I \subseteq \mathcal{I}$
Ensure: $\mathcal{F}[I](\mathcal{D}, \sigma)$:= Frequent sets with prefix I from $\mathcal{D}$ w.r.t. that particular threshold σ

```
1: F[I] := {}
2: for all i ∈ I occurring in D do
3:    F[I] := F[I] ∪ {I ∪ {i}}
4:    //Create D^i, a subset database of D, based on i
5:    D^i := {}
6:    for all j ∈ I occurring in D, such that j > i do
7:       C := cover({i}) ∩ cover({j})
8:       if |C| ≥ σ then
9:          D^i := D^i ∪ {(j, C)}
10:      end if
11:   end for
12:   //Depth-first recursion
13:   compute F[I ∪ {i}](D^i, σ)
14:   F[I] := F[I] ∪ F[I ∪ {i}]
15: end for
```

Both Eclat and FP-growth build a tree of frequent sets based on an *ordering* of the items. There are *three* different orderings that are commonly used: *least-frequent-first (LFF)*, *most-frequent-first (MFF)*, and *arbitrary or lexicographical* order. Both LFF and MFF can be applied as a *static* ordering, based on global frequencies, or as a *dynamic* ordering, based on frequencies for the current subtree [15, 17, 35, 43].

For Eclat and FP-growth, a set I is a candidate only if *two* particular subsets of I are frequent. These two test sets are the one obtained by omitting the last item from I and the one obtained by omitting the next-to-last item from I, when assuming a certain order on the items. Both test sets must be frequent before I can be a candidate.

Example 2.4.1. *Set* abcde *is a candidate, according to the Eclat and FP-growth condition, if* abcd *and* abce *both are frequent.*

3

A Probability Analysis for Frequent Set Mining Algorithms

Since the introduction of the frequent set mining problem, several algorithms for solving it were proposed and experimentally analyzed [11, 12, 15, 18, 29, 30, 35, 38, 41, 51, 57, 58, 59, 60]. The suggested references are just a small selection from the enormous amount of publications related to this topic. Compared to the extensive amount of literature on experimental analysis, relatively few papers have been devoted to *theoretical* analysis [28, 33, 50, 53]. The research[1] presented in this chapter can be classified into this last category. The inspiration can be found in important related work [50]. The results presented in this work are substantially different because of the more *general* setting [9].

We present a detailed *probabilistic study* of the performance of the frequent set mining algorithms presented in Chapter 2, for different data distributions. The properties of the frequent set mining problem and the performance of the associated algorithms and their interaction with different data distributions are studied analytically.

The generation of *candidates*, which is an important step in frequent set mining algorithms, is explored in detail. Other important notions in our probabilistic analysis are *success* and *failure*. The probabilities of these events are studied for a simple data model that describes the probability distribution of the database. For this data model, we assume that all the transactions are independent and each combination of items has its own probability, so any correlation between items is possible. The results of the analysis are used to compare the behavior of the algorithms for a variety of data distributions.

[1] It is joint work with Dirk Van Gucht and Paul W. Purdom from Indiana University in Bloomington, USA.

3.1 Introduction

The generation of *candidates*, i.e. candidate sets, is an important step in frequent set mining algorithms. The number of candidates has a major effect on the computational work that has to be done by the algorithm. The running time of a frequent set mining algorithm depends on the number of candidates and on how quickly the support of these candidates can be determined by counting. The number of candidates is a fundamental property of the particular algorithm because every algorithm has a different candidate definition (Chapter 2). The time performance and the memory usage for counting is dependent on implementation details [29, 30, 41]. In addition, different compilers and different machine architectures can show different behavior for the same algorithm [41].

Intuitively, a set I is called a *candidate* when its frequency status cannot be deduced by the algorithm based on previous knowledge from other sets, but has to be evaluated, i.e. counted, explicitly in the database. In practice, I is a candidate if certain associated *test sets* are already determined to be frequent. Chapter 2 shows that for all the algorithms, except for the Fast Completion Apriori (FCA) Algorithm, the test sets are obtained by omitting a single item from I; for FCA, the test sets are all those subsets of I whose size is equal to the level where the regular Apriori Algorithm was last used. Once all candidates are found, their exact frequency status is determined by explicitly counting the occurrences in the database. If a candidate set I is frequent, it is called a *success*; otherwise, it is a *failure*.

Example 3.1.1. *Set* abcd *is a candidate for the Apriori Algorithm when all its subsets of size one less,* abc*,* abd*,* acd *and* bcd *are frequent. For Eclat, the same set* abcd *is a candidate when there are two test sets, for example* abc *and* abd*, that are frequent.*

Because the candidacy definition differs for each algorithm, the *candidacy probability*, the probability that a set is a candidate, depends on the particular algorithm used. However, all correct algorithms have the same probability that a set is frequent, the *success probability*; it is a property of the data, not of the algorithm. Every algorithm that is required to output each success cannot avoid doing work associated with the success sets. The probability that a set is a candidate but not frequent is called the *failure probability*; it depends on both the problem instance and the algorithm and is particularly important because it is related to work that a better algorithm might hope to avoid. Indeed, the infrequent sets are not the output the algorithms are looking for, but the algorithms had to examine these sets to conclude that they are not wanted. This means that the algorithms lose some speed by examining these sets since they do not contribute to the output of the algorithm but they do consume running time.

This chapter investigates in detail the probabilities that a set is a candidate, success or failure. The average behavior of Apriori is considered in detail; for AIS, Eclat, FP-growth, which can be considered as a candidate-based algorithm, its title not withstanding [35], and FCA, the analysis is similar so only the main principles are sketched. The probabilistic analysis is done based on a general but simple probabilistic data model. The results of the analysis are used to compare the behavior of the algorithms for a variety of data distributions.

Our research shows that for each algorithm, the candidacy probability of a set is usually determined almost entirely by the frequency probability of a *particular* test set, but which test set this is, depends on the algorithm. It thus turns out that the various algorithms differ in

which test set is the most important one and has the main effect on the number of candidates that are generated. For both versions of the Apriori algorithm, this dominant test set turns out to be the one with the smallest probability; we call this the *best* test set. For the AIS algorithm, it is the test set with the highest probability; we call this the *worst* test set. These names sound a bit awkward, but a motivation for the choice is given in Subsection 3.5.4. For Eclat-like algorithms, including FP-growth, it is a set that is at least as good as the test set with the second-highest probability and there is a tendency for it to be the one with the second-highest probability. We prove that the candidacy probability of a set is near 1 when the probability of this dominant test set is significantly above σ/b, with σ the user-defined support threshold from the frequent set mining problem and b the number of transactions in the database, and it is near 0 when the probability is significantly below σ/b. Similar results are found for the success and failure probabilities.

The theoretical analysis presented in this chapter can help us understand the performance differences between various candidate-based frequent set mining algorithms. We will show that the algorithms considered have similar performance on uniform data, whereas they can have hugely differences in performance on other types of data. The theoretical approach is also useful for practical purposes; for example for designers and implementors of frequent set mining algorithms or for algorithms based on frequent set mining, such as for example Association Rule Mining Algorithms.

There exist a lot of good experimental data that illustrate the results and claims that will follow in the rest of this chapter [11, 12, 18, 57, 60, 35, 36]. These external experiments reflect our analytical results; this chapter gives a theoretical justification for these facts.

3.2 The Probability Model

As a first idea for a probability model for the distribution of the data in the database, we could suggest to reuse the very simple probability model in which all the transactions and all the items in the database are independent [50]. Additionally, all the items have the same probability, p. However, this model does *not* reflect a real-life database distribution.

To overcome this problem, we present a more *realistic* model: the *random data probability model*. In this data model, the set of transactions are identically distributed, independent and random. Each combination of items has its own probability. This model is very general. It allows us to accommodate the existence of dependencies: any correlation between items is possible. We assume that the distribution of the transactions does not change over time.

It is clear that our model is an important basic model, that is used to gain insights and have a good understanding of the average case performance of the frequent set mining algorithms. Even more, our model can handle the more general situation in which each transaction has its own distribution, as long as the transactions are random and independent. The idea here is to replace the original transactions by a single one whose behavior is a weighted linear combination of the behaviors of the original transactions. The weight factor is the probability that a transaction is of the particular type. By doing this, we obtain a corresponding average policy for a single random transaction: the basis for our probability model.

	cars	dolls	clothes	cars and clothes	dolls and clothes
boys	0.4	0.0	0.4	0.2	0.0
girls	0.0	0.4	0.4	0.0	0.2

Figure 3.1: Shopping behavior for boys and girls

	cars	dolls	clothes	cars and clothes	dolls and clothes
parents	0.2	0.2	0.4	0.1	0.1

Figure 3.2: Shopping behavior for parents

We assume that there are b transactions and n possible items in the database. Traditionally, the user-defined support threshold will be denoted by σ. To make the probability model more concrete, consider Examples 3.2.1 and 3.2.2. These examples illustrate the behavior and the interpretation of our probability model in the context of *market basket analysis*. The transactions in the database reflect the shopping behavior of shoppers that buy items in a shop. Both examples consider shoppers following the shopping behavior, thus the probability distribution of the data in the database, given in Table 3.1. Combinations of items not bought together are not shown in the table.

Example 3.2.1. *Suppose $b/2$ boys and $b/2$ girls come to the store. This example can* not *be handled exactly with our model because there are two different shopping behaviors and the different shoppers are not arriving in random order. Indeed, any excess of one sex in the initial shoppers is always balanced by an excess of the other sex in the remainder of shoppers.*

Example 3.2.2. *Suppose boys and girls come to the store randomly with* 0.5 *chance each, until a total of b shoppers come. This example can* not *be handled* directly *by our model because there is more than one shopping behavior, but this example can be handled* indirectly*, as the next Example 3.2.3 will show. For large b and for most results of interest, the results of the two Examples 3.2.1 and 3.2.2 are almost the same.*

Example 3.2.3. *Consider shoppers following the policy given in Table 3.2. This is just one policy for all shoppers, and fits perfectly in our data model. Since this policy is the average of the two policies from Table 3.1, the shoppers have the same shopping behavior as that which results from randomly selecting shoppers from Table 3.1. Therefore, our analysis can also indirectly handle Example 3.2.2.*

Associated with each set I, we have the probability $P(I)$ that expresses the probability that a particular transaction contains all the items of I, regardless of eventually other items.

Example 3.2.4. *If Table 3.2 gives the shopping policy, we have:*

$$P(\emptyset) = 0.2 + 0.2 + 0.4 + 0.1 + 0.1 = 1$$
$$P(\{\text{cars}\}) = 0.2 + 0.1 = 0.3$$
$$P(\{\text{dolls}\}) = 0.2 + 0.1 = 0.3$$
$$P(\{\text{clothes}\}) = 0.4 + 0.1 + 0.1 = 0.6$$
$$P(\{\text{cars, clothes}\}) = 0.1$$
$$P(\{\text{dolls, clothes}\}) = 0.1$$
$$P(\{\text{cars, clothes}\}) = 0$$
$$P(\{\text{cars, dolls, clothes}\}) = 0$$

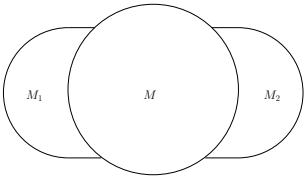


Figure 3.3: Graphical illustration of the region M associated with set I, and the ears M_1 and M_2 associated with test sets I_1 and I_2, respectively

Property 3.2.1. *$P(I)$ is the probability that a transaction contains all the items from set I, regardless of what other items are contained in the transaction. If $p(J)$ is used to denote the probability that a shopper buys* exactly *set J, then*

$$P(I) = \sum_{J \supseteq I} p(J) \ .$$

3.3 Notations

The work presented in this chapter focuses on what a candidate-based frequent set mining algorithm does while processing a *single* set, I. To determine the *total* work done by the algorithm, the results of this chapter have to be summed over *all* sets, but we are not going to focus on that. One can gain a lot of insight into the behavior of the algorithms by only focusing on what is important with respect to a single set.

In general, when a candidate-based algorithm is considering making set I a candidate, it will have already counted some of the subsets of I. These sets are going to be used as *test sets*. Various algorithms for the frequent set mining problem differ in which sets they use for test sets. According to the particular candidacy definition, we associate a selection of test sets for each candidate set I. For each of the considered sets, we can construct the set of transactions that contain all the items of the set. Figure 3.3 shows the situation in the transaction space when I is associated with *two* test sets, I_1 and I_2 ($I_1 \subset I$ and $I_2 \subset I$). The region M contains the transactions that contain all the items of I, along with perhaps additional items. When the database consists of b random transactions, the expected number of transactions in M will be the product $P(I)b$. The left ear M_1 contains all the transactions that contain all the items of I_1, except for those transactions that are in M, so contain all the items of I. Similarly, M_2 contains all the transactions with the items of I_2 except those that have all the items of I, and therefore are in M. The expected number of transactions in M_i ($i = 1, 2$) will therefore be the product $[P(I_i) - P(I)]b$. This is not equal to $p(I_i)b$, because it is possible that a transaction in M_i contains more items than exactly the items from I_i; it is just not allowed to contain all items from I.

Example 3.3.1. *We follow the shopping policy from Table 3.2. In the context of market basket analysis, we consider the following two examples in transaction space, i.e. basket space, when a particular set I is associated with two test sets, I_1 and I_2. In both cases, we give an illustration of the regions M, M_1 and M_2.*

- *When* $I = \{cars, clothes\}$, $I_1 = \{cars\}$, *and* $I_2 = \{clothes\}$, *then* M *can contain two types of baskets: baskets containing cars and clothes and baskets containing cars, dolls, and clothes.* M_1 *has two basket types: baskets containing cars and baskets containing cars and dolls.* M_2 *has two basket types: baskets containing clothes and baskets containing dolls and clothes. The expected number of baskets in* M *is* $P(I)b = 0.1b$*; the expected number of baskets in* M_1 *is* $[P(I_1) - P(I)]b = [0.3 - 0.1]b = 0.2b$*; the expected number of baskets in* M_2 *is* $[P(I_2) - P(I)]b = [0.6 - 0.1]b = 0.5b$.

- *When* $I = \{cars, dolls, clothes\}$, $I_1 = \{cars, clothes\}$, *and* $I_2 = \{cars, dolls\}$, *then* M *has one basket type: baskets consisting of cars, dolls, and clothes.* M_1 *has one basket type: baskets consisting of cars and clothes, and* M_2 *has also one basket type: baskets consisting of cars and dolls. The expected number of baskets in* M *is* $P(I)b = 0$*. The expected number of baskets in* M_1 *is* $[P(I_1) - P(I)]b = [0.1 - 0]b = 0.1b$*; the expected number of baskets in* M_2 *is* $[P(I_2) - P(I)]b = [0 - 0]b = 0$.

For all the algorithms (except for FCA) that are considered in this dissertation, the various ears are *disjoint*. In general, the number of ears associated with I will equal the number of test sets used for I. For set I to be a candidate, all the involved test sets have to be frequent. Thus, set I is a candidate if the number of transactions that are in $M \cup M_i$ is at least σ, for every test set I_i. Clearly, for every test set I_i, the number of transactions containing I is bounded by the number of transactions containing I_i. This is an upper bound that is too high by the number of transactions in M_i. Therefore, the fewer the number of transactions in M_i, the better the test set I_i.

We use the following **notations** in the analysis:

b, the total number of transactions in the database.

σ, the support threshold. A frequent set mining algorithm determines which sets occur in at least σ transactions.

I, the set being considered. We assume a natural order on the items and we name the elements of I with integers from 1 to $|I|$.

I_i, the i-th test set associated with I. For example, for the Apriori Algorithm, we have $I_i = I - \{i\}$ $(1 \leq i \leq |I|)$. Test sets are algorithm dependent.

$P(I)$, the probability that all the items in I occur in the same transaction, regardless of whether or not other items are purchased; $P(I)$ is the probability that a transaction is in M. This definition implies $P(\emptyset) = 1$, as illustrated in Example 3.2.4. Similarly, $P(I_i)$ is the probability that a transaction contains all the items in I_i; $P(I_i)$ is the probability that a transactions is in the set of transactions associated with the i-th test set I_i: $M \cup M_i$.

$Q_i(I) := P(I_i) - P(I)$, the probability that a transaction contains all the items in the i-th test set without having all the items from I. This is the probability that a transaction contributes to the i-th test set even though it does not contain all the items of I. In other words, the transaction is in the ear associated with the i-th test set, M_i, and $Q_i(I)$ is the probability of this ear.

$Q(I)$, the $Q_i(I)$ that dominates in the determination of the magnitude of the failure probability of set I.

ℓ, the number of i values for which $Q_i(I) = Q(I)$. It is the multiplicity of the controlling value.

For set I, we compute:

$S(I)$, the *success* probability, i.e., the probability that at least σ transactions contain all the items of I, so that I is frequent. Any correct algorithm for the frequent set mining problem has the same success probability. It is a property of the data, not of the algorithm.

$F(I)$, the *failure* probability, i.e., the probability that set I passes the candidacy test but fails the frequency test. Some algorithms are faster than others because they have a smaller failure probability. The failure probability depends on both the problem instance and the algorithm.

The probability that set I is a candidate is

$$C(I) = S(I) + F(I) \ .$$

3.4 General Probabilities

In this section, we focus on the general probabilities for being a candidate, a success or a failure. To confirm the results in Subsections 3.4.2 and 3.4.3, we first need to introduce some statistical background. We present two well-known distributions in Subsection 3.4.1 together with some interesting properties, that play an important role in the computation of the success, candidate and failure probabilities which will be studied in the rest of this chapter. The inspiration for the derivations can be found in important related work [50]. An overview of the main results can be found in Subsection 3.4.4.

3.4.1 Statistical Background

Definition 3.4.1. *The* Bernoulli *Distribution is the discrete distribution that is used to describe an experiment with* two *possible outcomes: a* success *outcome with probability p and a* failure *outcome with probability $q = 1 - p$. We use the notation $X \sim B(1, p)$ to express that stochast X follows a Bernoulli distribution with probability p.*

Example 3.4.1. *The classical example of the Bernoulli distribution is* tossing a coin *where both sides have equal probability* $1/2$ *to be on top.*

Given stochast $X = X_1 + \ldots + X_n$, where X_j $(1 \leq j \leq n)$ are independent and identical distributed (abbreviated i.i.d.), $X_j \sim B(1, p)$ following a Bernoulli Distribution. The result of the Bernoulli sum, X, is now defined to follow a *Binomial* Distribution, denoted $X \sim B(n, p)$.

Definition 3.4.2. *The* Binomial *distribution is a discrete distribution that represents the amount of successes in n independent Bernoulli experiments, so n independent repetitions of a random trial, with two possible outcomes,* success *with probability p and* failure *with probability $1 - p$. There are two parameters: n, the number of repetitions, and p, the success probability in the repeated Bernoulli experiment. The value for p has to be the same for all the n successive experiments.*

Property 3.4.1. *The probability that there are j successes in the n successive Bernoulli experiments is*

$$P(X = j) \;=\; \binom{n}{j}\; p^j\; (1-p)^{n-j}\;\;.$$

Property 3.4.2. *The probability of having at least k ($0 \leq k \leq n$) successes in the n successive Bernoulli experiments is*

$$P(X \geq k) \;=\; \sum_{j \geq k} \binom{n}{j}\; p^j\; (1-p)^{n-j} \;=\; \sum_{j=k}^{n} \binom{n}{j}\; p^j\; (1-p)^{n-j}\;\;.$$

Based on the Binomial Distribution, we can formulate the Binomial Theorem.

Definition 3.4.3. *Let n be a positive integer. The* Binomial Theorem *gives a formula for the expansion of the nth power of the sum of real numbers A and B:*

$$\sum_{i=1}^{n} \binom{n}{i}\; A^i\; B^{n-i} \;=\; (A+B)^n\;\;. \tag{3.1}$$

The Binomial Theorem is also valid for complex numbers A and B, and more generally for any elements A and B of a semi ring, as long as $AB = BA$. For this dissertation, we only need the real version of the theorem as presented in (3.1).

3.4.2 The Success Probability

Based on the statistical background, we can compute the success probability $S(I)$, the probability that at least σ of b transactions contain all of the items of set I, by using the Binomial Distribution.

According to Section 3.3, the probability that a transaction contains all the items of set I is expressed by $P(I)$. This means that if we pick a particular transaction, it has probability $P(I)$ to contain set I and probability $1 - P(I)$ to not contain I. We can now do this for all b transactions in the database, resulting in a repetition of b independent Bernoulli trails: a Binomial trial. We have independence between the trials because of the assumption of the model that all transactions are independent (Section 3.2). The success probability is the probability that at least σ out of b transactions contain items of set I. Using the Binomial Distribution, $S(I)$ can be expressed by the following expression.

$$S(I) \;=\; \sum_{j \geq \sigma} \binom{b}{j}\; [P(I)]^j\; [1-P(I)]^{b-j}\;\;. \tag{3.2}$$

Upper Bound for the Success Probability

Based on Chernoff techniques [24] and the Binomial Theorem (3.1), the following *upper* bound for $S(I)$ can be computed.

Property 3.4.3.

$$S(I) \;\leq\; x^{-\sigma} \sum_{j=1}^{b} \binom{b}{j}\; [xP(I)]^j\; [1-P(I)]^{b-j} \;=\; x^{-\sigma}\; [1+(x-1)\; P(I)]^b \tag{3.3}$$

for any $x \geq 1$.

Proof. The result (3.2) is an incomplete binomial sum that does not have a closed form, but an upper bound can be obtained using Chernoff techniques. The lower part of the sum is missing and the sum can be completed by using the following function

$$U_1(j) = \begin{cases} 0 & j < \sigma \\ 1 & j \geq \sigma \end{cases} .$$

When multiplying with U_1, $S(I)$ can be rewritten to

$$S(I) \;=\; S(I) \cdot U_1 \;=\; \sum_{j=1}^{b} \binom{b}{j} [P(I)]^j \; [1 - P(I)]^{b-j} \;.$$

As an approximation for U_1, we introduce U_2, defined by $U_2(j) = x^{-\sigma+j}$ with $x \geq 1$. It is known that $U_1(j) \leq U_2(j),\ \forall j$. Therefore, we replace U_1 by U_2 and the result is an upper bound for $S(I)$.

$$\begin{aligned} S(I) \;&\leq\; \sum_{j=1}^{b} \binom{b}{j} x^{-\sigma+j} \; [P(I)]^j \; [1 - P(I)]^{b-j} \\ &=\; x^{-\sigma} \sum_{j=1}^{b} \binom{b}{j} \; [xP(I)]^j \; [1 - P(I)]^{b-j} \\ &=\; x^{-\sigma} \; [1 + (x-1)P(I)]^b \end{aligned}$$

The last equality is based on the Binomial Theorem. □

Property 3.4.4. *The optimum for right-hand side $x^{-\sigma} \; [1 + (x-1) \; P(I)]^b$ in (3.3) is either on the boundary ($x = 1$) or when the derivative with respect to x is equal to zero. The second case is achieved when*

$$x_* \;=\; \frac{\sigma \; [1 - P(I)]}{(b - \sigma) \; P(I)} \;. \tag{3.4}$$

Proof. The optimum x_* is found by taking the derivative of the Chernoff bound (3.3) with respect to x and setting this expression equal to zero.

$$\begin{aligned} & \frac{\mathrm{d}}{\mathrm{d}x} \Big[x^{-\sigma} \; [1 + (x-1) \; P(I)]^b \Big] \;=\; 0 \\ \Leftrightarrow \;& (-\sigma) \; x^{-\sigma-1} \; [1 + (x-1) \; P(I)]^b + x^{-\sigma} \; b \; [1 + (x-1) \; P(I)]^{b-1} \; P(I) \;=\; 0 \\ \Leftrightarrow \;& (-\sigma) \; [1 + (x-1) \; P(I)] + x \; \sigma \; P(I) \;=\; 0 \\ \Leftrightarrow \;& x \; (b - \sigma) \; P(I) \;=\; \sigma \; [1 - P(I)] \\ \Leftrightarrow \;& x \;=\; \frac{\sigma \; [1 - P(I)]}{(b - \sigma) \; P(I)} \end{aligned}$$

□

Property 3.4.5. *When $x_* < 1$, the optimum is 1 (boundary case), and otherwise, when $x_* \geq 1$, x_* itself is the optimum for the equation. This is the case when*

$$\frac{\sigma}{b} \;\geq\; P(I) \;. \tag{3.5}$$

Proof. Indeed, the requirement $x_* \geq 1$ gives

$$\begin{aligned} & \frac{\sigma\ [1-P(I)]}{(b-\sigma)\ P(I)} \ \geq\ 1 \\ \Leftrightarrow\ & \sigma\ [1-P(I)] \ \geq\ (b-\sigma)\ P(I) \\ \Leftrightarrow\ & \sigma \ \geq\ bP(I)\ . \end{aligned}$$

□

We are now ready to calculate an *upper bound* to show that $S(I)$ is near zero in the region described by the above inequality (3.5). Therefore, we will introduce the notion α_1, that is defined by

$$\begin{aligned} \sigma &= b\ [P(I)+\alpha_1]\ , \\ \Leftrightarrow\ \alpha_1 &= \frac{\sigma}{b} - P(I)\ . \end{aligned} \tag{3.6}$$

In fact, α_1 describes the difference, reflecting the distance, between σ/b and $P(I)$. When we are in the region described by (3.5), thus when σ/b is strictly bigger than $P(I)$, resulting in α_1 being a positive value, we will show that $S(I)$ goes to zero rapidly with increasing α_1.

Theorem 3.4.1. *When $P(I) \leq \sigma/b$, the following upper bound for $S(I)$ can be found*

$$S(I) \leq e^{-b\alpha_1^2/\{2P(I)[1-P(I)]\}+\mathcal{O}(b\alpha_1^3[1-P(I)]^{-2})}\ , \tag{3.7}$$

with $\alpha_1 = \sigma/b - P(I)$. In this case $S(I)$ goes to 0 rapidly with increasing α_1.

Proof. We start from the Chernoff bound for $S(I)$, expressed by (3.3) and plug in the above found value for the optimum (3.4).

$$\begin{aligned} S(I) &\leq x^{-\sigma}\ [1+(x-1)\ P(I)]^b \\ &\leq \left[\frac{\sigma\ [1-P(I)]}{(b-\sigma)\ P(I)}\right]^{-\sigma} \left[1+\left(\frac{\sigma\ [1-P(I)]}{(b-\sigma)\ P(I)}-1\right)\ P(I)\right]^b \\ &= \sigma^{-\sigma}\ [1-P(I)]^{-\sigma}\ (b-\sigma)^{\sigma}\ P(I)^{\sigma}\ \left[1+\frac{\sigma\ [1-P(I)]-(b-\sigma)\ P(I)}{(b-\sigma)\ P(I)}\ P(I)\right]^b \\ &= \sigma^{-\sigma}\ [1-P(I)]^{-\sigma}\ (b-\sigma)^{\sigma}\ P(I)^{\sigma}\ \left[\frac{b\ (1-P(I))}{b-\sigma}\right]^b \\ &= \left(\frac{P(I)}{\sigma}\right)^{\sigma}\ \left(\frac{1-P(I)}{b-\sigma}\right)^{b-\sigma}\ b^b \end{aligned}$$

We now replace σ by its expression in terms of α_1, expressed by (3.6).

$$\begin{aligned} S(I) &\leq \left(\frac{P(I)}{b\ [P(I)+\alpha_1]}\right)^{b\ [P(I)+\alpha_1]}\ \left(\frac{1-P(I)}{b\ [1-P(I)-\alpha_1]}\right)^{b\ [1-P(I)-\alpha_1]}\ b^b \\ &= \left(\frac{1}{b\ [1+\frac{\alpha_1}{P(I)}]}\right)^{b\ [P(I)+\alpha_1]}\ \left(\frac{1}{b\ [1-\frac{\alpha_1}{1-P(I)}]}\right)^{b\ [1-P(I)-\alpha_1]}\ b^b \\ &= \left(\frac{1}{[1+\frac{\alpha_1}{P(I)}]}\right)^{b\ [P(I)+\alpha_1]}\ \left(\frac{1}{[1-\frac{\alpha_1}{1-P(I)}]}\right)^{b\ [1-P(I)-\alpha_1]} \end{aligned}$$

To further simplify this Chernoff bound for $S(I)$, we write it as

$$S(I) \;\leq\; e^{X(I)} \;, \tag{3.8}$$

with

$$\begin{aligned} X(I) &= \ln\left[\left(\frac{1}{[1+\frac{\alpha_1}{P(I)}]}\right)^{b\;[P(I)+\alpha_1]} \left(\frac{1}{[1-\frac{\alpha_1}{1-P(I)}]}\right)^{b\;[1-P(I)-\alpha_1]}\right] \\ &= -b\;[P(I)+\alpha_1]\;\ln\left(1+\frac{\alpha_1}{P(I)}\right) - b\;[1-P(I)-\alpha_1]\;\ln\left(1-\frac{\alpha_1}{1-P(I)}\right) \;. \end{aligned}$$

We thus have

$$\frac{X(I)}{b} \;=\; -\;[P(I)+\alpha_1]\;\ln\left(1+\frac{\alpha_1}{P(I)}\right) - [1-P(I)-\alpha_1]\;\ln\left(1-\frac{\alpha_1}{1-P(I)}\right) \;. \tag{3.9}$$

The formula $\ln(1+x) = x - \frac{1}{2}x^2 + \mathcal{O}(x^3)$ is now used to simplify (3.9).

$$\begin{aligned} \frac{X(I)}{b} &= -[P(I)+\alpha_1]\left[\frac{\alpha_1}{P(I)} - \frac{1}{2}\frac{\alpha_1^2}{P(I)^2} + \mathcal{O}\left(\frac{\alpha_1^3}{P(I)^3}\right)\right] \\ &\qquad -([1-P(I)]-\alpha_1)\left[\frac{-\alpha_1}{1-P(I)} - \frac{1}{2}\frac{(-\alpha_1)^2}{[1-P(I)]^2} + \mathcal{O}\left(\frac{(-\alpha_1)^3}{[1-P(I)]^3}\right)\right] \\ &= -[P(I)+\alpha_1]\left[\frac{\alpha_1}{P(I)} - \frac{1}{2}\frac{\alpha_1^2}{P(I)^2} + \mathcal{O}\left(\frac{\alpha_1^3}{P(I)^3}\right)\right] \\ &\qquad +([1-P(I)]-\alpha_1)\left[\frac{\alpha_1}{1-P(I)} + \frac{1}{2}\frac{\alpha_1^2}{[-P(I)]^2} + \mathcal{O}\left(\frac{\alpha_1^3}{[1-P(I)]^3}\right)\right] \\ &= -\alpha_1 + \frac{1}{2}\frac{\alpha_1^2}{P(I)} - \mathcal{O}\left(\frac{\alpha_1^3}{P(I)^2}\right) - \frac{\alpha_1^2}{P(I)} + \frac{1}{2}\frac{\alpha_1^3}{P(I)^2} - \mathcal{O}\left(\frac{\alpha_1^4}{P(I)^3}\right) + \alpha_1 + \frac{1}{2}\frac{\alpha_1^2}{1-P(I)} \\ &\qquad +\mathcal{O}\left(\frac{\alpha_1^3}{[1-P(I)]^2}\right) - \frac{\alpha_1^2}{1-P(I)} - \frac{1}{2}\frac{\alpha_1^3}{[1-P(I)]^2} - \mathcal{O}\left(\frac{\alpha_1^4}{[1-P(I)]^3}\right) \end{aligned}$$

The negative big $\mathcal{O}$ terms can be dropped because we are looking for an upper limit and what is left is

$$\frac{X(I)}{b} \;\leq\; -\frac{1}{2}\frac{\alpha_1^2}{P(I)} - \frac{1}{2}\frac{\alpha_1^2}{(1-P(I))} + \mathcal{O}\left(\frac{\alpha_1^3}{[1-P(I)]^2}\right) \;.$$

This expression can be plugged into (3.8), leading to the following upper bound for $S(I)$:

$$\begin{aligned} S(I) &\leq e^{b\left[-\frac{1}{2}\frac{\alpha_1^2}{P(I)} - \frac{1}{2}\frac{\alpha_1^2}{(1-P(I))} + \mathcal{O}\left(\frac{\alpha_1^3}{[1-P(I)]^2}\right)\right]} \\ \Leftrightarrow\; S(I) &\leq e^{\frac{-b\alpha_1^2}{2P(I)[1-P(I)]} + \mathcal{O}\left(\frac{b\alpha_1^3}{[1-P(I)]^2}\right)} \end{aligned} \tag{3.10}$$

In (3.10), the big $\mathcal{O}$ is w.r.t. α_1. We now expect that, when $P(I) \leq \sigma/b$, S(I) goes to zero rapidly with increasing α_1. This claim needs justification, because it is obvious that the big $\mathcal{O}$ plays an important role and possibly can disturb the asymptotical behavior of $S(I)$. Therefore, we refine the approximation starting from (3.9).

We thus have

$$\frac{X(I)}{b} \;=\; -\,[P(I)+\alpha_1]\;\ln\left(1+\frac{\alpha_1}{P(I)}\right)-[1-P(I)-\alpha_1]\;\ln\left(1-\frac{\alpha_1}{1-P(I)}\right)\;. \qquad (3.11)$$

The formula $\ln(1+x)=x-\frac{1}{2}x^2+\frac{1}{3(1+c)^3}x^3$, with c in between 0 and x and $|x|<1$ is now used to simplify (3.11). This formula is found by directly applying Taylor's theorem, with the Lagrange form of the remainder term. In particular, we use that

$$x-\frac{1}{2}x^2+\frac{1}{3(1+c)^3}x^3 \;\le\; x-\frac{1}{2}x^2+\frac{1}{3}x^3\;,$$

for $|x|<1$. This gives us the following upper bound for $X(I)/b$.

$$\begin{aligned}
\frac{X(I)}{b} &\le -[P(I)+\alpha_1]\left[\frac{\alpha_1}{P(I)}-\frac{1}{2}\frac{\alpha_1^2}{P(I)^2}+\frac{1}{3}\frac{\alpha_1^3}{P(I)^3}\right]\\
&\qquad -([1-P(I)]-\alpha_1)\left[\frac{-\alpha_1}{1-P(I)}-\frac{1}{2}\frac{(-\alpha_1)^2}{[1-P(I)]^2}-\frac{1}{3}\frac{\alpha_1^3}{(1-P(I))^3}\right]\\
&= -[P(I)+\alpha_1]\left[\frac{\alpha_1}{P(I)}-\frac{1}{2}\frac{\alpha_1^2}{P(I)^2}+\frac{1}{3}\frac{\alpha_1^3}{(1-P(I))^3}\right]\\
&\qquad +([1-P(I)]-\alpha_1)\left[\frac{\alpha_1}{1-P(I)}+\frac{1}{2}\frac{\alpha_1^2}{[1-P(I)]^2}+\frac{1}{3}\frac{\alpha_1^3}{(1-P(I))^3}\right]\\
&= -\alpha_1+\frac{1}{2}\frac{\alpha_1^2}{P(I)}-\frac{1}{3}\frac{\alpha_1^3}{P(I)^2}-\frac{\alpha_1^2}{P(I)}+\frac{1}{2}\frac{\alpha_1^3}{P(I)^2}-\frac{1}{3}\frac{\alpha_1^4}{P(I)^3}+\alpha_1+\frac{1}{2}\frac{\alpha_1^2}{1-P(I)}\\
&\qquad +\frac{1}{3}\frac{\alpha_1^3}{(1-P(I))^2}-\frac{\alpha_1^2}{1-P(I)}-\frac{1}{2}\frac{\alpha_1^3}{[1-P(I)]^2}-\frac{1}{3}\frac{\alpha_1^4}{(1-P(I))^3}\\
&= -\frac{1}{2}\frac{\alpha_1^2}{P(I)}-\frac{1}{2}\frac{\alpha_1^2}{(1-P(I))}+\frac{1}{6}\frac{\alpha_1^3}{P(I)^2}-\frac{1}{6}\frac{\alpha_1^3}{(1-P(I))^2}-\frac{1}{3}\frac{\alpha_1^4}{P(I)^3}-\frac{1}{3}\frac{\alpha_1^4}{(1-P(I))^3}
\end{aligned}$$

Therefore,

$$\begin{aligned}
\frac{X(I)}{b} &\le \frac{-\alpha_1^2}{2P(I)(1-P(I))}+\frac{\alpha_1^3}{6}\left(\frac{1}{P(I)^2}-\frac{1}{(1-P(I))^2}\right)-\frac{\alpha_1^4}{3}\left(\frac{1}{P(I)^3}+\frac{1}{(1-P(I))^3}\right)\\
&\le \frac{-\alpha_1^2}{2P(I)(1-P(I))}+\frac{\alpha_1^3}{6}\left(\frac{1-2P(I)}{P(I)^2(1-P(I))^2}\right) \qquad (3.12)
\end{aligned}$$

This expression can be plugged into (3.8), leading to the following upper bound for $S(I)$:

$$S(I) \;\le\; e^{\frac{-b\alpha_1^2}{2P(I)[1-P(I)]}+\frac{b\alpha_1^3}{6}\left(\frac{1-2P(I)}{P(I)^2(1-P(I))^2}\right)} \qquad (3.13)$$

We now want to express that the cubic term in (3.13) is not nearly as big as the quadratic term. We now can use the following upper bound for $S(I)$,

$$\begin{aligned}
S(I) &\le e^{\frac{-b\alpha_1^2}{2P(I)[1-P(I)]}+\frac{b\alpha_1^2}{4P(I)[1-P(I)]}}\\
&\le e^{\frac{-b\alpha_1^2}{4P(I)[1-P(I)]}}\;, \qquad (3.14)
\end{aligned}$$

if

$$\alpha_1 \;\leq\; \frac{3P(I)(1-P(I))}{2(1-2P(I))} \quad \text{and} \quad P(I) \;<\; 0.5 \;, \tag{3.15}$$

or

$$\alpha_1 \;\geq\; \frac{3P(I)(1-P(I))}{2(1-2P(I))} \quad \text{and} \quad P(I) \;>\; 0.5 \;, \tag{3.16}$$

The last expression (3.16) is not interesting, because α_1 is always positive. Eqn. (3.15) is found from

$$\begin{aligned}
& e^{\frac{-b\alpha_1^2}{2P(I)[1-P(I)]} + \frac{b\alpha_1^3}{6}\left(\frac{1-2P(I)}{P(I)^2(1-P(I))^2}\right)} \;\leq\; e^{\frac{-b\alpha_1^2}{4P(I)[1-P(I)]}} \\
\Leftrightarrow\;& \frac{-b\alpha_1^2}{2P(I)[1-P(I)]} + \frac{b\alpha_1^3}{6}\left(\frac{1-2P(I)}{P(I)^2(1-P(I))^2}\right) \;\leq\; \frac{-b\alpha_1^2}{4P(I)[1-P(I)]} \\
\Leftrightarrow\;& \frac{b\alpha_1^3}{6}\left(\frac{1-2P(I)}{P(I)^2(1-P(I))^2}\right) \;\leq\; \frac{b\alpha_1^2}{4P(I)[1-P(I)]} \\
\Leftrightarrow\;& \alpha_1\,(1-2P(I)) \;\leq\; \frac{3P(I)(1-P(I))}{2} \;.
\end{aligned}$$

Upper bound (3.14) shows that $S(I)$ is near zero, as long as α_1 is a limited positive value that is not near zero. Even if α_1 is small, for a fixed $P(I)$ and b large, we have a very small bound on $S(I)$ illustrating that $S(I)$ goes to zero rapidly.

If α_1 is larger than or equal to the above computed threshold (3.15) and continues to increase, it is still the case that $S(I)$ decreases, even though the current analysis no longer applies. This is clear from (3.6) which states that σ increases at the same rate as $b\alpha_1$, combined with (3.2). Obviously, $S(I)$ decreases as σ increases, because the bigger σ is, the less positive terms we add to $S(I)$. □

Lower Bound for the Success Probability

We are also interested in the behavior of $S(I)$ on the other side of the boundary. We can find an expression for a *lower* bound on $S(I)$ by using the definition

$$S(I) = 1 - \sum_{j<\sigma} \binom{b}{j} [P(I)]^j\,[1-P(I)]^{b-j} = 1 - \sum_{j\leq\sigma-1} \binom{b}{j} [P(I)]^j\,[1-P(I)]^{b-j}, \tag{3.17}$$

and compute an upper bound for the incomplete binomial sum

$$Y(I) = \sum_{j\leq\sigma-1} \binom{b}{j} [P(I)]^j\,[1-P(I)]^{b-j} \tag{3.18}$$

by using Chernoff techniques, analogously as we did before.

Property 3.4.6.

$$Y(I) \leq x^{-\sigma+1}\,[1+(x-1)P(I)]^b \;, \tag{3.19}$$

for any $x \leq 1$.

Proof. The Chernoff function that is used in this case differs from the previous one because we now have to complete the sum in the above direction. Therefore, we propose

$$L_1(j) = \begin{cases} 1 & j \leq \sigma - 1 \\ 0 & j > \sigma - 1 \end{cases} .$$

As an approximation for L_1, we introduce L_2, defined by $L_2(j) = x^{-\sigma+1+j}$ with $x \leq 1$. Multiplying $Y(I)$ with this function L_2 results in an upper bound for $Y(I)$.

$$\begin{aligned} Y(I) &\leq \sum_{j=1}^{b} \binom{b}{j} x^{-\sigma+1+j} \, [P(I)]^j \, [1 - P(I)]^{b-j} \\ &= x^{-\sigma+1} \sum_{j=1}^{b} \binom{b}{j} [xP(I)]^j \, [1 - P(I)]^{b-j} \\ &= x^{-\sigma+1} \, [1 + (x-1)P(I)]^b \end{aligned}$$

The last equality is based on the Binomial Theorem. □

Combined with the 1 minus in expression (3.17), this upper bound on $Y(I)$ yields a lower bound on $S(I)$.

Property 3.4.7. *The optimum for right-hand side $x^{-\sigma+1} \, [1+(x-1)P(I)]^b$ in (3.19) is either on the boundary ($x = 1$) or when the derivative with respect to x is equal to zero. The second case is achieved when*

$$x_* = \frac{(\sigma - 1) \, [1 - P(I)]}{(b - \sigma + 1) \, P(I)} .$$

Proof. We find the optimum x_* by taking the derivative w.r.t. x and setting it to zero.

$$\begin{aligned} & \frac{\mathrm{d}}{\mathrm{d}x} \left[x^{-\sigma+1} \, [1 + (x-1)P(I)]^b \right] = 0 \\ \Leftrightarrow \quad & (-\sigma + 1) \, x^{-\sigma} \, [1 + (x-1)P(I)]^b + x^{-\sigma+1} \, b \, [1 + (x-1)P(I)]^{b-1} \, P(I) = 0 \\ \Leftrightarrow \quad & (-\sigma + 1) \, [1 + (x-1)P(I)] + x \, b \, P(I) = 0 \\ \Leftrightarrow \quad & (b - \sigma + 1) \, P(I) \, x = (\sigma - 1) \, [1 - P(I)] \\ \Leftrightarrow \quad & x = \frac{(\sigma - 1) \, [1 - P(I)]}{(b - \sigma + 1) \, P(I)} \end{aligned}$$

□

Property 3.4.8. *The requirement $x_* \leq 1$ leads to*

$$\frac{\sigma - 1}{b} \leq P(I) . \tag{3.20}$$

Proof.

$$\begin{aligned} & \frac{(\sigma - 1) \, [1 - P(I)]}{(b - \sigma + 1) \, P(I)} \leq 1 \\ \Leftrightarrow \quad & (\sigma - 1) \, [1 - P(I)] \leq [b - (\sigma - 1)] \, P(I) \\ \Leftrightarrow \quad & \sigma - 1 \leq bP(I) \\ \Leftrightarrow \quad & \frac{\sigma - 1}{b} \leq P(I) \end{aligned}$$

□

This gives us a new region of interest: $\sigma/b \leq P(I) + 1/b$. Now, α_2 is defined to express the difference between $P(I)$ and $(\sigma - 1)/b$:

$$\begin{aligned} \sigma &= b\,[P(I) - \alpha_2] + 1 \\ \Leftrightarrow\ \alpha_2 &= P(I) - (\sigma - 1)/b\ . \end{aligned} \tag{3.21}$$

We are now ready to show that $S(I)$ goes to 1 rapidly in the region described by (3.20), thus when α_2 is a positive value.

Theorem 3.4.2. *When $P(I) \geq (\sigma - 1)/b$, the following lower bound for $S(I)$ can be found*

$$S(I) \geq 1 - e^{-b\alpha_2^2/\{2P(I)[1-P(I)]\} + \mathcal{O}(b\alpha_2^3 P(I)^{-2})}\ , \tag{3.22}$$

with $\alpha_2 = P(I) - (\sigma - 1)/b$. In this case $S(I)$ goes to 1 rapidly with increasing α_2.

Proof. We compute the Chernoff bound for $Y(I)$ by reusing the results from the Proof of Theorem 3.4.1.

$$Y(I) \leq \left(\frac{P(I)}{\sigma - 1}\right)^{\sigma - 1} \left(\frac{1 - P(I)}{b - \sigma + 1}\right)^{b - \sigma + 1} b^b\ . \tag{3.23}$$

In (3.23), we replace σ by its expression in terms of α_2, expressed by 3.21, leading to

$$\begin{aligned} Y(I) &\leq \left(\frac{P(I)}{b\,[P(I) - \alpha_2]}\right)^{b\,[P(I)-\alpha_2]} \left(\frac{1 - P(I)}{b\,[1 - P(I) + \alpha_2]}\right)^{b\,[1-P(I)+\alpha_2]} b^b \\ &= \left(\frac{1}{b\,[1 - \frac{\alpha_2}{P(I)}]}\right)^{b\,[P(I)-\alpha_2]} \left(\frac{1}{b\,[1 + \frac{\alpha_2}{1-P(I)}]}\right)^{b\,[1-P(I)+\alpha_2]} b^b \\ &= \left(\frac{1}{[1 - \frac{\alpha_2}{P(I)}]}\right)^{b\,[P(I)-\alpha_2]} \left(\frac{1}{[1 + \frac{\alpha_2}{1-P(I)}]}\right)^{b\,[1-P(I)+\alpha_2]}\ . \end{aligned}$$

To further simplify this Chernoff bound for $Y(I)$, we write it as

$$Y(I) \leq e^{T(I)}\ , \tag{3.24}$$

with

$$\begin{aligned} T(I) &= \ln\left[\left(\frac{1}{[1 - \frac{\alpha_2}{P(I)}]}\right)^{b\,[P(I)-\alpha_2]} \left(\frac{1}{[1 + \frac{\alpha_2}{1-P(I)}]}\right)^{b\,[1-P(I)+\alpha_2]}\right] \\ &= -b\,[P(I) - \alpha_2]\ \ln\left(1 - \frac{\alpha_2}{P(I)}\right) - b\,[1 - P(I) + \alpha_2]\ \ln\left(1 + \frac{\alpha_2}{1 - P(I)}\right)\ . \end{aligned}$$

We thus have

$$\frac{T(I)}{b} = -[P(I) - \alpha_2] \ln\left(1 - \frac{\alpha_2}{P(I)}\right) - [1 - P(I) + \alpha_2] \ln\left(1 + \frac{\alpha_2}{1 - P(I)}\right)\ . \tag{3.25}$$

The formula $\ln(1 + x) = x - \frac{1}{2}x^2 + \mathcal{O}(x^3)$ is again used to simplify (3.25).

$$\begin{aligned}
\frac{T(I)}{b} &= -[P(I)-\alpha_2]\left[\frac{-\alpha_2}{P(I)} - \frac{1}{2}\frac{(-\alpha_2)^2}{P(I)^2} + \mathcal{O}\left(\frac{(-\alpha_2)^3}{P(I)^3}\right)\right] \\
&\quad - ([1-P(I)]+\alpha_2)\left[\frac{\alpha_2}{1-P(I)} - \frac{1}{2}\frac{\alpha_2^2}{[1-P(I)]^2} + \mathcal{O}\left(\frac{\alpha_2^3}{[1-P(I)]^3}\right)\right] \\
&= [P(I)-\alpha_2]\left[\frac{\alpha_2}{P(I)} + \frac{1}{2}\frac{\alpha_2^2}{P(I)^2} + \mathcal{O}\left(\frac{\alpha_2^3}{P(I)^3}\right)\right] \\
&\quad - ([1-P(I)]+\alpha_2)\left[\frac{\alpha_2}{1-P(I)} - \frac{1}{2}\frac{\alpha_2^2}{[1-P(I)]^2} + \mathcal{O}\left(\frac{\alpha_2^3}{[1-P(I)]^3}\right)\right] \\
&= \alpha_2 + \frac{1}{2}\frac{\alpha_2^2}{P(I)} + \mathcal{O}\left(\frac{\alpha_2^3}{P(I)^2}\right) - \frac{\alpha_2^2}{P(I)} - \frac{1}{2}\frac{\alpha_2^3}{P(I)^2} - \mathcal{O}\left(\frac{\alpha_2^4}{P(I)^3}\right) - \alpha_2 + \frac{1}{2}\frac{\alpha_2^2}{1-P(I)} \\
&\quad -\mathcal{O}\left(\frac{\alpha_2^3}{[1-P(I)]^2}\right) - \frac{\alpha_2^2}{1-P(I)} + \frac{1}{2}\frac{\alpha_2^3}{[1-P(I)]^2} - \mathcal{O}\left(\frac{\alpha_2^4}{[1-P(I)]^3}\right)
\end{aligned}$$

Because we are looking for an upper bound, the negative big $\mathcal{O}$ terms can be dropped and what is left is

$$\frac{T(I)}{b} = -\frac{1}{2}\frac{\alpha_2^2}{P(I)} - \frac{1}{2}\frac{\alpha_2^2}{1-P(I)} + \mathcal{O}\left(\frac{\alpha_2^3}{[P(I)]^2}\right) \ .$$

This expression can be plugged into (3.24), leading to the following upper bound for $Y(I)$:

$$\begin{aligned}
Y(I) &\leq e^{b\left[-\frac{1}{2}\ \frac{\alpha_2^2}{P(I)} - \frac{1}{2}\ \frac{\alpha_2^2}{1-P(I)} + \mathcal{O}\left(\frac{\alpha_2^3}{[P(I)]^2}\right)\right]} \\
\Leftrightarrow\ Y(I) &\leq e^{\frac{-b\alpha_2^2}{2\ P(I)\ [1-P(I)]} + \mathcal{O}\left(\frac{b\alpha_2^3}{P(I)^2}\right)}
\end{aligned} \tag{3.26}$$

In (3.26) is the big $\mathcal{O}$ w.r.t. α_2. We thus have that

$$S(I) \ \geq\ 1 - e^{\frac{-b\alpha_2^2}{2\ P(I)\ [1-P(I)]} + \mathcal{O}\left(\frac{b\alpha_2^3}{P(I)^2}\right)} \ .$$

When $P(I) \geq (\sigma-1)/b$, so when α_2 is large, but not too large, this result indicates that $S(I)$ goes to 1 rapidly. Again, this claim needs justification. We refine the approximation starting from (3.25).

We thus have

$$\frac{T(I)}{b} = -[P(I)-\alpha_2]\ln\left(1-\frac{\alpha_2}{P(I)}\right) - [1-P(I)+\alpha_2]\ln\left(1+\frac{\alpha_2}{1-P(I)}\right) \ . \tag{3.27}$$

The formula

$$\ln(1+x) \ =\ x - \frac{1}{2}x^2 + \frac{1}{3(1+c)^3}x^3 \ \leq\ x - \frac{1}{2}x^2 + \frac{1}{3}x^3 \ ,$$

with c in between 0 and x and $|x| < 1$ is again used to simplify (3.27). This yields the following upper bound for $T(I)/b$.

$$\begin{aligned}
\frac{T(I)}{b} &\leq -[P(I)-\alpha_2]\left[\frac{-\alpha_2}{P(I)}-\frac{1}{2}\frac{\alpha_2^2}{P(I)^2}-\frac{1}{3}\frac{\alpha_2^3}{P(I)^3}\right] \\
&\quad -([1-P(I)]+\alpha_2)\left[\frac{\alpha_2}{1-P(I)}-\frac{1}{2}\frac{\alpha_2^2}{[1-P(I)]^2}+\frac{1}{3}\frac{\alpha_2^3}{(1-P(I))^3}\right] \\
&= [P(I)-\alpha_2]\left[\frac{\alpha_2}{P(I)}+\frac{1}{2}\frac{\alpha_2^2}{P(I)^2}+\frac{1}{3}\frac{\alpha_2^3}{P(I)^3}\right] \\
&\quad +([1-P(I)]+\alpha_2)\left[\frac{-\alpha_2}{1-P(I)}+\frac{1}{2}\frac{\alpha_2^2}{[1-P(I)]^2}-\frac{1}{3}\frac{\alpha_2^3}{(1-P(I))^3}\right] \\
&= \alpha_2+\frac{1}{2}\frac{\alpha_2^2}{P(I)}+\frac{1}{3}\frac{\alpha_2^3}{P(I)^2}-\frac{\alpha_2^2}{P(I)}-\frac{1}{2}\frac{\alpha_2^3}{P(I)^2}-\frac{1}{3}\frac{\alpha_2^4}{P(I)^3}-\alpha_2+\frac{1}{2}\frac{\alpha_2^2}{1-P(I)} \\
&\quad -\frac{1}{3}\frac{\alpha_2^3}{(1-P(I))^2}-\frac{\alpha_2^2}{1-P(I)}+\frac{1}{2}\frac{\alpha_2^3}{[1-P(I)]^2}-\frac{1}{3}\frac{\alpha_2^4}{(1-P(I))^3} \\
&= -\frac{1}{2}\frac{\alpha_2^2}{P(I)}-\frac{1}{2}\frac{\alpha_2^2}{1-P(I)}-\frac{1}{6}\frac{\alpha_2^3}{P(I)^2}+\frac{1}{6}\frac{\alpha_2^3}{(1-P(I))^2}-\frac{1}{3}\frac{\alpha_2^4}{P(I)^3}-\frac{1}{3}\frac{\alpha_2^3}{(1-P(I))^3}
\end{aligned}$$

Therefore,

$$\begin{aligned}
\frac{T(I)}{b} &\leq -\frac{\alpha_2^2}{2P(I)(1-P(I))}-\frac{\alpha_2^3}{6}\left(\frac{1}{P(I)^2}-\frac{1}{(1-P(I))^2}\right)-\frac{\alpha_2^4}{3}\left(\frac{1}{P(I)^3}+\frac{1}{(1-P(I))^3}\right) \\
&\leq \frac{-\alpha_2^2}{2P(I)(1-P(I))}-\frac{\alpha_2^3}{6}\left(\frac{1-2P(I)}{P(I)^2(1-P(I))^2}\right) .
\end{aligned}$$

This expression can be plugged into (3.24), leading to the following upper bound for $Y(I)$:

$$Y(I) \leq e^{\frac{-b\alpha_2^2}{2P(I)(1-P(I))}-\frac{b\alpha_2^3}{6}\left(\frac{1-2P(I)}{P(I)^2(1-P(I))^2}\right)} , \tag{3.28}$$

and the following lower bound for $S(I)$:

$$S(I) \geq 1-e^{\frac{-b\alpha_2^2}{2P(I)(1-P(I))}-\frac{b\alpha_2^3}{6}\left(\frac{1-2P(I)}{P(I)^2(1-P(I))^2}\right)} . \tag{3.29}$$

We now first continue focusing on $Y(I)$. We want to express that the cubic term in (3.28) is not nearly as big as the quadratic term. We now can use the following upper bound for $Y(I)$

$$\begin{aligned}
Y(I) &\leq e^{\frac{-b\alpha_2^2}{2P(I)(1-P(I))}-\frac{b\alpha_2^3}{6}\left(\frac{1-2P(I)}{P(I)^2(1-P(I))^2}\right)} \\
&\leq e^{\frac{-b\alpha_2^2}{4P(I)[1-P(I)]}} ,
\end{aligned} \tag{3.30}$$

if

$$\alpha_2 \leq \frac{3P(I)(1-P(I))}{2(P(I)-1)} \quad \text{and} \quad P(I) > 0.5 , \tag{3.31}$$

or

$$\alpha_2 \geq \frac{3P(I)(1-P(I))}{2(2P(I)-1)} \quad \text{and} \quad P(I) < 0.5 , \tag{3.32}$$

The last expression (3.32) is not interesting, because α_2 is always positive. Eqn. (3.31) is found from

$$\begin{aligned}
& e^{\frac{-b\alpha_2^2}{2P(I)(1-P(I))} - \frac{b\alpha_2^3}{6}\left(\frac{1-2P(I)}{P(I)^2(1-P(I))^2}\right)} \leq e^{\frac{-b\alpha_2^2}{4P(I)[1-P(I)]}} \\
\Leftrightarrow \quad & \frac{-b\alpha_2^2}{2P(I)(1-P(I))} - \frac{b\alpha_2^3}{6}\left(\frac{1-2P(I)}{P(I)^2(1-P(I))^2}\right) \leq \frac{-b\alpha_1^2}{4P(I)[1-P(I)]} \\
\Leftrightarrow \quad & \frac{-b\alpha_2^3}{6}\left(\frac{1-2P(I)}{P(I)^2(1-P(I))^2}\right) \leq \frac{b\alpha_2^2}{4P(I)[1-P(I)]} \\
\Leftrightarrow \quad & \alpha_2\,(2P(I)-1) \leq \frac{3P(I)(1-P(I))}{2} \ .
\end{aligned}$$

Upper bound (3.30) shows that Y is near zero, as long as α_2 is a limited positive value that is not near zero. Even if α_2 is small, for a fixed $P(I)$ and b large, we have a very small bound on $Y(I)$ illustrating that $Y(I)$ goes to zero rapidly. Combining this result with (3.29) gives that $S(I)$ goes to 1 rapidly, with increasing α_2.

If α_2 is larger than the above computed threshold (3.31) and continues to increase, it is still the case that $S(I)$ converges to 1, even though the current analysis no longer applies. This is clear from (3.21) which states that if α_2 increases, σ decreases at the same rate. If we combine this result with with (3.18), we have that $Y(I)$ decreases with decreasing σ, because the smaller σ is, the less positive terms we add to $Y(I)$. From (3.17), it is clear that $S(I) = 1 - Y(I)$, and thus a decreasing $Y(I)$ leads to an $S(I)$ that goes rapidly to 1. □

Remark

We have just computed an upper and lower bound on $S(I)$, showing that this probability is near 0 or 1, depending in which region $P(I)$ falls. These results hold for the situation in which $P(I)$, b, σ and thus α_1 and α_2 are fixed. There are some *additional interesting effects* in the bounds, when we also consider the case in which $P(I)$ and σ are fixed but b varies.

For example, in the case of the upper bound for $S(I)$, we make the following observation. The threshold σ that we use is in fact a count, and behaves proportional to b. We therefore keep the ratio σ/b fixed. In this case, whatever value $\alpha_1 = \sigma/b - P(I)$ takes, it will be greater than or equal to a large constant C times $1/\sqrt{b}$, once b becomes large enough. In this case, the bound in (3.14) is, for fixed $P(I)$ that is not close to 0 or 1, less than a large negative constant. Indeed,

$$\begin{aligned}
S(I) &\leq e^{\frac{-b\alpha_1^2}{4P(I)[1-P(I)]}} \\
&\leq e^{\frac{-bC^2/b}{4P(I)[1-P(I)]}} \\
&\leq e^{\frac{-C^2}{4P(I)[1-P(I)]}} \ .
\end{aligned}$$

This illustrates that in the case of a positive value for α_1, $S(I)$ is close to 0 as b increases.

In the case of the lower bound for $S(I)$, we focus on the upper bound for $Y(I)$. Here, the same observation holds. When α_2 is greater than or equal to a large constant C times $1/\sqrt{b}$, the bound in 3.30 is, for fixed $P(I)$ that is not close to 0 or 1, less than a large negative

constant. Indeed,

$$\begin{aligned} Y(I) &\leq e^{\frac{-b\alpha_2^2}{4P(I)[1-P(I)]}} \\ &\leq e^{\frac{-C^2}{4P(I)[1-P(I)]}} \quad . \end{aligned}$$

This illustrates that $Y(I)$ is close to 0 in this case, leading to $S(I)$ being close to 1.

Thus, when b is large, our probability of interest $S(I)$ turns out to approach 0 or 1, depending on which side of a bound $P(I)$ is on.

3.4.3 The Candidate and Failure Probability

To compute the candidate and failure probability, we define the following conditions with respect to a single transaction:

M: a transaction contains all items in I, and

M_i: a transaction contains all items of test set I_i, without containing all items of I.

For all the algorithms we consider (except FCA), these conditions are disjoint.

The probability that a transaction satisfies condition M is $P(I)$, the probability that it satisfies condition M_i is $Q_i(I)$.

As long as the M_is are disjoint, the probability that j_0 transactions satisfy condition M, j_1 transactions satisfy condition M_1, ..., j_n transactions satisfy condition M_n and the remaining $b - j_0 - \cdots - j_n$ transactions do not satisfy any of the conditions can be expressed by

$$\binom{b}{j_1, \ldots, j_n, b - j_0 - j_1 - \cdots - j_n} \times [P(I)]^{j_0} \left[\prod_{1 \leq i \leq n} Q_i(I)^{j_i} \right] \times \left[1 - P(I) - \sum_{1 \leq i \leq n} Q_i(I) \right]^{b - j_0 - \sum_{1 \leq i \leq n} j_i} . \quad (3.33)$$

If the M_i overlap, then the details are more complex, but the situation is similar and a variant of (3.33) can be derived. We give a sketch of how to find this variant. For this more complex case, define a set of $Q'_{i'}$ with an expanded index set. The expanded index set will give a name to each region of transactions space with regard to the test sets. There will be a region for the transactions that are in the ear of the first test set without being in any other test set, a region for the transactions that are in the ears of both test sets 1 and 2 but no others, etc. For each i', $Q'_{i'}$ gives the probability that a transactions is in the region for i'. With this notation, the probability associated with a set of $j_{i'}$ can be written with an equation that looks just like (3.33), except that many of the symbols will be primed and n will denote the number of regions.

Thus, for any algorithm we consider, (3.33) (or a variant) gives the probability related to a particular set of counts (the j values). If this equation is summed over the cases that lead a particular algorithm to make I a *candidate*, then the formula gives the probability that I is a candidate. If we sum over the conditions that lead to I being a *failure*, then we obtain the probability that I is a failure. The set of conditions depends on the particular algorithm.

3.4.4 Summary

Success Probability

The probability that at least σ of b transactions contain all the items of set I is

$$S(I) \;=\; \sum_{j \geq \sigma} \binom{b}{j} \; [P(I)]^j \; [1 - P(I)]^{b-j} \;\; .$$

When $P(I) \leq \sigma/b$, it can be shown that

$$S(I) \leq e^{-b\alpha_1^2/\{2P(I)[1-P(I)]\} + \mathcal{O}(b\alpha_1^3[1-P(I)]^{-2})} \;\; ,$$

with $\alpha_1 = \sigma/b - P(I)$. In this case $S(I)$ goes to 0 rapidly with increasing α_1.

When $P(I) \geq (\sigma - 1)/b$, we can find

$$S(I) \geq 1 - e^{-b\alpha_2^2/\{2P(I)[1-P(I)]\} + \mathcal{O}(b\alpha_2^3 P(I)^{-2})} \;\; ,$$

with $\alpha_2 = P(I) - (\sigma - 1)/b$. In this case $S(I)$ goes to 1 rapidly with increasing α_2.

Candidate and Failure Probability

W.r.t. disjoint conditions

M: a transaction contains all items in I, and

M_i: a transaction contains all items of test set I_i, without containing all items of I.

the probability that j_0 transactions satisfy condition M, j_1 transactions satisfy condition M_1, $\ldots$, j_n transactions satisfy condition M_n and the remaining $b - j_0 - \cdots - j_n$ transactions do not satisfy any of the conditions can be expressed by

$$\binom{b}{j_1, \ldots, j_n, b - j_0 - j_1 - \cdots - j_n} \times \; [P(I)]^{j_0} \left[\prod_{1 \leq i \leq n} Q_i(I)^{j_i} \right] \times \left[1 - P(I) - \sum_{1 \leq i \leq n} Q_i(I) \right]^{b - j_0 - \sum_{1 \leq i \leq n} j_i} \;\; .$$

If this equation is summed over the cases that lead a particular algorithm to make I a *candidate*, then the formula gives the probability that I is a candidate. If we sum over the conditions that lead to I being a *failure*, then we obtain the probability that I is a failure. The set of conditions depends on the particular algorithm.

3.5 The Failure Probability for Apriori

We now focus on the failure probability for the Apriori Algorithm. In this case, the number of tests associated with a set I is $|I|$. The distinct test sets are all the subsets of I, leaving one element out. Set I is a candidate if for every test set I_i $(1 \leq i \leq |I|)$ the number of transactions that are in $M \cup M_i$ is at least σ. This happens when

$$j_0 + j_1 \geq \sigma \text{ and } j_0 + j_2 \geq \sigma \text{ and } \cdots \text{ and } j_0 + j_{|I|} \geq \sigma \tag{3.34}$$

is true. Thus, to find the probability that I is a candidate for the Apriori Algorithm, we must sum eq. (3.33) (with $n = |I|$) subject to condition (3.34), what leads to

$$C(I) = \sum_{\substack{j_0 \\ j_1 \geq \sigma - j_0 \\ j_2 \geq \sigma - j_0 \\ \cdots \\ j_{|I|} \geq \sigma - j_0}} \binom{b}{j_0, j_1, \ldots, j_{|I|}, b - j_0 - j_1 \cdots - j_{|I|}} \times [P(I)]^{j_0} \left[\prod_{1 \leq i \leq |I|} Q_i(I)^{j_i} \right] \left[1 - P(I) - \sum_{1 \leq i \leq |I|} Q_i(I) \right]^{b - j_0 - \sum_{1 \leq i \leq |I|} j_i} .$$

The subset of these cases, where j_0 is smaller than σ, lead to I being a failure.

Based on the relationship between candidacy, success and failure probability, the following expression for $F(I)$ for Apriori can be found:

$$\begin{aligned} F(I) &= C(I) - S(I) \\ &= \sum_{\substack{j_0 < \sigma \\ j_1 \geq \sigma - j_0 \\ j_2 \geq \sigma - j_0 \\ \cdots \\ j_{|I|} \geq \sigma - j_0}} \binom{b}{j_0, j_1, \ldots, j_{|I|}, b - j_0 - j_1 - \cdots - j_{|I|}} \times [P(I)]^{j_0} \left[\prod_{1 \leq i \leq |I|} Q_i(I)^{j_i} \right] \\ &\qquad \times \left[1 - P(I) - \sum_{1 \leq i \leq |I|} Q_i(I) \right]^{b - j_0 - \sum_{1 \leq i \leq |I|} j_i} . \end{aligned} \tag{3.35}$$

3.5.1 Efficient Computation

We can calculate the number of operations needed to compute F by direct application of (3.35). This gives $\mathcal{O}(\sigma b^{|I|})$ as a result. However, we can compute $F(I)$ in time that is polynomial in b, σ and $|I|$ by using recurrences. Therefore, we write (3.35) as

$$F(I) = \sum_{j_0 < \sigma} \binom{b}{j_0} [P(I)]^{j_0} \; R_{\sigma - j_0}(b - j_0, I, I) \;\; , \tag{3.36}$$

where, for $I' \subseteq I$,

$$\begin{aligned} R_\sigma(b, I, I') &= \sum_{\substack{j_1 \geq \sigma \\ j_2 \geq \sigma \\ \cdots \\ j_{|I'|} \geq \sigma}} \binom{b}{j_1, \ldots, j_{|I'|}, b - j_1 - \cdots - j_{|I'|}} \\ &\qquad \times \left[\prod_{1 \leq i \leq |I'|} Q_i(I)^{j_i} \right] \left[1 - P(I) - \sum_{1 \leq i \leq |I|} Q_i(I) \right]^{b - \sum_{1 \leq i \leq |I'|} j_i} . \end{aligned} \tag{3.37}$$

Now, the following recursion can be used:

$$R_\sigma(b, I, I') = \sum_{j_{|I'|} \geq \sigma} \binom{b}{j_{|I'|}} Q_{|I'|}(I)^{j_{|I'|}} \; R_\sigma(b - j_{|I'|}, \; I, \; I' - \{|I'|\}). \tag{3.38}$$

In (3.38), we removed the last element ($|I'|$) from I' to develop the recurrence, but a valid recurrence could also be obtained by removing any element. The boundary condition is

$$R_\sigma(b, I, \emptyset) = \left[1 - P(I) - \sum_{1 \le i \le |I|} Q_i(I)\right]^b .$$

With (3.38) the Rs can be computed in time $\mathcal{O}(|I|b^2)$. With (3.36) we can compute F in time $\mathcal{O}(|I|b^2)$.

3.5.2 Chernoff Bound

Upper Bound for the Apriori Failure Probability

For most values of $P(I)$ and the $Q_i(I)$, $F(I)$ will either be close to 0 or close to 1. Chernoff bounds provide a good method to approximate $F(I)$ [24]. First, a Chernoff bound for R_σ (see Subsection 3.5.1) is computed, and based on this result a Chernoff bound for $F(I)$ is found.

A first Chernoff function can be applied leading to

$$R_\sigma(b, I, I') \;\le\; x_{|I'|}^{-\sigma} \sum_{j_{|I'|}} \binom{b}{j_{|I'|}} [x_{|I'|} Q_{|I'|}(I)]^{j_{|I'|}} \; R_\sigma(b - j_{|I'|}, I, I' - \{|I'|\}) \;, \quad (3.39)$$

with $x_{|I'|} \ge 1$. For the R_σ-term in the above result (3.39), again a Chernoff bound can be applied, and this procedure can be repeated until all elements in I' are done. In the last step, the basic recurrence for $I' = \emptyset$ is used. The final result can be written as

$$\begin{aligned} R_\sigma(b, I, I') \;\le\; & \sum_{j_1, \ldots, j_{|I'|}} \prod_{1 \le i \le |I'|} (x_i^{-\sigma}) \quad \binom{b}{j_1, \ldots, j_{|I'|}} \prod_{1 \le i \le |I'|} [x_i Q_i(I)]^{j_i} \\ & \times [1 - P(I) - \sum_{1 \le i \le |I|} Q_i(I)]^{b - j_1 - \ldots - j_{|I'|}} \;, \end{aligned} \quad (3.40)$$

with $x_i \ge 1$, for $1 \le i \le |I'|$.

The above Chernoff bound for R_σ (3.40) can be used to compute the Chernoff bound for $F(I)$. The idea is to plug in the Chernoff bound for $R_{\sigma - j_0}(b - j_0, I, I)$ in (3.36).

$$\begin{aligned} F(I) \;\le\; & \sum_{j_0 < \sigma} \binom{b}{j_0} P(I)^{j_0} \sum_{j_1, \ldots, j_{|I|}} \prod_{1 \le i \le |I|} (x_i^{-\sigma + j_0}) \binom{b - j_0}{j_1, \ldots, j_{|I|}} \prod_{1 \le i \le |I|} [x_i Q_i(I)]^{j_i} \\ & \times [1 - P(I) - \sum_{1 \le i \le |I|} Q_i(I)]^{b - j_0 - j_1 - \ldots - j_{|I|}} \end{aligned} \quad (3.41)$$

Inspired by the proof of Property 3.4.6, we rewrite (3.41) and we simplify the expression, resulting in

$$
\begin{aligned}
F(I) \;&\leq\; y^{-\sigma+1} \sum_{j_0,\ldots,j_{|I|}} \prod_{1\leq i\leq |I|} x_i^{-\sigma+j_0} \binom{b}{j_0, j_1, \ldots, j_{|I|}, b-j_0-j_1\cdots-j_{|I|}} \\
&\qquad \times [P(I)y]^{j_0} \left[\prod_{1\leq i\leq |I|} [Q_i(I)x_i]^{j_i}\right] \left[1-P(I)-\sum_{1\leq i\leq |I|} Q_i(I)\right]^{b-j_0-\sum_{1\leq i\leq |I|} j_i} \\
&=\; \left[1+P(I)\left(y\prod_{1\leq i\leq |I|} x_i - 1\right) + \sum_{1\leq i\leq |I|} Q_i(I)(x_i-1)\right]^b y^{-\sigma+1} \prod_{1\leq i\leq |I|} x_i^{-\sigma},
\end{aligned}
\tag{3.42}
$$

for any $y \leq 1$ and any set of $x_i \geq 1$. The last equality is achieved by applying the Binomial Theorem (3.1) twice.

To find the *best* Chernoff bound for $F(I)$, we take the partial derivative of (3.42) w.r.t. y and w.r.t. each x_i. To simplify the computations, we will work with *the logarithm* of (3.42) and take the partial derivative w.r.t. y and each x_i.

Setting the partial derivative with respect to y to zero gives

$$
\begin{aligned}
&\frac{b\ P(I)\ \prod_{1\leq i\leq |I|} x_i}{1+P(I)\ (y\prod_{1\leq i\leq |I|} x_i - 1) + \sum_{1\leq i\leq |I|} Q_i(I)(x_i-1)} - \frac{\sigma-1}{y} = 0 \\
\Leftrightarrow\quad &\left(b\ P(I)\ y \prod_{1\leq i\leq |I|} x_i - (\sigma-1)\ \left[1+P(I)\left(y\prod_{1\leq i\leq |I|} x_i - 1\right) + \sum_{1\leq i\leq |I|} Q_i(I)(x_i-1)\right]\right) = 0 \\
\Leftrightarrow\quad &(b-\sigma+1)\ P(I)\ y \prod_{1\leq i\leq |I|} x_i - (\sigma-1)\ \left[1-P(I)+\sum_{1\leq i\leq |I|} Q_i(I)(x_i-1)\right] = 0
\end{aligned}
\tag{3.43}
$$

Setting the partial derivative w.r.t. $x_{i'}$ to zero gives

$$
\begin{aligned}
&\frac{b\ [P(I)\ y\ \prod_{\substack{1\leq i\leq |I| \\ i\neq i'}} x_i + Q_{i'}(I)]}{1+P(I)\ (y\prod_{1\leq i\leq |I|} x_i - 1) + \sum_{1\leq i\leq |I|} Q_i(I)(x_i-1)} - \frac{\sigma}{x_{i'}} = 0 \\
\Leftrightarrow\quad &b\left[P(I)y\prod_{1\leq i\leq |I|} x_i + Q_{i'}(I)x_{i'}\right] \\
&\qquad -\sigma\left[1+P(I)\ \left(y\prod_{1\leq i\leq |I|} x_i - 1\right) + \sum_{1\leq i\leq |I|} Q_i(I)(x_i-1)\right] = 0 \\
\Leftrightarrow\quad &(b-\sigma)\ P(I)\ y \prod_{1\leq i\leq |I|} x_i + b\ Q_{i'}(I)\ x_{i'} \\
&\qquad -\sigma\ \left[1-P(I)+\sum_{1\leq i\leq |I|} Q_i(I)(x_i-1)\right] = 0\ .
\end{aligned}
\tag{3.44}
$$

To find the optimum Chernoff bound, we now need to solve (3.43) and (3.44), except that we omit those equations that would lead to their variable being out of range. Thus, (3.43) is

included if and only if the implied value for y is less than or equal to 1. Otherwise, we set $y = 1$ and ignore (3.43). Analogously, for each $x_{i'}$, we include (3.44) if and only if the implied value for $x_{i'}$ is greater than or equal to 1. In order to do so, we assume that we can order the values in I according to their corresponding x-value. Therefore, let I' be a subset of I such that for i in I', $x_i > 1$ and for i in $I - I'$, $x_i = 1$. We order the elements in I so that for $1 \leq i \leq |I'|$, i is an element of I' while for $|I'| < i \leq |I|$, i is an element of $I - I'$. Now, (3.43) can be written as

$$(b - \sigma + 1)\ P(I)\ y \prod_{1 \leq i \leq |I'|} x_i - (\sigma - 1)\ \left[1 - P(I) + \sum_{1 \leq i \leq |I'|} Q_i(I)(x_i - 1)\right] = 0\ , \quad (3.45)$$

and (3.44) can be written as

$$(b - \sigma)\ P(I)\ y \prod_{1 \leq i \leq |I'|} x_i + b\ Q_{i'}(I)\ x_{i'} - \sigma\ \left[1 - P(I) + \sum_{1 \leq i \leq |I'|} Q_i(I)(x_i - 1)\right] = 0\ . \quad (3.46)$$

For (3.46) the following interesting fact is noticed. When we take the difference of two versions from (3.46) with different values for i', for example i'_1 and i'_2, we conclude that

$$bQ_{i'_1}(I)x_{i'_1} - bQ_{i'_2}(I)x_{i'_2} = 0\ . \quad (3.47)$$

This means that the value of $Q_i(I)x_i$ does not vary with i. Therefore, the largest value of x_i is associated with the smallest value for $Q_i(I)$ and vice versa.

Suppose we have a solution for (3.46), or alternately a joint solution for (3.45) and (3.46), with all variables within their required range. Now suppose we change σ so as to reduce the value of each x_i for i in I' toward 1. The various x_i hit 1 at different times. Since the ratio of the x_i is given by (3.47), the x_i associated with the largest $Q_i(I)$ will be the first x_i to be no longer greater than 1. At this point, we are on the boundary and element i can be dropped from I', because we cannot use the formula of the derivative anymore. The equation corresponding to that i has to be dropped together with i in I. We can continue like this as long as we leave at least one element in I'. This will leave us with all the remaining elements in I' having the same value for Q. We call this value $Q(I)$. Likewise, all the relevant x_i are equal, so we call them x. For the I' that has only the smallest $Q_i(I)$, we define $\ell = |I'|$. Therefore, when σ is small enough to leave only the x_i associated with the smallest $Q_i(I)$ in range, we can write (3.45) as

$$(b - \sigma + 1)\ P(I)\ y\ x^\ell - (\sigma - 1)\ [1 - P(I) + \ell Q(I)(x - 1)] = 0\ , \quad (3.48)$$

and we can write (3.46) as

$$(b - \sigma)\ P(I)\ y\ x^\ell + b\ Q(I)\ x - \sigma\ [1 - P(I) + \ell Q(I)(x - 1)] = 0\ . \quad (3.49)$$

Since σ must remain an integer as it decreases, the above argument sometimes requires b to be large.

Case $Q(I) + P(I) \leq \sigma/b$ First, we are interested in the boundary case $x \geq 1$ and $y = 1$. Therefore, we substitute $x = 1$ and $y = 1$ in (3.49). This leads to

$$\begin{aligned} & (b - \sigma)\ P(I) + b\ Q(I) - \sigma\ [1 - P(I)] = 0 \\ \Leftrightarrow\ & b\ P(I) - \sigma\ P(I) + b\ Q(I) - \sigma + \sigma\ P(I) = 0 \\ \Leftrightarrow\ & b\ [P(I) + Q(I)] = \sigma\ . \end{aligned} \quad (3.50)$$

Property 3.5.1. *The requirement $x \geq 1$ in (3.50) leads to $\sigma \geq b[P(I) + Q(I)]$.*

Proof. This is the case because an increasing value of x corresponds with a decreasing value of b. To proof the statement, we start from (3.46) and we set $y = 1$, leading to

$$(b - \sigma)\ P(I)\ x^{\ell} + b\ x\ Q(I) - \sigma\ [1 - P(I) + \ell(x-1)Q(I)] = 0\ , \tag{3.51}$$

with $x \geq 1$. We now perform implicit differentiation with x as a function of b.

$$\begin{aligned}
& \frac{\mathrm{d}}{\mathrm{d}b}\left[(b-\sigma)\ P(I)\ x(b)^{\ell} + b\ x(b)\ Q(I) - \sigma\ [1 - P(I) + \ell\ (x(b)-1)\ Q(I)]\right] = 0 \\
\Leftrightarrow\ & \frac{\mathrm{d}}{\mathrm{d}b}\left(b\ P(I)\ x(b)^{\ell}\right) - \frac{\mathrm{d}}{\mathrm{d}b}\left(\sigma\ P(I)\ x(b)^{\ell}\right) + \frac{\mathrm{d}}{\mathrm{d}b}\left(b\ x(b)\ Q(I)\right) - \frac{\mathrm{d}}{\mathrm{d}b}\left(\sigma\ \ell\ x(b)\ Q(I)\right) = 0 \\
\Leftrightarrow\ & P(I)\ x\ (b)^{\ell} + b\ P(I)\ \frac{\mathrm{d}}{\mathrm{d}b}\left(x(b)^{\ell}\right) - \sigma\ P(I)\frac{\mathrm{d}}{\mathrm{d}b}\left(x(b)^{\ell}\right) \\
& \qquad + x(b)\ Q(I) + b\ Q(I)\ \frac{\mathrm{d}}{\mathrm{d}b}\left(x(b)\right) - \sigma\ \ell\ Q(I)\frac{\mathrm{d}}{\mathrm{d}b}\left(x(b)\right) = 0 \\
\Leftrightarrow\ & P(I)\ x(b)^{\ell} + b\ P(I)\ \frac{\mathrm{d}}{\mathrm{d}x}\left(x(b)^{\ell}\right)\ \frac{\mathrm{d}x}{\mathrm{d}b} - \sigma\ P(I)\ \frac{\mathrm{d}}{\mathrm{d}x}\left(x(b)^{\ell}\right)\ \frac{\mathrm{d}x}{\mathrm{d}b} \\
& \qquad + x(b)\ Q(I) + b\ Q(I)\ \frac{\mathrm{d}}{\mathrm{d}x}\left(x(b)\right)\frac{\mathrm{d}x}{\mathrm{d}b} - \sigma\ \ell\ Q(I)\ \frac{\mathrm{d}}{\mathrm{d}x}\left(x(b)\right)\ \frac{\mathrm{d}x}{\mathrm{d}b} = 0 \\
\Leftrightarrow\ & P(I)\ x(b)^{\ell} + b\ P(I)\ \ell\ x(b)^{\ell-1}\ \frac{\mathrm{d}x}{\mathrm{d}b} - \sigma\ P(I)\ \ell\ x(b)^{\ell-1}\frac{\mathrm{d}x}{\mathrm{d}b} \\
& \qquad + x(b)\ Q(I) + b\ Q(I)\frac{\mathrm{d}x}{\mathrm{d}b} - \sigma\ \ell\ Q(I)\ \frac{\mathrm{d}x}{\mathrm{d}b} = 0 \\
\Leftrightarrow\ & P(I)\ x^{\ell} + Q(I)\ x + \left[(b-\sigma)\ \ell\ P(I)\ x^{\ell-1} + (b - \sigma\ell)\ Q(I)\right]\ \frac{\mathrm{d}x}{\mathrm{d}b} = 0 \\
\Leftrightarrow\ & \frac{\mathrm{d}x}{\mathrm{d}b} = -\frac{P(I)\ x^{\ell} + Q(I)\ x}{(b-\sigma)\ \ell\ P(I)\ x^{\ell-1} + (b - \sigma\ell)\ Q(I)}
\end{aligned} \tag{3.52}$$

Because we are interested in the behavior of x when b changes, we study the sign of the right-hand side of (3.52).

Nominator

We have that $P(I)\ x^{\ell} + Q(I)\ x \geq 0$, because $P(I)$, $Q(I)$, ℓ and x are positive.

Denominator

We start with rewriting (3.51).

$$\begin{aligned}
& (b-\sigma)\ P(I)\ x^{\ell} + b\ x\ Q(I) - \sigma\ [1 - P(I) + \ell\ (x-1)\ Q(I)] = 0, \quad x \geq 1 \\
\Leftrightarrow\ & (b-\sigma)\ P(I)\ x^{\ell} + (b - \sigma\ell)\ x\ Q(I) - \sigma\,[1 - P(I) - \ell\ Q(I)] = 0, \quad x \geq 1 \\
\Leftrightarrow\ & (b-\sigma)\ P(I)\ x^{\ell} + (b - \sigma\ell)\ x\ Q(I) = \sigma\,[1 - P(I) - \ell\ Q(I)]\,, \quad x \geq 1
\end{aligned} \tag{3.53}$$

We now show that $\sigma\,[1 - P(I) - \ell\ Q(I)]$ is positive.

$$\begin{aligned} \ell\ Q(I) + P(I) - 1 &= \ell\ P(I_i) - \ell\ P(I) + P(I) - 1 \\ &= \ell\ P(I_i) + (\ell - 1)\ P(I) - 1 \\ &\leq \ell\ P(I_i) + (\ell - 1)\ P(I_i) - 1 \\ &= P(I_i) - 1 \\ &\leq 0\ , \end{aligned}$$

based on the definition of $Q(I)$, the monotonicity principle and the fact that a probability is always sandwiched between 0 and 1.

Based on this result, the denominator of (3.52) itself now can be rewritten to

$$\begin{aligned} (b - \sigma)\ \ell\ P(I)\ x^{\ell-1} + (b - \sigma\ell)\ Q(I) &= \frac{(b - \sigma)\ \ell\ P(I)\ x^{\ell} + (b - \sigma\ell)\ x\ Q(I)}{x} \\ &= \frac{(b - \sigma)\ P(I)\ x^{\ell} + (b - \sigma\ell)\ x\ Q(I)}{x} \\ &\quad + (\ell - 1)\ (b - \sigma)\ P(I)\ x^{\ell-1}\ . \end{aligned} \tag{3.54}$$

This result is positive, because the nominator of the first fraction, x, $\ell - 1$, $b - \sigma$ and $P(I)$ are positive.

Combining all the information together, we have that

$$\frac{\mathrm{d}x}{\mathrm{d}b} \leq 0\ .$$

Therefore, if we start at the value b that results in $x = 1$ and b decreases, then x increases. At a certain point x will be strictly bigger than 1. □

We are in the region described by $Q(I) + P(I) \leq \sigma/b$. We now introduce α_3 to express the difference:

$$\alpha_3 = \frac{\sigma}{b} - [Q(I) + P(I)]\ . \tag{3.55}$$

Theorem 3.5.1. *In the region $Q(I) + P(I) \leq \sigma/b$, $F(I)$ goes rapidly to 0. In particular, when α_3 is small enough, i.e.,*

$$\alpha_3 = \{\ell\ P(I) + Q(I) - \ell\ [Q(I) + P(I)]^2\} o(1) \tag{3.56}$$

and b is large enough that the second smallest $Q_i(I)$ is not important,

$$F(I) \leq e^{-b\ \ell\ \theta\ \alpha_3^2/2\left(Q(I)+P(I)+(\ell-1)P(I)-\ell[Q(I)+P(I)]^2\right)} \tag{3.57}$$

gives an upper bound on the failure probability. Here θ is used to represent a function that approaches 1 in the limit.

Proof. Before we prove the statement in detail, we intuitively show that is has to be true. By substituting $\ell = 1$ in (3.56) and substituting this result in (3.57), we can find that

$$F(I) \leq e^{-\frac{b}{2}[P(I)+Q(I)-[P(I)+Q(I)]^2]} \ .$$

Because $\alpha_3 = P(I)+Q(I)-[P(I)+Q(I)]^2$ is small and b is very large, $F(I)$ is approximately near 0 in this region.

In (3.51), we replace x by $1+\delta$, with δ small.

$$\begin{aligned} & (b-\sigma)\ P(I)\ (1+\delta)^\ell + b\ (1+\delta)Q(I) - \sigma\ [1-P(I)+\ell((1+\delta)-1)\ Q(I)] = 0 \\ \Leftrightarrow\ & (b-\sigma)\ P(I)\ (1+\delta)^\ell + (b-\sigma\ell)\ (1+\delta)Q(I) = \sigma\ [1-P(I)-\ell\ Q(I)] \end{aligned} \tag{3.58}$$

Expand $(1+\delta)^\ell$ to second order:

$$(1+\delta)^\ell = 1 + \ell\delta + \frac{\ell(\ell-1)}{2}\delta^2\theta, \tag{3.59}$$

with θ a function that approaches 1 in the limit as δ approaches 0. In fact, θ is shorthand for $1+o(1)$ where δ is the variable approaching 0. Now, (3.58) becomes

$$\begin{aligned} & (b-\sigma)\ P(I)\ \left[1+\ell\delta+\frac{\ell(\ell-1)}{2}\delta^2\theta\right] + (b-\sigma\ell)\ (1+\delta)\ Q(I) = \sigma\ [1-P(I)-\ell Q(I)] \\ \Leftrightarrow\ & (b-\sigma)\ P(I)\ \left[\ell\delta+\frac{\ell(\ell-1)}{2}\delta^2\theta\right] + (b-\sigma\ell)\ \delta\ Q(I) \\ & \qquad = \sigma\ [1-P(I)-\ell Q(I)] - (b-\sigma)\ P(I) - (b-\sigma\ell)\ Q(I) \\ \Leftrightarrow\ & \delta\ \left[(b-\sigma)\ P(I)\ \ell\ \left[1+\frac{\ell-1}{2}\delta\theta\right] + (b-\sigma\ell)\ Q(I)\right] \\ & \qquad = \sigma\ [1-P(I)-\ell Q(I)] - (b-\sigma)\ P(I) - (b-\sigma\ell)\ Q(I) \\ \Leftrightarrow\ & \delta = \frac{\sigma\ [1-P(I)-\ell Q(I)] - (b-\sigma)\ P(I) - (b-\sigma\ell)\ Q(I)}{(b-\sigma)\ P(I)\ \ell\ \left[1+\frac{\ell-1}{2}\delta\theta\right] + (b-\sigma\ell)\ Q(I)} \\ \Leftrightarrow\ & \delta = \frac{\sigma - b\ [P(I)+Q(I)]}{b\ [\ell P(I)+Q(I)] - \sigma\ \ell\ [P(I)+Q(I)] + (b-\sigma)\ \ell\ P(I)\ \frac{\ell-1}{2}\ \delta\ \theta} \\ \Leftrightarrow\ & \delta = \frac{\sigma - b\ [P(I)+Q(I)]}{b\ [\ell P(I)+Q(I)] - \sigma\ \ell\ [P(I)+Q(I)]} \frac{1}{1+\frac{(b-\sigma)\ \ell\ P(I)\ \frac{\ell-1}{2}\ \delta\ \theta}{b\ [\ell P(I)+Q(I)]-\sigma\ \ell\ [P(I)+Q(I)]}} \\ \Leftrightarrow\ & \delta = \frac{\sigma - b\ [P(I)+Q(I)]}{b\ [\ell P(I)+Q(I)] - \sigma\ \ell\ [P(I)+Q(I)]} \left(1+\frac{(b-\sigma)\ \ell\ P(I)\ \frac{\ell-1}{2}\ \delta\ \theta}{b\ [\ell P(I)+Q(I)] - \sigma\ \ell\ [P(I)+Q(I)]}\right)^{-1} \end{aligned} \tag{3.60}$$

This is an expression for δ in terms of δ. A first approximation for δ, called δ_0, can be found by substituting zero for the right most δ in (3.60):

$$\delta_0 = \frac{\sigma - b\ [P(I)+Q(I)]}{b\ [\ell P(I)+Q(I)] - \sigma\ \ell\ [P(I)+Q(I)]}.$$

The next approximation for δ, δ_1, can be found by substituting the expression for δ_0 in the rightmost δ in (3.60). This gives

$$\begin{aligned}\delta_1 &= \frac{\sigma - b\ [P(I)+Q(I)]}{b\ [\ell P(I)+Q(I)] - \sigma\ \ell\ [P(I)+Q(I)]} \\ &\qquad \left(1 + \frac{(b-\sigma)\ \ell\ P(I)\ \frac{\ell-1}{2}\ \theta\ (\sigma - b\ [P(I)+Q(I)])}{(b\ [\ell P(I)+Q(I)] - \sigma\ \ell\ [P(I)+Q(I)])^2}\right)^{-1} .\end{aligned}$$

We now replace σ by its expression in terms of α_3 (3.55), leading to

$$\begin{aligned}\delta_1 &= \frac{b\ \alpha_3}{b\ [\ell P(I)+Q(I)] - b\ \ell\ [P(I)+Q(I)]^2 - b\ \alpha_3\ \ell\ [P(I)+Q(I)]} \\ &\qquad \left(1 + \frac{b\ (1-P(I)-Q(I)-\alpha_3)\ \ell\ (\ell-1)\ b\ \alpha_3\ P(I)\ \frac{\theta}{2}}{(b\ [\ell P(I)+Q(I)] - b\ \ell\ [P(I)+Q(I)]^2 - b\ \alpha_3\ [P(I)+Q(I)])^2}\right)^{-1} \\ &= \frac{\alpha_3}{[\ell P(I)+Q(I)] - \ell\ [P(I)+Q(I)]^2 - \alpha_3\ \ell\ [P(I)+Q(I)]} \\ &\qquad \left(1 + \frac{(1-P(I)-Q(I)-\alpha_3)\ \ell\ (\ell-1)\ \alpha_3\ P(I)\ \frac{\theta}{2}}{([\ell P(I)+Q(I)] - \ell\ [P(I)+Q(I)]^2 - \alpha_3\ [P(I)+Q(I)])^2}\right)^{-1} . \qquad (3.61)\end{aligned}$$

We continue with approximating δ_1. Therefore, we take care of α_3 in (3.61). The α_3 in the second part of (3.61) is set to zero, resulting in

$$\delta_{1,0} = \frac{\alpha_3}{[\ell P(I)+Q(I)] - \ell\ [P(I)+Q(I)][P(I)+Q(I)+\alpha_3]} . \qquad (3.62)$$

In (3.62), we assume that α_3 is almost zero:

$$\delta_{1,0} = \frac{\alpha_3\theta}{[\ell P(I)+Q(I)] - \ell\ [P(I)+Q(I)]^2} . \qquad (3.63)$$

This value (3.63) is from now on used as an approximation for δ.

We now rewrite the upper bound on $F(I)$ (3.42) in terms of ℓ with $y = 1$ as

$$F(I) \leq e^X , \qquad (3.64)$$

with

$$\begin{aligned}X &= \ln\Big([1 + P(I)\ (x^\ell - 1) + \ell\ Q(I)\ (x-1)]^b\ x^{-\sigma\ell}\Big) \\ &= -\sigma\ \ell\ \ln x + b\ \ln\Big(1 + (x^\ell - 1)P(I) + \ell(x-1)Q(I)\Big)\end{aligned}$$

In this expression, x is replaced by its expression in terms of δ.

$$X = -\sigma\ \ell\ \ln(1+\delta) + b\ \ln\Big(1 + ((1+\delta)^\ell - 1)P(I) + \ell((1+\delta)-1)Q(I)\Big)$$

We expand $(1+\delta)^\ell$ (to the second order) as we did before.

$$\begin{aligned}X &= -\sigma\ \ell\ \ln(1+\delta) + b\ \ln\left(1 + (1 + \ell\delta + \frac{\ell(\ell-1)}{2}\delta^2\theta - 1)P(I) + \ell\delta Q(I)\right) \\ &= -\sigma\ \ell\ \ln(1+\delta) + b\ \ln\left(1 + \ell\delta P(I) + \ell(\ell-1)\delta^2\frac{\theta}{2}P(I) + \ell\delta Q(I)\right) \qquad (3.65)\end{aligned}$$

Now, the approximation $\ln(1+x) = x - \frac{x^2}{2}\theta$ is used.

$$\begin{aligned} X &= -\sigma\ell\delta + \sigma\ell\frac{\delta^2}{2}\theta + b\left[\ell\delta P(I) + \ell(\ell-1)\delta^2\frac{\theta}{2}P(I) + \ell\delta Q(I)\right] \\ &\quad -\frac{b\theta}{2}\left[\ell\delta P(I) + \ell(\ell-1)\delta^2\frac{\theta}{2}P(I) + \ell\delta Q(I)\right]^2. \end{aligned} \tag{3.66}$$

From the square in (3.66), we only take the term in δ^2 (this is the biggest term), because δ is very small.

$$\begin{aligned} & X \approx -\sigma\ell\delta + \sigma\ell\frac{\delta}{2}\theta + b[\ell\delta P(I) + \ell(\ell-1)\delta^2\frac{\theta}{2} + \ell\delta Q(I)] - \frac{b\theta}{2}\ell^2\delta^2[P(I)+Q(I)]^2 \\ \Leftrightarrow\quad & X \approx -\ell\left[\sigma - b[P(I)+Q(I)]\right]\delta + \frac{\delta^2}{2}\theta\left[\sigma\ell + b\ell(\ell-1)P(I) - b\ell^2[P(I)+Q(I)]^2\right] \end{aligned}$$

The expression for σ in terms of α_3, $\sigma = b[P(I)+Q(I)+\alpha_3]$ is now substituted, leading to

$$\begin{aligned} & X \approx -\ell b\alpha_3\delta + \frac{\delta^2}{2}\theta\left[b\ell[P(I)+Q(I)] + b\ell\alpha_3 + b\ell(\ell-1)P(I) - b\ell^2[P(I)+Q(I)]^2\right] \\ \Leftrightarrow\quad & X \approx -\ell b\alpha_3\delta + \frac{\delta^2}{2}\theta b\ell\left[[P(I)+Q(I)] + \alpha_3 + (\ell-1)P(I) - \ell[P(I)+Q(I)]^2\right] \\ \Leftrightarrow\quad & X \approx \left(-\ell b\alpha_3 + \frac{\delta}{2}\theta b\ell\left[[P(I)+Q(I)] + \alpha_3 + (\ell-1)P(I) - \ell[P(I)+Q(I)]^2\right]\right)\delta\ . \end{aligned}$$

In this expression, we replace δ by its approximation (3.63).

$$\begin{aligned} & X \approx (-\ell b\alpha_3 + \frac{\theta}{2}b\ell\left[P(I)+Q(I)+\alpha_3 + (\ell-1)P(I) - \ell[P(I)+Q(I)]^2\right] \\ & \qquad \frac{\alpha_3\theta}{[\ell P(I)+Q(I)] - \ell[P(I)+Q(I)]^2})(\frac{\alpha_3\theta}{[\ell P(I)+Q(I)] - \ell[P(I)+Q(I)]^2}) \\ \Leftrightarrow\quad & X \approx (-\ell b\alpha_3 + \frac{\ell b\theta^2\alpha_3}{2} + \frac{\ell b\theta^2\alpha_3^2}{2\left[[\ell P(I)+Q(I)] - \ell[P(I)+Q(I)]^2\right]}) \\ & \qquad\qquad (\frac{\alpha_3\theta}{[\ell P(I)+Q(I)] - \ell[P(I)+Q(I)]^2}) \\ \Leftrightarrow\quad & X \approx \frac{-\ell b\theta\alpha_3^2}{[\ell P(I)+Q(I)] - \ell[P(I)+Q(I)]^2} \\ & \qquad + \frac{\ell b\theta^3\alpha_3^2}{2\Big([\ell P(I)+Q(I)] - \ell[P(I)+Q(I)]^2\Big)} \\ & \qquad\qquad + \frac{\ell b\theta^3\alpha_3^3}{2\Big([\ell P(I)+Q(I)] - \ell[P(I)+Q(I)]^2\Big)^2} \end{aligned} \tag{3.67}$$

The term with α_3^3 is neglected and all the powers of θ are replaced by the same θ'.

$$X \approx \frac{-\ell b\theta'\alpha_3^2}{[\ell P(I)+Q(I)] - \ell[P(I)+Q(I)]^2} + \frac{\ell b\theta'\alpha_3^2}{2\left[[\ell P(I)+Q(I)] - \ell[P(I)+Q(I)]^2\right]} \tag{3.68}$$

$$\approx \frac{-\ell b\theta'\alpha_3^2}{2\left[[\ell P(I)+Q(I)] - \ell[P(I)+Q(I)]^2\right]} \tag{3.69}$$

We now have an upper bound for $F(I)$:

$$F(I) \leq e^{\frac{-\ell b \theta' \alpha_3^2}{2[[\ell P(I)+Q(I)]-\ell[P(I)+Q(I)]^2]}} \ .$$

We have to remark that this upper bound is only correct when α_3 is small. This condition is used to have (3.63) as an approximation for δ and to find (3.69) as an approximation for X. Therefore, the step from (3.67) to (3.68) implies that

$$\alpha_3 = \left[\ell P(I) + Q(I) - \ell[P(I) + Q(I)]^2\right] o(1).$$

□

Case $P(I) \geq (\sigma - 1)/b$ Second, we are interested in the boundary case $x = 1$ and $y \leq 1$. Therefore, we substitute $x = 1$ in (3.48). This leads to

$$(b - \sigma + 1)\ y\ P(I) + (-\sigma + 1)\ [1 - P(I)] = 0 \ ,$$

with $y \leq 1$. For y, the following condition can be found:

$$y = \frac{(\sigma - 1)\ [1 - P(I)]}{(b - \sigma + 1)\ P(I)} \ ,$$

and the requirement $y \leq 1$ results in $\sigma \leq bP(I) + 1$ or

$$\frac{\sigma}{b} \leq P(I) + \frac{1}{b} \ . \tag{3.70}$$

This second region, $P(I) \geq (\sigma - 1)/b$, is the region where $S(I) \approx 1$ (see Subsection 3.4.2), so nearly all sets pass the test. Because $C(I) = S(I) + F(I)$, so $F(I) = C(I) - S(I)$, $F(I)$ cannot be larger than $1 - S(I)$. We therefore can use the bound for $S(I)$ from Subsubsection 3.4.2 to find a bound for $F(I)$.

Theorem 3.5.2.

$$F(I) \leq 1 - S(I) \leq e^{\left(\frac{-b\alpha_2^2}{2P(I)[1-P(I)]}\right)+\mathcal{O}\left(\frac{b\alpha_2^3}{[P(I)]^2}\right)} \ , \tag{3.71}$$

where α_2 is defined by $\sigma = b[P(I) - \alpha_2] + 1$ to express the difference between $P(I) + 1/b$ and σ/b. The big $\mathcal{O}$ is with respect to α_1. If α_2 is positive, $F(I)$ goes to zero rapidly.

Lower Bound for the Apriori Failure Probability

To obtain a lower bound on $F(I)$, we use inclusion-exclusion arguments.

Theorem 3.5.3.

$$F(I) \geq 1 - e^{-b\alpha_1^2/\{2P(I)[1-P(I)]\}+\mathcal{O}(b\alpha_1^3[1-P(I)]^{-2})} \sum_{1 \leq i \leq |I|} e^{-b\beta_i^2/\{2[P(I_i)][1-P(I_i)]\}+\mathcal{O}(b\beta_i^3[P(I_i)]^{-2})} \tag{3.72}$$

where $P(I_i) = P(I) + Q_i(I)$ and where α_1 and β_i related to σ by $\sigma = b[P(I) + \alpha_1]$ and $\sigma = b[P(I) + Q_i(I) - \beta_i] + 1$ when α_1 and all the βs are positive.

Proof. In (3.37) we defined $R_\sigma(b, I, I')$ by using sums in which each $j_i \geq \sigma$ $(1 \leq i \leq |I'|)$. With inclusion-exclusion arguments, an alternative way to compute R_σ can be found. We focus on $R_\sigma(b, I, |I|)$, because this is what we need in (3.36).

Inclusion-exclusion arguments give

$$R_\sigma(b, I, I) = \sum_{I' \subseteq I} (-1)^{|I'|}\ r_\sigma(b, I, I')\ , \tag{3.73}$$

where

$$\begin{aligned} r_\sigma(b, I, I') &= \sum_{\substack{\text{For all } h \text{ in } I',\ j_h < \sigma \\ \text{For all } h \text{ in } I - I',\ j_h}} \binom{b}{j_1, \ldots, j_{|I|}, b - j_1 - \cdots - j_{|I|}} \\ &\qquad \times \left[\prod_{1 \leq i \leq |I|} Q_i(I)^{j_i}\right] \left[1 - P(I) - \sum_{1 \leq i \leq |I|} Q_i(I)\right]^{b - \sum_{1 \leq i \leq |I|} j_i} \\ &= \sum_{\text{For all } h \text{ in } I',\ j_h < \sigma} \binom{b}{j_1, \ldots, j_{|I'|}, b - j_1 - \cdots - j_{|I'|}} \\ &\qquad \times \left[\prod_{1 \leq i \leq |I'|} Q_i(I)^{j_i}\right] \left[1 - P(I) - \sum_{1 \leq i \leq |I'|} Q_i(I)\right]^{b - \sum_{1 \leq i \leq |I'|} j_i}. \end{aligned}$$

The inclusion-exclusion formula starts with an initial case (no limits on the summation indices), corrects by considering a single feature (limit on one sum), corrects the correction for double counting, etc. Therefore, if the sum in (3.73) omits terms when $|I'|$ is above a certain limit, one has an upper or lower limit on the true value, depending on the sign of the leading omitted term. If we limit the terms in the right side of (3.73) to those with $|I'| \leq 1$, we obtain the lower bound

$$R_\sigma(b, I, I) \geq [1 - P(I)]^b - \sum_{1 \leq i \leq |I|} \sum_{j < \sigma} \binom{b}{j}\ Q_i(I)^j\ [1 - P(I) - Q_i(I)]^{b-j}.$$

Plugging this bound into (3.36) gives

$$\begin{aligned} F(I) \ \geq\ & \sum_{j_0 < \sigma} \binom{b}{j_0} P(I)^{j_0} \Bigg([1 - P(I)]^{b - j_0} \\ & - \sum_{1 \leq i \leq |I|} \sum_{j < \sigma} \binom{b - j_0}{j} Q_i(I)^j [1 - P(I) - Q_i(I)]^{b - j_0 - j} \Bigg)\ . \end{aligned} \tag{3.74}$$

Proceeding as in [50], associativity leads to

$$\begin{aligned} F(I) \geq & \sum_{j_0 < \sigma} \binom{b}{j_0} P(I)^{j_0} [1 - P(I)]^{b - j_0} \\ & - \sum_{j_0 < \sigma} \binom{b}{j_0} P(I)^{j_0} \sum_{1 \leq i \leq |I|} \sum_{j < \sigma} \binom{b - j_0}{j} Q_i(I)^j [1 - P(I) - Q_i(I)]^{b - j_0 - j}\ . \end{aligned} \tag{3.75}$$

First, we concentrate on the first part

$$\sum_{j_0<\sigma} \binom{b}{j_0} P(I)^{j_0}[1-P(I)]^{b-j_0} \ . \tag{3.76}$$

A lower bound for (3.76) can be found by using the result (3.10). It is known that

$$\begin{aligned} 1 &= \sum_{j_0} \binom{b}{j_0} P(I)^{j_0} \ [1-P(I)]^{b-j_0} \\ &= \sum_{j_0<\sigma} \binom{b}{j_0} P(I)^{j_0} \ [1-P(I)]^{b-j_0} + \sum_{j_0\geq\sigma} \binom{b}{j_0} P(I)^{j_0} \ [1-P(I)]^{b-j_0} \\ &= (1-S(I)) + S(I) \end{aligned}$$

Eqn. (3.10) gives an upper bound on $S(I)$ resulting in a lower bound for $1-S(I)$:

$$1-S(I) \geq 1-e^{\left(\frac{-b\alpha_1^2}{2P(I)[1-P(I)]}\right)+\mathcal{O}\left(\frac{b\alpha_1^3}{[1-P(I)]^2}\right)} \tag{3.77}$$

with $\sigma = b[P(I)+\alpha_1]$.

Now, we concentrate on the second part

$$\sum_{j_0<\sigma} \binom{b}{j_0} P(I)^{j_0} \sum_{1\leq i\leq |I|} \sum_{j<\sigma} \binom{b-j_0}{j} Q_i(I)^j \ [1-P(I)-Q_i(I)]^{b-j_0-j} \ ,$$

or stated otherwise

$$\sum_{1\leq i\leq |I|} \sum_{j_0<\sigma} \binom{b}{j_0} P(I)^{j_0} \sum_{j<\sigma} \binom{b-j_0}{j} Q_i(I)^j \ [1-P(I)-Q_i(I)]^{b-j_0-j} \ . \tag{3.78}$$

In this expression, we first consider the rightmost sum

$$\sum_{j<\sigma} \binom{b-j_0}{j} Q_i(I)^j \ [1-P(I)-Q_i(I)]^{b-j_0-j} \ .$$

We can rewrite this to

$$\sum_{j\leq\sigma-1} \binom{b-j_0}{j} Q_i(I)^j \ [1-P(I)-Q_i(I)]^{b-j_0-j} \ , \tag{3.79}$$

and with Chernoff techniques, an upper bound for (3.79) can be found:

$$(3.79) \ \leq \ \sum_j x^{-\sigma+1} \binom{b-j_0}{j} [xQ_i(I)]^j \ [1-P(I)-Q_i(I)]^{b-j_0-j} \ , \tag{3.80}$$

with $x \leq 1$. This upper bound can be computed, based on the Binomial Theorem, with $x \leq 1$:

$$\begin{aligned} & x^{-\sigma+1} \sum_j \binom{b-j_0}{j} [xQ_i(I)]^j \ [1-P(I)-Q_i(I)]^{b-j_0-j} \\ = \ & x^{-\sigma+1} \ [xQ_i(I)+1-P(I)-Q_i(I)]^{b-j_0} \\ = \ & x^{-\sigma+1} \ [1-P(I)+(x-1)Q_i(I)]^{b-j_0} \ . \end{aligned}$$

We now plug in this expression in (3.78) and consider

$$\sum_{j_0<\sigma} \binom{b}{j_0} [xP(I)]^{j_0}\ x^{-\sigma+1}\ [1-P(I)+(x-1)Q_i(I)]^{b-j_0}\ .$$

This can be rewritten to

$$\sum_{j_0\leq\sigma-1} \binom{b}{j_0} x^{-\sigma+1}\ [1-P(I)+(x-1)Q_i(I)]^{b-j_0}\ , \tag{3.81}$$

and with the same Chernoff technique an upper bound for (3.81) can be computed:

$$(3.81) \ \leq\ x^{-\sigma+1} \sum_{j_0} y^{-\sigma+1} \binom{b}{j_0} [xP(I)]^{j_0}\ [1-P(I)+(x-1)Q_i(I)]^{b-j_0}\ ,$$

with $y \leq 1$. By using the Binomial Theorem, this upper bound is

$$\begin{aligned} & x^{-\sigma+1}\ y^{-\sigma+1} \sum_{j_0} \binom{b}{j_0} [xyP(I)]^{j_0}\ [1-P(I)+(x-1)Q_i(I)]^{b-j_0} \\ =\ & x^{-\sigma+1}\ y^{-\sigma+1}\ [xyP(I)+1-P(I)+(x-1)Q_i(I)]^b \\ =\ & x^{-\sigma+1}\ y^{-\sigma+1}\ [1+(xy-1)P(I)+(x-1)Q_i(I)]^b \end{aligned} \tag{3.82}$$

with $y \leq 1$.

Plugging this in (3.78) we find that

$$(3.78) \leq \sum_{1\leq i\leq I} x^{-\sigma+1}\ y^{-\sigma+1}\ [1+(xy-1)P(I)+(x-1)Q_i(I)]^b\ ,$$

for $x \leq 1$ and $y \leq 1$. If we set $y = 1$, we find

$$(3.78) \leq \sum_{1\leq i\leq I} x^{-\sigma+1}\ [1+(x-1)[P(I)+Q_i(I)]]^b\ .$$

We do can substitute $y = 1$ because the Chernoff bound is correct for $y \leq 1$. If we set y equal to 1, we find *an* upper bound. This may not be the best bound we can find, but it is a useful one.

Inspired by Subsubsection (3.4.2) where we computed an upper bound for Y, we can do here the same. For $\sigma = b\,[[P(I)+Q_i(I)]-\beta_i]+1$ we have that

$$x^{-\sigma+1}\ [1+(x-1)[P(I)+Q_i(I)]]^b\ \leq\ e^{\frac{-b\beta_i^2}{2[P(I)+Q_i(I)][1-P(I)-Q_i(I)]}+\mathcal{O}\left(\frac{b\beta_i^3}{[P(I)+Q_i(I)]^2}\right)}\ . \tag{3.83}$$

Combining (3.77) and (3.83) together in (3.75) leads to a lower bound on $F(I)$:

$$\begin{aligned} F(I) \geq 1 &- e^{-b\alpha_1^2/(2P(I)[1-P(I)])+\mathcal{O}(b\alpha_1^3[1-P(I)]^{-2})} \\ &- \sum_{1\leq i\leq|I|} e^{-b\beta_i^2/(2[P(I)+Q_i(I)][1-P(I)-Q_i(I)])+\mathcal{O}(b\beta_i^3[P(I)-Q_i(I)]^{-2})}\ , \end{aligned} \tag{3.84}$$

where $P(I_i) = P(I)+Q_i(I)$ and where α_1 and β_i are related to σ by $\sigma = b[P(I)+\alpha_1]$ and $\sigma = b[P(I)+Q_i(I)-\beta_i]+1$ when α_1 and all the βs are positive. For those cases where α_1 or some β is too large for the big $\mathcal{O}$ term to be small, a correct bound can still be obtained by replacing the too-large parameter with a smaller value. As long as α_1 and none of the βs are near 0, the value of F is near 1. □

3.5.3 Summary

Upper Bound for the Apriori Failure Probability

When $Q(I) + P(I) \leq \sigma/b$, it can be shown that

$$F(I) \leq e^{-b\ell\theta\alpha_3^2/(2\{Q(I)+P(I)+(\ell-1)P(I)-\ell[Q(I)+P(I)]^2\})} \tag{3.85}$$

with $\alpha_3 = \sigma/b - [P(I) + Q(I]$; θ is used to represent a function that approaches 1 in the limit. In this case, $F(I)$ goes rapidly to 0.

When $P(I) \geq (\sigma - 1)/b$, $F(I)$ goes rapidly to 0 because $F(I) \leq 1 - S(I)$. In particular, we can find

$$F(I) \leq e^{-b\alpha_2^2/\{2P(I)[1-P(I)]\}+\mathcal{O}(\alpha_2^3 bP(I)^{-2})} \tag{3.86}$$

when $\alpha_2 = P(I) - (\sigma - 1)/b$.

Lower Bound for the Apriori Failure Probability

To obtain a lower bound on $F(I)$, we use inclusion-exclusion arguments leading to

$$\begin{aligned} F(I) \quad \geq \quad & 1 - e^{-b\alpha_1^2/\{2P(I)[1-P(I)]\}+\mathcal{O}(b\alpha_1^3[1-P(I)]^{-2})} \\ & - \sum_{1\leq i\leq |I|} e^{-b\beta_i^2/\{2[P(I_i)][1-P(I_i)]\}+\mathcal{O}(b\beta_i^3[P(I_i)]^{-2})} \end{aligned} \tag{3.87}$$

where $P(I_i) = P(I) + Q_i(I)$ and where α_1 and β_i related to σ by $\sigma = b[P(I) + \alpha_1]$ and $\sigma = b[P(I) + Q_i(I) - \beta_i] + 1$ when α_1 and all the βs are positive.

3.5.4 Interpretation

We now interpret the theoretical results from the previous sections: we visualize the knowledge from the general success probability $S(I)$ and the failure probability $F(I)$ for the Apriori Algorithm on a probability line from 0 to 1. The diagram in Figure 3.4 illustrates the behavior of Apriori with regard to a single set, I, where the associated test sets are $I_1, \ldots, I_{|I|}$. We know that these test sets are all frequent, because of the candidacy definition of Apriori. Set I has probability $P(I)$. It is obvious that $P(I) \leq P(I_i)$, for each $1 \leq i \leq |I|$. Each test set I_i has some larger (or equal) probability than set I because each transaction that contains all the items of I also contains all the items of each test set I_i. Therefore, we do not have to take care of all individual probabilities for the test sets separately, but we focus on the *smallest* one: $P(I_{dominant})$. If $P(I) \leq P(I_{dominant})$, this relationship is also true for all other test sets. This test set $I_{dominant}$ is therefore called the *best* test set.

It is clear that if $P(I) > P(I_{dominant})$, set I can never be a candidate. Therefore, in the region to the right of $P(I_{dominant})$), the candidacy probability $C(I)$ equals zero, implying that the success and failure probabilities $S(I)$ and $F(I)$ in that region also are equal to zero. From Theorem 3.4.1, it is clear that $S(I)$ is approximately equal to zero in the middle region in between $P(I)$ and $P(I_{dominant})$, but also in the region to the right of $P(I_{dominant})$, as was already noticed. Theorem 3.4.2 states that $S(I)$ is approximately 1 in the only region that is

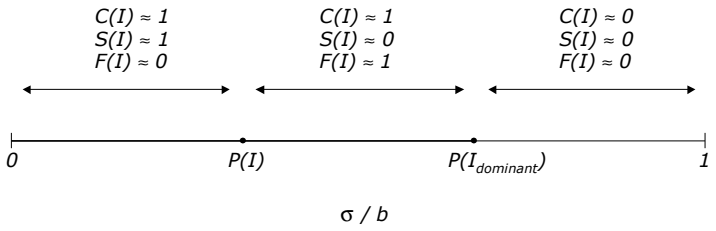


Figure 3.4: Whether set I is frequent (a success), or a candidate which is infrequent (a failure), or not even a candidate, is determined with high probability by where the threshold ratio σ/b falls in relation to the probability of set I and the probability of I's most important test set $I_{dominant}$

left, the region to the left of $P(I)$. Theorems 3.5.1 and 3.5.2 describe the behavior for $F(I)$ in the left-most region, while Theorem 3.5.3 describes the behavior of $F(I)$ in the middle region. Even more, now that it is clear what $F(I)$ and $S(I)$ are like in the left-most and the middle region, we can derive what the behavior of $C(I)$ is in both regions.

To summarize:

- If the fractional threshold σ/b is less than $P(I)$, then set I is always a success $[S(I) \approx 1]$. In this case, the failure probability is (almost) zero $[F(I) \approx 0]$.

- If σ/b is above $P(I_{dominant})$, where $I_{dominant}$ is the *best* test set, i.e., the one with the smallest probability, then there is almost no chance that I is even a candidate $[C(I) \approx 0]$, so both the success and failure probabilities are (nearly) zero $[S(I) \approx 0,\ F(I) \approx 0]$.

- If σ/b is between $P(I)$ and $P(I_{dominant})$, the probability that I is a candidate is almost one $[C(I) \approx 1]$, but the probability that I is a success is almost zero $[S(I) \approx 0]$, so the probability that I is a failure is almost one $[F(I) \approx 1]$.

Nearly all the failures for the Apriori Algorithm, i.e. counting sets which are not frequent, come from those sets where the set's probability $P(I)$ is below the threshold σ/b but where the probability of the *best* test set $P(I_{dominant})$is above the threshold. The other test sets have some effect on the probability, but their effect is insignificant unless their probability of purchase is near that of $I_{dominant}$, and even then their effect is not very important. The *best* test set has the main importance for the performance of the Apriori Algorithm.

3.6 The Failure Probability for the Other Algorithms

In this section, we consider the failure probability for the remaining algorithms. For AIS, Eclat and FP-growth, the relevant test sets for set I are still the sets I_i with one item less. For FCA, the relevant test sets are all the subsets of I that have size n, where n is the last level where the regular Apriori Algorithm was used.

3.6.1 The AIS Algorithm

For AIS, a set I is a candidate if (1) I is a 1-extension of a frequent set of size one less, and (2) I occurs in a transaction. For a simple analysis, we take the viewpoint that a set I is a candidate if it satisfies the first condition. We briefly indicate how to do the more complex analysis where both conditions must be true for I to be a candidate.

With the simple approach, set I will be a *candidate* exactly when the condition

$$j_0 + j_1 \geq \sigma \text{ or } j_0 + j_2 \geq \sigma \text{ or } \ldots \text{ or } j_0 + j_{|I|} \geq \sigma \tag{3.88}$$

is true. It is a *failure* when (3.88) is true with $j_0 < \sigma$. Thus, the failure probability for AIS is given by (3.35) except that the condition on the summation is

$$j_0 < \sigma \text{ and } (j_0 + j_1 \geq \sigma \text{ or } j_0 + j_2 \geq \sigma \text{ or } \ldots \text{ or } j_0 + j_{|I|} \geq \sigma) \ . \tag{3.89}$$

This gives us

$$F(I) = \sum_{\substack{(j_0<\sigma) \\ \text{and} \\ \begin{pmatrix} j_1 \geq \sigma - j_0, \text{ or} \\ j_2 \geq \sigma - j_0, \text{ or} \\ \cdots \\ j_{|I|} \geq \sigma - j_0 \end{pmatrix}}} \binom{b}{j_0, j_1, \ldots, j_{|I|}, b - j_0 - j_1 - \cdots - j_{|I|}} \times [P(I)]^{j_0} \left[\prod_{1 \leq i \leq |I|} Q_i(I)^{j_i} \right] \times \left[1 - P(I) - \sum_{1 \leq i \leq |I|} Q_i(I) \right]^{b - j_0 - \sum_{1 \leq i \leq |I|} j_i} .$$

Inclusion-exclusion type reasoning can be used on the logical-or operation: the sum is equivalent to the sum of a collection of subsums, some occurring with a negative sign. The i-th subsum of the first group has the condition

$$j_0 < \sigma \text{ and } j_0 + j_i \ \geq \ \sigma \ . \tag{3.90}$$

In the subsum, each index not mentioned in condition (3.90) must be summed out by summing over all possible values of the index. There is a second group of subsums that must be subtracted with the condition

$$j_0 < \sigma \text{ and } j_0 + j_{i_1} \geq \sigma \text{ and } j_0 + j_{i_2} \ \geq \ \sigma \ ,$$

because we have double counted the situations where two j_i's satisfy (3.88). These groups continue until we run out of corrections. From the detailed analysis that we did for the Apriori Algorithm, we can approximate all these subsums.

Because a set I of size n is a candidate for AIS if there exists a frequent $(n-1)$-sized subset of I, the only subsum in this collection of inclusion-exclusion subsums that is important is the one associated with the weakest of the test sets, the test set with the *highest* probability, as long as the largest $Q_i(I)$ is not too close to the others. Thus, when we have a unique

worst test set (no near ties), the failure probability for the AIS Algorithm is bounded by eqs. (3.85), (3.86), and (3.87) with $Q(I)$ being the largest of the $Q_i(I)$ and $\ell = 1$.

The performance of AIS is determined by the *worst* test set. If I_{worst} is the $(m-1)$-sized subset of a candidate set I of size m with largest probability of being purchased, then I is a candidate with probability near 1 when the probability of buying all the items of I_m is significantly above σ/b, and it is near 0 when the probability of buying the items is significantly below σ/b. The other test sets have only a slight effect on the probability that set I will be a candidate. When several sets tie for worst, the bound on the failure probability is worse than that given by eqs. (3.85), (3.86), and (3.72) but only by a factor that is no larger than the number of ties. Whether $F(I)$ is near 0 or 1 still depends on whether or not σ/b is between $P(I)$ and $P(I_{worst})$. Figure 3.4 is again applicable, but with $I_{dominant} = I_{worst}$.

If we want to analyze the effect of requiring a candidate to have an associated transaction, we need to reduce the failure probability from the previous calculation by removing the cases where set I does not occur in any transaction. Usually, the correction is not that important.

3.6.2 The Fast Completion Apriori Algorithm

The test sets for a set I are its subsets consisting of n elements, where n is the last level where the regular Apriori Algorithm is used. Unlike the previous cases, we now have *overlapping* test set ears. The presence of overlapping test set ears reduces the effectiveness compared to the no-overlap case but we will show that the performance of the algorithm on set I is determined primarily by the *best* of I's test sets. Of course, this best test set comes from the level where Apriori stopped (several levels back), so it usually is a much worse test set than the one Apriori would use. This is the only significant weakness of the Fast Completion Apriori Algorithm with regard to generating candidates: the test sets used come from several levels back and thus are likely to have much higher probabilities than test sets from one level back would have.

Rather than do a detailed calculation of the effect of the overlap, we will use some simple ideas to show that eqs. (3.85), (3.86), and (3.72) still bound the failure probability with $Q(I)$ being the $Q_i(I)$ for the best test set and $\ell = 1$, as long as there is no tie for the best Q.

The presence of overlapping ears reduces the effectiveness of the test sets compared to the no-overlap case. However, even in the presence of overlapping ears the collection of test sets is at least as effective as the best of the test sets in the collection of test sets. When ℓ (the number of ties for best) is one, the bounds are all in terms of the Q for the best test set. Thus, they still apply even when there are overlaps. When there is a tie for best, a more complex analysis is needed to obtain the best possible bounds, but the previous bounds still apply if one uses a modified ℓ, where the modified ℓ is somewhere between 1 and the original ℓ. Eq. (3.72) requires a modification so that the sum is over all test sets.

3.6.3 The Eclat and FP-growth Algorithm

In Section 2.4, we showed that for Eclat and FP-growth, a set I is a candidate only if *two* particular subsets of I are frequent. These two test sets are the one obtained by omitting the last item from I — this test set is called the *father* test set — and the one obtained by

omitting the next-to-last item from I — this test set is called the *special-uncle* test set—, when assuming a certain order on the items. Both the father and the special-uncle must be frequent for I to be a candidate. It is important to realize that both test sets are dependent of the assumed *order* on the items. We will discuss this later in this subsection.

Since both test sets must be frequent for a set I to be a candidate, for our probability model in which all items are independent, the *best* of those two test sets has the main effect on whether I is a candidate. Thus, the failure probability is bounded by eqs. (3.85), (3.86) and (3.72) with $Q(I)$ equal to the smaller of the $Q_i(I)$ for the father and the $Q_i(I)$ for the special-uncle. When these two Qs are different, $\ell = 1$; when they are equal, $\ell = 2$.

As already mentioned, which two Qs control the failure probability depends on the *ordering* that is used. It is clear that we should use an ordering that leads to the two best test sets being the father and the special-uncle. If the order is dynamic MFF, the father test set will be the worst test set and the special uncle test set will be the second worst. If the order is static MFF, there will be a strong tendency for the father test set to be the worst one and for the special-uncle test set to be the second-worst one. If this happens, the *second-worst* test set has the main effect on the probability of candidacy, but the actual situation will depend on what correlations are present in the data. If the order is dynamic LFF, the father test set will be the *best* test set and has the main effect. If the order is static LFF, the father test set will likely be the best test set. If the items are chosen in a random lexicographical order, both the parent and the special uncle will each be randomly-chosen test sets. Each pair of possible Qs is thus equally likely to be chosen and the *best* of the two randomly-chosen test sets will have the main effect. In rare cases, correlations in the data can lead to other than the usual results.

Once we have determined which test set has the main effect, the situation is the same as for the previous algorithms, as illustrated in Figure 3.4; the set I is a candidate with probability near 1 when the probability of buying all the items of the important test set is significantly above σ/b and it is near 0 when the probability of buying is significantly below σ/b. Set I is a failure when the threshold σ/b falls between the probability of I and the probability of the important test set.

3.7 Discussion

The algorithms considered in this chapter count every frequent set plus those infrequent sets that pass the candidacy test. If N is the number of items, then the number of frequent sets of size n is bounded by $\binom{N}{n}$, and the number of candidates of size $n+1$ is never more than $\binom{N}{n+1}$, and it is always at least the number of frequent sets of size $n+1$ [28]. This implies that the amount of work done by the best of the algorithms we consider is never more than a factor of N less than that of the worst algorithm (provided there is at least one frequent set). Since N is usually large, this linear dependence still leaves a lot of room for variation in the amount of work the various algorithms do.

The algorithms discussed in this chapter all have a candidacy test that considers a set I when only some of I's subsets, the test sets, are frequent. The various algorithms differ in which subsets they consider for the candidacy test. It is clear that an algorithm that considers

all subsets that are missing one item (such as Apriori) will sometimes need to consider fewer candidates than those that have a weaker candidacy test. We will illustrate this with an example of anti-correlated data.

Example 3.7.1. *Suppose items have the following frequency ordering: $a < b < c$. Let $a =$ hot dog, $b =$ Pepsi Cola and $c =$ Coca Cola, and assume that we have the following doubleton frequency ordering $bc < ab < ac$. For thresholds in the range $bc < \sigma < ab$, Eclat counts the set abc but Apriori does not.*

It is clear that Apriori does less work compared to Eclat, but it is less obvious how significant these differences are.

We compare the number of candidates generated by Apriori, various versions of Eclat, and AIS on some hypothetical datasets. We choose these datasets to be easy to describe, but at the same time, each dataset is selected to illustrate a property that is likely to arise in some real datasets. We use the probabilistic analysis from the previous sections to show that the significance in difference between the various algorithms depends both on the *algorithm* and on *properties of the dataset* being processed (and in the case of Eclat-like algorithms, the *ordering assumption*). The conclusions from this section are in agreement with previous experimental results [15, 41, 57]. Some of these conclusions could be obtained by considering particular fixed datasets but our probabilistic analysis shows that these conclusions are quite general. If we have a dataset with randomly filled transactions, the probability analysis shows that the expected behavior of each algorithm occurs with high probability, unless the parameters are near the boundary of a region.

For an infrequent set, a bad test set is a test set that is frequent; a good test set is one that is infrequent. For two algorithms to have a difference in performance (with regard to candidate generation) on a particular set I, the better algorithm must use at least one good test set for I while the worse algorithm must use no good test sets. Since the test sets for any form of Eclat are a subset of those of Apriori and a superset of those for AIS, the performance of an Eclat algorithm is intermediate between those of Apriori and of AIS. To keep the discussion simple, we ignore the condition in AIS that to be a candidate, a set, I, must occur in at least one transaction. For each type of dataset considered below, this restriction is insignificant for most datasets within the type.

Uniform Random Data

For uniform data, we assume that the various test sets of the same size all have similar probabilities. In general, we can say that for suitable uniform data, all the algorithms (other than FCA) have essentially the same performance in terms of the number of candidates generated.

To illustrate this claim, we focus on Apriori and AIS. For suitable uniform random data, each item has a fixed probability, p, of being in a transaction. For any set I all the test sets of the same size have similar probabilities. When set I is infrequent, for I to be a candidate for AIS and not for Apriori, we must have the best test set be infrequent and the worst test set be frequent. This can happen only when the relative threshold, σ/b, is close to the expected support of I's test sets. The standard deviation of their supports is proportional to $b^{-1/2}$. The standard deviation of the spread between the support of the best and worst test

set is proportional to $(|I| - 1)b^{-1/2}$. When b is large, for most thresholds σ, σ/b will not be close enough to the support of I's test sets for there to be any significant difference in the performance of Apriori and AIS. Still, there are small ranges for the threshold where Apriori performs much better than AIS.

Independent Random Data

We call data suitably independent if it is the case that the singleton support of the item missing from a test set is a good predictor of the support of the test set. If the data is non-uniform but suitably independent, then Apriori has essentially the same performance as LFF Eclat; Random Eclat is worse; MFF Eclat is worse yet, and AIS is worst of all.

These claims for non-uniform data depend on the data having many sets where the support of the best test set is much less than that of the second-best test set, and the support of the second-worst test set is much less than that of the worst test set. This spread in probabilities in turn makes it likely that for many of the sets the relative threshold will be between the probability of the test sets that are most important to the algorithms with the better performance and the probability of the test sets that are most important for the rest of the algorithms. These conclusions are in agreement with previous experimental results [15, 41, 16, 57]. Some of these conclusions could be obtained by considering particular fixed datasets but our probabilistic analysis shows that these conclusions are quite general.

Data with A Single Maximal Frequent Set

We now consider a very simple interpretation of what is meant by peaky data in the introduction of the shopping model (Section 3.2). For an example of data with a single maximal frequent set, consider a dataset with items 1 to n where sets with any combination of items 1 to m are frequent (2^m frequent sets) while all other sets are infrequent. The database is thus characterized by one single maximal frequent set of size m. Now consider sets that contain exactly one item in the range $m + 1$ to n with the rest of the items in the range 1 to m ($(n - m)2^m$ sets). Such a set has just one bad test set, the test set missing the item in range $m + 1$ to n. Any algorithm that uses this bad test set (AIS) will have each of these sets as candidates and these sets will be failures. On the other hand, any frequent set algorithm with at least two test sets will reject these $(n - m)2^m$ sets. All algorithms will have the 2^m frequent sets as candidates. Thus, for such data, the number of candidates for AIS is larger than the number of candidates for the other algorithms by a factor of $n - m + 1$.

Data with Overlapping Maximal Frequent Sets

We now consider another, more complex interpretation of what is meant by peaky data in the introduction of the shopping model (Section 3.2). For an example of data with overlapping maximal frequent sets, consider a dataset with items 1 to n where the frequent sets come from two overlapping groups of items from two overlapping maximal frequent sets. The first group consists of items 1 to h. The second group consists of items from from 1 to g and from $h + 1$ to m (where $g \leq h$). There are 2^h frequent sets where all the items come from the first group, 2^{m-h+g} frequent sets where all the items come from the second group, and 2^g frequent sets that have been counted twice (because all their items come from the overlap of the two groups). This is a total of $2^h + 2^{m-h+g} - 2^g$ frequent sets. All other sets are infrequent.

Now consider a set I, where all items but two come from 1 to g (the overlap region), one comes from $g+1$ to h (non-overlap part of group 1), and one comes from $h+1$ to m (non-overlap part of group 2). There are $(h-g)(m-h)2^g$ such sets. Each such set has two bad test sets, the one where all items come from group 1 and the one where all items come from group 2. The set I is a candidate for those algorithms where these two bad test sets are the only test sets considered, and it is not a candidate for those algorithms that consider some other test sets. Now suppose the items in the range 1 to g are even more frequent than the remaining items. Then Most-Frequent-First Eclat will pick these two bad test sets. Apriori and more fortunate versions of Eclat will have at least one good test set.

If we have a pair of overlapping maximal frequent sets where the items in the overlapping region (1 to g) have higher singleton frequencies that the other items, then Most-Frequent-First Eclat is worse than Apriori and also worse than most other versions of Eclat, but no parameter values make this effect large. When the dataset has many highly overlapping maximal frequent sets, however, the effect can be large.

Anti-correlation

When the data has strong anti-correlations, Apriori is the best, followed by LFF Eclat, and then by the remaining algorithms in the same order as before, w.r.t. amounts of candidates generated.

For an example of anti-correlated data consider a dataset where any frequent set has at most one of items 1 through h. This will be the case when the h values are associated with disjoint regions of a range variable. Also suppose that the frequency status of the set does not depend on which of these values is in the set. Now consider a set I with two of these items (thus I is infrequent) where either test set obtained by omitting one of these two special items is frequent. Those are the only two bad test sets of I. The other $|I|-2$ test sets are good. In this case, I is not a candidate for Apriori or for those versions of Eclat that have an ordering that leads to at least one good test set; I is a candidate for AIS. Let's count how bad this situation can be for various versions of Eclat. Consider a base set J that (1) has none of the items 1 through h, (2) is frequent, and (3) where J remains frequent when adding any single item in the range 1 to h. For each J this gives $h+1$ frequent sets: J and the h extensions to J obtained by adding an item in the range 1 to h. The set I, built from J, has two items in the range 1 to h. For each J this gives $h(h-1)/2$ infrequent sets. Suppose the singleton frequencies of the items in J are higher than the singleton frequencies of the items 1 to h. Then the two candidates generated by MFF Eclat are obtained by omitting one of the items in the range 1 to h. MFF has two bad test sets. LFF has at least one good test set (when J has at least one item). On the other hand, suppose the singleton frequencies of the items in J are lower than the singleton frequencies of the items 1 to h. Now LFF has two bad test sets. Thus, for each J, there are $h(h-1)/2$ infrequent sets that are candidates (bad cases) for the unfortunate version of Eclat and $h+1$ frequent sets (good cases). When h is large, the ratio of bad cases to good cases is large. Apriori has no problem with these cases.

While a realistic dataset won't normally have a single range variable with a really large number of values, a dataset may well have a number of range variables that together have a large number of values. In such cases, some versions of Eclat do much worse than Apriori.

Remarks

The least-frequent-first orderings are best for reducing candidate generation, but other orderings also have their own advantages. For example, most-frequent-first orderings lead to early detection of large sets [15].

The Apriori Algorithm, before testing a set, needs to know the support of each test set obtained by omitting one item. Such an algorithm must consider the support of each frequent set. The various Eclat Algorithms need the support of only two test sets. Thus, it is possible for them to determine the support of a set, I, after examining $\mathcal{O}(|I|^2)$ smaller sets. The Eclat algorithm, as originally stated, does not gain any advantage from this fact. However, if one adds tests to omit testing of small sets that are subsets of larger sets that are already known to be frequent, such a modified Eclat Algorithm no longer needs to test every frequent set. A modified Eclat Algorithm has the potential of counting the support of many fewer sets than Apriori does. Such a technique was introduced by Bayardo in his Max-Miner Algorithm [15].

Conclusion

For any dataset the number of candidates generated by Apriori will be less than or equal to the number generated by any version of Eclat. As long as certain sets occur at least once, the number of candidates generated by any version of Eclat will be less than or equal to the number of candidates generated by AIS. Our analysis shows that for uniform random datasets these differences usually are not significant, but they are often important for other datasets.

If some items are more likely to occur in sets than other items, any version of Eclat will be significantly better than AIS. When large sets of the same size have relative frequencies in the same order as the singleton support of their items (independence being an extreme case of this), LFF Eclat generates significantly less candidates than MFF Eclat. On such datasets lexicographical Eclat has intermediate performance. For such datasets Apriori has less candidates than LFF Eclat, but the difference is not significant. Many real datasets have properties that lead to these differences in performance. For some datasets with range variables MFF Eclat will be significantly worse than Apriori, while for some other datasets with range variables LFF Eclat will do significantly worse than Apriori.

Some of these conclusions could be obtained by considering particular fixed datasets. Our probabilistic analysis shows that the conclusions are quite general. Most of our conclusions depend on the relative support of the test sets derived from a given set. Obviously, in a fixed dataset the least frequent test set is least frequent. Our probabilistic analysis shows that even when the sets are arriving from a random process, the test set whose support is smallest is almost for sure to be the one with the lowest support for any dataset generated by the process. This is the case even for a set that has many test sets. Eclat uses only two test sets while Apriori has as many test sets as a set has items. None-the-less, when Eclat chooses the two test sets that are probabilistically expected to be best for its two test sets, the expected number of candidates it generates is only insignificantly larger than the number Apriori generates.

4

Maximal Frequent Set Mining Algorithms

Possible difficulties with frequent set mining can arise when mining very large databases. The size of the data can be massive, containing millions of transactions. This can make the support counting a tough problem, because of the exponential sized search space. Even with a user-defined support threshold σ, the output can still be large and the process to find all frequent sets very hard. Indeed, if σ is set too low or when the data are highly correlated, the process of mining can result in an immense amount of frequent sets. Even the most efficient mining algorithms cannot cope with this combinatorial blow-up. To overcome this problem, *condensed representations* can be used. Instead of trying to find *all* frequent sets, we can search for a *specific* kind of frequent sets that still covers enough[1] information about all frequent sets. In this chapter, we focus on a very basic type of condensed representation: the *maximal* frequent sets.

More specifically, we analyze algorithms that, under the right circumstances, permit efficient mining for *large* frequent sets. We introduce the concept of *jumping* to find large frequent sets and develop a family of level-by-level bottom-up jumping algorithms that are studied using a simple probability model[2]. The analysis clarifies why the jumping idea sometimes works well, and which properties the data needs to have for this to be the case. The link with Max-Miner [15] arises in a natural way and the analysis makes clear the role and importance of each major idea used in this algorithm [8].

[1] We assume that it is *enough* when, based on the condensed representation, *all* frequent sets can be reconstructed without their corresponding frequencies. In this sense, our interpretation of condensed representations differs from others in data mining literature, where it is necessary that all frequent sets with their corresponding frequencies can be reconstructed, for example closed sets or non-derivable sets.

[2] The topic presented in this chapter is joint work with Dirk Van Gucht and Paul W. Purdom from Indiana University in Bloomington, USA.

4.1 Introduction

In Definition 1.1.13, the *Monotonicity Property* of the support function was introduced. This property states that, if sets $J \subseteq I$, then $support(J) \geq support(I)$. Therefore, if I contains z items and

(1) if I is frequent, then all of its $2^z - 1$ strict subsets are also frequent,

and, on the other hand,

(2) if I is infrequent, then all of its $2^{m-z} - 1$ strict supersets are also infrequent.

These two consequences are at the core of frequent set mining algorithms because of their potential to significantly reduce the exponential-sized search space of the sets.

In their search for *maximal frequent sets*, algorithms such as Max-Miner [15], MAFIA [19], certain versions of Eclat [57] and FP-growth [35], and GenMax [32] make clever use of the monotonicity property, by avoiding outputting *every* frequent set.

Definition 4.1.1. *Assume we have a database $\mathcal{D}$, based on items from $\mathcal{I}$. A set $I \subseteq \mathcal{I}$ is a* maximal frequent set *if I has* no *superset that is frequent.*

Based on monotonicity, it is clear that any subset of a maximal frequent set is automatically frequent. This makes that these maximal frequent set mining algorithms are very well suited to run on data that is not purely random or uniform, but in which a few combinations of items occur more than others.

Inspired by the candidate-based frequent set mining algorithms from the previous chapters, our goal is to develop a level-wise bottom-up candidate-based algorithm that *jumps* to the maximal frequent sets, or as close as possible to these sets, and this as early as possible in the processing. By jumping, we thus can identify *large* frequent sets, with size *close* to the size of the maximal frequent sets in $\mathcal{D}$, early. More concrete, we can specify in general what a jump is for a level-wise bottom-up frequent set mining algorithm.

Definition 4.1.2. *Given a database $\mathcal{D}$ and a level-wise, bottom-up frequent set mining algorithm η. After processing level ℓ and discovering a collection of frequent sets of size ℓ, the algorithm η performs a* jump *of size $y > 1$ if it, somehow, decides to continue with level $\ell + y := x$, and ignores the intermediate levels. Based on frequent sets of size ℓ, η constructs candidates of size x and continues on level x.*

Maximal Frequent Set Jumping is the attempt to perform jumps early in the processing, in order to discover the maximal frequent sets. It is a lookahead pruning strategy, that tries to discover large frequent sets in an early phase and exploits this information, based on monotonicity, to save time. Of course, when a large frequent set is found, the challenge is to *quickly avoid* processing its subsets. This is in fact a special case of the so-called *subsumption problem.*

Definition 4.1.3. *Given two collections (sets of sets), $\mathcal{R}$ and $\mathcal{S}$. The* subsumption problem *is to determine in $\mathcal{S}$ each set S that is a subset of some set R in $\mathcal{R}$. In the special case that $\mathcal{R}$ and $\mathcal{S}$ are the same, it is required that each selected S is a proper subset of some R.*

This is exactly the type of test needed to make use of the information obtained from maximal frequent set jumping; $\mathcal{R}$ is the set of already found frequent sets and $\mathcal{S}$ is the set of candidate

sets that are being considered for additional processing. For each set S from $\mathcal{S}$ we have to test if it is frequent by checking if it is contained in some set R from $\mathcal{R}$. If $\mathcal{R}$ consists of one set and $\mathcal{S}$ of a lot of sets, or, alternatively, if $\mathcal{R}$ consists of a lot of sets and $\mathcal{S}$ of a single set, each set in the large set has to be matched with the single set in the other set. When $\mathcal{R}$ consists of a lot of sets and $\mathcal{S}$ too, there exist straightforward algorithms that solve the subsumption problem in time $\mathcal{O}(|\mathcal{R}||\mathcal{S}|)$. More information about the subsumption problem can be found in [27]. Fortunately, more efficient ways to solve the subsumption problem in mining for maximal frequent sets exist; for example, the techniques used in GenMax [32]. However, for our maximal frequent set jumping analysis, the subsumption problem is a hard problem to cope with.

This chapter uses a simple probabilistic analysis to show that if you want to mine for maximal frequent sets by jumping, and you want to use only simple ideas and avoid subsumption testing except for those cases where efficient algorithms are known, you are more or less forced into an algorithm that is very close to Max-Miner [15]. We also show that if any key idea is omitted from Max-Miner, its performance will be much poorer for certain datasets that can be found in the real world.

4.2 Perfect Jumps

4.2.1 Introduction to Perfect Jumps

To efficiently mine for maximal frequent sets, we already suggested to jump. Inspired by the candidate-based frequent set mining algorithms in general, we are looking for a *candidate-based, level-wise, bottom-up* algorithm, that we can adapt to jump. After a jump to a high level, we hope that the algorithm discovers some large sets that are frequent and we have to make sure that the algorithm makes good use of the information found. This leaves us with *two* important questions that we have to solve: first, once we have jumped, how to use the information about the large frequent sets, and second, how to do the jumps.

To answer the *first* question, we can use *monotonicity*, as already discussed in Section 4.1. If the algorithm jumps from level ℓ to level x, with of course $x > \ell$, and it finds some frequent sets on level x, then the *only* candidates that it needs to consider on level $\ell + 1$ are those sets that are not subsets of the frequent sets on level x. Otherwise, the algorithm already knows that the candidate is frequent, based on monotonicity.

To answer the *second* question, we now introduce the concept of a *perfect jump*. This simple idea is easy to incorporate into an algorithm and helps us to understand the conditions that have to be fulfilled before a jump is possible. Before we can formulate the definition of a perfect jump, it is important to recall that we discuss level-wise, bottom-up algorithms that all follow a candidate generation and testing strategy. The most fundamental part of such an algorithm is the *candidate-generation module*, that generates candidates based on a certain collection of frequent sets. More concrete, at a certain point in the processing, the algorithm has discovered a collection of frequent sets of a certain size, say ℓ. The candidate-generation module, which is algorithm dependent, is able to generate, based on these sets, candidates of size $\ell + y$, with $y \geq 1$. For example, Apriori generates candidates for level $\ell + 1$, based on all frequent sets found on level ℓ. Eclat on the other hand works depth-first and uses a

candidate-generation procedure based on some selection of frequent sets of size ℓ, to generate candidates of size $\ell+1$. Max-Miner on his turn generates candidates of size $\ell+y$ with $y>1$. With this algorithm-dependent candidate-generation module in mind, we are now ready to define the concept of a *perfect jump*.

Definition 4.2.1. *Given a dataset $\mathcal{D}$ and a level-wise, bottom-up frequent set mining algorithm η with a certain candidate-generation module. A* perfect jump *of η is a jump from level ℓ to level x, with $x>\ell$, such that* every *candidate set on level x, based on the candidate-generation module and frequent sets of size ℓ, is frequent.*

Example 4.2.1. *Assume that we have the following simple database $\mathcal{D}$, based on 8 items, and support threshold $\sigma=3$.*

$\mathcal{D}=$

TID	*a*	*b*	*c*	*d*	*p*	*q*	*r*	*s*
1	*1*	*1*	*1*	*1*	*0*	*1*	*0*	*0*
2	*1*	*1*	*1*	*1*	*0*	*0*	*1*	*0*
3	*1*	*1*	*1*	*1*	*0*	*0*	*0*	*0*
4	*0*	*0*	*0*	*0*	*1*	*1*	*1*	*1*
5	*1*	*0*	*0*	*0*	*1*	*1*	*1*	*1*
6	*0*	*0*	*0*	*0*	*1*	*1*	*1*	*1*
7	*1*	*0*	*0*	*0*	*0*	*1*	*0*	*0*
8	*1*	*0*	*0*	*0*	*0*	*0*	*1*	*0*

We will first give an overview of jumping from every level to every other possible level, when we follow the Apriori candidate-generation module, as described in Algorithm 1 - lines 13–19. Second, we will give the details for the case that we ran Apriori up to level 2. From level 2, we then decide to jump, based on the Apriori candidate-generation module. We will show that jumps to levels 3 and 5 are not perfect, but a jump to level 4 is.

Overview

1	$\rightarrow$	2	*not perfect*
1	$\rightarrow$	3	*not perfect*
1	$\rightarrow$	4	*not perfect*
1	$\rightarrow$	5	*not perfect*
2	$\rightarrow$	3	*not perfect*
2	$\rightarrow$	4	*perfect*
2	$\rightarrow$	5	*not perfect*
3	$\rightarrow$	4	*not perfect*
3	$\rightarrow$	5	*stop (no candidates of size 5)*
4	$\rightarrow$	5	*stop (no candidates of size 5*

Detailed computations, when jumping from level 2

Level 1:
W.r.t. the singletons, we have the following results: $support(a)=6$, $support(b)=3$, $support(c)=3$, $support(d)=3$, $support(p)=3$, $support(q)=5$, $support(r)=5$ and $support(s)=3$, illustrating that all items in the database are frequent.

Level 2:
Based on the frequent sets of size 1*, Apriori generates candidates of size* 2*. This candidate-generating step consists of a* join *step and a* prune *step (see Algorithm 1 - lines 13–19), yielding* $\binom{8}{2} = 28$ *candidate sets of size* 2*, from which only* 14 *sets turn out to be frequent:* $support(ab) = 3$*,* $support(ac) = 3$*,* $support(ad) = 3$*,* $support(aq) = 3$*,* $support(ar) = 3$*,* $support(bc) = 3$*,* $support(bd) = 3$*,* $support(cd) = 3$*,* $support(pq) = 3$*,* $support(pr) = 3$*,* $support(ps) = 3$*,* $support(qr) = 3$*,* $support(qs) = 3$ *and* $support(rs) = 3$*.*

Jump to Level 3*:*
Assume that the algorithm jumped from Level 2 *to Level* 3*, instead of to Level* 4*. This means that Apriori continues as usual. The join and prune step in the candidate-generating module generates the following candidates: abc, abd, acd, aqr, bcd, pqr, pqs, prs and qrs, from which aqr would turn out not to be frequent in* $\mathcal{D}$*. This jump from Level* 2 *to Level* 3 *therefore is* not *a perfect jump.*

Jump to Level 4*:*
Assume that the algorithm now decides to jump to Level 4*. Therefore, it uses the frequent sets of size* 2*, to generate candidate sets of size* 4*, by joining and pruning as in the traditional Apriori algorithm. This means that we start from a prefix of size* 1*, and combine this prefix with items that follow in the lexicographical order. Based on this principle, we generate sets abcd, abcq, abcr, abdq, abdr, abqr, acdq, acdr, adqr, pqrs in the join step, leading to only two candidate sets abcd and pqrs after the prune step. When we now count the occurrences of these candidates in* $\mathcal{D}$*, it turns out that both sets are frequent:* $support(abcd) = 3$ *and* $support(pqrs) = 3$*. By jumping to Level* 4*, we thus performed a* perfect jump*.*

Jump to Level 5*:*
Assume that the algorithm jumped from Level 2 *to Level* 5*, instead of to Level* 4*. The candidate generation step would have joined and pruned sets of size* 5*, ending with a single candidate set of size* 5*, abcdq. Counting the support of this single candidate set of size* 5 *in* $\mathcal{D}$ *gives us that abcdq is not frequent. Again, this jump is* not *a perfect jump.*

4.2.2 The (Truncated) Single Maximal Frequent Set Assumption

We can now determine *when* a perfect jump is possible. Therefore, we assume the realistic but simple probability distribution for the data, defined in Section 4.4, that will be used in the rest of this chapter. We will first define the most ideal situation to work with: the *single maximal frequent set assumption.* Based on this idea, we will slightly weaken this assumption to the *truncated maximal frequent set assumption.*

Assumption 4.2.1. ***Single Maximal Frequent Set Assumption*** *We assume that we have* one *single maximal frequent set in database* $\mathcal{D}$*, which is the superset of all other smaller frequent sets in* $\mathcal{D}$*. There are no other frequent sets in the database than the subsets of the single maximal frequent set. This single maximal frequent set consists of a items. To simplify our vision, these a items are all items of one particular type in the data distribution.*

Example 4.2.2. *Consider a database* $\mathcal{D}$*, based on items* a, b, c, d, e, f, g *and* h*. We assume that any combination of items* a, b, c, d, e *is frequent, while all other sets are infrequent. Thus,* $\mathcal{D}$ *contains a single maximal frequent set, abcde. In this case,* $a = 5$*.*

The database in Example 4.2.1 does not follow the single maximal frequent set assumption, because there are two maximal frequent sets, abcd and pqrs.

We will now define a *more general* assumption, based on the following definitions of virtual frequent sets and virtual maximal frequent sets.

Definition 4.2.2. *A non-empty set I is a* virtual frequent set *if there exists a certain v, with $1 \leq v \leq |I|$, such that all subsets of I of size $|I|-v$ are frequent. We will call v the* truncation size *of I and we will call $|I|-v$ the* truncation level *of I.*

Corollary 4.2.1. *If I is a virtual frequent set with truncation size v, and truncation level $|I|-v$, then for each v', with $v < v' \leq |I|$, v' is also a truncation size and $|I|-v'$ is also a truncation level for I.*

Proof. I is a virtual frequent set with truncation size v. This means that all subsets of I of size $|I|-v$ are frequent. Based on monotonicity, all subsets of the frequent sets of size $|I|-v$ are also frequent. □

Corollary 4.2.2. *Each non-empty frequent set I is a virtual frequent set.*

Proof. Indeed, for a non-empty frequent set I, there exists a truncation size $v = 0$ and truncation level $|I|$ such that all subsets of I of size $|I|$, i.e. I itself, is frequent. □

Corollary 4.2.3. *Each candidate set I of size at least 2, generated by the Apriori Algorithm, is a virtual frequent set.*

Proof. Indeed, for each candidate set I of size at least 2, generated by the Apriori Algorithm, these exists a truncation size 1 and a truncation level $|I|-1$, such that all subsets of I of size $|I|-1$ are frequent. □

Definition 4.2.3. *Let I be a virtual frequent set. I is called a* virtual maximal frequent set*, if there exists no superset of I that is also a virtual frequent set.*

Assumption 4.2.2. ***Truncated Single Maximal Frequent Set Assumption*** *It not required that we have a single maximal frequent set of size a, but it is necessary that we have a single* virtual *maximal frequent set of size a, leading to $\binom{a}{a-v}$ real maximal frequent sets of size $a-v$ in the database. The virtual single maximal frequent set of size a is the mother of all maximal frequent sets of size $a-v$ in $\mathcal{D}$. All other frequent sets in $\mathcal{D}$ are subsets of these maximal frequent sets of size $a-v$. In this situation, we have a* truncated maximal frequent set*, based on a items, truncated down to truncation level $a-v$, with truncation size v.*

Remark that the Single Maximal Frequent Set Assumption 4.2.1 is a special case of the Truncated Single Maximal Frequent Set Assumption 4.2.2, with the virtual single maximal frequent set equal to the single maximal frequent set, and thus truncation size $v = 0$.

Example 4.2.3. *We consider a database $\mathcal{D}$, that contains a virtual single maximal frequent set, abcde. In reality, we have $\binom{5}{4}$ sets of size 4 that are the maximal frequent sets of size 4. We thus have a truncated maximal frequent set of size 5, truncated down to level 4, with truncation size 1. This can be the case in market basket analysis, when shoppers all want the same 5 items, but they are limited to budgets that can pay for at most 4 items. If the shoppers vary in which of the 4 (instead of 5) items they actually buy and each combination of items is omitted equally often, their shopping will result in a truncated maximal frequent set based on 5 items, truncated down to level 4.*

Theorem 4.2.1. *We assume that we are under the (Truncated) Single Maximal Frequent Set Assumption, where the (virtual) single maximal frequent set consists of a items. Suppose that on level w, the frequent sets that are found contain a distinct items. In this case, the number of frequent sets on level w is $\binom{a}{w}$. In this situation, it is always possible to make a perfect jump.*

Proof. Because of the (Truncated) Single Maximal Frequent Set Assumption, we know that all frequent singletons in the database are these a items. On level w, all frequent sets of size w are combinations of these a items. Because of monotonicity, we know that all $\binom{a}{w}$ subsets of size w based on a items, have to be frequent. It is clear that a perfect jump is possible. In the truncated case, this can be a jump to the truncated level, and in the general case, this jump can be to the single maximal frequent set. □

In general, when we are processing a dataset with a single maximal frequent set consisting of a items under the (Truncated) Single Maximal Frequent Set Assumption, and the number of frequent sets on a certain level w is not given by the formula specified in Theorem 4.2.1, there is no possibility that a perfect jump will occur.

Suppose we meet the condition where a perfect jump is possible under the (Truncated) Single Maximal Frequent Set Assumption, and we jump from level ℓ to level x, there are $\binom{a}{x}$ candidate sets on level x, according to Theorem 4.2.1. These sets are formed by taking each combination of x items from the a items that occur in the frequent sets on level ℓ.

4.2.3 Saving Work with Maximal Frequent Set Jumping

The idea behind maximal frequent set jumping is to save time by not processing intermediate levels, thanks to monotonicity. The perfect jump situation under the (Truncated) Maximal Frequent Set Assumption ensures that we can save time by not counting a considerable amount of sets.

Property 4.2.1. *We assume that we are under the (Truncated) Single Maximal Frequent Set Assumption, where the (virtual) single maximal frequent set consists of a items. If we make a perfect jump from level ℓ to level x, every candidate set between levels $\ell + 1$ and x is also frequent. Therefore, the algorithm does not have to explicitly determine the frequency status of the $\sum_{j=\ell+1}^{x-1} \binom{a}{j}$ sets that occur strictly between levels ℓ and x.*

Proof. Because we make a perfect jump from level ℓ to level x, all candidate sets on level x are frequent. Monotonicity gives that all subsets are frequent too. This means that we do not have to explicitly count all candidate sets from the levels in between ℓ and x. According to Theorem 4.2.1, on each level j we have $\binom{a}{j}$ sets, leading to

$$\sum_{j=\ell+1}^{x-1} \binom{a}{j}$$

sets that we do not have to count. □

The biggest savings come from doing a jump from a level ℓ, with ℓ small compared to a, to a level x, with x near a. Therefore, we will tacitly assume that jumps occur from small levels to high levels.

Property 4.2.2. *We assume that we are under the (Truncated) Single Maximal Frequent Set Assumption, where the (virtual) single maximal frequent set consists of a items. If we make a perfect jump from a level ℓ to a level x, we can save considerable time when ℓ is below $a/2 - \sqrt{a}/\sqrt{2}$.*

Proof. The work for each level j equals $\binom{a}{j}$. Because we are dealing with binomial coefficients, it is clear that to save time, we have to jump from a level ℓ that is below $a/2$. We now try to find out how far below $a/2$ we have to go to enjoy the benefits of jumping.

We first give an intuitive approach to find a solution. To visualize our problem, we could plot the total work for all levels by plotting the binomial coefficients. They will have a peak around $a/2$. When we assume that a is large enough to approximate this Binomial Distribution by a Normal Distribution with the same mean and standard deviation, it is clear that most of the work is done within the region of standard deviation around $a/2$, as stated by the confidence intervals for the Normal Distribution. To save work, we thus have to jump from a level ℓ that is below $a/2$ minus the standard deviation of the corresponding distribution.

We now want to have an idea what this *low* level ℓ is. Therefore, it is important to think about the total work a jumping algorithm has to do. In the jumping process from level ℓ to level x, the algorithm first has to reach level l. When it decides to jump to level x, it then also has to do all the work that comes starting with level x, until the end. Schematically, all the work that a jumping algorithm for sure has to do is

$$\sum_{i=0}^{\ell} \binom{a}{i} + \sum_{i=x}^{a} \binom{a}{i} \ .$$

To have a more concrete idea what this *low* level ℓ is, we now compute a Chernoff Bound for the work that we cannot avoid, when jumping from level ℓ: $\sum_{i=0}^{\ell} \binom{a}{i}$. We will show that this bound remains very small for a selection of values of ℓ and we will specify how small ℓ is supposed to be for this to be the case. Inspired by the analysis in Section 3.4.2, we start with

$$\begin{aligned}
\sum_{i=0}^{\ell} \binom{a}{i} &\leq \sum_{i=0}^{a} \binom{a}{i} z^{-\ell+i}, \quad z \leq 1 \\
&= z^{-\ell} \sum_{i=0}^{a} \binom{a}{i} z^{i}, \quad z \leq 1 \\
&= z^{-\ell} (1+z)^{a}, \quad z \leq 1 \ . \qquad (4.1)
\end{aligned}$$

The optimum for the right-hand side of (4.1) is reached either on the boundary $z = 1$ or in $z_* = \dfrac{\ell}{a-\ell}$. We can find this optimum z_* by taking the derivative w.r.t. z and setting it to zero.

$$\begin{aligned}
& \frac{\mathrm{d}}{\mathrm{d}z}\left(z^{-\ell} (1+z)^{a}\right) = 0 \\
\Leftrightarrow \quad & (-\ell)\, x^{-\ell-1} (1+z)^{a} + z^{-\ell}\, a\, (1+z)^{a-1} = 0 \\
\Leftrightarrow \quad & (-\ell)\, (1+z) + z\, a = 0 \\
\Leftrightarrow \quad & (a-\ell)\, z = \ell \\
\Leftrightarrow \quad & z = \frac{\ell}{a-\ell} \qquad (4.2)
\end{aligned}$$

The requirement $z_* \leq 1$ leads to the region

$$\begin{aligned} & \frac{\ell}{a-\ell} \leq 1 \\ \Leftrightarrow \quad & \ell \leq a-\ell \\ \Leftrightarrow \quad & 2\ell \leq a \\ \Leftrightarrow \quad & \ell \leq \frac{a}{2} \ . \end{aligned}$$

This is what we intuitively already were expecting. We now introduce the notation β, to express the distance between $a/2$ and ℓ:

$$\beta = \frac{a}{2} - \ell \quad \Leftrightarrow \quad \ell = \frac{a}{2} - \beta \ . \tag{4.3}$$

We now continue with Chernoff bound (4.1) and plug in the value for the optimum (4.2).

$$\begin{aligned} \sum_{i=0}^{\ell} \binom{a}{i} &\leq z^{-\ell}\,(1+z)^a, \quad z \leq 1 \\ &\leq \left(\frac{\ell}{a-\ell}\right)^{-\ell} \left(1+\frac{\ell}{a-\ell}\right)^a \\ &= a^a\ \ell^{-\ell}\,(a-\ell)^{\ell-a} \end{aligned}$$

We now replace ℓ by its expression in terms of β as expressed in (4.3).

$$\begin{aligned} \sum_{i=0}^{\ell} \binom{a}{i} &\leq a^a \left(\frac{a}{2}-\beta\right)^{\beta-\frac{a}{2}} \left(\frac{a}{2}+\beta\right)^{-\frac{a}{2}-\beta} \\ &= a^a \left(\frac{a}{2}(1-\frac{2\beta}{a})\right)^{\frac{2\beta-a}{2}} \left(\frac{a}{2}(1+\frac{2\beta}{a})\right)^{-(\frac{a+2\beta}{2})} \\ &= a^a\ \frac{a^{\frac{2\beta-a}{2}}}{2^{\frac{2\beta-a}{2}}} \left(1-\frac{2\beta}{a}\right)^{\frac{2\beta-a}{2}} \frac{a^{-\frac{a+2\beta}{2}}}{2^{-\frac{a+2\beta}{2}}} \left(1+\frac{2\beta}{a}\right)^{-\frac{a+2\beta}{2}} \\ &= 2^a \left(1-\frac{2\beta}{a}\right)^{\frac{2\beta-a}{2}} \left(1+\frac{2\beta}{a}\right)^{-\frac{a+2\beta}{2}} \end{aligned}$$

We now write

$$\sum_{i=0}^{\ell} \binom{a}{i} \leq e^Z \ ,$$

with

$$\begin{aligned} Z &= \ln\left(2^a \left(1-\frac{2\beta}{a}\right)^{\frac{2\beta-a}{2}} \left(1+\frac{2\beta}{a}\right)^{-\frac{a+2\beta}{2}}\right) \\ &= a\ln 2 + \frac{2\beta-a}{2} \ln\left(1-\frac{2\beta}{a}\right) - \frac{a+2\beta}{2} \ln\left(1+\frac{2\beta}{a}\right) \ . \end{aligned}$$

Inspired by the previous Chernoff bound analysis in Chapter 3, we now use the following approximation in the case that $|x| < 1$:

$$\ln(1+x) \leq x - \frac{x^2}{2} + \frac{x^3}{3} \ .$$

This gives us

$$\begin{aligned} Z &\leq a\ln 2 + \frac{2\beta - a}{a}\left(\frac{-2\beta}{a} - \frac{4\beta^2}{2a^2} - \frac{8\beta^3}{3a^3}\right) - \frac{a+2\beta}{a}\left(\frac{2\beta}{a} - \frac{4\beta^2}{2a^2} + \frac{8\beta^3}{3a^3}\right) \\ &= a\ln 2 - \frac{2\beta^2}{a} - \frac{16\beta^4}{3a^3} \\ &\leq a\ln 2 - \frac{2\beta^2}{a}\ , \end{aligned}$$

because a and β are positive. We thus have that

$$\begin{aligned} \sum_{i=0}^{\ell}\binom{a}{i} &\leq e^Z \\ &\leq e^{a\ln 2 - \frac{2\beta^2}{a}} \\ &= 2^a\, e^{\frac{-2\beta^2}{a}} \\ &= \frac{2^a}{e^{\frac{2\beta^2}{a}}}\ . \end{aligned} \tag{4.4}$$

In this expression (4.4), we can compute when $\frac{2\beta^2}{a}$ becomes large. This is the case if

$$\begin{aligned} 2\beta^2 &\gg a \\ \Leftrightarrow\quad \beta &\gg \sqrt{\frac{a}{2}}\ . \end{aligned}$$

If we now replace β by its expression in terms of a and ℓ, it is clear that

$$\begin{aligned} \frac{a}{2} - \ell &\gg \frac{\sqrt{a}}{\sqrt{2}} \\ \Leftrightarrow\quad \ell &\ll \frac{a}{2} - \frac{\sqrt{a}}{\sqrt{2}} \end{aligned} \tag{4.5}$$

If ℓ is smaller than the value expressed in (4.5), it is clear that (4.4) is very small.

Of course, in those cases where we jump from a level near $a/2 - \sqrt{a}/\sqrt{2}$, the total work may be so large that the savings are not useful.

Because of symmetry in the binomial coefficients, we can reuse the above result for the Chernoff Bound for $\sum_{i=0}^{\ell}\binom{a}{i}$ to prove that the *rest* of the work that we still have to do, when we jump from an early level ℓ to a certain level x, i.e. $\sum_{i=x}^{a}\binom{a}{i}$, will also be very small as long as x is considerably larger than $a/2$. Combining both results, we have that the *total* work in case of jumping from ℓ to x, i.e. $\sum_{i=0}^{\ell}\binom{a}{i} + \sum_{i=x}^{a}\binom{a}{i}$, is small, as we would like. □

4.3 The Perfect Jump Algorithm

Based on the concept of perfect jumps, we now define the *Perfect Jump Algorithm*. This level-wise bottom-up algorithm with built-in candidate-generation module is designed exclusively to bring out and analyze central issues related to maximal frequent set jumping. In

Algorithm 3 The Perfect Jump Algorithm

Require: Items $\mathcal{I} = \{i_1, i_2, \ldots, i_m\}$, database $\mathcal{D}$, user-defined support threshold σ

1: [Start] Set $\ell = 0$. If the number of transactions is greater than σ, then set the empty set to frequent, otherwise set the empty set to infrequent.
2: [Check if done] If there are no frequent sets on level ℓ then stop. Otherwise set $\ell = \ell + 1$.
3: [Process level ℓ] Determine the support of the sets of size ℓ that can be built from the frequent sets from level $\ell - 1$.
4: [Try to jump] Set a to the number of items that appear in the frequent sets on the current level. If the number of frequent sets on this level is strictly smaller than $\binom{a}{\ell}$ or $\ell \geq a/2$ then go to Step 2.
5: [Try jumping] Set $x = a - \ell$. For every set that can be formed by choosing x of the a items, determine whether or not the set is frequent. If every such set is frequent, then set $\ell = x + 1$ and go to Step 3.
6: [Continue] Go to Step 2.

particular, it is not proposed as an algorithm for implementation and experimental analysis. Therefore, we never implemented it, but only used it as a tool for reasoning about maximal frequent set jumping. A detailed description can be found in Algorithm 3. Intuitively, you can think about the algorithm as Apriori with a slight modification of performing jumps.

When the conditions for attempting to jump are met, as described by Theorem 4.2.1, the algorithm tries jumping to level $a - \ell$, because the number of candidate sets to be tested at that level, i.e. $\binom{a}{a-\ell}$, is the same as the number of frequent sets on level ℓ, i.e. $\binom{a}{\ell}$. This balances the work attempting jumps with the other work, thereby ensuring that the algorithm is never slower than the Apriori Algorithm by more than a factor of two. In favorable cases, i.e., when a high jump occurs (see Property 4.2.2), the Perfect Jump Algorithm is *much faster*.

For the rest of this chapter, we will work with the Perfect Jump Algorithm, based on the pseudo code defined in Algorithm 3. Conceptually, it is possible to work with other specifications. For example, there are *variations* of the algorithm possible, that jump to *some higher* level than $a - \ell$. Attempting jumps to level $a - \ell + j$ $(1 \leq j \leq \ell)$ ensures that the work done attempting jumps is small compared to the non-jump work already done by the algorithm. Attempting jumps to higher levels decreases the chances of a successful jump, but for many datasets, including data following the simple probability model that we have, consisting of two transaction types and two item types, the added speed from trying high jumps is more important than the decreased chance of success. To illustrate, the algorithm could try to jump to level a and determine whether or not the single set of all a items is frequent. Remark that if you want to work with an alternate definition, the pseudo code of the algorithm has to be rewritten.

4.4 The Probability Model

To gain insight in the performance of maximal frequent set jumping algorithms, we will use a very simple probabilistic data model. In general, the model assumes that all the transactions are *random* and that both the transactions and the items are divided into several *categories*.

Items / Transactions	Type 1	Type 2	$\cdots$	Type j
t_1	p_{11}	p_{12}	$\cdots$	p_{1j}
t_2	p_{21}	p_{22}	$\cdots$	p_{2j}
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
t_i	p_{i1}	p_{i2}	$\cdots$	p_{ij}

Figure 4.1: A General Probability Model for Maximal Frequent Set Jumping

Transactions	Item Type 1	Item Type 2
t_1	p_{11}	p_{12}
t_2	p_{21}	p_{22}

Figure 4.2: The Two Transaction Types Two Item Types Probability Model

Definition 4.4.1. *The general probability model that is used to study maximal frequent set jumping is characterised by*

- t_i *transactions of type* i*,*
- n_j *items of type* j*,*
- *probability* p_{ij} *that a transaction of type* i *contains an item of type* j*.*

A schematical overview of this model can be found in Figure 4.1. By convention, we number the transactions and items such that p_{11} *is the* largest *of the probabilities.*

This model is so *general* that it can represent any particular dataset by having a type for each transaction and a type for each item, resulting in each p_{ij} being either 0 or 1. The probabilistic aspects of the model are become more and more interesting if the amount of transaction types and item types are large. However, for the sake of simplicity and presentation, we concentrate on the special case with just *two* types of transactions and *two* types of items, as illustrated in Figure 4.2, because it already brings out the most important aspects of maximal frequent set jumping algorithm performance.

Based on this model with only two types of transactions and two types of items, we can explicitly write down some important characteristics.

Property 4.4.1. *The* probability *that a transaction of type* i *(*$i = 1, 2$*) contains* m_1 *different items of type 1 and* m_2 *different items of type 2, regardless of what other items are in the transaction, is denoted* $P_i(m_1, m_2)$ *and is, based on the simple probability model, computed as*

$$P_i(m_1, m_2) \;=\; {p_{i1}}^{m_1}\, {p_{i2}}^{m_2} \;. \tag{4.6}$$

This probability is the same regardless of the particular items, as long as m_1 and m_2 are fixed.

Property 4.4.2. *The* expected support *of a set that contains m_1 particular items of the first kind and m_2 particular items of the second kind is*

$$t_1\ P_1(m_1, m_2)\ +\ t_2\ P_2(m_1, m_2)\ , \tag{4.7}$$

and can be computed, using the simple probability model and Property 4.4.1, as

$$t_1\ {p_{11}}^{m_1}\ {p_{12}}^{m_2}\ +\ t_2\ {p_{21}}^{m_1}\ {p_{22}}^{m_2}\ . \tag{4.8}$$

4.5 Analysis of the Perfect Jump Algorithm

In Subsection 4.2.2, we assumed that the dataset that we consider has a (truncated) *single* maximal frequent set, where the (virtual) single maximal frequent set consists of a items. To simplify the mathematical analysis, we will make this assumption more concrete, by assuming that this single maximal frequent set is based on m_1 items of item type 1. We will show that this requirement is necessary to have a useful perfect jump, when we have a dataset that follows the two transaction types two item types probability model. Recall that p_{11} is the largest probability.

Property 4.5.1. *For the two transaction types two item types probability model, there is no (truncated) single maximal frequent set unless p_{11} is* close to one.

Proof. This is implied by Properties 4.4.1 and 4.4.2. Under the assumption that p_{11} is the largest probability, the support of a set that contains m_1 items of type 1 and m_2 items of type 2 is bounded from above by ${p_{11}}^{m_1+m_2}$. This quantity becomes rapidly small unless p_{11} is close to one. □

Inspired by Property 4.4.2, the (Truncated) Single Maximal Frequent Set based on m_1 items of type 1 Assumption can be mathematically formalized as follows.

Property 4.5.2. *For the two transaction types two item types probability model, to have a (truncated) single maximal frequent set that is only formed by m_1 items of the fixed type 1, at level $x = m_1 - \ell$, we need to have the following inequalities:*

$$t_1\ P_1(m_1 - \ell, 0)\ +\ t_2\ P_2(m_1 - \ell, 0)\ \geq\ \sigma\ , \tag{4.9}$$

$$t_1\ {p_{11}}^{m_1-\ell}\ +\ t_2\ {p_{21}}^{m_1-\ell}\ \geq\ \sigma\ . \tag{4.10}$$

To meet the condition for a perfect jump, the frequent sets on level ℓ must be those sets that contain ℓ items of type 1 and no items of type 2. Obeying (4.10) already ensures that every set with ℓ items of type 1 is frequent. To ensure that no other sets are frequent, we need the following conditions.

Property 4.5.3. *For the two transaction types two item types probability model, to have a (truncated) single maximal frequent set, that is only formed by m_1 items of the fixed type 1, at level $x = m_1 - \ell$ and to ensure that* no other *sets are frequent, except those sets that contain ℓ items of type 1, we need that for every r ($1 \leq r \leq \ell$),*

$$t_1\ P_1(\ell - r, r)\ +\ t_2\ P_2(\ell - r, r)\ <\ \sigma\ , \tag{4.11}$$

$$t_1\ {p_{11}}^{\ell-r}\ {p_{12}}^{r}\ +\ t_2\ {p_{21}}^{\ell-r}\ {p_{22}}^{r}\ <\ \sigma\ . \tag{4.12}$$

The first term on the left side of (4.12) is biggest at $r = 1$, since $p_{11} > p_{12}$. The second term is biggest at $r = 1$ when $p_{21} \geq p_{22}$ and at $r = \ell$ when $p_{21} \leq p_{22}$.

Situation 1	p_{21}, p_{22} zero
	p_{11}, p_{12} small p_{11} large, p_{12} small p_{11}, p_{12} large
Situation 2	$p_{11}, p_{12}, p_{21}, p_{22}$ small
Situation 3	p_{11} large, p_{12}, p_{21}, p_{22} small
Situation 4	p_{11}, p_{21} large, p_{12}, p_{22} small
Situation 5	p_{11}, p_{22} large, p_{12}, p_{21} small
Situation 6	p_{11}, p_{12} large, p_{21}, p_{22} small
Situation 7	p_{11}, p_{12}, p_{21} large and p_{22} small
Situation 8	$p_{11}, p_{12}, p_{21}, p_{22}$ large

Figure 4.3: Possible Interpretations for the Two Transaction Types Two Item Types Probability Model

4.5.1 Case Analysis

Assume that we have a dataset following the two transaction types two item types probability distribution, as introduced in Section 4.4. We now discuss some interesting situations for a selection of parameter settings, reflecting datasets that can occur in practice. Figure 4.3 groups all cases that have to be considered. The goal is to perform a case analysis of the conditions for a perfect jump to occur on level ℓ. Remark that the perfect jump algorithm, as it is defined in Algorithm 3, is designed to work perfectly on datasets that have a single (truncated) maximal frequent set. The results presented in this Subsection are based on Property 4.5.2 and Property 4.5.3.

As a **first**, introductory case, we assume that we are in Situation 1. In this case, we only have one type of transactions in the data distribution, reflecting in $p_{21} = 0$ and $p_{22} = 0$. We will show that this is a situation in which the (truncated) single maximal frequent set, based on m_1 items of type 1 Assumption can hold. To have a perfect jump we need to have

$$p_{11}{}^{m_1-\ell} \;\geq\; \frac{\sigma}{t_1} \;, \tag{4.13}$$

and

$$p_{12} \;<\; \frac{\sigma}{t_1 \; p_{11}{}^{\ell-1}} \;. \tag{4.14}$$

If we write $p_{11} = 1 - \varepsilon$, then the logarithm of (4.13) gives

$$\begin{aligned} & \ln\Big((1-\varepsilon)^{m_1-\ell}\Big) \;\geq\; \ln(\sigma/t_1) \\ \Leftrightarrow \quad & (m_1-\ell)\;\ln(1-\varepsilon) \;\geq\; \ln(\sigma/t_1) \;. \end{aligned}$$

Based on the approximation

$$\ln(1+x) \;=\; x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \cdots \;,$$

that converges when $-1 < x \leq +1$, we can find:

$$(m_1 - \ell)\ (-\varepsilon + \text{higher order terms})\ \geq\ \ln(\sigma/t_1)\ , \tag{4.15}$$

$$\Leftrightarrow\ \ \varepsilon\ \leq\ \frac{-\ln(\sigma/t_1)}{m_1 - \ell}\ \text{(neglecting higher order terms)}\ . \tag{4.16}$$

Essentially, to have a perfect jump from a low level, p_{11} must be close to 1 and p_{12} must be less than σ/t_1, as illustrated by (4.14). Indeed, for a particular dataset, (4.14) shows that the threshold σ must be large enough to filter out the noise from the items that are not part of the maximal frequent set, and (4.13) will determine whether there is any value of σ that will work. The value of ℓ only plays a small role in what is happening. If p_{11} is not extremely close to 1 (eq. (4.16)), we don't have a single maximal frequent set. If p_{11} is quite close to 1, then ${p_{11}}^{\ell-1}$ is close to 1 for small ℓ. Thus, if this noise, resulting from transactions that contain the maximal frequent set items and also occasional other items, causes trouble on level 1, it probably will cause trouble on the higher levels too.

As a **second** case, consider Situation 2, in which all probabilities are small. Here, it is not likely that our Assumption can hold. To have a (truncated) single maximal frequent set, only based on m_1 of item type 1, Property 4.5.2 states that

$$t_1\ {p_{11}}^{m_1-\ell}\ +\ t_2\ {p_{21}}^{m_1-\ell}\ \geq\ \sigma\ . \tag{4.17}$$

When p_{11} and p_{21} are small, (4.17) is not likely to be the case. We can conclude that we do not have a pronounced (truncated) single maximal frequent set, in this case.

For the **third** case, we are in Situation 3. In this case, we have two types of transactions in the data distribution, but the second type is characterised by p_{21} and p_{22} small: both are not equal to 0 but neither of them is close to 1. Even more, we have that p_{11} is large, but p_{12} is small. In this case, we may *not* be able to make perfect jumps for *small* values of ℓ. But, since in this case the second term in (4.12)

$$t_1\ {p_{11}}^{\ell-r}\ {p_{12}}^{r}\ +\ t_2\ {p_{21}}^{\ell-r}\ {p_{22}}^{r}\ <\ \sigma$$

decreases rapidly with ℓ and the second term in (4.10)

$$t_1\ {p_{11}}^{m_1-\ell}\ +\ t_2\ {p_{21}}^{m_1-\ell}\ \geq\ \sigma$$

only helps, we will be able to make a perfect jump with a *moderate* value of ℓ.

For a **fourth** case, we assume that we are in Situation 4. We have two types of transactions in the data distribution, and in this case, both p_{11} and p_{21} are large, while p_{12} and p_{22} are small. When this is the case, *both* types of transactions contribute to the (truncated) maximal frequent set of items of type 1. We therefore need both p_{12} and p_{22} to be small (close to 0) to have a perfect jump. Indeed, if we assume, for the sake of simplicity, that $p_{11} = p_{21} = 1$, and $p_{12} = p_{22}$, eqn. (4.12) gives

$$\begin{aligned} & t_1\ {p_{11}}^{\ell-r}\ {p_{12}}^{r}\ +\ t_2\ {p_{11}}^{\ell-r}\ {p_{12}}^{r}\ <\ \sigma \\ \Leftrightarrow\ & (t_1 + t_2)\ {p_{11}}^{\ell-r}\ {p_{12}}^{r}\ <\ \sigma \\ \Leftrightarrow\ & {p_{12}}^{r}\ <\ \frac{\sigma}{t_1 + t_2}\ . \end{aligned}$$

Consider Situation 5 as a **fifth** case. The distribution is characterised by p_{11} and p_{22} near 1, but p_{12} and p_{21} near 0. In this situation, the Perfect Jump Algorithm is not jumping, because a perfect jump is not possible. This happens because the model is generating data with *two* maximal frequent sets, and the algorithm is designed to run on data with a single maximal frequent set. The perfect jump algorithm needs an *additional idea* before it can perform well on such data. This implies that we have to relax the single maximal frequent set assumption. More information can be found in Subsection 4.6.2. Intuitively, we want to *separate* the dataset, based on the two maximal frequent sets, and apply *locally* the (truncated) single maximal frequent set, based on m_1 items of type 1 assumption. This idea is also exploited by Max-Miner, as we will show in the following sections in this chapter.

If we consider Situation 6 as a **sixth** case, we are in a situation in which the (truncated) single maximal frequent set is based on two item types. When considering the two item types as one, we can reuse the results for one item type.

For a **seventh** case, assume that we are in Situations 7 or 8. We now have three or even four of the probabilities in the distribution table that are large. This means that we may have a (truncated) single maximal frequent set based on the items of *both* item types. When the amount of transactions of the first type, t_1, is small, and the amount of transactions of the second type, t_2, is large, and p_{22} is large, we may have a single maximal frequent set based *just* on items of type 2.

4.5.2 Summary

We can conclude that the Perfect Jump Algorithm will do well on data wit a *single* maximal frequent set, as long as the subset of transactions that contain most of the maximal items are unlikely to contain any other items. In the two transactions type two shopper type probability model, this situation is reflected by high probability values for the maximal items, small probability values for all other items and no intermediate values. The Perfect Jump Algorithm will do well on this kind of data, even in the presence of more other types of transactions, just as long as these transaction types do not lead to other maximal frequent sets. However, when we have several items with intermediate probabilities, the possible maximal frequent set(s) are not that pronounced, and the performance of the Perfect Jump Algorithm is not good. This shows that, to cope with all situations that can occur in real-life data, we have to adapt the Perfect Jump Algorithm.

4.6 Adaptations of the Perfect Jump Algorithm

The results from the previous section confirm that the Perfect Jump Algorithm performs well on data that have a (truncated) single maximal frequent set. We now will try to improve the idea of maximal frequent set jumping, by modifying the Perfect Jump Algorithm. As a first modification, we will stop considering exact support counts and work with lower bounds. The second modification that we will introduce is an adaptation of the algorithm such that it can run on datasets with several maximal frequent sets. This adaptation is necessary, because the way the algorithm is designed now will never cause a jump to happen, as illustrated in the previous section. Both modifications of the Perfect Jump Algorithm are independent of each other and therefore can be combined.

Algorithm 4 The Perfect Jump Algorithm with Lower Bounds

Require: Items $\mathcal{I} = \{i_1, i_2, \ldots, i_m\}$, database $\mathcal{D}$, user-defined support threshold σ

1: [Start] Set $\ell = 0$. If the number of transactions is greater than σ, then set the empty set to frequent, otherwise set the empty set to infrequent.
2: [Check if done] If there are no frequent sets on level ℓ then stop. Otherwise set $\ell = \ell + 1$.
3: [Process level ℓ] Determine the support of the sets of size ℓ that can be built from the frequent sets from level $\ell - 1$.
4: [Try to jump] Set a to the number of items that appear in the frequent sets on the current level. If the number of frequent sets on this level is strictly smaller than $\binom{a}{\ell}$ or $\ell \geq a/2$ then go to Step 2.
5: [Try jumping] Set $x = a - \ell$. For every candidate set K of size x, that can be formed by choosing x of the a items, determine whether or not the set is frequent, by using the lower bound formula (4.18). This formula has to hold for every frequent set $I \subset K$ of size $\ell - 1$, and every frequent set of size ℓ, based on I and singleton items $\{j\}$ from $J = K \setminus I$. If every such set K is found frequent, because the lower bound on its support count exceeds σ, then count its actual support, set $\ell = x + 1$ and go to Step 3.
6: [Continue] Go to Step 2.

4.6.1 The Perfect Jump Algorithm with Lower Bounds

We will weaken Step 5 in Algorithm 3, such that this slightly modificated version of the Perfect Jump Algorithm still performs good under the (truncated) single maximal frequent set assumption. We therefore use the notion of a support lower bound, as introduced by Bayardo.

Theorem 4.6.1. ***Support Lower Bounding*** *[15] For disjoint sets I and J, we have that*

$$Support(I \cup J) \;\geq\; \sum_{j \in J} Support(I \cup \{j\}) - (|J| - 1) Support(I) \;. \tag{4.18}$$

In Step 5 in Algorithm 3, we now can replace the support counting procedure by the support lower bounding procedure. This is perfectly possible when we know the support counts of all frequent sets on levels $\ell - 1$ and ℓ. Instead of counting the actual support of the candidate sets of size x, this will give us a lower bound on the support of the candidate sets. In practice, we will use the inequality (4.18) presented in Theorem 4.6.1 with I a set on level $\ell - 1$ and J the set of items other than those in I that make up the potential maximal frequent set. Therefore, in the case of a (truncated) single maximal frequent set consisting of m_1 items of type 1, if we jump to level $x = m_1 - \ell$, we have that $|I| = \ell - 1$ and $|J| = m_1 - \ell - |I|$. The pseudo code of this adaptation of the Perfect Jump Algorithm, called the Perfect Jump Algorithm with Lower Bounds, can be found in Algorithm 4.

To show the effects of the Perfect Jump Algorithm with Lower Bounds, we will analyze its behavior when mining a dataset with a (truncated) single maximal frequent set.

Property 4.6.1. *To analyze the conditions for a successful perfect jump in Algorithm 4 with data based on the two transaction types two item types probability model and a (truncated) maximal frequent set based on m_1 items of type 1, we still need (4.14), but we need to replace (4.9) and (4.10) with*

$$
\begin{aligned}
& t_1\ [(m_1 - 2\ell + 1)P_1(\ell, 0) - (m_1 - 2\ell)P_1(\ell - 1, 0)] \\
& \qquad +\ t_2\ [(m_1 - 2\ell + 1)P_2(\ell, 0) - (m_1 - 2\ell)P_2(\ell - 1, 0)]\ \geq\ \sigma\ , \qquad (4.19) \\
\Leftrightarrow\ & t_1\ [(m_1 - 2l + 1){p_{11}}^{\ell} - (m_1 - 2\ell){p_{11}}^{\ell-1}] \\
& \qquad +\ t_2\ [(m_1 - 2\ell + 1){p_{21}}^{\ell} - (m_1 - 2\ell){p_{21}}^{\ell-1}]\ \geq\ \sigma\ , \\
\Leftrightarrow\ & t_1\ {p_{11}}^{\ell-1}\ [(m_1 - 2\ell + 1)p_{11} - (m_1 - 2\ell)] \\
& \qquad +\ t_2\ {p_{21}}^{\ell-1}\ [(m_1 - 2\ell + 1)p_{21} - (m_1 - 2\ell)]\ \geq\ \sigma\ . \qquad (4.20)
\end{aligned}
$$

Proof. When we assume that $|I| = \ell - 1$ and $|J| = m_1 - \ell - |I|$, it is clear that $m_1 - \ell = |J| + |I|$, $|J| = m_1 - \ell - \ell + 1 = m_1 - 2\ell + 1$ and therefore$|J| - 1 = m_1 - 2\ell$.

If we now rewrite $P_1(m_1 - \ell, 0)$, based on (4.18) and the fact that we do not focus on individual items but on item types, we find

$$
\begin{aligned}
P_1(m_1 - \ell, 0) &= |J| \cdot P_1(\ell - 1 + 1, 0) - (|J| - 1) \cdot P_1(\ell - 1, 0)\ , \\
&= (m_1 - 2\ell + 1)\ P_1(\ell, 0) - (m_1 - 2\ell)\ P_1(\ell - 1, 0)\ .
\end{aligned}
$$

Analogously, we find

$$
P_2(m_1 - \ell, 0)\ =\ (m_1 - 2\ell + 1)P_2(\ell, 0) - (m_1 - 2\ell)P_2(\ell - 1, 0)\ .
$$

When plugging in these results in (4.9), we find (4.19). By plugging in the corresponding probability formulas for P_1 and P_2, we can find (4.20). □

To simplify the analysis, we now assume that we only have one type of transactions.

Property 4.6.2. *Assume that we only have one type of transactions in the two transaction types two item types probability model, reflecting in $p_{21} = 0$. Write $p_{11} = 1 - \varepsilon$. We can find the following upper bound for ε:*

$$
\varepsilon\ \leq\ \frac{1 - \sigma/t_1}{m_1 - \ell}\ .
$$

Proof. Because p_{11} was supposed to be close to 1, we can write $p_{11} = 1 - \varepsilon$. We now replace p_{11} in (4.20) by its expression in ε, apply the Binomial Theorem (3.1) and carry out the calculations to first order in ε.

$$
\begin{aligned}
& (1 - \varepsilon)^{\ell-1}\ [(m_1 - 2\ell + 1)(1 - \varepsilon) - (m_1 - 2\ell)]\ \geq\ \frac{\sigma}{t_1} \\
\Leftrightarrow\ & (1 - \varepsilon)^{\ell-1}\ [1 - (m_1 - 2\ell + 1)\varepsilon]\ \geq\ \frac{\sigma}{t_1} \quad \text{(apply now the Binomial Theorem)} \\
\Leftrightarrow\ & (1 - (\ell - 1)\varepsilon)\ [1 - (m_1 - 2\ell + 1)\varepsilon]\ \geq\ \frac{\sigma}{t_1} \quad \text{(neglecting higher order terms)} \\
\Leftrightarrow\ & 1 - (m_1 - 2\ell + 1)\varepsilon - (\ell - 1)\varepsilon\ \geq\ \frac{\sigma}{t_1} \quad \text{(neglecting higher order terms)} \\
\Leftrightarrow\ & 1 - (m_1 - \ell)\varepsilon\ \geq\ \frac{\sigma}{t_1} \\
\Leftrightarrow\ & \varepsilon\ \leq\ \frac{1 - \sigma/t_1}{m_1 - \ell} \qquad (4.21)
\end{aligned}
$$

This leads to the postulated upper bound for ε. □

Motivation for Bounds instead of Counts

To reason about this upper bound for ε, we introduce the notion δ, that expresses the distance between σ/t_1 and 1: $\sigma/t_1 = 1 - \delta$. We will refer to the basic Perfect Jump Algorithm, introduced in Section 4.3, as the algorithm with *exact counts*, because it explicitly counts occurrences of candidates in the database in Step 5. According to both upper bounds for ε, ε_{EC} for the exact count model (cfr. eqn. (4.16))

$$\varepsilon_{EC} \leq \frac{-\ln(\sigma/t_1)}{m_1 - \ell} , \tag{4.22}$$

and ε_{LB} for the lower bound model (cfr. eqn. (4.21)),

$$\varepsilon_{LB} \leq \frac{1 - \sigma/t_1}{m_1 - \ell} , \tag{4.23}$$

it is clear that, when σ is almost as large as t_1, illustrating that δ is near 0, the Perfect Jump Algorithm needs p_{11} equally close to 1 in *both* cases - whether it uses estimates, i.e. lower bounds, or exact counts - in order for the jump to be successful. When σ is substantially less than t_1, for example by a few powers of ten, the version of the algorithm that uses exact counts will succeed in making perfect jumps with values of p_{11} that are much further from 1, compared to the lower bound version. This is easy to see, because $\ln(1 + x) \leq x$ for $x \geq 0$, leading to

$$\frac{-\ln(\sigma/t_1)}{m_1 - \ell} \geq \frac{1 - \sigma/t_1}{m_1 - \ell} ,$$

and therefore, $\epsilon_{LB} \leq \epsilon_{EC}$.

This gives the impression that the lower bound version is not really an improvement of the Perfect Jump Algorithm. This is not completely true, because when using lower bounds, it is usually appropriate to use a value of the level to jump to, x, that is *one smaller* than the value appropriate for exact counting. This is because the lower bound algorithm does not need to actually count the support of sets on level x, to know if they are frequent. Indeed, instead of counting, it uses lower bounds based on information from lower levels. Only once it has found that a certain candidate set is frequent, because the lower bound on its support exceeds σ, the algorithms counts the actual support, to be able to continue with candidate generation based on frequent sets for the next level. We can conclude that lower bounds are practical to use when searching for slightly shorter truncated maximal frequent sets.

Even more, if you would be interested in implementing the Perfect Jump Algorithm, it is not necessary to choose between the basic model with exact counting *or* the optimization with lower bounds. It is easy to modify the original Perfect Jump Algorithm to first try a lower bound on level $x = m_1 - \ell - 1$ and then, if this does not lead to a successful perfect jump, use exact counting on level $x = m_1 - \ell$. Of course, adding this slight complexity to the code has a payoff only for datasets with truncated maximal frequent sets.

4.6.2 The Perfect Jump Algorithm with Connected Components

We will now present an optimization of the Perfect Jump Algorithm, that can handle databases that are more general than the databases following the (truncated) single maximal

frequent set assumption. The general analysis of the Perfect Jump Algorithm illustrates that items with intermediate probability interfere with the performance of the algorithm. However, when this interference is coming from a *second* type of transactions, it can often be eliminated by breaking the data into disjoint components. Therefore, in this subsection, we introduce an adaptation of the Perfect Jump Algorithm that works with *connected components*. The basic idea behind the principle of breaking up the data is to make the Perfect Jump Algorithm *local*, thus able to cope with one maximal frequent set at a time.

Definition 4.6.1. *Given dataset $\mathcal{D}$. The* connectedness relation $\kappa(\mathcal{D})$ *is a relation that is defined over all frequent sets of $\mathcal{D}$. We say that frequent sets I and J are* connected *if*

$$|I| = |J| = |I \cap J| + 1 \ .$$

Example 4.6.1. *The join step in the generating part of the Apriori Algorithm (see Chapter 2, Algorithm 1 - lines 14–15) is a special situation of this definition. Two frequent sets X and Y of a particular size s are joined, if their $(s-1)$-sized prefix, based on some assumed order, is the same. We have that $|X| = |Y| = s$ and $|X \cap Y| = s - 1$. X and Y thus are* connected frequent sets*. They are both 1-extensions (Definition 2.1.1) of the same prefix and therefore differ in only one item.*

Property 4.6.3. *The resulting connectedness relation κ is* symmetric*, but* not transitive*.*

Proof. The symmetry of the relation is clear. To proof that the relation is not transitive, we refer to [57], in which this relation was previously used in some versions of the Eclat Algorithm. As a simple illustration of the non-transitivity of the connectedness relation, consider sets ab and bc, that are obviously connected, and sets bc and cd that are also connected. It is clear that transitivity does not hold: ab connected to bc and bc connected to cd does not mean that ab is connected with cd. □

The transitive closure of any symmetric relation is reflexive. Moreover, it is an equivalence relation. Therefore, we can divide the frequent sets into equivalence classes.

Definition 4.6.2. *Given a dataset $\mathcal{D}$ and a connectedness relation κ over all frequent sets. We can divide all frequent sets into* equivalence classes*, also called* connected components*, by putting into the same class or component any two frequent sets that are connected.*

We now have the following important property w.r.t. equivalence classes.

Property 4.6.4. *Given frequent set mining algorithm η, based on the Apriori candidate generation module. Sets from two different equivalence classes* never *interact when η runs.*

Proof. Let I and J be frequent sets at level ℓ and I and J are in different equivalence classes. This means that I and J are not connected. The join part in the Apriori generation step will never join I and J, because they do not have a prefix in common. □

Based on the above explanation, we can conclude that the Perfect Jump Algorithm remains correct if we partition the frequent sets on level ℓ into equivalence classes and process each class independently. By doing so, we introduce a recursive algorithm: the Perfect Jump Algorithm with Connected Components, with pseudo code in Algorithm 5. According to Definition 4.6.1, all frequent singletons are connected, and form 1 equivalence class. While on level 1, all frequent sets are in the same class, on higher levels there can be many classes.

Algorithm 5 The Perfect Jump Algorithm with Connected Components

Require: Items $\mathcal{I} = \{i_1, i_2, \ldots, i_m\}$, database $\mathcal{D}$, user-defined support threshold σ

1: [Start] Set $\ell = 0$. If the number of transactions is greater than σ, then set the empty set to frequent, otherwise set the empty set to infrequent.
2: [Check if done] If there are no frequent sets on level ℓ then stop. Otherwise set $\ell = \ell + 1$.
3: [Process level ℓ] Determine the support of the sets of size ℓ that can be built from the frequent sets from level $\ell - 1$. Split the selection of all frequent sets of size ℓ into connected components, by grouping all connected sets in a component.
4: [Split Algorithm over Connected Components] Process now all components individually. For component i, set a_i to the number of items that appear in the frequent sets of size ℓ of that connected component. Project $\mathcal{D}$ on $\mathcal{D}_i$, the dataset that only considers transactions that contain the items from the connected component.
5: [Try to jump] If the number of frequent sets on level ℓ in connected component i is strictly smaller than $\binom{a_i}{\ell}$, or $\ell \geq a_i/2$, then go to Step 2.
6: [Try jumping] For component i, set $x_i = a_i - \ell$. For every set that can be formed by choosing x_i of the a_i items, determine whether or not the set is frequent. If every such set is frequent, then set $\ell = x_i + 1$ and go to Step 3.
7: Go to Step 2.

It is important to prove that this adaptation of the Perfect Jump Algorithm is correct, i.e. that it will find all maximal frequent sets, and will never miss sets.

Theorem 4.6.2. *The Perfect Jump Algorithm with Connected Components always finds all frequent sets.*

Proof. This is easy to see, because the algorithm is based on the candidate generation part of Apriori, consisting of a join and a prune step (see Chapter 2 - Algorithm 1). On a certain level ℓ, Apriori generates candidates for the next level, based on all frequent sets of size ℓ. In this generation process, Apriori joins frequent sets of size ℓ, based on a common prefix of size $\ell - 1$, that then are pruned according to the monotonicity principle. This shows that two frequent sets I and J, if they are joined, have to be in the same connected component. Two frequent sets from two different connected components will never be joined. This shows that the join-step from Apriori forms the basis for connectedness. Therefore, the correctness of the recursive Perfect Jump Algorithm with Connected Components is the correctness of the Apriori Algorithm, since they generate the same candidates. □

Remark that, for datasets with a (truncated) single maximal frequent set, the Perfect Jump Algorithm with Connected Components equals the basic Perfect Jump Algorithm. In this case, there is only one connected component, reflecting the (truncated) single maximal frequent set.

For our random two transaction types two item types probability model, it is interesting to consider the conditions when the sets made up of items of type 1 are in a *different* equivalence class than those of type 2.

Property 4.6.5. *To have two classes on level ℓ, all those sets made up of $\ell - t$ items of type 1 and t items of type 2 with $1 \leq t \leq \ell - 1$ must be infrequent. The* expected number *of such*

sets is

$$t_1 \; P_1(\ell - t, t) \; + \; s_2 \; P_2(\ell - t, t) \; . \tag{4.24}$$

When this number is significantly below σ, the typical dataset from the model will have two *equivalence classes.*

The condition for each maximal frequent set to be in its own equivalence class is the same as before - when we only had one maximal set - and is expressed by (4.12), except that (4.12) permits $t = \ell$ while (4.24) does not allow that case. The $t = \ell$ case is associated with a maximal frequent set that will be in a separate component.

We see that by combining the idea of perfect jumps with partitions, we obtain a simple algorithm that is efficient at finding large frequent sets in some interesting datasets with several maximal frequent sets.

4.7 Other Important Design Principles for Maximal Frequent Set Mining

We will now focus on a selection of other design principles that will play an important role in maximal frequent set mining, but that are orthogonal to all ideas introduced in the previous sections, i.e. perfect jumps, lower bounds and connected components. The first important idea that we study is the ordering of items in the search space. As a second important concept we talk about subsumption. Both ideas are independent and therefore can be combined. An application of both ideas is found in Max-Miner, probably the best-known algorithm to find maximal frequent sets [15].

4.7.1 The Ordering Aspect in Maximal Frequent Set Mining

In the previous section 4.6, we introduced the Perfect Jump Algorithm with Connected Components, to solve the problem with interference that was coming from other types of transactions. Our solution was to break up the data into disjoint components. However, this solution still does *not* completely solve the major problem of perfect jumps: having *intermediate probabilities*. The Perfect Jump Algorithm With Connected Components turns out to be a fine algorithm, except for datasets with *several* items of intermediate probability.

We now discuss an important heuristic to overcome the existence of several items of intermediate probability. We suggest to partition the search space of sets, so that many of the partitions don't have sets that contain items of intermediate probability. On each level, we therefore will partition the candidates according to the already found frequent sets in the previous level, based on a certain *order*. The ordering assumption makes it able to avoid considering too many sets with intermediate probability and therefore improves the mining of maximal frequent sets.

The ordering idea can be found in the following Algorithm 6, that mines for maximal frequent sets.

Algorithm 6 The Ordering Idea in Maximal Frequent Set Mining

Require: Items $\mathcal{I} = \{i_1, i_2, \ldots, i_m\}$, database $\mathcal{D}$, user-defined support threshold σ
Ensure: $\mathcal{M}$:= Maximal Frequent Sets from $\mathcal{D}$ w.r.t. that particular threshold σ

```
1: pick a total ordering ρ on the items
2: M := ∅
3: C_1 := { {i} | i ∈ I}
4: k := 1
5: while C_k ≠ ∅ do
6:    partition C_k in prefix-blocks cell(I), ∀ possible (k−1)-sized prefixes I, based on ρ
7:    for all non-empty cell(I) do
8:       construct set Peak(I), which consists of all items that occur in cell(I)
9:       if Peak(I) is frequent then
10:         add Peak(I) to M
11:      else
12:         add frequent sets from cell(I) to M, but only those not subsumed in M
13:      end if
14:   end for
15:   generate C_{k+1} := {X | |X| = k+1 and ∄M ∈ M such that X ⊆ M and ∀Y ⊂ X :
      ∃M ∈ M such that Y ⊆ M}
16:   k++
17: end while
18: return M
```

Remark that in Subsection 4.6.2, we discussed a partitioning of the dataset based on connected components. In Algorithm 6, we in fact have a decomposition by means of cells. Both decomposition principles are based on two completely different ideas and therefore are orthogonal. The partitioning idea that is the basis for Algorithm 6 is an ordering on the items.

We will now step by step discuss Algorithm 6. At the start, the algorithm picks a total ordering ρ on all the items. It then begins a level-wise search for maximal frequent sets in the lexicographically ordered search space. In the end of processing level $\ell - 1$, it constructs candidates of size ℓ, grouped together in C_ℓ. Only sets of size ℓ that are not frequent, but that have all $(\ell - 1)$-sized subsets frequent are considered to be candidates. This candidacy test is clearly inspired by Apriori, but in the case of Algorithm 6, the amount of candidates is less compared to Apriori, because the algorithm probably already discovered some large frequent sets. It is not necessary to consider ℓ-sized subsets from sets in $\mathcal{M}$, because it is obvious that they are frequent. As a first step in processing level ℓ, C_ℓ is partitioned, based on prefix sets of size $(\ell - 1)$, according to ordering ρ. All candidate sets with the same prefix I_1 are grouped in the same cell, $cell(I_1)$. From the remaining candidates, Algorithm 6 finds all those that contain I_2 and puts them in $cell(I_2)$. It then determines $cell(I_3)$ etc. This is done for all lexicographically ordered possible prefixes. After this partitioning phase, the algorithm processes each cell as follows. Consider $cell(I)$. For each set in this cell, based on the lexicographical ordering, the algorithm constructs $Peak(I)$, which consists of all the items that occur in $cell(I)$. The algorithm then enters its maximal frequent set jumping phase. For each $peak(I)$ it determines whether or not it is frequent, either by explicit counting or by

using the lower bound formula (4.18), in which J is set to J_I, the set that consists of all items that occur in $cell(I)$, except those that are in I. If it is frequent, $Peak(I)$ is added to $\mathcal{M}$, otherwise the frequent sets in $cell(I)$ are added to $\mathcal{M}$. The algorithm then generates new candidates for the next level $\ell + 1$ and moves on.

Remark that, when determining the support of $Peak(I)$, Algorithm 6 can use explicit counting in the database. However, it is also possible to incorporate support lower bounding to improve the performance. The idea here is to obtain a lower bound on the support of a set by exploiting the available support information provided by its subsets.

If the initial ordering ρ picked by Algorithm 6 is a randomly selected ordering, then the frequent sets at level $\ell - 1$ will also occur in some random order. When the algorithm then enters its partitioning phase of the candidates at level ℓ, i.e. $\mathcal{C}_\ell$, it can be expected that for a frequent set I of size $\ell - 1$, the set J_I will have a mixture of low- to high-probability items. Even if J_I is large, the analysis strongly suggests that $Peak(I) = I \cup J_I$ will be infrequent and the algorithm will not be able to gain much from its maximal frequent set jumping abilities. Therefore, rather than picking a random ordering, the algorithm picks an ordering ρ based on the supports of the items that increases the effectiveness of pruning. More specifically, the ordering is determined by arranging the items in *increasing* order of their supports. By doing so, the algorithm forces the most frequent items to appear in the most candidates. This is simply because items with high support are more likely to be part of long frequent sets. Because of the way that the algorithm is constructed, items that appear last in the ordering, will appear in the most candidates. Therefore, item ordering is used to position the most frequent items last: items are ordered in increasing order of support. This ordering is much more likely to decrease the generation of many J_I's which have a mixture of low- to high probability items. Consequently, the algorithm considerably improves its opportunity to obtain significant performance improvement on account of maximal frequent set jumping.

4.7.2 Subsumption in Maximal Frequent Set Mining

The Max-Miner Algorithm [15] adds one more heuristic to Algorithm 6 to improve its performance: *subsumption*. When mining for maximal frequent sets, the idea behind subsumption is not to remember subsets of sets that are already found to be frequent, but only remember the largest sets. This idea is independent of all other ideas already mentioned in this chapter. By adding subsumption to the list of key ideas, we rediscovered all issues that the Max-Miner Algorithm covers: jumping, ordering and subsumption [15].

As you would expect from its name, Max-Miner is, inspired by the condensed representations[3] idea, designed to mine for *maximal* frequent sets [15]. The basic idea is that Max-Miner attempts to look ahead in order to quickly identify long frequent sets – this is the idea of *jumping*. By identifying such sets early on, it can prune all the subsets from consideration, based on monotonicity. Therefore, its output implicitly and concisely represents all frequent sets. Because of this, Max-Miner perfectly fits in our maximal frequent set jumping story.

[3]Recall that for this dissertation, it is sufficient to be able to reconstruct all frequent sets based on the condensed representation. It is not required that the corresponding frequencies are provided automatically with each frequent set.

Algorithm 7 The Max-Miner Algorithm

Require: Items $\mathcal{I} = \{i_1, i_2, \ldots, i_m\}$, database $\mathcal{D}$, user-defined support threshold σ
Ensure: $\mathcal{M}$:= Maximal Frequent Sets from $\mathcal{D}$ w.r.t. that particular threshold σ

1: pick a total ordering ρ on the items
2: $\mathcal{M} := \emptyset$
3: $C_1 := \{ \{i\} \mid i \in \mathcal{I}\}$
4: $k := 1$
5: **while** $C_k \neq \emptyset$ **do**
6: partition C_k in prefix-blocks $cell(I)$, $\forall$ possible $(k-1)$-sized prefixes I, based on ρ
7: **for all** $cell(I)$ **do**
8: construct set $Peak(I)$, which consists of all items that occur in $cell(I)$
9: **if** $Peak(I)$ is frequent **then**
10: add $Peak(I)$ to $\mathcal{M}$, while removing all subsumed sets from $\mathcal{M}$
11: **else**
12: add frequent sets from $cell(I)$ to $\mathcal{M}$, but only those not subsumed in $\mathcal{M}$
13: **end if**
14: **end for**
15: generate $C_{k+1} := \{X \mid |X| = k+1$ and $\nexists M \in \mathcal{M}$ such that $X \subseteq M$ and $\forall Y \subset X : \exists M \in \mathcal{M}$ such that $Y \subseteq M\}$
16: $k++$
17: **end while**
18: **return** $\mathcal{M}$

Max-Miner is a breadth-first, level-wise frequent set mining algorithm that works much like Apriori, except that it, thanks to its goal to find maximal frequent sets efficiently, also has maximal frequent set jumping capabilities. During its running, the algorithm maintains a set $\mathcal{M}$, which at the end will contain all maximal frequent sets. A rough outline of Max-Miner, only focusing on the basic ideas, can be found in Algorithm 7. It is the pseudo code for Algorithm 6, that partitions the search space using a certain *ordering*, in which one extra heuristic is added: subsumption. For the exact (implementation) details, we refer to [15]. Note that we do not have an explicit step for counting in Algorithm 7, because counting is implementation dependent. Several places are possible; for example, Bayardo counts after Step 5.

We now focus on the subsumption part in the algorithm. Max-Miner uses a subtle strategy to update $\mathcal{M}$. When it adds $Peak(I)$, it removes from $\mathcal{M}$ all the sets that are subsumed by it. And, when it considers adding the frequent sets in $cell(I)$, it only adds those that are not subsumed by an existing set in $\mathcal{M}$. Clearly, under this update strategy, there is the guarantee that at the end, $\mathcal{M}$ consist exactly of all maximal frequent sets. Obviously, this update strategy can be *costly* because each of the subsumption tests has to be done explicitly, and as far as we know, no general algorithm exists that can do this efficiently.

Regardless of which update strategy is used, in the cases where $Peak(I)$ is frequent and J_I is large, Max-Miner can be expected to gain substantial performance improvements, as illustrated by its experimental results [15].

Max-Miner searches for frequent sets in a breadth-first, level-wise, order. If it is changed to depth-first order, it is necessary to give up the full Apriori candidacy test. Doing so leads to Eclat-like versions of Max-Miner [19, 32, 35, 57]. However, with respect to the performance gain due to maximal frequent set jumping, these Eclat-like algorithms get essentially the same benefits compared to Max-Miner.

4.8 Conclusion and Future Work

When studying algorithms to process data in order to quickly find maximal frequent sets, it is necessary to consider the *variation of probability* of various items and the *correlations* in the data. If there are no items with a high support, at least among some subset of the transactions, then the data has no large maximal frequent set that catches the eye. The Apriori Algorithm is efficient at processing the low support items. If there are large maximal frequent sets in the data, several different solutions, introduced in this chapter, can be used to solve the problem.

If the data has some low support items, some high support items, no intermediate support items, and no important correlations, then the data may be characterized by *one* large maximal frequent set. The Perfect Jump Algorithm has good performance in this case.

If the data has some low support items, some high support items, no intermediate support items but there are important correlations, then there may be *several* large maximal frequent sets in the data. The Perfect Jump Algorithm needs to be augmented with a division of the data into independent components before it can handle such data efficiently. This is achieved by the Perfect Jump Algorithm with Connected Components.

These ideas still do not lead to an efficient algorithm if the dataset has several items with an intermediate probability. Many datasets have the property that the occurrence of items is close enough to independent, such that the relative probability of items on one level is similar to that on other levels. In such cases, jumps can be done relative to a set that is missing the items of intermediate probability. This is the essential reason why Max-Miner types of algorithms are efficient.

This line of reasoning makes it clear that *ordering items* from least-frequent to most-frequent is critical to the success of Max-Miner. If for example, Max-Miner had processed items in random order, it is unlikely that it would produce a set that had all of the highly frequent but none of the moderately frequent items. Likewise, it will not do well on some datasets where the correlation structure makes the support of items on one level greatly different from that on other levels. Similar conclusions can be drawn for the other maximal frequent set mining algorithms.

Remark that the candidate generation in the Perfect Jump Algorithm is based on the candidate generation method of Apriori. Our analysis is also applicable to any other candidate generation method; it is in fact independent of the candidate generation method.

Also remark that the difference between depth-first and breadth-first is not in conflict with all presented ideas. It is orthogonal to it.

To conclude, we can now focus on possible further work. The research presented in Chapter 3 considers a general, in-depth analysis of candidate-based frequent set mining algorithms. This chapter can be seen as a first part of a trilogy about frequent set mining algorithms. A logical step to more complex algorithms is performed in Chapter 4, in which we focus on a particular kind of frequent set mining algorithm: algorithms that mine for maximal frequent sets. This chapter therefore can be seen as the second part of the trilogy. A next step now could be to analyze algorithms that mine for closed sets. This future research can be seen as the third part, that concludes the trilogy.

5

Frequent Set Mining in Streams

We now change the focus from offline mining static databases to online mining streaming data. Data streams are massive, unbounded sequences of data elements, that are continuously generated at a rapid rate. The amount of information passing by is typically huge, and there is no possibility to store it. Therefore, streaming data can only be read once and if an item has passed, it can not be revisited.

In this chapter, we study the problem of finding frequent sets in a continuous stream of sets, called transactions. Most previous work on mining frequently occurring items or sets over streams requires an in advance fixed window length or decay factor, given by the user. Our work[1] is the first to change this viewpoint by defining a new measure for frequency, based on a flexible window length [4]. An algorithm, based on this new measure is derived, that maintains a summary consisting of crucial information about the history of the stream, that is dynamically adjusted each time a new transaction enters the stream. The information in the summary is exact information, as compared to the many approximations considered in related work, and it allows to produce the current frequency of a particular set immediately. Obviously, it is important that the space requirements of the algorithm remain small. An empirical and theoretical analysis is performed to illustrate that this is the case.

An introduction to the concepts and notations that are necessary for the rest of this chapter is given in Section 5.2. All the details about the new frequency measure can be found in Section 5.3. Section 5.4 considers the new algorithm and its properties and possible adaptations. In Section 5.5, a worst-case analysis for the size of the summary is performed, based on a special type of streams, the Farey streams. The experiments in Section 5.6 illustrate that the size of the summary in real-life situations remains very small.

[1] The material presented in this chapter is joint work with Toon Calders (Eindhoven University of Technology) and Bart Goethals (University of Antwerp).

5.1 Motivation

In data mining literature, a lot of algorithms are introduced to compute frequency counts exceeding a user-defined threshold over data streams. Because streaming data can only be scanned once, it is important to compute frequency counts in a single pass with some error guarantees. Two important algorithms that follow this procedure are Sticky Sampling and Lossy Counting [47]. Both algorithms are designed to run on streams of singleton items and streams of variable sized sets of items. The algorithms remember a small summary of the stream and produce an approximate output, with a guaranteed error, not exceeding a user-defined threshold. Sticky Sampling is a probabilistic algorithm, while Lossy Counting is deterministic. In practice, Lossy Counting performs better than Sticky Sampling, although it theoretically has a worse worst-case bound. The goal of our work is to find a way to produce an exact result instead of an approximation.

In Section 1.2 in the Introduction, we also illustrated that most of the previous work on mining frequently occurring items or sets over data streams use models that work with a window length or a decay factor which is *fixed* in advance. In many applications, however, it is *not* possible to fix a window length or a decay factor that is most optimal for every item at every time point in an evolving stream. For example, consider a large retail chain of which sales can be considered as a stream. It is very difficult to choose in which period you are particularly interested. For many products, the amount sold depends highly on the period of the year. E.g., sales of ice cream increase in summer time, and sales of beer increase during the world cup. Such seasonal variations of changes in frequency of a set can only be discovered when choosing the correct window size for that set. This size, however, can hide a similar behavior of other sets in another window.

Therefore, in this dissertation, we introduce a *new frequency measure*, based on a flexible window length, by defining the current frequency of a specific set in the stream of sets of items as the maximal frequency of the set over all windows in the stream from any point in the past until the current state. When a stream evolves, the length of the window containing the highest frequency for a given set changes continuously. This new frequency measure for finding frequent sets in a stream is quite different from the ones typically found in the literature on stream mining. To the best of our knowledge, we are the first to present an approach in which no fixed window length or decay factor is used. Even more, this new frequency measure is very suitable to identify sudden bursts of occurrences of sets quickly, while still taking into account the history of the stream. Therefore, our approach can be useful in applications in which hot topics or combinations of topics need to be tracked. Examples of such applications include identifying stocks with a strong growth in trading volumes or with unusually high price fluctuations, tracking popular search terms on the internet, detecting network anomalies in telecommunication if the number of packages lost within a certain time period exceeds some threshold, detecting Gamma Ray bursts in astrophysical data, research in medical and bio-medical data, etc.

5.2 Notations

Before we can start mining streams, it is important to unambiguously define what a stream is. We assume that a stream is build from items in universe $\mathcal{I} = \{i_1, i_2, \ldots, i_n\}$, a finite set.

In general, on every time stamp t, a *transaction* I_t over $\mathcal{I}$, i.e. $I_t \subseteq \mathcal{I}$, arrives in the stream.

Definition 5.2.1. *A* stream $\langle I_1\ I_2\ \ldots\ I_m\rangle$ *is a sequence of transactions over* $\mathcal{I}$*, denoted* $\mathbb{S}$*, where* $m = |\mathbb{S}|$ *is the* length *of the stream.*

Notation 5.2.1. *The stream of length* 3*, with on the first time stamp the singleton* $\{a\}$*, on the second time stamp the singleton* $\{b\}$ *and on the third time stamp the singleton* $\{c\}$*, is denoted* $\langle a\ b\ c\rangle$*. The stream of length* 3*, with on the first time stamp the set* $\{a, b\}$*, on the second time stamp the singleton* $\{c\}$ *and on the third time stamp the set* $\{a, d, f\}$*, is denoted* $\langle ab\ c\ adf\rangle$*. The order of the items within a set is not important.*

Definition 5.2.2. *The number of transactions over* $\mathcal{I}$ *in a stream* $\mathbb{S}$ *that contain set* I *is denoted* $count(I, \mathbb{S})$*.*

Example 5.2.1. *For stream* $\langle a\ b\ c\rangle$*,* $count(a, \langle a\ b\ c\rangle) = 1$*, since* a *occurs in exactly one set. For stream* $\langle ab\ c\ adf\rangle$*,* $count(a, \langle ab\ c\ adf\rangle) = 2$ *and* $count(af, \langle ab\ c\ adf\rangle) = 1$*.*

Definition 5.2.3. *The* frequency of I in $\mathbb{S}$ *is defined as*

$$freq(I, \mathbb{S}) := \frac{count(I, \mathbb{S})}{|\mathbb{S}|} \ .$$

Example 5.2.2. *By definition, we have* $freq(a, \langle a\ b\ c\rangle) = 1/3$*,* $freq(a, \langle ab\ c\ adf\rangle) = 2/3$ *and* $freq(af, \langle ab\ c\ adf\rangle) = 1/3$*.*

Definition 5.2.4. *Let* $\mathbb{S}_1$ *be* $\langle I_1^1\ I_2^1\ \ldots\ I_{m_1}^1\rangle$*,* $\mathbb{S}_2$ *be* $\langle I_1^2\ I_2^2\ \ldots\ I_{m_2}^2\rangle$*, ... and* $\mathbb{S}_r$ *be* $\langle I_1^r\ I_2^r\ \ldots\ I_{m_r}^r\rangle$*. The* concatenation of the streams $\mathbb{S}_1, \ldots, \mathbb{S}_r$ *equals*

$$\mathbb{S}_1 \cdot \mathbb{S}_2 \cdot \ldots \cdot \mathbb{S}_r := \langle I_1^1\ I_2^1\ \ldots\ I_{m_1}^1\ I_1^2\ I_2^2\ \ldots\ I_{m_2}^2 \ldots\ I_1^r\ I_2^r\ \ldots\ I_{m_r}^r\rangle \ .$$

Definition 5.2.5. *The notation* $\mathbb{S}[s, t]$ *is used to denote the* substream *or* block *or* window from position s to position t in stream $\mathbb{S}$*,*

$$\mathbb{S}[s, t] := \langle I_s\ I_{s+1}\ \ldots\ I_t\rangle \ .$$

The substream of $\mathbb{S}$ *consisting of the last* k *transactions of* $\mathbb{S}$*, denoted* $last(k, \mathbb{S})$*, is*

$$last(k, \mathbb{S}) := \mathbb{S}[|\mathbb{S}| - k + 1, |\mathbb{S}|] \ .$$

Until now, a stream is defined as a statical object. In reality, however, a stream is an *evolving* object: at every time point, a new transaction will be inserted at the end of the stream, making the stream continuously grow. As such, evolving streams are essentially unbounded, time-dependent sequences, and when processing them, it is to be assumed that only a small part can be kept in memory.

Definition 5.2.6. $\mathbb{S}_t$ *will denote the stream* $\mathbb{S}$ *up to time stamp* t*; that is, the part of the stream that already passed at time stamp* t*.*

We assume that streams grow from left to right; this means that the earlier transactions are on the left of the stream, and more recent transactions on the right. For simplicity, we assume that the first transaction arrived at time stamp 1, and since then, at every time stamp a new transaction was inserted.

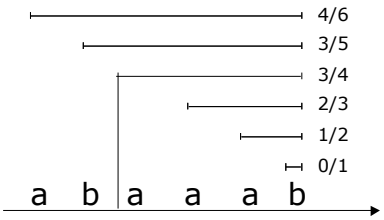


Figure 5.1: Visualization of the window in which a occurs the most

5.3 New Frequency Measure for Streams

We are now ready to define our new frequency measure for streams and to formally introduce the mining problem, that will be studied in the rest of this chapter. Several properties of the new measure are studied, that will play an important role in the development of an incremental algorithm for mining frequent sets in streams.

5.3.1 The Max-Frequency Measure

Because it is not possible to fix a window length or a decay factor that is most optimal for every set at every time point in an evolving stream, we will give every set the benefit of the doubt and consider for each set the window in which it has the highest frequency. Intuitively, Figure 5.1 illustrates our view. More specifically, we define the current frequency of a set as the maximum over all windows from the past until the current state.

Definition 5.3.1. *The* max-frequency $maxfreq(I, \mathbb{S})$ of a set I in a stream $\mathbb{S}$ *is defined as the maximum of the frequencies of I over all windows extending to the end of the stream; that is:*

$$maxfreq(I, \mathbb{S}) := \max_{k=1..|\mathbb{S}|} (freq(I, last(k, \mathbb{S}))) \ .$$

The longest window in which the maximum frequency is reached is called the maximal window for I in $\mathbb{S}$ *and its starting point is denoted $startmax(I, \mathbb{S})$. That is, $startmax(I, \mathbb{S})$ is the smallest index such that*

$$maxfreq(I, \mathbb{S}) = freq(I, \mathbb{S}[startmax(I, \mathbb{S}), |\mathbb{S}|]) \ .$$

Example 5.3.1.

$$\begin{aligned} maxfreq(a, \langle a\ b\ a\ a\ a\ b\rangle) &= \max_{k=1..6} (freq(a, last(k, \langle a\ b\ a\ a\ a\ b\rangle))) \\ &= \max\left(\frac{0}{1}, \frac{1}{2}, \frac{2}{3}, \frac{3}{4}, \frac{3}{5}, \frac{4}{6}\right) = \frac{3}{4} \\ maxfreq(a, \langle b\ c\ d\ a\ b\ c\ d\ a\rangle) &= \max\left(\frac{1}{1}, \dots\right) = 1 \\ maxfreq(a, \langle x\ a\ a\ x\ a\ a\ x\rangle) &= \max\left(\frac{0}{1}, \frac{1}{2}, \frac{2}{3}, \frac{2}{4}, \frac{3}{5}, \frac{4}{6}, \frac{4}{7}\right) = \frac{2}{3} \\ maxfreq(ab, \langle ab\ cd\ ef\ abgh\ cdp\rangle) &= \max\left(\frac{0}{1}, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{2}{5}\right) = \frac{1}{2} \end{aligned}$$

5.3.2 Remarks

The Min-Frequency Measure

Not only the maximal frequency of a set can be of interest to an analyst, also the *minimal frequency* can sometimes reveal interesting knowledge. Inspired by the max-frequency measure, we can also define the min-frequency of a set I as the minimum of the frequency of I over all windows extending from the end of the stream.

Definition 5.3.2. *The* min-frequency $minfreq(I, \mathbb{S})$ *of a set I in a stream $\mathbb{S}$ is defined as the minimum of the frequencies of I over all windows extending to the end of the stream; that is:*

$$minfreq(I, \mathbb{S}) := \min_{k=1..|\mathbb{S}|} (freq(I, last(k, \mathbb{S}))) \ .$$

The longest window in which the minimal frequency is reached is called the minimal window for I in $\mathbb{S}$ *and its starting point is denoted $startmin(I, \mathbb{S})$. That is, $startmin(I, \mathbb{S})$ is the smallest index such that*

$$minfreq(I, \mathbb{S}) = freq(I, \mathbb{S}[startmin(I, \mathbb{S}), |\mathbb{S}|]) \ .$$

Notice that maintaining and finding the minimal frequency $minfreq()$ does not pose a new challenge over the maximal frequency. Indeed, consider $\overline{I}$, the negation of I, meaning that if I is contained in a transaction, $\overline{I}$ is not and vice versa.

Definition 5.3.3. *Given, a stream $\mathbb{S}$ of transactions I_t over $\mathcal{I}$, and a set I. The negation $\overline{I}$ of set I is characterized, for each transaction I_t in $\mathbb{S}$, by*

$$count(\overline{I}, \langle I_t \rangle) = 1 - count(I, \langle I_t \rangle) \ ,$$

Property 5.3.1. *Given, a stream $\mathbb{S}$ and a set I. Now, $startmax()$ and $startmin()$ are related:*

$$startmin(I, \mathbb{S}) \ = \ startmax(\overline{I}, \mathbb{S}) \ .$$

Then, Example 5.3.2 illustrates that at any time point the maximal frequency of $\overline{I}$ and the minimal frequency of I add up to 1.

Property 5.3.2. *Given a stream $\mathbb{S}$ and a set I. Now, $minfreq()$ and $maxfreq()$ are related:*

$$minfreq(I, \mathbb{S}) \ = \ 1 - maxfreq(\overline{I}, \mathbb{S}) \ .$$

Example 5.3.2. *First, we focus on $minfreq()$ and $startmin()$ for target a.*

$$\begin{aligned}
minfreq(a, \langle a\ b\ a\ a\ a\ b \rangle) &= \min_{k=1..6} (freq(a, last(k, \langle a\ b\ a\ a\ a\ b \rangle))) \\
&= \min\left(\frac{0}{1}, \frac{1}{2}, \frac{2}{3}, \frac{3}{4}, \frac{3}{5}, \frac{4}{6}\right) = 0 \\
startmin(a, \langle a\ b\ a\ a\ a\ b \rangle) &= 6 \\
minfreq(a, \langle b\ c\ d\ a\ b\ c\ d\ a \rangle) &= \min\left(\frac{1}{1}, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{2}{5}, \frac{2}{6}, \frac{2}{7}, \frac{2}{8}\right) = \frac{1}{4} \\
startmin(a, \langle b\ c\ d\ a\ b\ c\ d\ a \rangle) &= 1
\end{aligned}$$

$$
\begin{aligned}
minfreq(a, \langle x\ a\ a\ x\ a\ a\ x\rangle) &= \min\left(\frac{0}{1}, \frac{1}{2}, \frac{2}{3}, \frac{2}{4}, \frac{3}{5}, \frac{4}{6}, \frac{4}{7}\right) = 0\\
startmin(a, \langle x\ a\ a\ x\ a\ a\ x\rangle) &= 7\\
minfreq(ab, \langle ab\ cd\ ef\ abgh\ cdp\rangle) &= \min\left(\frac{0}{1}, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{2}{5}\right) = 0\\
startmin(ab, \langle ab\ cd\ ef\ abgh\ cdp\rangle) &= 5
\end{aligned}
$$

Now, we focus on maxfreq() and startmax() for target $\overline{a}$ *in the same streams.*

$$
\begin{aligned}
maxfreq(\overline{a}, \langle a\ b\ a\ a\ a\ b\rangle) &= \max\left(\frac{1}{1}, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{2}{5}, \frac{2}{6}\right) = 1\\
startmax(\overline{a}, \langle a\ b\ a\ a\ a\ b\rangle) &= 6\\
maxfreq(\overline{a}, \langle b\ c\ d\ a\ b\ c\ d\ a\rangle) &= \max\left(\frac{0}{1}, \frac{1}{2}, \frac{2}{3}, \frac{3}{4}, \frac{3}{5}, \frac{4}{6}, \frac{5}{7}, \frac{6}{8}\right) = \frac{3}{4}\\
startmax(\overline{a}, \langle b\ c\ d\ a\ b\ c\ d\ a\rangle) &= 1\\
maxfreq(\overline{a}, \langle x\ a\ a\ x\ a\ a\ x\rangle) &= \max\left(\frac{1}{1}, \frac{1}{2}, \frac{1}{3}, \frac{2}{4}, \frac{2}{5}, \frac{2}{6}, \frac{3}{7}\right) = 1\\
startmax(\overline{a}, \langle x\ a\ a\ x\ a\ a\ x\rangle) &= 7\\
maxfreq(\overline{ab}, \langle ab\ cd\ ef\ abgh\ cdp\rangle) &= \max\left(\frac{1}{1}, \frac{1}{2}, \frac{2}{3}, \frac{3}{4}, \frac{3}{5}\right) = 1\\
startmax(\overline{ab}, \langle ab\ cd\ ef\ abgh\ cdp\rangle) &= 5
\end{aligned}
$$

It is clear that $startmin(I, \mathbb{S}) = maxwin\overline{I}, \mathbb{S}$ *and* $minfreq(I, \mathbb{S}) = 1 - maxfreq(\overline{I}, \mathbb{S})$.

Therefore, in order to find $minfreq(I, \mathbb{S})$, we can in fact monitor the set $\overline{I}$, and when we need the minimal frequency of I, it is easily found as $1 - maxfreq(\overline{I}, \mathbb{S})$.

The Minimal Window Length

By definition, the max-frequency of a set I in a stream that ends with a transaction containing I is always 100%, independently of the overall frequency of I. Hence, even in a stream where I is extremely rare, at some points, the max-frequency will be 1. This disadvantage of max-frequency, however, can easily be resolved by setting a *minimal length all windows must obey.* A discussion on the minimal window length is included in Subsection 5.4.3.

Comparing Max-Frequency with Other Existing Measures

Note that our definition of frequency is *quite different from the usual approaches*, explained in Section 1.2. To illustrate this claim, we incorporate the three most important stream mining approaches in our notation: the sliding window model [26, 31, 40, 42, 45, 46, 55], the time-fading model [23, 44] and the landmark model [40, 42, 56]. When using our notation, the frequency of a set I in a stream $\mathbb{S}$ in the *sliding window* model is defined as

$$freq(I, last(W, \mathbb{S}))\ , \tag{5.1}$$

for a fixed window length W. In the *time-fading* model, a time-decaying factor $0 < d < 1$ is used to compute the frequency of set I:

$$freq(I, \mathbb{S}) = \sum_{j=1}^{|\mathbb{S}|/W} d^{(|\mathbb{S}|/W)-j}\ freq(I, \mathbb{S}[(j-1)W + 1, jW])\ . \tag{5.2}$$

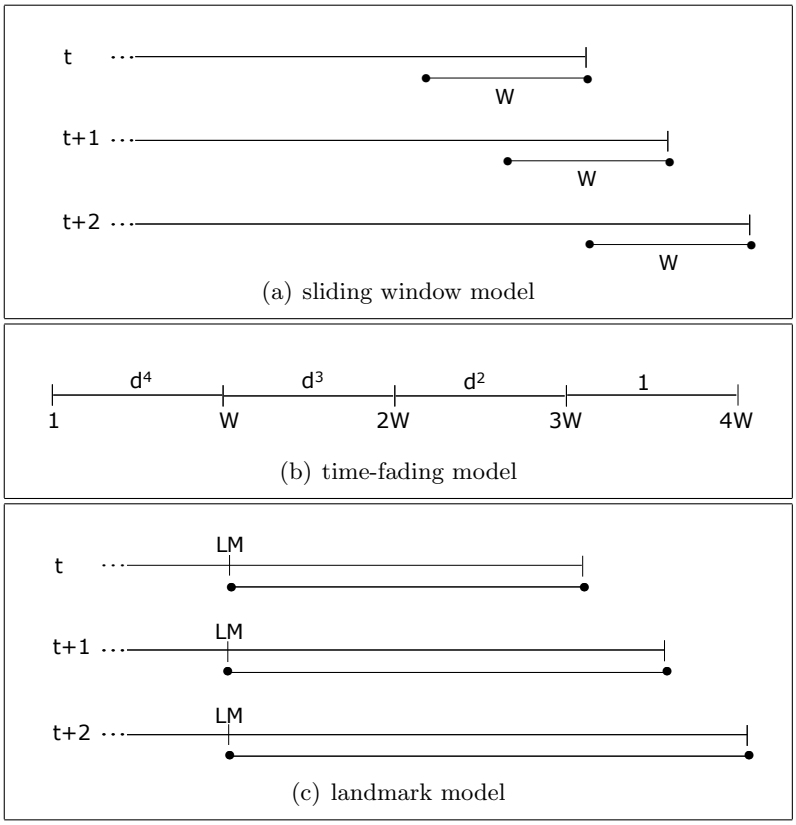


Figure 5.2: Illustration of the measure in the sliding window, time-fading and landmark model

For the *landmark* model, the frequency of set I in $\mathbb{S}$ is defined as

$$freq(I, \mathbb{S}[LM, |\mathbb{S}|]) \ , \tag{5.3}$$

for a window between the landmark LM and the current time. A schematical illustration of the different views can be found in Figure 5.2. Here, each stream is presented as a line, to simplify the figure and clarify the use of the different windows.

To compare the existing models with our new max-frequency model, we focus on their behavior for a particular set in a certain stream of length 10000, for which we assume that on each time stamp a new transaction enters the stream, and we group all the results in Figure 5.3. The distribution of the number of occurrences of this set in the stream is given in the bottom graph. This line illustrates the probability of the set for which the frequency is reported, at each indicated time point. First the probability fluctuates, but after some time, it becomes zero. Notice that this bottom graph only gives the probability; the actual frequency can be slightly different. The graphs above indicate how the different measures in the models behave for this target set. At every time stamp, the frequency of the target has been plotted for the different measures. Each drawing is computed based on the above introduced formulas (5.1), (5.2) and (5.3).

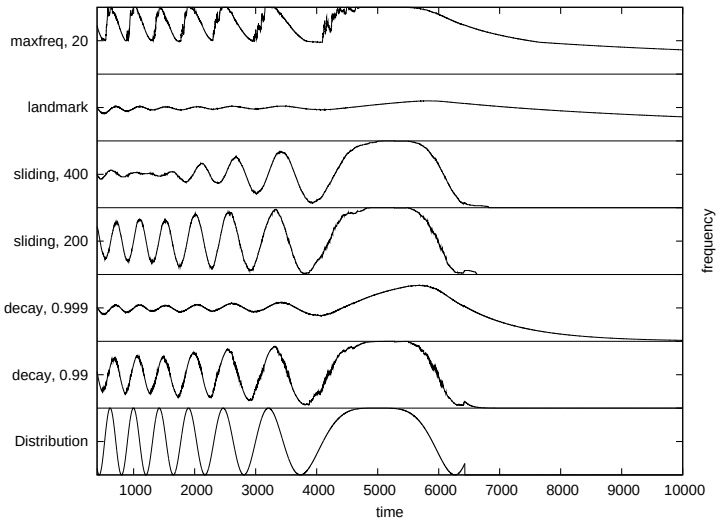


Figure 5.3: Comparison of different frequency measures

The sliding window frequency is plotted for window lengths 200 and 400. Notice that for window length 400, some of the frequency jumps of the target set go by unnoticed and others are significantly lowered, because of the smoothing implied by a large window length. The time-fading frequency has been given for two different decay factors, 0.99 and 0.999. Notice that although these two values are very similar, the frequency is actually very different. Clearly, the landmark model is less suitable for evaluating evolving and unbounded streams. The new measure max-frequency is plotted on the top line of the illustration. Max-frequency takes into account the history without fading away sudden jumps in the frequency. As a direct result of this, the graph for the max-frequency is less smooth than the other graphs, because the actual frequency of the target only approximates the given distribution. Notice that these deviations also show in the time-fading model (for example with decay 0.99), although in a far less pronounced way. The other methods do not show the existing short deviations from the ideal distribution, as they are less sensitive to short-time changes. Max-frequency is much less dependent on finding the exact right parameter setting than the other models, because it only determines a lower bound on the window size. For example, for a stream of length 10000 and extreme minimum window lengths of 10 or 1000, still 9010 of the 10000 windows are treated in the same way for both paramater settings. In the other models, the parameters completely determine the weight of every point in the stream. For different parameter settings, all 10000 points in the stream will be handled differently.

5.3.3 The Problem Statement

In Figure 5.4, the max-frequency of a singleton target set $\{a\}$ in an evolving stream consisting of singleton transactions is given. The starting point $startmax(a, \mathbb{S})$ of each maximal window is marked by a vertical bar. Note that if two (or more) time points are candidate to be the starting point of the maximal window, it is the *oldest* time point that wins, as defined in Definition 5.3.1 and illustrated in $\mathbb{S}_8$.

Stream $\mathbb{S}_t$ at time stamp t	$maxfreq(a, \mathbb{S}_t)$
$\mathbb{S}_1 = \langle \mid \text{a} \rangle$	1
$\mathbb{S}_2 = \langle \mid \text{a} \quad \text{a} \rangle$	1
$\mathbb{S}_3 = \langle \mid \text{a} \quad \text{a} \quad \text{a} \rangle$	1
$\mathbb{S}_4 = \langle \mid \text{a} \quad \text{a} \quad \text{a} \quad \text{b} \rangle$	3/4
$\mathbb{S}_5 = \langle \mid \text{a} \quad \text{a} \quad \text{a} \quad \text{b} \quad \text{b} \rangle$	3/5
$\mathbb{S}_6 = \langle \mid \text{a} \quad \text{a} \quad \text{a} \quad \text{b} \quad \text{b} \quad \text{b} \rangle$	3/6
$\mathbb{S}_7 = \langle \quad \text{a} \quad \text{a} \quad \text{a} \quad \text{b} \quad \text{b} \quad \text{b} \mid \text{a} \rangle$	1
$\mathbb{S}_8 = \langle \mid \text{a} \quad \text{a} \quad \text{a} \quad \text{b} \quad \text{b} \quad \text{b} \quad \text{a} \quad \text{b} \rangle$	4/8 = 1/2
$\mathbb{S}_9 = \langle \mid \text{a} \quad \text{a} \quad \text{a} \quad \text{b} \quad \text{b} \quad \text{b} \quad \text{a} \quad \text{b} \quad \text{b} \rangle$	4/9
...	...

Figure 5.4: Max-frequency of a target a in a stream at every time point

The **main problem** we study in the rest of this chapter is the following:

> ***Stream Mining with Flexible Windows***
> *For an evolving stream $\mathbb{S}$ and a fixed target set A, maintain a small summary of the stream in time, such that, at any time point t, $maxfreq(A, \mathbb{S}_t)$ can be produced instantly from the summary.*

More formally, we will introduce a summary for target set A in the stream at time t, $Summary(A, \mathbb{S}_t) := S_t(A)$, an update procedure *Update*, and a procedure *Get_maxfreq*, such that, if we assume that on time stamp $t+1$ a transaction T is added to the stream, resulting in a new stream $\mathbb{S}_{t+1} := \mathbb{S}_t \cdot \langle T \rangle$, $Update(S_t(A), T)$ equals $S_{t+1}(A) = Summary(A, \mathbb{S}_{t+1})$ and $Get\_maxfreq(S_{t+1}(A)) = maxfreq(A, \mathbb{S}_{t+1})$. The procedure *Get_maxfreq* picks on each time stamp t' the current maximal frequency for the target set A from the summary. Therefore, $Get\_maxfreq(S_{t'}(A))$ equals $maxfreq(A, \mathbb{S}_{t'})$ for target set A. Because *Update* has to be executed every time a new set arrives, it has to be extremely efficient, in order to be finished before the next set arrives. Similarly, because the stream continuously grows, the summary must be independent of the number of items seen so far, or, at least grow very slowly as the stream evolves. The method we develop will indeed meet these criteria, as the experiments in Section 5.6 will show. Section 5.5 will give the theoretical proof for a specific type of streams.

5.3.4 Properties of Max-Frequency

We are now ready to show some properties of max-frequency that will be crucial for the incremental algorithm that maintains the summary of the stream. In Figure 5.4, when looking for the maximal window for target set a, we manually checked all possible windows to find the maximal one. Obviously, in long and evolving streams, checking all possible windows to find the maximal one is infeasible algorithmically, given the constraints of stream problems: it is not possible to store the stream, nor large parts of it, and therefore it is also not possible to revisit information that already passed by. Fortunately, to find the maximal window for a certain target set, not every time point in the stream needs to be checked. We will show exactly which points need to be inspected and we will call these points the *borders* in the stream. The summary of the stream then consists exactly of the recording of these borders, together with some information about the corresponding frequencies of the target set. Remark that the amount of borders possibly is unlimited[2].

[2]Section 5.5 will consider in detail the size of the summary that is possible in worst case.

Definition 5.3.4. *Time stamp q in stream $\mathbb{S}$ is called a* border for set A in $\mathbb{S}$ *if there exists a stream $\mathbb{B}$ such that $q = startmax(A, \mathbb{S} \cdot \mathbb{B})$, i.e. q is the starting point of a maximal window.*

We will now give an exact characterization of the borders w.r.t frequency counts in the stream.

Lemma 5.3.1. *Given $a, b, c, d \in \mathbb{N}_0$.*

$$\frac{a}{b} \geq \frac{c}{d} \Leftrightarrow \frac{a}{b} \geq \frac{a+c}{b+d} \tag{5.4}$$

$$\frac{a}{b} \leq \frac{c}{d} \Leftrightarrow \frac{a}{b} \leq \frac{a+c}{b+d} \tag{5.5}$$

Proof.

$$\frac{a}{b} \geq \frac{c}{d} \Leftrightarrow ad \geq bc \Leftrightarrow ab+ad \geq ab+bc \Leftrightarrow \frac{a}{b} \geq \frac{a+c}{b+d}$$

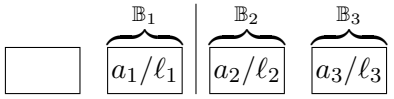
$$\frac{a}{b} \leq \frac{c}{d} \Leftrightarrow ad \leq bc \Leftrightarrow ab+ad \leq ab+bc \Leftrightarrow \frac{a}{b} \leq \frac{a+c}{b+d}$$

□

Theorem 5.3.1. *Let $\mathbb{S}$ be a stream of length L, and let $\mathbb{S}[q, L]$ be the maximal window for the target set A. Then, for any p, r with $p < q \leq r$:*

$$freq(A, \mathbb{S}[p, q-1]) < freq(A, \mathbb{S}[q, r]) \ .$$

Proof. Let $\mathbb{B}_1$ denote $\mathbb{S}[p, q-1]$, $\mathbb{B}_2$ denote $\mathbb{S}[q, r]$, and $\mathbb{B}_3$ denote $\mathbb{S}[r+1, L]$. Because $\mathbb{B}_2 \cdot \mathbb{B}_3$ is the maximal window for A in $\mathbb{S}$, it holds that the frequency of A in $\mathbb{B}_2 \cdot \mathbb{B}_3$ is strictly higher than in $\mathbb{B}_1 \cdot \mathbb{B}_2 \cdot \mathbb{B}_3$ and at least as high as in $\mathbb{B}_3$. This is so, because in the case of multiple windows with maximal frequency, the largest one is selected. Now, let $\ell_1 = |\mathbb{B}_1|$, $\ell_2 = |\mathbb{B}_2|$, and $\ell_3 = |\mathbb{B}_3|$, and let $a_1 = count(A, \mathbb{B}_1)$, $a_2 = count(A, \mathbb{B}_2)$, and $a_3 = count(A, \mathbb{B}_3)$, as depicted in:



Then, the conditions on the frequency translate into:

$$\frac{a_2+a_3}{\ell_2+\ell_3} > \frac{a_1+a_2+a_3}{\ell_1+\ell_2+\ell_3} \quad \text{and} \quad \frac{a_2+a_3}{\ell_2+\ell_3} \geq \frac{a_3}{\ell_3} \ . \tag{5.6}$$

The first condition in (5.6) is, according to Lemma 5.3.1 (5.4), equivalent to

$$\frac{a_2+a_3}{\ell_2+\ell_3} > \frac{a_1}{\ell_1} \ . \tag{5.7}$$

The second condition in (5.6) is, according to Lemma 5.3.1 (5.5), equivalent to $\frac{a_2}{\ell_2} \geq \frac{a_3}{\ell_3}$, and this can be rewritten to

$$\frac{a_2}{\ell_2} \geq \frac{a_2+a_3}{\ell_2+\ell_3} \ , \tag{5.8}$$

based on Lemma 5.3.1 (5.4). Combining (5.7) and (5.8) yields

$$freq(A, \mathbb{B}_1) = \frac{a_1}{\ell_1} < \frac{a_2}{\ell_2} = freq(A, \mathbb{B}_2) \ .$$

□

Theorem 5.3.2. *Let* $\mathbb{S}$ *be a stream, and let* $1 \leq q \leq |\mathbb{S}|$. *Position* q *is a* border *for target set* A *in* $\mathbb{S}$ if and only if *for all indices* j, k *with* $1 \leq j < q$ *and* $q \leq k \leq |\mathbb{S}|$, *it holds that*

$$freq(A, \mathbb{S}[j, q-1]) \;<\; freq(A, \mathbb{S}[q, k]) \;.$$

Proof.

$\Rightarrow$ Suppose that there exist a stream $\mathbb{B}$ such that $q = startmax(A, \mathbb{S} \cdot \mathbb{B})$, and that there exist indices j and k with $1 \leq j < q$ and $q \leq k \leq |\mathbb{S}|$ such that $freq(A, \mathbb{S}[j, q-1]) \geq freq(A, \mathbb{S}[q, k])$. This situation is *in contradiction* with Theorem 5.3.1: split the stream $\mathbb{S} \cdot \mathbb{B}$ as

$$(\mathbb{S}[1, j-1]) \cdot (\mathbb{S}[j, q-1]) \cdot (\mathbb{S}[q, k]) \cdot (\mathbb{S}[k+1, |\mathbb{S}|] \cdot \mathbb{B}) \;.$$

In this stream, it is known that $(\mathbb{S}[q, |\mathbb{S}|]) \cdot \mathbb{B}$ is the maximal window, while we also have that $freq(A, \mathbb{S}[j, q-1]) \geq freq(A, \mathbb{S}[q, k])$.

$\Leftarrow$ We need to show that there exists a continuation $\mathbb{S}'$ of stream $\mathbb{S}$, resulting in stream $\mathbb{S} \cdot \mathbb{S}'$, in which q is the starting point of the maximal window. We consider *two* cases: either q is the rightmost border in $\mathbb{S}$, or not.

If q *is the rightmost border*, then q is the maximal border in $\mathbb{S}$, because for any other border p that is smaller than q, the frequency in the window starting in p is smaller than that in the window starting in q. Indeed, for every border p such that $1 \leq p < q$, it is given that $freq(A, \mathbb{S}[p, q-1]) < freq(A, \mathbb{S}[q, |\mathbb{S}|])$ which implies

$$freq(A, \mathbb{S}[p, |\mathbb{S}|]) < freq(A, \mathbb{S}[q, |\mathbb{S}|]) \;.$$

To illustrate, if $freq(A, \mathbb{S}[p, q-1]) = a/b$ and $freq(A, \mathbb{S}[q, |\mathbb{S}|]) = c/d$, it is obvious that $a/b < c/d \Leftrightarrow (a+c)/(b+d) < c/d$. In this case, we do not need an extra extension $\mathbb{S}'$ to show that q is the maximal border.

In the other case, let q' be the leftmost border in $\mathbb{S}$ that is on the right of q. That is, q' is a border, $q < q'$, and there is no other border r such that $q < r < q'$. We are now going to construct a stream $\mathbb{S}'$ such that, if we append $\mathbb{S}'$ to $\mathbb{S}$, resulting in $\mathbb{S} \cdot \mathbb{S}'$, the frequency of A from q until the end of the extended stream will be the same as the frequency of A in the window from q to $q' - 1$:

$$freq(A, (\mathbb{S} \cdot \mathbb{S}')[q, |\mathbb{S} \cdot \mathbb{S}'|]) \;=\; freq(A, \mathbb{S}[q, q'-1]) \;. \tag{5.9}$$

If we can find such an extension $\mathbb{S}'$, we will show that q is indeed the maximal border in $\mathbb{S} \cdot \mathbb{S}'$.

To find this extension $\mathbb{S}'$, we introduce two parameters, x and y. Let y be the length of the extension $\mathbb{S}'$ that we are going to construct, and let x be the number of occurrences of the target set A in extension $\mathbb{S}'$. If the two parameters satisfy the following equalities

$$freq(A, (\mathbb{S} \cdot \mathbb{S}')[q, |\mathbb{S} \cdot \mathbb{S}'|]) \;=\; \frac{count(A, \mathbb{S}[q', |\mathbb{S}|]) + x}{|\mathbb{S}| - q' + 1 + y} \;=\; freq(A, \mathbb{S}[q, q'-1]) \;, \tag{5.10}$$

our condition (5.9) is reached.

Notice that for any solution x, y of (5.10), indeed always $x \leq y$, as is required by the definition of the two parameters. This is because

$$\mathit{freq}(A, \mathbb{S}[q, q'-1]) \ < \ \mathit{freq}(A, \mathbb{S}[q', |\mathbb{S}|]) \ = \ \frac{\mathit{count}(A, \mathbb{S}[q', |\mathbb{S}|])}{|\mathbb{S}| - q' + 1} \ ,$$

since q' is a border, and thus every block that occurs before q' must have a frequency that is lower than any block that occurs after q', according to the "only if" part which is already shown above. Obviously, $\mathit{count}(A, \mathbb{S}[q', |\mathbb{S}|]) \ \leq \ |\mathbb{S}| - q' + 1$.

Based on parameters x and y, we construct $\mathbb{S}'$ as follows:

$$\langle \overbrace{a \ a \ \cdots \ a}^{x} \overbrace{\emptyset \ \emptyset \ \cdots \ \emptyset}^{y-x} \rangle \ .$$

All what is left to show is that q is, indeed, the maximal border in $\mathbb{S} \cdot \mathbb{S}'$. We therefore first prove that it is not possible to have a new border in the extension $\mathbb{S}'$, nor in the part of $\mathbb{S}$, to the right of q.

Obviously, because all target singleton sets in $\mathbb{S}'$ occur in the beginning of the extension, the maximal frequency of A in $\mathbb{S}'$ is x/y, which is smaller than $\mathit{freq}(A, (\mathbb{S} \cdot \mathbb{S}')[q, |\mathbb{S} \cdot \mathbb{S}'|])$. To prove this statement, we split $(\mathbb{S} \cdot \mathbb{S}')[q, |\mathbb{S} \cdot \mathbb{S}'|]$ into *three* parts:

$$\mathbb{S}[q, q'-1] \ \cdot \ \mathbb{S}[q', |\mathbb{S}|] \ \cdot \ \mathbb{S}' \ .$$

- The first part has a frequency of A equal to that of the whole: $\mathit{freq}(A, \mathbb{S}[q, q'-1]) \ = \ \mathit{freq}(A, (\mathbb{S} \cdot \mathbb{S}')[q, |\mathbb{S} \cdot \mathbb{S}'|])$
- The second part has a frequency of A that is strictly larger than that of the first part: $\mathit{freq}(A, \mathbb{S}[q, q'-1]) \ < \ \mathit{freq}(A, \mathbb{S}[q', |\mathbb{S}|])$, because q' is a border in $\mathbb{S}$.
- The third part $\mathbb{S}'$ consequently needs to have a frequency of A lower than that of the whole.

Mathematically seen, if

$$\mathit{freq}(A, \mathbb{S}[q, q'-1]) \ = \ \frac{a}{b}, \ \mathit{freq}(A, \mathbb{S}[q', |\mathbb{S}|]) \ = \ \frac{c}{d} \ \text{ and } \ \mathit{freq}(A, \mathbb{S}') \ = \ \frac{x}{y} \ ,$$

we have that $\mathit{freq}(A, (\mathbb{S} \cdot \mathbb{S}')[q, |\mathbb{S} \cdot \mathbb{S}'|]) = \dfrac{a+c+x}{b+d+y}$. The first condition in the above enumeration translates to

$$\frac{a}{b} \ = \ \frac{a+c+x}{b+d+y} \ . \tag{5.11}$$

The second condition can be written as

$$\frac{a}{b} \ < \ \frac{c}{d} \ \Leftrightarrow \ \frac{a}{b} \ < \ \frac{a+c}{b+d} \ , \tag{5.12}$$

according to Lemma 5.3.1 (5.5) Combining (5.11) and (5.12), based on Lemma 5.3.1, yields

$$\frac{a+c+x}{b+d+y} \ < \ \frac{a+c}{b+d} \ \Leftrightarrow \ \frac{x}{y} \ < \ \frac{a+c}{b+d} \ \Leftrightarrow \ \frac{x}{y} \ < \ \frac{a+c+x}{b+d+y} \ ,$$

that shows that x/y is always smaller than $freq(A, (\mathbb{S} \cdot \mathbb{S}')[q, |\mathbb{S} \cdot \mathbb{S}'|])$.

Furthermore, for any position $q'' \geq q'$ after q in $\mathbb{S}$, we have that

$$\begin{aligned} freq(A, (\mathbb{S} \cdot \mathbb{S}')[q, q'' - 1]) &\geq freq(A, (\mathbb{S} \cdot \mathbb{S}')[q, q' - 1]) \\ &= freq(A, (\mathbb{S} \cdot \mathbb{S}')[q, |\mathbb{S} \cdot \mathbb{S}'|]) \ , \end{aligned} \tag{5.13}$$

because q' was a border and because of the way $\mathbb{S}'$ was constructed. If we call

$$freq(A, \mathbb{S}[q, q' - 1]) \ = \ \frac{a}{b}, \ freq(A, \mathbb{S}[q', q'' - 1]) \ = \ \frac{u}{v}, \ freq(A, \mathbb{S}[q'', |\mathbb{S}|]) \ = \ \frac{n}{m},$$

and $freq(A, \mathbb{S}') \ = \ \frac{x}{y}$, the above condition (5.13) gives

$$\frac{a+u}{b+v} \ \geq \ \frac{a+u+v+x}{b+v+m+y} \ \Leftrightarrow \ \frac{a+u}{b+v} \ \geq \ \frac{v+x}{m+y} \ ,$$

based on Lemma 5.3.1, and hence,

$$freq(A, (\mathbb{S} \cdot \mathbb{S}')[q, q'' - 1]) \ \geq \ freq(A, (\mathbb{S} \cdot \mathbb{S}')[q'', |\mathbb{S} \cdot \mathbb{S}'|]) \ .$$

There thus exists a block before q'' that has a larger frequency of A than a block after q''. Therefore, q'' cannot be a border.

We also have to show that there is no border on the left-hand side of q that can play the same role. Indeed, for all positions p before q in $\mathbb{S}$,

$$\begin{aligned} freq(A, \mathbb{S}[p, q - 1]) &< freq(A, \mathbb{S}[q, q' - 1]) \\ &= freq(A, \mathbb{S}[q, |\mathbb{S} \cdot \mathbb{S}'|]) \ . \end{aligned}$$

Therefore, $\mathbb{S}[q, |\mathbb{S} \cdot \mathbb{S}'|]$ is the maximal window for A in $\mathbb{S} \cdot \mathbb{S}'$.

This if-direction of the proof shows that we will never store too many borders.

□

Example 5.3.3. *Consider the stream $\mathbb{S}_{27}$ in Figure 5.5. In this stream, two positions have been marked with a backslash. Both these points do* not *meet the criterion to be a border given in Theorem 5.3.2. Indeed, for both positions, a block before and after the position is indicated such that the frequency in the before-block is higher than the frequency in the after-block. The only positions that* meet *the requirement are indicated by vertical bars.*

Consider border q at position 21 in Figure 5.5. The frequency of a between this border q and the next border q' at position 27 is $3/6 = 0.5$. According to the proof of Theorem 5.3.2, we can compute a continuation of stream $\mathbb{S}_{27}$ such that position 21 becomes the maximal border. *This is done as follows. We have to find integers x and y, $x \leq y$, such that*

$$0.5 = \frac{count(a, \mathbb{S}[27, |\mathbb{S}_{27}|]) + x}{|\mathbb{S}_{27}| - 27 + 1 + y} = \frac{1+x}{1+y} \ .$$

This condition is satisfied by $x = 0$ and $y = 1$. (Another solution is $x = 2$ and $y = 5$.) This means that we can add a stream of length 1 starting with 0 times the target; that is, e.g., $\langle b \rangle$, resulting in $\mathbb{S}_{27} \cdot \langle b \rangle$. This extension of the stream is constructed in such a way that the frequency of a from position 21 until the end of the stream, $4/8$, is equal to the frequency of a in the block between the border at position 21 and the next border at position 27, $3/6$. Position 21 is the maximal border in $\mathbb{S}_{28} := \mathbb{S}_{27} \cdot \langle b \rangle$.

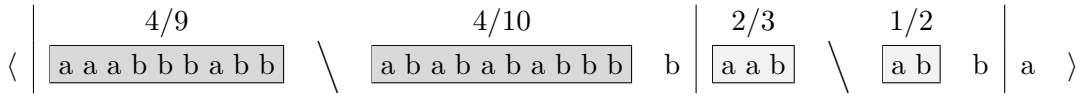


Figure 5.5: Example of dropping borders

5.4 New Algorithm for Mining Streams

Based on the new frequency measure *maxfreq*(), we present an incremental algorithm that efficiently maintains a summary for the target set A, and allows us to produce the current max-frequency of the target set instantly at any time. The algorithm and the summary maintained by it are studied in detail. A selection of adaptations of the algorithm is explained, including using a user-defined minimal frequency threshold σ, taking into account a minimal window length *mwl*, combining both σ and *mwl* and mining all frequent sets instead of focusing on one particular target set.

5.4.1 The Summary

The summary that our algorithm is going to construct contains all the relevant information of the history of the stream, which allows to produce the current frequency of a specific set immediately at any time. That is, when a new transaction arrives in the stream, the summary is updated, and when at a certain point in time, the current frequency of a set is required, the result can be obtained instantly from the summary.

The relevant information of the history of the stream that is kept in the summary are the border positions and their corresponding frequencies. We only focus on this information, because we observed that many points in the stream can never become the starting point of a maximal window, no matter what the continuation of the stream will be. The summary thus consists of some statistics about the few points in the stream that are still candidate starting points of a maximal window, the *borders*. Remark that the amount of borders can possibly be unlimited[3].

Definition 5.4.1. *Consider a stream $\mathbb{S}_t$ and target set A. At time t, let $p_1 < p_2 < \ldots < p_r$ be the border positions of stream $\mathbb{S}_t$, ordered from old to most recent. Let $a_i = count(A, \mathbb{S}_t[p_i, p_{i+1} - 1])$ be the number of occurrences of the target set A in between two subsequent border positions p_i and p_{i+1} (for $i = 1, \ldots, r-1$). Let $a_r = count(A, \mathbb{S}_t[p_r, t])$ denote the number of occurrences of A since the last border. The summary $S_t(A)$ for target set A in $\mathbb{S}_t$ is defined as the array*

$$S_t(A) := \begin{array}{|c|c|c|} \hline p_1 & \ldots & p_r \\ \hline a_1 & \ldots & a_r \\ \hline \end{array} \ .$$

We often use the following space-saving shorthand notation:

$$S_t(A) := [(p_1, a_1), \ldots, (p_r, a_r)] \ .$$

If it is clear from the situation for which target set A the summary is, we will simply use S_t.

[3]Section 5.5 will consider in detail the size of the summary in worst case.

Property 5.4.1. *Given a summary $S_t(A) = [(p_1, a_1), \ldots, (p_r, a_r)]$ for target set A at time stamp t in $\mathbb{S}_t$. The frequencies of set A for any of the border positions in $\mathbb{S}_t$ can be computed from this summary. For $1 \leq i \leq r$, we have that*

$$freq(A, \mathbb{S}_t[p_i, t]) \;=\; \frac{\sum_{j=i}^{r} a_j}{t - p_i + 1} \;\; .$$

Example 5.4.1. *Given stream $\mathbb{S}_{17} = \langle b\ a\ a\ a\ b\ a\ a\ b\ a\ b\ b\ a\ a\ a\ a\ b\ a \rangle$, and target set a. It can be computed (we will show this later in detail in Example 5.4.2) that the summary $S_{17}(a)$ looks like*

$$S_{17}(a) = \begin{array}{|c|c|c|} \hline 2 & 12 & 17 \\ \hline 6 & 4 & 1 \\ \hline \end{array} \;\; .$$

Based on this summary, we can easily compute the frequencies of a since any of the border positions:

$$\begin{aligned} freq(a, \mathbb{S}_t[2, 17]) &= (6 + 4 + 1)/(17 - 2 + 1) = 11/16 \\ freq(a, \mathbb{S}_t[12, 17]) &= (4 + 1)/(17 - 12 + 1) = 5/6 \\ freq(a, \mathbb{S}_t[17, 17]) &= 1/(17 - 17 + 1) = 1/1 \;\; . \end{aligned}$$

The border positions, ordered from old to most recent, are $2 < 12 < 17$, reflecting in $\langle\ b\ |a\ a\ a\ b\ a\ a\ b\ a\ b\ b\ |a\ a\ a\ a\ b\ |a\ \rangle$.

We will now give some important properties of the summary that will be used by the algorithm. First of all, we show that the frequency of the target set in the blocks in between two subsequent border positions are increasing. These fractions can be computed as

$$\frac{a_i}{p_{i+1} - p_i + 1} \text{ (with } 1 \leq i \leq r - 1) \text{ and } \frac{a_r}{t - p_r + 1} \;\; .$$

As a consequence, among all borders p_i, we have that $freq(A, \mathbb{S}_t[p_i, t])$ is maximal for $i = r$.

Property 5.4.2. *Let $\mathbb{S}_t$ be a stream and focus on target set A. Let $S_t(A) = [(p_1, a_1), \ldots, (p_r, a_r)]$. Then, it is always the case that*

$$\frac{a_1}{p_2 - p_1} \;<\; \frac{a_2}{p_3 - p_2} \;<\; \cdots \;<\; \frac{a_{r-1}}{p_r - p_{r-1}} \;<\; \frac{a_r}{t - p_r + 1} \;\; ,$$

and

$$freq(A, \mathbb{S}_t[p_1, t]) \;<\; freq(A, \mathbb{S}_t[p_2, t]) \;<\; \ldots \;<\; freq(A, \mathbb{S}_t[p_r, t]) \;\; .$$

Proof. First of all, we show that

$$\frac{a_i}{p_{i+1} - p_i} \;\geq\; \frac{a_{i+1}}{p_{i+2} - p_{i+1}}$$

(for $1 \leq i \leq r - 1$) is *not* possible, because p_{i+1} is a border and therefore, according to Theorem 5.3.2, it is not possible to find a before-block with a frequency that is higher than or equal to the frequency of an after-block.

We now show that if we have increasing individual fractions for the blocks, we also have increasing corresponding frequencies. More formally, we prove that

$$freq(A, \mathbb{S}_t[p_i, t]) \ < \ freq(A, \mathbb{S}_t[p_{i+1}, t]) \ ,$$

for two consecutive borders p_i and p_{i+1} $(1 \leq i \leq r-1)$. The corresponding frequencies x_i/y_i for p_i and x_{i+1}/y_{i+1} for p_{i+1} are constructed:

$$freq(A, \mathbb{S}_t[p_i, t]) = \frac{x_i}{y_i} = \frac{a_i + a_{i+1} + \ldots + a_r}{(p_{i+1} - p_i) + \ldots + (p_r - p_{r-1}) + (t - p_r + 1)} \ ,$$

$$freq(A, \mathbb{S}_t[p_{i+1}, t]) = \frac{x_{i+1}}{y_{i+1}} = \frac{a_{i+1} + a_{i+2} + \ldots + a_r}{(p_{i+2} - p_{i+1}) + \ldots + (p_r - p_{r-1}) + (t - p_r + 1)} \ .$$

Suppose for the sake of contradiction, that $x_i/y_i \geq x_{i+1}/y_{i+1}$. From this, it follows that

$$\begin{array}{ccccc} \dfrac{x_i - x_{i+1}}{y_i - y_{i+1}} & \geq & \dfrac{x_i}{y_i} & \geq & \dfrac{x_{i+1}}{y_{i+1}} \ . \\ \| & & & & \| \\ freq(A, \mathbb{S}[p_i, p_{i+1} - 1]) & & & & freq(A, \mathbb{S}[p_{i+1}, |\mathbb{S}|]) \end{array}$$

Because of Theorem 5.3.2, this is in contradiction with the fact that p_{i+1} is a border. □

Property 5.4.3. *The last entry of a summary always contains the max-frequency. That is: for a stream $\mathbb{S}_t$, target set A and summary $S_t(A) = [(p_1, a_1), \ldots, (p_r, a_r)]$,*

$$maxfreq(A, \mathbb{S}_t) \ = \ freq(A, \mathbb{S}_t[p_r, t]) \ = \ \frac{a_r}{t - p_r + 1} \ .$$

Proof. This follows directly from Property 5.4.2. □

5.4.2 The Algorithm

After introducing a summary for target set A in $\mathbb{S}_t$, i.e. $Summary(A, \mathbb{S}_t)$, it is time to introduce the algorithm that maintains this summary. The core of the algorithm is the procedure *Update*, which adjusts the summary each time a new transaction enters the stream. It is obvious that a new transaction that enters the stream either contains the target set or does not contain the target set.

Definition 5.4.2. *Given a stream $\mathbb{S}_t$, a target set A and a transaction T. If $A \subseteq T$, T is called a* target transaction. *If $A \not\subseteq T$, T is called a* non-target transaction.

Algorithm 8 presents the pseudo-code of the procedure *Update*, which updates the summary of target set A at time t, i.e. $S_t(A)$, according to the new information contained in transaction T that enters the stream at time stamp $t+1$, resulting in $S_{t+1}(A)$. In practice, a new summary is created, based on the old one. To start, the new summary is created empty. After the first target transaction entered the stream, the summary is initialized. Then, we consider the following *two* important cases.

Algorithm 8 *Update*$(S_t(A), T)$ on time $t+1$

Require: $S_t(A) = Summary(A, \mathbb{S}_t)$
$= [(p_1, a_1), \ldots, (p_r, a_r)]$
Ensure: $S_{t+1}(A) = Summary(A, \mathbb{S}_{t+1})$
$= Summary(A, \mathbb{S}_t \cdot \langle T \rangle)$

1: Set $S_{t+1}(A) := [\ ]$
2: **if** ($S_t(A)$ is empty) **then**
3: **if** (target set $A \subseteq T$) **then**
4: $S_{t+1}(A) := [(t+1, 1)]$
5: **end if**
6: **else**
7: **if** (target set $A \subseteq T$) **then**
8: **if** $a_r = t - p_r + 1$ **then**
9: $S_{t+1}(A) := [(p_1, a_1), \ldots, (p_{r-1}, a_{r-1}), (p_r, a_r + 1)]$
10: **else**
11: $S_{t+1}(A) := [(p_1, a_1), \ldots, (p_r, a_r), (t+1, 1)]$
12: **end if**
13: **else**
14: $S_{t+1}(A) := S_t(A)$
15: $i := r$
16: **while** $i > 1$ **do**
17: **if** $\dfrac{a_i}{t - p_i + 1} \leq \dfrac{a_i + a_{i-1}}{t - p_{i-1} + 1}$ **then**
18: $a_{i-1} := a_{i-1} + a_i$
19: remove (p_i, a_i) from $S_{t+1}(A)$
20: $i := i - 1$
21: **else**
22: $i := 1$
23: **end if**
24: **end while**
25: **end if**
26: **end if**

1. A *target transaction* arrives in the stream (lines 7–12):
   (a) (lines 8–9) If the frequency of the last entry is 1, and hence, the previous transaction in the stream also contained the target set, we need to increment the number of occurrences in the last entry of the summary. Otherwise (lines 10–11), the previous transaction was a non-target transaction and thus a new border $(t+1, 1)$ needs to be added.
   (b) None of the existing borders can be removed from the summary.
2. A *non-target transaction* arrives in the stream (lines 13–25):
   (a) No new borders need to be added to the summary.
   (b) This is the only case in which borders can actually be *removed* from the summary. Therefore, according to Theorem 5.3.2, we have to compare the frequencies of every

two blocks adjacent to a border. That is, for a certain border p to be dropped, we have to find a before-block and an after-block such that the before-block has a higher frequency for the target set than the after-block. In reality, we do not have to physically check every two combinations of before-blocks and after-blocks for p. It is enough to pick the before-block with the highest frequency, and the only new after-block that contains the newly entered information, and check if the first is smaller than the latter. If this is not the case, p is not a border anymore and has to be dropped. It is simple to see why these two frequencies are the only frequencies that have to be considered. Obviously, the before-block with the highest frequency is exactly the block represented by the border before p. Indeed, at time point p, that particular border represented the maximum window, according to Property 5.4.2. We now only have to compare this frequency with the frequency from p until the current time point t. Indeed, any other after-block with a lower frequency would have caused the border p to have been removed earlier.

Furthermore, we do not have to consider every border p for removal independently, but, as stated in the following Property 5.4.4, only the most recent borders need to be considered for removal. In other words, if a border can not be removed, then all earlier borders can not be removed either, and hence, we must only consider the removal of borders from right to left, until one can not be removed (lines 16–24).

To summarize, we can only have a new border in the summary when a target transaction enters the stream, and then only in the special case that the previous transaction was a non-target transaction. In this situation, no borders will be dropped. Dropping can only be the case when a non-target transaction enters the stream. In this case, no borders can be added, the old summary has to be corrected according to the new information and eventually, borders can be removed if the increasing property of the summary is violated.

Definition 5.4.3. *Given $\mathbb{S}_t$, target set A and summaries $S_t(A) = [(p_1, a_1), \ldots, (p_r, a_r)]$ and $S_{t+1}(A) = [(p'_1, a'_1), \ldots, (p'_r, a'_r)]$. We use the notation $borders(S_t(A))$ to denote* the set of all borders $\{p_1, \ldots, p_r\}$ for target set A in $\mathbb{S}_t$ at time t.

If $borders(S_{t+1}(A)) \subset borders(S_t(A))$, than the borders that are dropped from the summary, i.e. the borders in $\{p_1, \ldots, p_r\} \setminus \{p'_1, \ldots, p'_r\}$, are called disappearing borders.

Property 5.4.4. *Let $\mathbb{S}_t$ be a stream with borders $p_1, \ldots, p_r$ for target set A. When, at a certain time stamp t', a border p_i is removed, also all borders p_j, with $1 < i < j \leq r$, disappear. That is, if p_i is not a border in the stream $\mathbb{S}_{t'}$ anymore, then neither are all p_j.*

Proof. To prove that if a border disappears, then also all borders to the right of this border, so all borders at higher and more recent positions, we assume that we have three borders p_1, p_2 and p_3, at time t'. The summary thus looks like: $S_{t'}(A) = [(p_1, a_1), (p_2, a_2), (p_3, a_3)]$. For simplicity, we denote $p_2 - p_1 = \ell_1$, $p_3 - p_2 = \ell_2$ and $t - p_3 + 1 = \ell_3$. We will show that if at time t' p_2 disappears, also p_3 will disappear and p_1 is the only border that is left in the summary. To prove is

$$\frac{a_1 + a_2 + a_3}{\ell_1 + \ell_2 + \ell_3} \geq \frac{a_2 + a_3}{\ell_2 + \ell_3} \Rightarrow \frac{a_2 + a_3}{\ell_2 + \ell_3} \geq \frac{a_3}{\ell_3} \ .$$

According to Lemma 5.3.1 (5.5), we thus have

$$\frac{a_1 + a_2 + a_3}{\ell_1 + \ell_2 + \ell_3} \geq \frac{a_2 + a_3}{\ell_2 + \ell_3} \Leftrightarrow \frac{a_1}{\ell_1} \geq \frac{a_2 + a_3}{\ell_2 + \ell_3} \ .$$

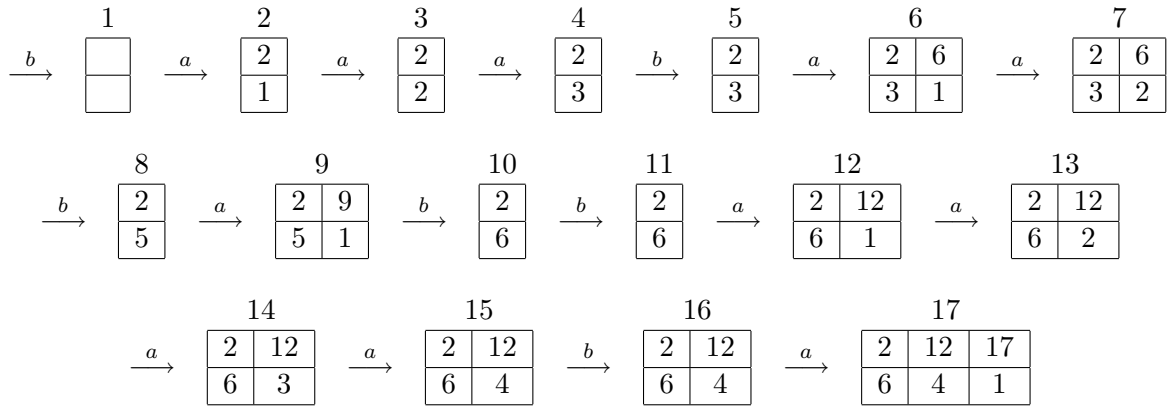


Figure 5.6: Example for stream $\langle b\ a\ a\ a\ b\ a\ a\ b\ a\ b\ b\ a\ a\ a\ a\ b\ a\rangle$

We know that $a_1/\ell_1 < a_2/\ell_2$. If we combine both results, we can find

$$\frac{a_2}{\ell_2} > \frac{a_2+a_3}{\ell_2+\ell_3} \Leftrightarrow \frac{a_2}{\ell_2} > \frac{a_3}{\ell_3} \Leftrightarrow \frac{a_2+a_3}{\ell_2+\ell_3} \geq \frac{a_3}{\ell_3} ,$$

based on Lemma 5.3.1 (5.5). □

Example 5.4.2. *An illustration of the algorithm is given for the following stream* $\mathbb{S}_{17} = \langle b\ a\ a\ a\ b\ a\ a\ b\ a\ b\ b\ a\ a\ a\ a\ b\ a\rangle$ *and target set* a*. For each time point, the summary* $S_t(a)$ *is illustrated in Figure 5.6.*

The algorithm starts with $S_0(a) = [\ ]$*. The stream now starts at time* $t = 1$ *with a transaction* $\{b\}$*, that does not contain the target set* $\{a\}$*. Therefore, at time stamp* 1

$$\mathit{Update}(S_0(a), \{b\}) = \mathit{Update}([\ ], \{b\}) = S_1(a) = \mathit{Summary}(a, \langle b\rangle) = [\ ] .$$

At time stamp 2, $\mathit{Update}([\ ], \{a\})$ *results in* $S_2(a) = [(2, 1)]$*, corresponding to the stream* $\langle b\ |a\rangle$ *with a border at position* 2 *and the corresponding frequency* $1/(2-2+1) = 1/1$*.*

At time stamp 8*, for the first time a border is dropped. We had* $S_7(a) = [(2, 3), (6, 2)]$*, corresponding with stream* $\langle b\ |a\ a\ a\ b\ |a\ a\rangle$*.* $\mathit{Update}(S_7(a), \{b\})$ *will yield* $S_8(a) = [(2, 5)]$*, and* not $[(2, 3), (6, 2)]$*. Indeed, because at time* $t = 8$ *the frequencies from the border at position* 2 *to the border at position* 6 *decrease, namely*

$$\frac{3+2}{8-2+1} = \frac{5}{7} > \frac{2}{8-6+1} = \frac{2}{3} ,$$

we can conclude that position 6 *is no longer a border at time* $t = 8$*. This is reflected in the summary* $S_8(a) = [(2, 3+2)]$ *and* $\langle b\ |a\ a\ a\ b\ a\ a\ b\rangle$*.*

5.4.3 Adaptations of the Stream Mining Algorithm

In this subsection, we will now consider a selection of adaptations of our basic Algorithm 8. First of all, it is possible to adjust the algorithm to take care of a *minimal frequency threshold*

σ and reproduce only those sets that have a frequency that is larger than σ[4]. Another possible adaptation is the introduction of a *minimal window length mwl* to settle up with peaks in the max-frequency. Of course, the *combination* of both σ and *mwl* is possible too. As a last important remark, we focus on mining *all* frequent sets in the stream.

Minimal Frequency Threshold

Based on our basic Algorithm 8, we are able to report the frequency of the target set exactly at any time point. Inspired by the frequent set mining problem in static data, we will now introduce a user-defined *minimal frequency threshold* σ. That is, for the target set, we should be able, at any time point, to produce its exact frequency only if it is *above* the minimal frequency threshold σ. This relaxation will allow us to decrease the size of the summary and make the tracking of all frequent sets over the stream even more feasible. Adding a minimal frequency threshold σ is an optimization that will make our algorithm more efficient.

According to the following Theorem 5.4.1, we can reuse the basic algorithm and we just have to check in the summary if the frequencies are above the minimal frequency threshold. If this is not the case, the corresponding border can be dropped from the summary.

Theorem 5.4.1. *Let $\mathbb{S}_t$ be a stream with summary $S_t(A) = [(p_1, a_1), \ldots, (p_r, a_r)]$ for target set A, and assume that we have a user-defined minimal frequency threshold σ. Suppose that*

$$\mathit{freq}(A, \mathbb{S}_t[p_1, t]) \ < \ \sigma \ .$$

We now can remove (p_1, a_1) from the summary. Even though it is possible that p_1 can still become the starting point of a maximal window in the future, it can never be the starting point of a maximal window in which the target set is above the threshold.

Proof. To show that p_1 can never be the starting point of a maximal window in which the target set is above the threshold σ, we will extend the stream $\mathbb{S}_t$ with $\mathbb{B}$. The number of occurrences of the target set A in $\mathbb{B}$ is denoted by b. For simplicity, we abbreviate $a_1 + \ldots + a_r := a_L$ and $t - p_1 + 1 := L$, and we introduce the notation $\sigma = X/Y$.

If we assume that $\mathit{freq}(A, (\mathbb{S}_t \cdot \mathbb{B})[p_1, |\mathbb{S}_t| + |\mathbb{B}|])$ exceeds the minimal threshold σ,

$$\mathit{freq}(A, (\mathbb{S}_t \cdot \mathbb{B})[p_1, |\mathbb{S}_t| + |\mathbb{B}|]) \ = \ \frac{a_L + b}{L + |\mathbb{B}|} \ > \ \frac{X}{Y} \ ,$$

then it is easy to show that $\mathit{freq}(A, \mathbb{B})$ must be even larger, and hence p_1 is *not* the maximal border. Indeed, for the sake of contradiction, assume that

$$\mathit{freq}(A, \mathbb{B}) = \frac{b}{|\mathbb{B}|} \ < \ \frac{X}{Y} \ \Leftrightarrow \ bY \ < \ |\mathbb{B}|X \ .$$

Because it is assumed that we are in the situation in which $\frac{a_L}{L} < \frac{X}{Y}$, it is easy to see that

$$a_L Y + bY \ < \ LX + |\mathbb{B}|X \ \Leftrightarrow \ \frac{a_L + b}{L + |\mathbb{B}|} \ < \ \frac{X}{Y} \ ,$$

[4]Remark that we have a slightly other interpretation of σ, compared to the previous chapters. Indeed, σ is no longer an absolute count of occurrences as introduced in Chapter 1, but a percentage. It is used to expressing a minimal frequency count in the stream. As such, the σ used in this chapter compares with the relative approach for σ, mentioned in Chapter 1.

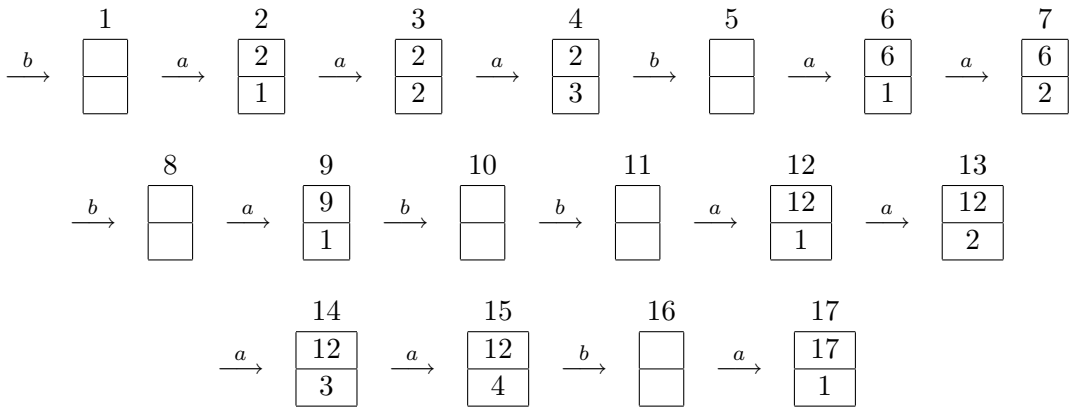


Figure 5.7: Example for stream $\langle b\ a\ a\ a\ b\ a\ a\ b\ a\ b\ b\ a\ a\ a\ a\ b\ a\rangle$, with minimal frequency threshold 85%

which is in contradiction with our assumption. This thus shows that $b/|\mathbb{B}| \geq \sigma$. Even more, we have $b/|\mathbb{B}| \geq a_L + b/(L + |\mathbb{B}|)$. Indeed, if we assume that this is not true, we have

$$\frac{b}{|\mathbb{B}|} < \frac{a_L + b}{L + |\mathbb{B}|} \Leftrightarrow \frac{b}{|\mathbb{B}|} < \frac{a_L}{L} < \sigma \ ,$$

which is in contradiction with our previous result. Therefore, indeed $\mathit{freq}(A, \mathbb{B})$ is larger than the minimal frequency threshold σ and p_1 is not the maximal border. It can be removed safely from the summary. □

Example 5.4.3. *An illustration is given in Figure 5.7. For the following stream* $\mathbb{S}_{17} = \langle b\ a\ a\ a\ b\ a\ a\ b\ a\ b\ b\ a\ a\ a\ a\ b\ a\rangle$*, target set* a *and* $\sigma = 85\%$*, the summary* $S_t(a)$ *is computed for each time point.*

In order to be able to perform this pruning efficiently in our algorithm, we store and maintain for the summaries also the total count $\mathit{total} = a_L = a_1 + a_2 + \ldots + a_r$. When the left-most border is pruned, *total* is decreased by a_1 to reflect the new total.

Minimal Window Length

We already mentioned the one major disadvantage of our new frequency measure *maxfreq*() over the more traditional fixed-window approaches. The max-frequency of a set A in a stream that ends with a target transaction is always 100%, independently of the overall frequency of A. Hence, even in a stream where A is extremely rare, at some points, the max-frequency will be maximal. At every time stamp where A appears, suddenly, the frequency will jump up to 1, and after that quickly go down again. In many applications such peaks are undesirable. Obviously, the key problem causing the peaks is that when a target transaction arrives in the stream, the window with the maximal frequency will become the window of length 1. This undesirable effect of max-frequency of leading to a frequency of 100% in a window of length 1 every time the target arrives in the stream, can therefore easily be resolved by setting a minimum $\mathit{mwl} > 1$ on the length that all windows must obey.

Given a minimal window length *mwl*, i.e. an explicit lower bound on the size of the windows in which the frequencies are considered, we can adjust our new frequency measure as follows. At first sight, this will lead to a new algorithm, but we will show that it turns out to be an adaptation of the basic algorithm.

Definition 5.4.4. *The* max-frequency under minimal window length *mwl of a set I in the stream $\mathbb{S}$, denoted $maxfreq_{mwl}(I, \mathbb{S})$, is defined as the maximum of the frequencies of I over all windows of size* at least *mwl, extending to the end of $\mathbb{S}$; that is:*

$$maxfreq_{mwl}(I, \mathbb{S}) := \max_{k=mwl..|\mathbb{S}|}(freq(I, last(k, \mathbb{S}))) \ .$$

If the length of the stream $\mathbb{S}$ is less than mwl, the max-frequency is defined to be 0.

Example 5.4.4. *We compute the max-frequency under $mwl = 3$ of target set a.*

$$\begin{aligned}
maxfreq_3(a, \langle a\ b\ a\ a\ a\ b\rangle) &= \max_{k=3..6}(freq(a, last(k, \langle a\ b\ a\ a\ a\ b\rangle))) \\
&= \max\left(\frac{2}{3}, \frac{3}{4}, \frac{3}{5}, \frac{4}{6}\right) = \frac{3}{4} \\
maxfreq_3(a, \langle b\ c\ d\ a\ b\ c\ d\ a\rangle) &= \max\left(\frac{1}{3}, \frac{1}{4}, \frac{2}{5}, \frac{2}{6}, \frac{2}{7}, \frac{2}{8}\right) = \frac{2}{5} \\
maxfreq_3(a, \langle x\ a\ a\ x\ a\ a\ x\rangle) &= \max\left(\frac{2}{3}, \frac{2}{4}, \frac{3}{5}, \frac{4}{6}, \frac{4}{7}\right) = \frac{2}{3} \\
maxfreq_3(ab, \langle ab\ cd\ ef\ abgh\ cdp\rangle) &= \max\left(\frac{1}{3}, \frac{1}{4}, \frac{2}{5}\right) = \frac{2}{5}
\end{aligned}$$

In Figure 5.8 the evolution of max-frequency for target set $\{a\}$ in the indicated stream of length 65 has been given for different minimal window lengths. On every time stamp, a new singleton transaction enters the stream and the max-frequency of the target set is plotted for three different values of the minimal window length. If we would have plotted $mwl = 1$, we are in the case of our basic algorithm and it is clear from our previous analysis that there are high jumps in the max-frequency every time set $\{a\}$ occurs in the stream. For $mwl = 3$, $mwl = 5$ and $mwl = 10$, the illustration shows that there are still jumps in the frequency, but they are far less pronounced. Hence, setting an appropriate minimal window length effectively resolves the instability of the max-frequency measure.

Now we have introduced the max-frequency with a minimum window length, it is very important to consider the *pruning criterion* that will be used in the algorithm. In the algorithm *without* minimal window length, we used the fact that a border q in stream $\mathbb{S}$ can be pruned if we can find, for any possible value of $p < q$ and $q \leq r$, two blocks $\mathbb{B}_1 = \mathbb{S}[p, q-1]$ and $\mathbb{B}_2 = \mathbb{S}[q, r]$ such that the frequency of the target in $\mathbb{B}_1$ is higher that in $\mathbb{B}_2$ (Theorem 5.3.2). The intuition behind the proof of this theorem was that in such a situation, q can never become a border again, because either the window starting at p will have higher frequency, or the window starting at $r+1$ has. When we are working with a minimal window length, however, this observation does *no longer* imply that q can be pruned. Indeed; it could be the case that the suffix of the stream starting at $r+1$ does *not* meet the minimal window length requirement. In that case, even though the window starting at q has a lower frequency for the

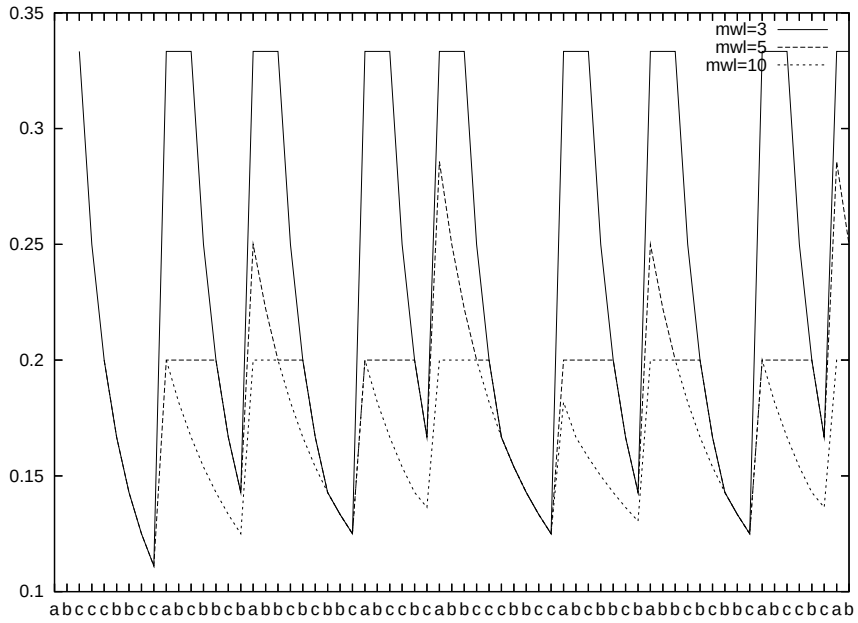


Figure 5.8: Evolution of max-frequency for target a in the given stream, for minimal window lengths 3, 5, and 10

target than the window starting at $r+1$, it can still have the highest frequency of all windows *that meet the minimal window requirement.* The next example illustrates this situation.

Example 5.4.5. *Consider target set* $\{a\}$ *and stream* $\mathbb{S} = \langle |a\ a\ a\ b\ |a\ a\rangle$ *in which the borders at positions* 1 *and* 5 *are marked with a vertical bar, according to the algorithm without minimal window length. When transaction* $\{b\}$ *arrives in the stream, resulting in* $\langle |a\ a\ a\ b\ a\ a\ b\rangle$*, then position* 5 *is no longer a border for target* a*, as the block* $a\ a\ a\ b$ *before position* 5 *has a higher frequency of the target set than the block* $a\ a\ b$ *after position* 5*. Therefore, in the algorithm without minimal window length, the border at position* 5 *is pruned, because no matter how the stream evolves, position* 5 *will never be a border again for target set* $\{a\}$*.*

However, consider now the case where we do have a minimal window length of size 3*. Then, position* 5 *can* still *become a border again for the target* $\{a\}$*. Indeed, suppose two more target transactions are added to the stream, resulting in:* $\langle |a\ a\ a\ b\ .a\ a\ b\ |a\ a\rangle$*, when we do not take care of mwl. But, if we do take care of mwl, in this stream, the window starting at position* 5 *has the highest frequency of the target item among the windows satisfying the minimal window length. Therefore, this position* 5 *cannot be dropped.*

Because the pruning criterion for borders given in Theorem 5.3.1 is no longer valid with minimal window length, we will use the following, slightly weaker pruning criterion:

Theorem 5.4.2. *Let* $\mathbb{S}$ *be a stream of length* L*, and let* $\mathbb{S}[q, L]$ *be the maximal window of length at least mwl for the target set* A*. Then, for any* p*,* r *with* $p < q \leq r \leq L - mwl$*, it holds that*

$$freq(A, \mathbb{S}[p, q-1]) \ < \ freq(A, \mathbb{S}[q, r]) \ .$$

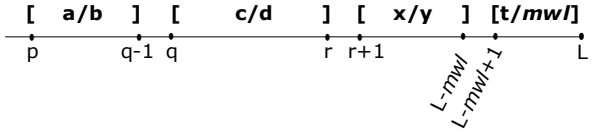
Proof. The proof is very similar to the proof of Theorem 5.3.1. Let $\mathbb{B}_1$ denote $\mathbb{S}[p, q-1]$, $\mathbb{B}_2$ denote $\mathbb{S}[q, r]$, and $\mathbb{B}_3$ denote $\mathbb{S}[r+1, L]$. Notice that, because of the extra requirement $r \leq L - mwl$, $\mathbb{B}_3$ obeys the minimal window length constraint. Indeed, $|\mathbb{B}_3| = L-(r+1)+1 = L - r \geq mwl$. Because $\mathbb{B}_2 \cdot \mathbb{B}_3$ is the maximal window for A in $\mathbb{S}$, it holds that the frequency of A in $\mathbb{B}_2 \cdot \mathbb{B}_3$ is strictly higher than in $\mathbb{B}_1 \cdot \mathbb{B}_2 \cdot \mathbb{B}_3$ and at least as high as in $\mathbb{B}_3$. From this point on, the proof is the same as for Theorem 5.3.1. □

The following observation gives us a solution for the algorithm to proceed when dealing with a certain minimal window length *mwl*.

Observation 5.4.1. *Let $\mathbb{S}$ be a stream of length L, and let mwl be the minimal window length. Let $\mathbb{S}[q, L]$ be the maximal window of length at least mwl for the target set A. Then*

- *either $q = L - mwl + 1$, and therefore the maximal window covers $last(mwl, \mathbb{S})$,*
- *or, q is a border in $\mathbb{S}[1, L - mwl]$.*

Proof. First of all, because the length of the maximal window is at least *mwl*, we have that $q \leq L - mwl + 1$. We now can have that $q = L - mwl + 1$ or $q < L - mwl + 1$. In the latter case, we have to show that q is a border in $\mathbb{S}[1, L - mwl]$. According to Theorem 5.3.2, we have to prove that the frequency of target A in every block before position q is strictly less than the frequency in every block following q. More concrete, we pick a certain $1 \leq p < q$ and we denote the number of occurrences of target A in $\mathbb{S}[p, q-1]$ by a and we use the shorthand notation b for the length of this substream, $q - p$. We also pick a certain $q \leq r \leq L - mwl$ and we denote the number of occurrences of the target A in $\mathbb{S}[q, r]$ by c and we introduce the shorthand notation d for the length $r - q + 1$. Finally, the number of occurrences of the target A in $\mathbb{S}[r+1, L - mwl]$ are denoted by x and y is used to express the length of this substream, $L - mwl - r$. Remark that in the case of $q = L - mwl$ or $q < L - mwl$ but $r = L - mwl$, $x = 0$ and $y = 0$. We also denote the number of occurrences of the target A in $last(mwl, \mathbb{S})$ by t, and the length of this last substream equals *mwl*.



It is sufficient to show that $freq(A, \mathbb{S}[p, q-1]) < freq(A, \mathbb{S}[q, r])$, i.e. $a/b < c/d$. Because q is the starting point of the maximal window of length at least *mwl* in $\mathbb{S}$, we know that

$$\frac{c+x+t}{d+y+mwl} \geq \frac{a+c+x+t}{b+d+y+mwl} \quad \text{and} \quad \frac{c+x+t}{d+y+mwl} \geq \frac{x+t}{y+mwl} ,$$

which is equivalent with

$$\frac{c+x+t}{d+y+mwl} \geq \frac{a}{b} , \tag{5.14}$$

and

$$\frac{c}{d} \geq \frac{x+t}{y+mwl} , \tag{5.15}$$

Stream	Summary	Max-frequency for target a
$\mathbb{S}_4 = \langle\|a\ \underline{a\ a\ b}\rangle$	[(1,1)]	$\max(2/3, 3/4) = 3/4$
$\mathbb{S}_5 = \langle\|a\ a\ \underline{a\ b\ a}\rangle$	[(1,2)]	$\max(2/3, 4/5) = 4/5$
$\mathbb{S}_6 = \langle\|a\ a\ a\ \underline{b\ a\ a}\rangle$	[(1,3)]	$\max(2/3, 5/6) = 5/6$
$\mathbb{S}_7 = \langle\|a\ a\ a\ b\ \underline{a\ a\ b}\rangle$	[(1,3)]	$\max(2/3, 5/7) = 5/7$
$\mathbb{S}_8 = \langle\|a\ a\ a\ b\ \|a\ \underline{a\ b\ a}\rangle$	[(1,3), (5,1)]	$\max(2/3, 3/4, 6/8) = 3/4$
$\mathbb{S}_9 = \langle\|a\ a\ a\ b\ \|a\ a\ \underline{b\ a\ a}\rangle$	[(1,3), (5,2)]	$\max(2/3, 4/5, 7/9) = 4/5$
$\mathbb{S}_{10} = \langle\|a\ a\ a\ b\ a\ a\ b\ \underline{a\ a\ a}\rangle$	[(1, 5)]	$\max(3/3, 8/10) = 1$

Figure 5.9: Maintenance of the summary and *maxfreq*() by a minimal window length of 3

according to Lemma 5.3.1. Assume now for the sake of contradiction that $a/b \geq c/d$. We then have, according to (5.14) and Lemma 5.3.1, that

$$\frac{c+x+t}{d+y+mwl} \geq \frac{a}{b} \geq \frac{c}{d} \Rightarrow \frac{x+t}{y+mwl} \geq \frac{c}{d} ,$$

which is not possible, as shown in (5.15). Therefore, the assumption is wrong, illustrating that $a/b < c/d$. □

Based on the above observation, for the algorithm to proceed, we assume that the size of the minimal window length and the contents of the most recent window of minimal window length are kept in main memory. Therefore, the number of occurrences of the target set in the last *mwl* transactions is known. In practice, for a stream $\mathbb{S}$ of length L and a minimal window length *mwl*, we no longer keep the frequencies of the complete stream but only for $\mathbb{S}[1, L - mwl]$. Hence, the summary will always be exactly *mwl* sets behind the current situation and we can reuse our basic algorithm, as we did before. Of course, when the length of the stream does not exceed the minimal window length, it does not make sense to analyze the stream. The advantage of working this way is that we can continue pruning borders in the summary like we did before, without worrying about the *mwl*. Indeed; every pair of blocks that needs to be tested according to Theorem 5.4.2, are also blocks in $\mathbb{S}[1, L - mwl]$. Thus, applying Theorem 5.4.2 on $\mathbb{S}$ is exactly the same as applying Theorem 5.3.1 on $\mathbb{S}[1, L - mwl]$. To find the maximal frequency when taking care of minimum window length *mwl*, we have to make a scan over the summary, and take into account the *mwl* transactions not summarized but kept in main memory.

Theorem 5.4.3. *Let $\mathbb{S}$ be a stream of length L, with minimal window length mwl. For target set A, we have Summary$(A, \mathbb{S}[1, L - mwl]) = [(p_1, a_1), \ldots, (p_r, a_r)]$. Then,*

$$maxfreq_{mwl}(A, \mathbb{S}) = \max(freq(A, last(mwl, \mathbb{S})), freq(A, \mathbb{S}[p_r, L]), \ldots, freq(A, \mathbb{S}[p_1, L])) \ .$$

When we need the max-frequency for target set A when taking care of the minimum window length *mwl*, we compute the frequency of A for all border positions from $\mathbb{S}[1, L - mwl]$ in the complete stream $\mathbb{S}$, together with the frequency of A in the minimum window itself, $last(mwl, \mathbb{S})$. We illustrate the algorithm in the case of $mwl = 3$.

Example 5.4.6. *Consider stream $\langle a\ a\ a\ b\ a\ a\ b\ a\ a\ a\rangle$ of length 10 and $mwl = 3$. An overview of the summaries for target set a for each time stamp can be found in Figure 5.9. The underlined parts are kept in memory. The max-frequency is computed for target a.*

Stream	Summary	Max-frequency for target a
$\mathbb{S}_4 = \langle \vert a\ \underline{a\ a\ b} \rangle$	[(1,1)]	3/4
$\mathbb{S}_5 = \langle \vert a\ a\ \underline{a\ b\ a} \rangle$	[(1,2)]	4/5
$\mathbb{S}_6 = \langle \vert a\ a\ a\ \underline{b\ a\ a} \rangle$	[(1,3)]	5/6
$\mathbb{S}_7 = \langle \vert a\ a\ a\ b\ \underline{a\ a\ b} \rangle$	[(1,3)]	5/7
$\mathbb{S}_8 = \langle \vert a\ a\ a\ b\ \vert a\ \underline{a\ b\ a} \rangle$	[(1,3), (5,1)]	**6/8**
$\mathbb{S}_9 = \langle \vert a\ a\ a\ b\ \vert a\ a\ \underline{b\ a\ b} \rangle$	[(1,3), (5,2)]	**6/9**
$\mathbb{S}_{10} = \langle \vert a\ a\ a\ b\ a\ a\ b\ \underline{a\ b\ b} \rangle$	[(1, 5)]	6/10

Figure 5.10: *maxfreq*() is not always determined by the last summary-entry, in case of *mwl*

Remark 5.4.1. *Property 5.4.3 is not valid in the case of stream mining when taking care of a minimal window length.*

Proof. Given a stream $\mathbb{S}$, a target set A and summary $S_t(A)$ based on minimal window length *mwl*, it is clear that Property 5.4.3 is valid for the substream $\mathbb{S}_t - mwl$. This means that the maximal frequency for stream $\mathbb{S}_t - mwl$ can be found in the last entry of the summary. For the complete stream up to time t in which *mwl* is included, this extra part can change the frequencies such that the maximal frequency is not determined by the last entry in the summary. For example, consider stream $\langle a\ a\ a\ b\ a\ a\ b\ a\ b\ b \rangle$. In Figure 5.10, an overview of the summaries for target set a is given and the max-frequency is computed. For example, from $\mathbb{S}_9$ it is clear that *maxfreq*() is not determined by the last entry in the summary. □

Remark 5.4.2. *In Theorem 5.4.3, it is not necessary to check* all *borders from the summary. It is sufficient to check where the increasing property of the frequencies is broken.*

Example 5.4.7. *Consider Figure 5.10, where we focus on target a.*

- *For* $\mathbb{S}_7$*, we have that* $\frac{3+2}{7-1+1} > \frac{2}{3}$*, leading to a max-frequency of* 5/7
- *For* $\mathbb{S}_8$*, we have that* $\frac{3+1+2}{8-1+1} = \frac{1+2}{8-5+1} > \frac{2}{3}$*, leading to a max-frequency of* 6/8
- *For* $\mathbb{S}_9$*, we have that* $\frac{3+2+1}{9-1+1} > \frac{2+1}{9-5+1} > \frac{1}{3}$*, leading to a max-frequency of* 6/9

Combination of Minimal Frequency Threshold and Minimal Window Length

We can now combine both minimal frequency σ and minimal window length *mwl*. We can expect that the minimal window length has an important influence on the pruning of summary entries based on the minimal frequency.

In the case without minimal window length but with minimal frequency σ, if we have to check whether we have to remove the last entry of a summary $[(p, a)]$ of a stream $\mathbb{S}_t$, we test whether $a/(t-p+1) < \sigma$, and the reasoning is that if this is the case, p can be removed as a border. This is because every extension $\mathbb{B}$ that would turn p into a maximal and frequent border, would be even more frequent itself (as illustrated in the proof of Theorem 5.4.1). If we now are in the case in which the minimal window length is taken into account, together

with the minimal frequency, this is no longer true, though, as this $\mathbb{B}$ could be too short, i.e. shorter than *mwl*. In this case, p might be the starting point of the maximal window *of length at least mwl* and can not be thrown away. We show that if this happens, it is possible by adding a block with only target transactions.

Theorem 5.4.4. *Given a stream $\mathbb{S}_t$, a summary $S_t(A) = [(p_1, a_1), \ldots, (p_r, a_r)]$ for target set A at time t based on minimal frequency threshold σ, and minimal window length mwl. If $a_r/(t - p_r + 1) < \sigma$, p_r is a border for A in $\mathbb{S}$ following the mwl requirement, meaning that p_r can still be the starting point of a maximal window of length at least mwl, if there exists a certain b with $0 \leq b < mwl$, such that*

$$(a_r + b)/(t - p_r + 1 + b) \geq b/mwl, \text{ and } (a_r + b)/(t - p_r + 1 + b) \geq \sigma \ .$$

Proof. We will construct an extension $\mathbb{B}$, such that p_r is the starting point of a maximal window of length at least *mwl* in stream $\mathbb{S}_t \cdot \mathbb{B}$. We therefore add a block $\mathbb{B}$ of length b with b times a target transaction. Then, p_r is a frequent border according to the minimal frequency threshold σ if

$$(a_r + b)/(t - p_r + 1 + b) \geq \sigma \ .$$

Furthermore, $\mathbb{B}$ must be shorter than the minimal window length *mwl*. Indeed, if $\mathbb{B}$ was larger than *mwl*, we would be in the situation comparable with the situation without *mwl* (Theorem 5.4.1). It also has to be the case that the minimal window itself must not become more frequent than $(a_r + b)/(t - p_r + 1 + b)$; otherwise p is not maximal. When we have such a block $\mathbb{B}$, the minimal window has at least b targets, and thus a frequency of at least b/mwl. Thus, p_r can become the maximal border again only if there exists a b with $0 \leq b < mwl$, such that $(a_r + b)/(t - p_r + 1 + b) \geq b/mwl$, and $(a_r + b)/(t - p_r + 1 + b) \geq \sigma$. □

Example 5.4.8. *Consider the following stream with $mwl = 2$ and $\sigma = 51\%$.*

Stream	Summary	Max-frequency for target a
$\mathbb{S}_3 = \langle \lvert a\ \underline{a\ b} \rangle$	[(1,1)]	$2/3 > \sigma$
$\mathbb{S}_4 = \langle \lvert a\ a\ \underline{b\ b} \rangle$	[(1,2)]	$2/4 < \sigma$
$\mathbb{S}_5 = \langle \lvert a\ a\ b\ \underline{b\ a} \rangle$	[(1,2)]	$3/5 > \sigma$

At time stamp 4, the max-frequency is 50%, which is smaller than σ, thus could suggest to throw away the border. But, at time stamp 5, we constructed an extension of the stream such that the border appears again. This happens because the extension $\langle a \rangle$ falls within the minimum window length.

Thus, again, we have to refine the pruning criterion. We can use the above observation in Theorem 5.4.4 to specify a new pruning condition.

Theorem 5.4.5. *Given a stream $\mathbb{S}_t$, a summary $S_t(A) = [(p_1, a_1), \ldots, (p_r, a_r)]$ for target set A, based on minimal window length mwl at time t, and a minimal frequency threshold σ. Position p_r can be pruned from the summary if*

$$(a_r + b_{min})/(t - p_r + 1 + b_{min}) < b_{min}/mwl \ ,$$

with b_{min} the smallest b that satisfies the condition $(a_r + b)/(t - p_r + 1 + b) \geq \sigma$.

Proof. The smallest b that satisfies $(a_r + b)/(t - p_r + 1 + b) \geq \sigma$ can be found by solving

$$\begin{aligned} & \frac{a_r + b}{t - p_r + 1 + b} \ \geq \ \sigma \\ \Leftrightarrow \quad & a_r + b \ \geq \ \sigma(t - p_r + 1) + \sigma b \\ \Leftrightarrow \quad & b(1 - \sigma) \ \geq \ \sigma(t - p_r + 1) - a_r \\ \Leftrightarrow \quad & b \ \geq \ \frac{\sigma(t - p_r + 1) - a_r}{1 - \sigma} = \frac{a_r - \sigma(t - p_r + 1)}{\sigma - 1} \end{aligned}$$

and equals

$$b_{min} = \frac{a - mwl(t - p + 1)}{mwl - 1} \ .$$

According to Theorem 5.4.4, it is clear that p_r can be pruned if

$$(a_r + b_{min})/(t - p_r + 1 + b_{min}) \ < \ b_{min}/mwl \ .$$

□

Mining All Sets

Until now, we merely focused on mining a single frequent target set. However, the goal of stream mining is to find *all* sets that are frequent in the stream. We thus can refine our problem statement.

> ***Stream Mining with Flexible Windows***
> *Given a minimal frequency threshold σ and a minimal window length mwl, for an evolving stream $\mathbb{S}$, maintain a small summary of the stream in time, such that, at any time point t, all current frequent sets can be produced instantly from the summary.*

A straightforward way to do this is to apply Algorithm 8 for all sets at the same time and keep different summaries for each of the sets at the same time. That is, for every set we maintain a summary for the stream minus the last mwl transactions, together with the last mwl transactions in memory. Of course, this is impossible to do for all sets. Fortunately, this naive solution can be improved using the following observations.

Instead of remembering summaries for all sets, the idea is to only consider sets that are *actually frequent*. More specifically, when the minimum window length is equal to 1, every subset of a transaction entering the stream is frequent. From there on, we simply need to keep track of the max-frequency of each of these sets using Algorithm 8, and we can remove a set with its summary as soon as it is infrequent. As such, we only remember summaries of sets that are frequent. When the minimum window length is strictly larger than 1, however, this is not valid anymore. It is not sufficient to only consider summaries for those sets that are frequent within the minimum window, according to the traditional definition of frequency. In order to know these frequent sets within the minimum window, many efficient incremental algorithms have already been proposed in the literature [54]. Interesting further research could study the behavior of the combination of our algorithm with one of these new algorithms and conclude if optimizations are possible. However, when we only consider the sets that are frequent within the minimum window, we do miss sets from consideration.

Example 5.4.9. *Assume* $mwl = 3$ *and* $\sigma = 49\%$. *Consider* $\mathbb{S}_4 = \langle a\ \underline{b\ b\ a}\rangle$. *Within the minimum window, the frequency of target* a *is* $1/3 < \sigma$ *and we therefore are not willing to compute the summary for* a. *However,* $maxfreq(a, \mathbb{S}_4) = 2/4 > \sigma$, *meaning that* a *is frequent in the stream, and thus it is necessary to remember the summary.*

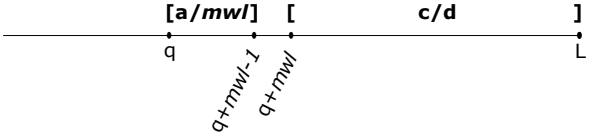
Therefore, it is important to find another criterium to discover when to start remembering a summary for a certain set. As a result, we have the following observation.

Observation 5.4.2. *Let* $\mathbb{S}$ *be a stream of length* L *and let* mwl *be the minimal window length and* σ *the minimal frequency threshold. We focus on target set* A. *Suppose that* $maxfreq_{mwl}(A, \mathbb{S}) \geq \sigma$. *If* $\mathbb{S}[q, L]$ *is the maximal window of length at least* mwl *for* A, *then,*

- *either* $L - 2mwl + 2 \ \leq\ q \ \leq\ L - mwl + 1$,
- *or, the following conditions are all fulfilled:*
  - $freq(A, \mathbb{S}[q, q + mwl - 1]) \ \geq\ \sigma$
  - $maxfreq(A, \mathbb{S}[1, L - mwl]) \ \geq\ \sigma$
  - q *is a border in* $\mathbb{S}[1, L - mwl]$

Proof. First of all, because the length of the maximal window is at least mwl, we have that $q \leq L - mwl + 1$. We now can have that $q > L - 2mwl + 1$ or $q \leq L - 2mwl + 1$. The first case, $q > L - 2mwl + 1$, leads to the situation $L - 2mwl + 2 \ \leq\ q \ \leq\ L - mwl + 1$. In the case that $q \leq L - 2mwl + 1$, we have to prove the above three statements.

Assume for the sake of contradiction that $freq(A, \mathbb{S}[q, q + mwl - 1]) < \sigma$. We denote the number of occurrences of the target set A in $\mathbb{S}[q, q + mwl - 1]$ with a. We thus have that $a/mwl < \sigma$. We also denote the number of occurrences of A in $\mathbb{S}[q + mwl, L]$ by c and the length of $\mathbb{S}[q + mwl, L]$ by d.

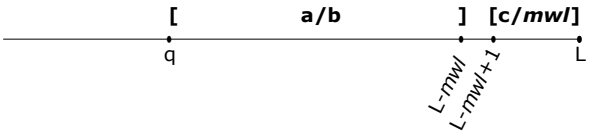


Because q is the starting point of the maximal window of length at least mwl for target A in $\mathbb{S}$, we have that $\dfrac{a + c}{mwl + d} \ \geq\ \sigma$. Combining these results, together with Lemma 5.3.1 gives

$$\frac{a + c}{mwl + d} \ \geq\ \sigma \ > \frac{a}{mwl} \ \Rightarrow\ \frac{c}{d} \ >\ \frac{a}{mwl} \ \Leftrightarrow\ \frac{c}{d} \ >\ \frac{a + c}{mwl + d}\ .$$

This result shows that target set A is even more frequent in $\mathbb{S}[q + mwl, L]$ than it is in $\mathbb{S}[q, L]$, but this is not possible because $\mathbb{S}[q, L]$ is the maximal window for A. Our assumption is therefore false, reflecting that $a/mwl \geq \sigma$, thus $freq(A, \mathbb{S}[q, q + mwl - 1]) \geq \sigma$.

The next statement that we have to prove is that $maxfreq(A, \mathbb{S}[1, L - mwl]) \geq \sigma$. We will show that $freq(A, \mathbb{S}[q, L - mwl]) \geq \sigma$, and hence $maxfreq(A, \mathbb{S}[1, L - mwl]) \geq \sigma$, because the latter can only be larger than any other frequency count. We therefore denote the number of occurrences of A in $\mathbb{S}[q, L - mwl]$ by a, and the length of $\mathbb{S}[q, L - mwl]$ by b. The number of occurrences of A in stream $last(mwl, \mathbb{S})$ are denoted by c and the length of this last substream equals mwl.

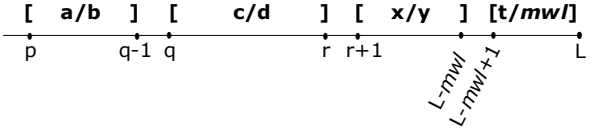


We know that $maxfreq_{mwl}(A, \mathbb{S}) \geq \sigma$, i.e. $\frac{a+c}{b+mwl} \geq \sigma$, and we have to show that the frequency of A in $\mathbb{S}[q, L-mwl]$, i.e. a/b, is also greater than or equal to σ. Therefore, for the sake of contradiction, we assume that $a/b < \sigma$. We then have, based on Lemma 5.3.1,

$$\frac{a+c}{b+mwl} \geq \sigma > \frac{a}{b} \Rightarrow \frac{c}{mwl} > \frac{a}{b} \Leftrightarrow \frac{c}{mwl} > \frac{a+c}{b+mwl} .$$

This result shows that A is even more frequent in $last(mwl, \mathbb{S})$ than it is in $\mathbb{S}[q, L]$, which is not possible because q is the starting point of the maximal window for A in $\mathbb{S}$. Hence, our assumption is wrong, meaning that $a/b \geq \sigma$.

We will now show that q is a border in $\mathbb{S}[1, L-mwl]$. According to Theorem 5.3.2, we have to prove that the frequency of target A in every block before position q is strictly less than the frequency in every block following q. More concrete, we pick a certain $1 \leq p < q$ and we denote the number of occurrences of target A in $\mathbb{S}[p, q-1]$ by a and we use the shorthand notation b for the length of this substream, $q-p$. We also pick a certain $q \leq r \leq L-mwl$ and we denote the number of occurrences of the target A in $\mathbb{S}[q, r]$ by c and we introduce the shorthand notation d for the length $r-q+1$. Finally, the number of occurrences of the target A in $\mathbb{S}[r+1, L-mwl]$ are denoted by x and y is used to express the length of this substream, $L-mwl-r$. Remark that in the case of $r = L-mwl$, $x = 0$ and $y = 0$. We also denote the number of occurrences of the target A in $last(mwl, \mathbb{S})$ by t, and the length of this last substream equals mwl.



It is sufficient to show that $freq(A, \mathbb{S}[p, q-1]) < freq(A, \mathbb{S}[q, r])$, i.e. $a/b < c/d$. Because q is the starting point of the maximal window of length at least mwl in $\mathbb{S}$, we know that

$$\frac{c+x+t}{d+y+mwl} \geq \frac{a+c+x+t}{b+d+y+mwl} \quad \text{and} \quad \frac{c+x+t}{d+y+mwl} \geq \frac{x+t}{y+mwl} ,$$

which is equivalent with

$$\frac{c+x+t}{d+y+mwl} \geq \frac{a}{b} , \tag{5.16}$$

and

$$\frac{c}{d} \geq \frac{x+t}{y+mwl} , \tag{5.17}$$

according to Lemma 5.3.1. Assume now for the sake of contradiction that $a/b \geq c/d$. We then have, according to (5.16) and Lemma 5.3.1, that

$$\frac{c+x+t}{d+y+mwl} \geq \frac{a}{b} \geq \frac{c}{d} \Rightarrow \frac{x+t}{y+mwl} \geq \frac{c}{d} ,$$

which is not possible, as shown in (5.17). Therefore, the assumption is wrong, illustrating that $a/b < c/d$. □

Observation 5.4.2 shows that we do not need to maintain the summaries of all sets, but only of those that (1) were once frequent in the minimal window (first condition of the second case), and (2) are frequent now in $\mathbb{S}[1, L - mwl]$ (second condition of the second case). Furthermore, we need to find the frequent sets in the mwl windows $\mathbb{S}[L - 2mwl + 1, L]$, $\mathbb{S}[L - 2mwl + 2, L]$, $\ldots$, $\mathbb{S}[L - mwl, L]$.

It is also important to know when a summary can be thrown away. The following theorem shows us when this is possible.

Theorem 5.4.6. *Given a stream $\mathbb{S}$ of length L and let mwl be the minimal window length and σ the minimal frequency threshold. We focus on target set A and have a summary $S_t^{-mwl}(A)$ for A in $\mathbb{S}[1, L-mwl]$. Suppose that the max-frequency of A in $\mathbb{S}[1, L-mwl]$ is not frequent any more, i.e. $maxfreq_{mwl}(A, \mathbb{S}[1, L - mwl]) = freq(A, \mathbb{S}[p_i, L - mwl]) < \sigma$ for one of the borders p_i from the summary $S_t^{-mwl}(A)$. This border will never contribute to the computation of the maximal frequency again, and therefore, the summary can be dropped from consideration.*

Proof. We will show that if this border p_i will be frequent in the future again, there is another more recent border that has a higher frequency and that is therefore important to find the maximal frequency. Therefore, we assume that stream $\mathbb{S}_t$ is extended with a certain new stream $\mathbb{B}$. We denote the number of occurrences of A in $\mathbb{S}[p_i, L - mwl]$ by a_i, and use the shorthand ℓ_i to denote the length of $\mathbb{S}[p_i, L - mwl]$. The number of occurrences of target set A in $\mathbb{B}$ is denoted by b. We introduce the notation $\sigma = X/Y$.

If we assume that $freq(A, (\mathbb{S} \cdot \mathbb{B})[p_i, L + |\mathbb{B}|])$ exceeds the minimal threshold σ,

$$freq(A, (\mathbb{S} \cdot \mathbb{B})[p_i, L + |\mathbb{B}|]) \;=\; \frac{a_i + b}{\ell_i + |\mathbb{B}|} \;\geq\; \frac{X}{Y} \;,$$

then it is easy to show that $freq(A, \mathbb{B})$ must be even larger, and hence p_i will never be the maximal border anymore. Indeed, for the sake of contradiction, assume that

$$freq(A, \mathbb{B}) = \frac{b}{|\mathbb{B}|} \;<\; \frac{X}{Y} \;\Leftrightarrow\; bY \;<\; |\mathbb{B}|X \;.$$

Because it is assumed that we are in the situation in which $\frac{a_i}{\ell_i} < \frac{X}{Y}$, it is easy to see that

$$a_iY + bY \;<\; \ell_iX + |\mathbb{B}|X \;\Leftrightarrow\; \frac{a_i + b}{\ell_i + |\mathbb{B}|} \;<\; \frac{X}{Y} \;,$$

which is in contradiction with our assumption. This thus shows that $b/|\mathbb{B}| \geq \sigma$. Even more, we have $b/|\mathbb{B}| \geq a_i + b/(\ell_i + |\mathbb{B}|)$. Indeed, if we assume that this is not true, we have

$$\frac{b}{|\mathbb{B}|} \;<\; \frac{a_i + b}{\ell_i + |\mathbb{B}|} \;\Leftrightarrow\; \frac{b}{|\mathbb{B}|} \;<\; \frac{a_i}{\ell_i} \;<\; \sigma \;,$$

which is in contradiction with our previous result. Therefore, indeed $freq(A, \mathbb{B})$ is larger than the minimal frequency threshold σ and p_i will never be the maximal border anymore. The summary can therefore be safely dropped from consideration. □

We now have the following algorithm to update the summary when a new transaction enters the stream. For every set A for which we are already maintaining a summary, we update its summary based on the information from the transaction that leaves the minimal window. Remark that we indeed are mwl behind in mining the stream. We now check if the max-frequency of A in the part of the stream without the minimal window, $\mathbb{S}[1, L - mwl]$, is still frequent. If this is not the case, we remove the summary, according to Theorem 5.4.6. Then, for all items that are frequent in the minimal window, and for which we are not yet maintaining a summary, we start a summary. By doing so, we guarantee that we are able to capture all maximal windows with $q \leq L - 2mwl$. Furthermore, we always keep the last $2mwl$ transactions in memory. Now, when all frequent sets are required, we generate all frequent sets from the summaries, plus all sets that are frequent in one of the windows $\mathbb{S}[L - 2mwl + 1, L]$, $\mathbb{S}[L - 2mwl + 2, L]$, ..., $\mathbb{S}[L - mwl, L]$. The goal is to find frequent sets in all windows. A straightforward way to do this is to perform mwl times frequent set mining, based on existing efficient incremental algorithms [54], which can be very costly. In future work, it is possible to think about how to improve the performance with an adaptation to these algorithms that have already been proposed in literature.

5.5 Worst-Case Analysis

We already mentioned that the amount of borders can possibly be unlimited. In this section we study how *large* the summary can become in worst case. For a specific stream length ℓ, we will identify a stream of this length that *maximizes* the number of borders. Farey sequences play an important role in this analysis. Before we will actually compute bounds on the size of the summary, we will introduce the mathematical background.

5.5.1 Farey Streams

Definition 5.5.1. *Consider a stream $\mathbb{S}$ of length L in which we have N borders for target set A. Assume that the N blocks separated by these borders have lengths $\ell_1, \ldots, \ell_N$, and contain respectively $a_1, \ldots, a_N$ times the target A:*

$$\langle \;\big|\; \boxed{a_1/\ell_1} \;\big|\; \boxed{a_2/\ell_2} \;\big|\; \ldots \;\big|\; \boxed{a_N/\ell_N} \;\rangle \quad .$$

From Theorem 5.3.1, we know that the frequencies of the target set in the blocks must be increasing:

$$\frac{a_1}{\ell_1} \;<\; \frac{a_2}{\ell_2} \;<\; \cdots \;<\; \frac{a_N}{\ell_N} \quad .$$

We thus know that with every stream $\mathbb{S}$ with N borders for a certain target set A, there is a corresponding increasing sequence of N fractions. This sequence of fractions is called the block frequency sequence *of the target set in the stream. The* length *of the stream is the sum of the denominators: $|\mathbb{S}| = \ell_1 + \ldots + \ell_N = L$.*

It is very interesting to see that the other direction is also true.

Definition 5.5.2. *For every increasing sequence of numbers*

$$0 \;<\; \frac{a'_1}{\ell'_1} \;<\; \frac{a'_2}{\ell'_2} \;<\; \cdots \;<\; \frac{a'_N}{\ell'_N} \;\leq\; 1 \quad ,$$

it is possible to find a stream of length $\ell'_1 + \ldots + \ell'_N$ *with* N *borders for target set* A*, namely:*

$$\langle |\overbrace{a\ldots a}^{a'_1}\overbrace{b\ldots b}^{\ell'_1-a'_1}|\overbrace{a\ldots a}^{a'_2}\overbrace{b\ldots b}^{\ell'_2-a'_2}|\ldots|\overbrace{a\ldots a}^{a'_N}\overbrace{b\ldots b}^{\ell'_N-a'_N}\rangle \ .$$

In this stream, a *denotes a target transaction and* b *denotes a non-target transaction. This stream is called the* canonical stream associated with the sequence of increasing fractions $a'_1/\ell'_1 < a'_2/\ell'_2 < \ldots < a'_N/\ell'_N$.

It is obviously clear that finding the maximal number of borders for a stream of length ℓ corresponds to finding the largest number of different fractions between 0 and 1, of which the sum of the denominators adds up to ℓ. In this context, the notion of *Farey sets* and *Farey sequences* will be very useful.

Definition 5.5.3. *The* Farey set of order k*, denoted* F_k*, is the following set of completely reduced* Farey fractions*:*

$$F_k \ := \ \left\{ \frac{a}{b} \ \middle| \ gcd(a,b) = 1, \quad 0 < a \leq b \leq k \right\} \ .$$

Note that in this definition, the fraction $0/k$ *is not included in* F_k*, in contrast to some other definitions of Farey fractions. The* Farey sequence of order k *is the list in which the elements of* F_k *are ordered in increasing order [25]. The notation* F_k *will be used for both the Farey set and the Farey sequence.*

Example 5.5.1. *The Farey sequences of orders* 1 *to* 5*:*

$$\begin{aligned}
F_1 &= \frac{1}{1} \\
F_2 &= \frac{1}{2}, \frac{1}{1} \\
F_3 &= \frac{1}{3}, \frac{1}{2}, \frac{2}{3}, \frac{1}{1} \\
F_4 &= \frac{1}{4}, \frac{1}{3}, \frac{1}{2}, \frac{2}{3}, \frac{3}{4}, \frac{1}{1} \\
F_5 &= \frac{1}{5}, \frac{1}{4}, \frac{1}{3}, \frac{2}{5}, \frac{1}{2}, \frac{3}{5}, \frac{2}{3}, \frac{3}{4}, \frac{4}{5}, \frac{1}{1}
\end{aligned}$$

Property 5.5.1. *The Farey set of order* k*,* F_k*, contains all the members of the Farey sets of lower orders* F_i *(with* $i \leq k-1$*). In particular,* F_k *contains all the members of* F_{k-1}*, plus an additional fraction based on each number that is less than* k *and co-prime to* k*, divided by* k*.*

Example 5.5.2. F_6 *consists of* F_5*, together with the fractions* $\frac{1}{6}$ *and* $\frac{5}{6}$*.*

$$F_6 \ = \ \frac{1}{6}, \frac{1}{5}, \frac{1}{4}, \frac{1}{3}, \frac{2}{5}, \frac{1}{2}, \frac{3}{5}, \frac{2}{3}, \frac{3}{4}, \frac{4}{5}, \frac{5}{6}, \frac{1}{1}$$

Property 5.5.2. *Except for* F_1*, each* F_k *has an* even *number of terms. If we would include the starting term* $\frac{0}{1}$ *in the Farey sequences, the middle term of each sequence is always* $\frac{1}{2}$*, for* $k > 1$*.*

Just like any other increasing sequence of fractions, also the Farey sequence F_k can be associated with its canonical stream $\mathbb{F}_k$. This stream has $|F_k|$ borders for a certain target set A, and a length that equals the sum of the denominators of the elements in F_k.

Example 5.5.3. *Consider F_5, the Farey sequence of the fifth order:*

$$\frac{1}{5} < \frac{1}{4} < \frac{1}{3} < \frac{2}{5} < \frac{1}{2} < \frac{3}{5} < \frac{2}{3} < \frac{3}{4} < \frac{4}{5} < \frac{1}{1} \ .$$

The corresponding Farey stream *of the fifth order, $\mathbb{F}_5$, is*

$$\langle\, |a\ b\ b\ b\ b\ |a\ b\ b\ b\ |a\ b\ b\ |a\ a\ b\ b\ b\ |a\ b\ |a\ a\ a\ b\ b\ |a\ a\ b\ |a\ a\ a\ b\ |a\ a\ a\ a\ b\ |a\rangle \ .$$

In this stream, a denotes a transaction that contains a certain target set A and b denotes a non-target transaction. It is clear that this stream has $|F_5| = 10$ borders for target set A and a total length of $5+4+3+5+2+5+3+4+5+1 = 37$.

5.5.2 Borders in Farey Streams

We will now show (Theorem 5.5.1) that the Farey streams have the *maximal* number of borders; that is, for every stream $\mathbb{S}$ of length equal to the length of $\mathbb{F}_k$, for a certain k, the number of borders in $\mathbb{S}$ is less than or equal to the number of borders in $\mathbb{F}_k = |F_k|$.

Definition 5.5.4. *The sum of the denominators of the fractions in a sequence S is denoted by $dsum(S)$ and can be computed as follows:*

$$dsum(\{a_1/\ell_1, \ldots, a_N/\ell_N\}) \ := \ \sum_{i=1}^{N} \ell_i \ .$$

Lemma 5.5.1. *Let $S = \{a_1/\ell_1, \ldots, a_N/\ell_N\}$ be a set of N different fractions, with $0 < a_i < \ell_i$, for all $i = 1..N$. Let k be such that $|S| > |F_k|$, then*

$$dsum(S) \ > \ dsum(F_k) \ .$$

Proof. It is easy to see that $dsum(S) - dsum(F_k) = dsum(S - F_k) - dsum(F_k - S)$ and $|S - F_k| > |F_k - S|$. Here, $S - F_k$ denotes all the fractions that are in S, but not in F_k, and vice versa, $F_k - S$ denotes all the fractions that are in F_k, but not in S. Furthermore, any fraction in $S - F_k$ must have a denominator of at least $k+1$, and every fraction in $F_k - S$ has a denominator of at most k. Therefore,

$$\begin{aligned} dsum(S) - dsum(F_k) &= dsum(S - F_k) - dsum(F_k - S) \\ &\geq (k+1) \cdot |S - F_k| - k \cdot |F_k - S| \\ &> |S - F_k| \\ &> 0 \end{aligned}$$

Hence, $dsum(S)$ must be larger than $dsum(F_k)$. □

Theorem 5.5.1. *Let $\mathbb{S}$ be a stream with $|\mathbb{S}| = |\mathbb{F}_k|$. Then, the number of borders in $\mathbb{S}$ is at most the number of borders in $\mathbb{F}_k$.*

Proof. Consider the block frequency sequence $S = \{a1/\ell_1, \ldots, a_N/\ell_N\}$ of $\mathbb{S}$ for target set A. The number of borders in $\mathbb{S}$ equals $|S|$ $(= N)$, and the number of borders in $\mathbb{F}_k$ equals $|F_k|$. Suppose now, for the sake of contradiction, that the number of borders in $\mathbb{S}$ is larger than the number of borders in $\mathbb{F}$. Then, $|S| > |F_k|$, and thus, because of Lemma 5.5.1, $dsum(S) > dsum(F_k)$. This is in contradiction with the fact that $dsum(S) = |\mathbb{S}| = |\mathbb{F}_k| = dsum(F_k)$. Hence, the number of borders in $\mathbb{S}$ can maximally be the number of borders in $\mathbb{F}_k$. □

Corollary 5.5.1. *Let $\ell = dsum(F_k)$, and $N = |F_k|$, for a fixed k. A stream of length ℓ has maximally N borders.*

5.5.3 Bounds on the Number of Borders in Farey Streams

The number of borders for a certain target set A in a Farey stream $\mathbb{F}_k$ of order k equals $|F_k|$ and the length of the stream equals $dsum(F_k)$. This representation does not reveal the *actual ratio* between the *size* of a stream and the *number of borders* of a stream. Therefore, the asymptotic behavior of these quantities (Theorem 5.5.2 and 5.5.3) has been worked out in the rest of this section, based on known results in number theory about *Euler's totient function* and the *Möbius function*, resulting in the important relation stated in theorem 5.5.4.

Definition 5.5.5. *The* Euler's totient function φ *maps positive integers r to the number of positive integers i that are co-prime to r; that is, $gcd(r, i) = 1$, $(1 \leq i \leq r)$.*

Example 5.5.4. *$\varphi(5) = 4$, as $1, 2, 3, 4$ are all co-prime to 5. $\varphi(8) = 4$, as only $1, 3, 5, 7$ are co-prime to 8.*

From this definition, it follows that $\varphi(1) = 1$. Based on the relationship between two consecutive Farey sequences F_{k-1} and F_k, the lengths of F_{k-1} and F_k can be related using Euler's totient function $\varphi(k)$.

Property 5.5.3. *Given two consecutive Farey sequences F_{k-1} and F_k. We have*

$$|F_k| \;=\; |F_{k-1}| \;+\; \varphi(k) \;,$$

based on Euler's totient function $\varphi(k)$. It can also be derived that

$$|F_k| \;=\; \sum_{i=1}^{k} \varphi(i) \;.$$

Proof. For $k = 1$, this identity obviously holds. Furthermore, the number of new elements in $F_k - F_{k-1}$ are exactly those fractions $\frac{x}{k}$ such that $1 \leq x \leq k$ and $gcd(x, k) = 1$ (Property 5.5.1). Here, $\frac{x}{k}$ must be completely reduced. Hence, the number of new fractions equals the number of integers x, $1 \leq x \leq k$ that are co-prime to k, which is $\varphi(k)$. □

Property 5.5.4. *The number of fractions in a Farey sequence F_k with denominator $d \leq k$ is $\varphi(d)$. Therefore, the length of the corresponding Farey stream $\mathbb{F}_k$ can be computed as*

$$|\mathbb{F}_k| \;=\; dsum(F_k) \;=\; \sum_{i=1}^{k} i \cdot \varphi(i) \;.$$

k	Length of $\mathbb{F}_k$	# borders of $\mathbb{F}_k$
1	$\lvert\mathbb{F}_1\rvert = \varphi(1) = 1$	$\lvert F_1\rvert = \varphi(1) = 1$
2	$\lvert\mathbb{F}_2\rvert = \lvert\mathbb{F}_1\rvert + (2 \cdot \varphi(2)) = 1 + (2 \cdot 1) = 3$	$\lvert F_2\rvert = \lvert F_1\rvert + \varphi(2) = 1 + 1 = 2$
3	$\lvert\mathbb{F}_3\rvert = \lvert\mathbb{F}_2\rvert + (3 \cdot \varphi(3)) = 3 + (3 \cdot 2) = 9$	$\lvert F_3\rvert = \lvert F_2\rvert + \varphi(3) = 2 + 2 = 4$
4	$\lvert\mathbb{F}_4\rvert = \lvert\mathbb{F}_3\rvert + (4 \cdot \varphi(4)) = 9 + (4 \cdot 2) = 17$	$\lvert F_4\rvert = \lvert F_3\rvert + \varphi(4) = 4 + 2 = 6$
5	$\lvert\mathbb{F}_5\rvert = \lvert\mathbb{F}_4\rvert + (5 \cdot \varphi(5)) = 17 + (5 \cdot 4) = 37$	$\lvert F_5\rvert = \lvert F_4\rvert + \varphi(5) = 6 + 4 = 10$
6	$\lvert\mathbb{F}_6\rvert = \lvert\mathbb{F}_5\rvert + (6 \cdot \varphi(6)) = 37 + (6 \cdot 2) = 49$	$\lvert F_6\rvert = \lvert F_5\rvert + \varphi(6) = 10 + 2 = 12$
7	$\lvert\mathbb{F}_7\rvert = \lvert\mathbb{F}_6\rvert + (7 \cdot \varphi(7)) = 49 + (7 \cdot 6) = 91$	$\lvert F_7\rvert = \lvert F_6\rvert + \varphi(7) = 12 + 6 = 18$
...	...	...

Figure 5.11: The length and the number of borders for $\mathbb{F}_k$, a Farey stream of order k

Example 5.5.5. *For a selection of values of k, the length of the corresponding Farey stream and the amount of borders is given in Figure 5.11.*

We are now ready to prove some asymptotic behaviors, that will help to understand the relation between the number of borders in a stream and the size of the stream.

Theorem 5.5.2. *Let k be a positive integer.*

$$\sum_{i=1}^{k} \varphi(i) \;=\; \frac{3k^2}{\pi^2} \;+\; \mathcal{O}(k \log(k))$$

Proof. It is well-known from the literature about Euler's totient function [10] that

$$\frac{1}{k^2} \sum_{i=1}^{k} \varphi(i) \;=\; \frac{3}{\pi^2} \;+\; \mathcal{O}\left(\frac{\log(k)}{k}\right) \;. \tag{5.18}$$

Hence, asymptotically, the number of borders $|F_k|$ becomes $\frac{3k^2}{\pi^2}$. □

Unfortunately, we could not find an existing similar result in the literature, that gives the asymptotic behavior of $\sum_{i=1}^{k} i \cdot \varphi(i)$. Therefore, we will show the following result:

$$\frac{1}{k^3} \sum_{i=1}^{k} i \cdot \varphi(i) \;=\; \frac{2}{\pi^2} \;+\; \mathcal{O}\left(\frac{\log(k)}{k}\right) \;. \tag{5.19}$$

The proof of this result will use similar techniques as the proofs for the asymptotic behavior of (5.18). The new result (5.19) shows that asymptotically, the length of the stream $dsum(F_k)$ becomes $\frac{2k^3}{\pi^2}$. Before we can actually write down the proof, we need some lemmas and properties related to the Möbius function μ.

Lemma 5.5.2. *For all integers $1 \leq i \leq k$, it holds that*

$$\left(2\left\lfloor\frac{k+1}{i}\right\rfloor^3 + 3\left\lfloor\frac{k+1}{i}\right\rfloor^2 + \left\lfloor\frac{k+1}{i}\right\rfloor\right) - \left(2\left\lfloor\frac{k}{i}\right\rfloor^3 + 3\left\lfloor\frac{k}{i}\right\rfloor^2 + \left\lfloor\frac{k}{i}\right\rfloor\right) = \frac{6(k+1)^2}{i^2} \tag{5.20}$$

if $i|k+1$, and is 0 otherwise. As usual, $i|k+1$ denotes that i is a divisor of $k+1$.

Proof.

If i is not a divisor of $k+1$, then $\left\lfloor \frac{k+1}{i} \right\rfloor$ equals $\left\lfloor \frac{k}{i} \right\rfloor$. Therefore, in this case, the expression (5.20) trivially evaluates to 0.

On the other hand, if $i|(k+1)$, then $\left\lfloor \frac{k+1}{i} \right\rfloor = \frac{k+1}{i}$, and $\left\lfloor \frac{k}{i} \right\rfloor = \frac{k+1-i}{i}$, as $k+1-i$ is the largest integer divisible by i that is smaller than k. Therefore, in this case, the expression (5.20) is equal to

$$2\left(\frac{k+1}{i}\right)^3+3\left(\frac{k+1}{i}\right)^2+\left(\frac{k+1}{i}\right)-2\left(\frac{k+1-i}{i}\right)^3-3\left(\frac{k+1-i}{i}\right)^2-\left(\frac{k+1-i}{i}\right) \ .$$

After some straightforward computations, this expression can be simplified to

$$\frac{6k^2+12k+6}{i^2} \ .$$

□

Definition 5.5.6. *Let $\mu(r)$ denote the* Möbius function. *$\mu(r)$ is defined as follows:*

$\mu(r) = 1$ if the prime factorization of r is square-free and has an even number of factors,

$\mu(r) = -1$ if the factorization is square-free and has an odd number of factors, and

$\mu(r) = 0$ otherwise.

Example 5.5.6. *We give an example for each of the three possible cases.*
$\mu(6) = 1$, because the factorization $6 = 2 \cdot 3$ is square free and has an even number of terms.
$\mu(3) = -1$, because the prime factorization has 1 factor, 3, and is square-free.
$\mu(8) = 0$, as the prime factorization is $8 = 2^3$, which is not square free.

Property 5.5.5. *$\mu(1)$ is defined to be 1.*

There is a fundamental link between Euler's totient function and the Möbius function, that is proved in the literature [10].

Property 5.5.6. *Let k be a positive integer.*

$$\varphi(k) \ = \ \sum_{d|k} d \cdot \mu(k/d)$$

Corollary 5.5.2. *Let k be a positive integer.*

$$\frac{\varphi(k)}{k} \ = \ \sum_{i|k} \frac{\mu(i)}{i} \tag{5.21}$$

Proof.

$$\varphi(k) \ = \ \sum_{d|k} d \cdot \mu(k/d) \ \Leftrightarrow \ \frac{\varphi(k)}{k} \ = \ \sum_{d|k} \frac{d}{k} \cdot \mu(k/d) \ = \ \sum_{d|k} \frac{\mu(k/d)}{k/d}$$

□

Lemma 5.5.3. *Let k be a positive integer.*

$$\sum_{i=1}^{k} i \cdot \varphi(i) \;=\; \frac{1}{6} \sum_{i=1}^{k} i \cdot \mu(i) \cdot \left(2 \left\lfloor \frac{k}{i} \right\rfloor^3 + 3 \left\lfloor \frac{k}{i} \right\rfloor^2 + \left\lfloor \frac{k}{i} \right\rfloor \right)$$

Proof. We will prove this identity by induction.

$k = 1$ $\quad \varphi(1) = 1 = \frac{1}{6} \cdot \mu(1) \cdot (2 + 3 + 1)$

$k + 1$ Suppose that the equality holds for $1..k$.
Then, we need to prove the following equality in order to extend to $k + 1$:

$$\begin{aligned} 6 \cdot \left[\sum_{i=1}^{k+1} i \cdot \varphi(i) - \sum_{i=1}^{k} i \cdot \varphi(i) \right] &= \sum_{i=1}^{k+1} i \cdot \mu(i) \cdot \left(2 \left\lfloor \frac{k+1}{i} \right\rfloor^3 + 3 \left\lfloor \frac{k+1}{i} \right\rfloor^2 + \left\lfloor \frac{k+1}{i} \right\rfloor \right) \\ &\quad - \sum_{i=1}^{k} i \cdot \mu(i) \cdot \left(2 \left\lfloor \frac{k}{i} \right\rfloor^3 + 3 \left\lfloor \frac{k}{i} \right\rfloor^2 + \left\lfloor \frac{k}{i} \right\rfloor \right) \\ \Leftrightarrow 6 \cdot (k+1) \cdot \varphi(k+1) &= \sum_{i=1}^{k+1} i \cdot \mu(i) \cdot \left(2 \left\lfloor \frac{k+1}{i} \right\rfloor^3 + 3 \left\lfloor \frac{k+1}{i} \right\rfloor^2 + \left\lfloor \frac{k+1}{i} \right\rfloor \right) \\ &\quad - \sum_{i=1}^{k} i \cdot \mu(i) \cdot \left(2 \left\lfloor \frac{k}{i} \right\rfloor^3 + 3 \left\lfloor \frac{k}{i} \right\rfloor^2 + \left\lfloor \frac{k}{i} \right\rfloor \right) \end{aligned}$$

The right-hand side can be simplified into

$$\begin{aligned} &\sum_{i=1}^{k} i \cdot \mu(i) \cdot \left[\left(2 \left\lfloor \frac{k+1}{i} \right\rfloor^3 + 3 \left\lfloor \frac{k+1}{i} \right\rfloor^2 + \left\lfloor \frac{k+1}{i} \right\rfloor \right) - \left(2 \left\lfloor \frac{k}{i} \right\rfloor^3 + 3 \left\lfloor \frac{k}{i} \right\rfloor^2 + \left\lfloor \frac{k}{i} \right\rfloor \right) \right] \\ &\qquad + 6 \cdot (k+1) \cdot \mu(k+1) \\ = \; & \sum_{\substack{i \mid (k+1) \\ i < k+1}} i \cdot \mu(i) \cdot \frac{6(k+1)^2}{i^2} + 6 \cdot (k+1) \cdot \mu(k+1) \qquad \text{(By Lemma 5.5.2)} \\ = \; & 6 \cdot (k+1)^2 \cdot \sum_{\substack{i \mid (k+1) \\ i \leq k+1}} \frac{\mu(i)}{i} \\ = \; & 6 \cdot (k+1)^2 \cdot \frac{\varphi(k+1)}{(k+1)} \qquad \text{(Identity (5.21))} \\ = \; & 6 \cdot (k+1) \cdot \varphi(k+1) \end{aligned}$$

□

Property 5.5.7. *Let k be a positive integer. It is well-known from the literature [10] that*

$$\frac{1}{k} \sum_{i=1}^{k} \frac{\varphi(i)}{i} \;=\; \frac{6}{\pi^2} \;+\; \mathcal{O}\left(\frac{\log(k)}{k} \right) \;. \tag{5.22}$$

Property 5.5.8. *Let k be a positive integer. It is well-known from the literature [10] that*

$$\sum_{i=1}^{k} \frac{\varphi(i)}{i} \;=\; \sum_{i=1}^{k} \frac{\mu(i)}{i} \left\lfloor \frac{k}{i} \right\rfloor \quad .$$

Corollary 5.5.3. *Let k be a positive integer. $\displaystyle\sum_{i=1}^{k} \frac{\mu(i)}{i^2}$ converges to $\dfrac{6}{\pi^2}$.*

Proof. When we combine Properties 5.5.7 and 5.5.8, it is obviously clear that $\displaystyle\sum_{i=1}^{k} \frac{\mu(i)}{i^2}$ converges to $\dfrac{6}{\pi^2}$. □

We are now finally ready to proof the asymptotic behavior stated in (5.19).

Theorem 5.5.3. *Let k be a positive integer.*

$$\frac{1}{k^3} \sum_{i=1}^{k} i \cdot \varphi(i) \;=\; \frac{2}{\pi^2} + \mathcal{O}\left(\frac{\log(k)}{k}\right) \quad .$$

Proof. The proof is based on the identity given in Lemma 5.5.3, and the fact that we can lower and upper bound this identity, using $x-1 < \lfloor x \rfloor \leq x$. Inspired by similar proofs [10, 39], we get the following *lower bound* on $\dfrac{1}{k^3} \displaystyle\sum_{i=1}^{k} i \cdot \varphi(i)$:

$$\begin{aligned}
& \frac{1}{6k^3} \sum_{i=1}^{k} i \cdot \mu(i) \cdot \left(2\left(\frac{k}{i} - 1\right)^3 + 3\left(\frac{k}{i} - 1\right)^2 + \left(\frac{k}{i} - 1\right) \right) \\
= \;& \frac{1}{6k^3} \sum_{i=1}^{k} i \cdot \mu(i) \cdot \left(2\left(\frac{k}{i}\right)^3 - 3\left(\frac{k}{i}\right)^2 + \left(\frac{k}{i}\right) \right) \\
= \;& \frac{2}{6} \sum_{i=1}^{k} \frac{\mu(i)}{i^2} + \mathcal{O}\left(\frac{\log(k)}{k}\right) \quad .
\end{aligned}$$

Similarly, the following *upper bound* can be found.

$$\begin{aligned}
& \frac{1}{6k^3} \sum_{i=1}^{k} i \cdot \mu(i) \cdot \left(2\left(\frac{k}{i}\right)^3 + 3\left(\frac{k}{i}\right)^2 + \left(\frac{k}{i}\right) \right) \\
= \;& \frac{2}{6} \sum_{i=1}^{k} \frac{\mu(i)}{i^2} + \mathcal{O}\left(\frac{\log(k)}{k}\right) \quad .
\end{aligned}$$

Because $\displaystyle\sum_{i=1}^{k} \frac{\mu(i)}{i^2}$ converges to $6/\pi^2$ (Corollary 5.5.3), we get the following asymptotic behavior for both the lower and the upper bound:

$$\frac{2}{\pi^2} + \mathcal{O}\left(\frac{\log(k)}{k}\right) \quad .$$

□

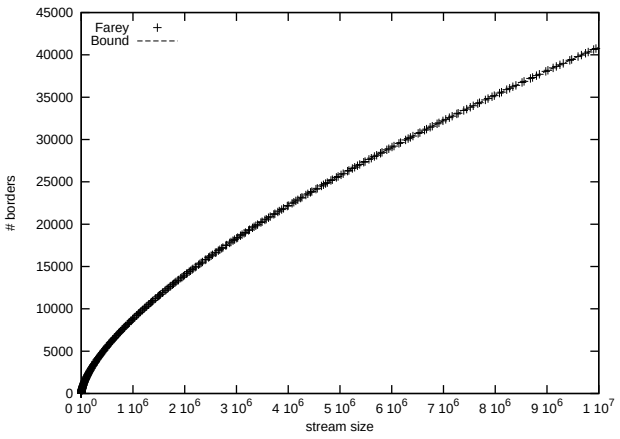


Figure 5.12: Worst-case number of borders

To conclude, we summarize the results. If k is a positive integer, we proved in Theorem 5.5.2 and Theorem 5.5.3 that the following two asymptotic behaviors are true.

$$\sum_{i=1}^{k} \varphi(i) \;=\; \frac{3k^2}{\pi^2} \;+\; \mathcal{O}(k\log(k)) \;,$$
$$\sum_{i=1}^{k} i\cdot\varphi(i) \;=\; \frac{2k^3}{\pi^2} \;+\; \mathcal{O}(k^2\log(k)) \;.$$

Based on these two characteristics, we can find the relationship between the size of a stream and the number of borders.

Theorem 5.5.4. *Asymptotically, in worst case, the number of borders N and the length of the stream L are related as*

$$N \;=\; \left(\frac{\pi^2 L}{2}\right)^{2/3} \frac{3}{\pi^2} \;.$$

Proof. Because of the shown asymptotic behaviors, we have that

$$N \;\sim\; \frac{3k^2}{\pi^2} \;\Leftrightarrow\; k \;\sim\; \left(\frac{\pi^2 N}{3}\right)^{1/2}$$

and

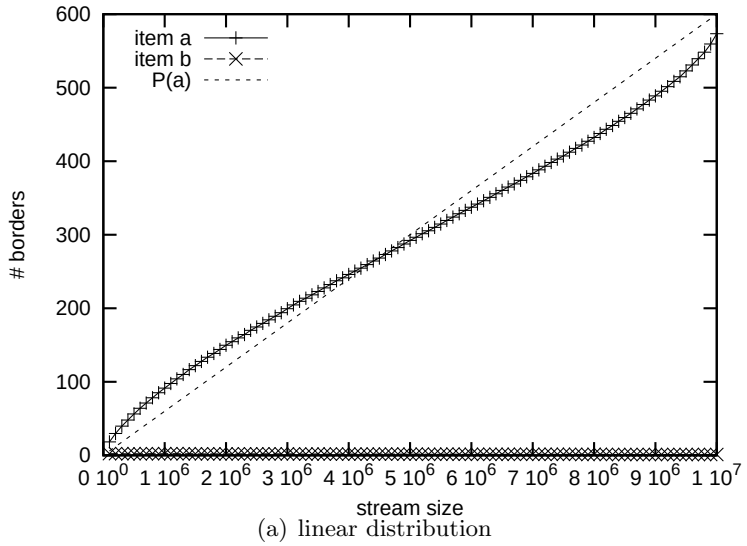
$$L \;\sim\; \frac{2k^3}{\pi^2} \;\Leftrightarrow\; k \;\sim\; \left(\frac{\pi^2 L}{2}\right)^{1/3}$$

We thus can find

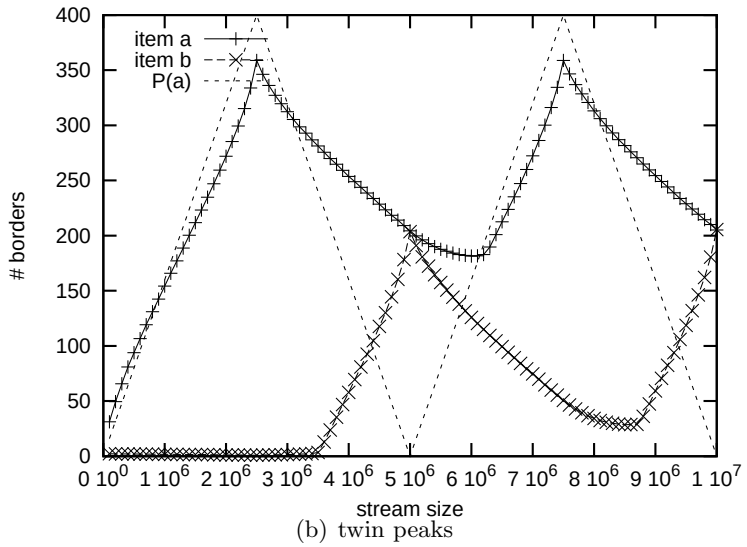
$$\left(\frac{\pi^2 N}{3}\right)^{1/2} \;=\; \left(\frac{\pi^2 L}{2}\right)^{1/3} \;\Leftrightarrow\; N \;=\; \left(\frac{\pi^2 L}{2}\right)^{2/3} \frac{3}{\pi^2} \;.$$

□

Figure 5.12 shows the number of borders for Farey streams of lengths up to 10^7 together with the upper bound given by this formula. As can be seen, the bound on the number of borders is almost exactly the actual worst-case number.



(a) linear distribution



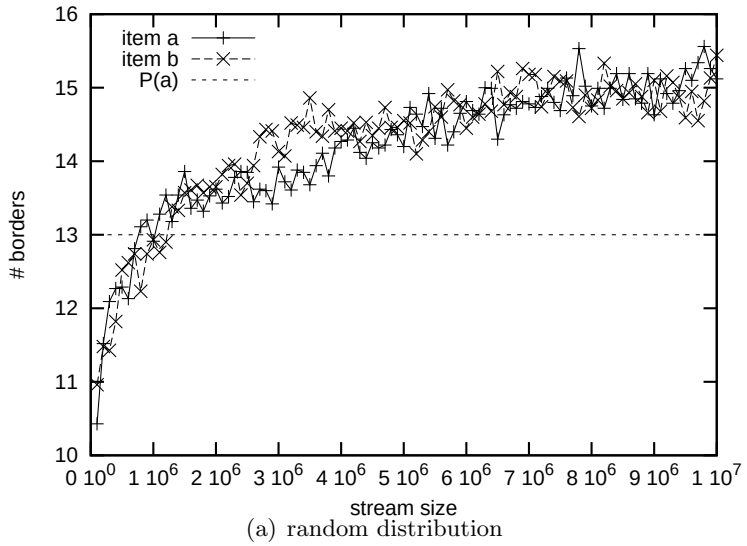
(b) twin peaks

Figure 5.13: Size of the summaries for two items a and b (linear - twin peaks)

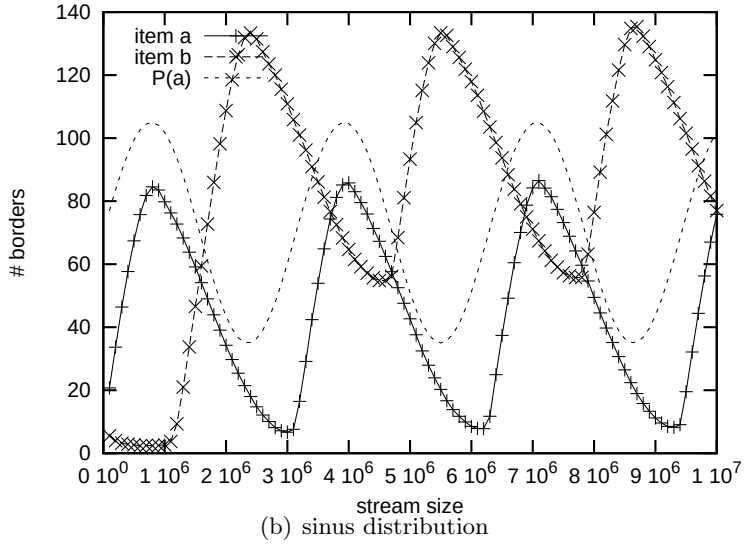
5.6 Experiments

From the description of the basic algorithm, it is clear that the update procedure is very efficient, given that the summaries remain small. Producing the current frequency of the target set is obviously very efficient, as it amounts to simply a lookup of the most recent entry. Hence, the complete approach will be feasible if and only if *the summaries remain small.*

To show that this is indeed the case, we have recorded the size of the summary for different synthetical streams and different target sets. The results are reported in Figures 5.13 and 5.14. The considered streams are streams of length 10^7 over two singleton transactions $\{a\}$ and $\{b\}$, abbreviated a and b, that we will both consider as target sets. Note that it does not matter



(a) random distribution



(b) sinus distribution

Figure 5.14: Size of the summaries for two items a and b (random - sinus)

whether we focus on two target sets or on 1000 target sets, as the sets do not influence the size of each others summary. The streams have a length of 10^7 and after every 10000 transactions, the size of the summary for the items a and b are reported. The streams are randomly generated. The probability of having transaction a in the stream is given by the line $P(a)$. Thus, in the random graph, the probability of having a is 1/2 in the whole stream, independent of the moment. In the other graphs, the probability of a is given by a linear or a sinus distribution, or by a distribution reflecting in two peaks. The probability of b is 1 minus the probability of a. The graphs report the average over 100 streams, generated with the indicated distributions. After every 10 000 transactions, the sizes of the summaries for both the targets a and b are reported.

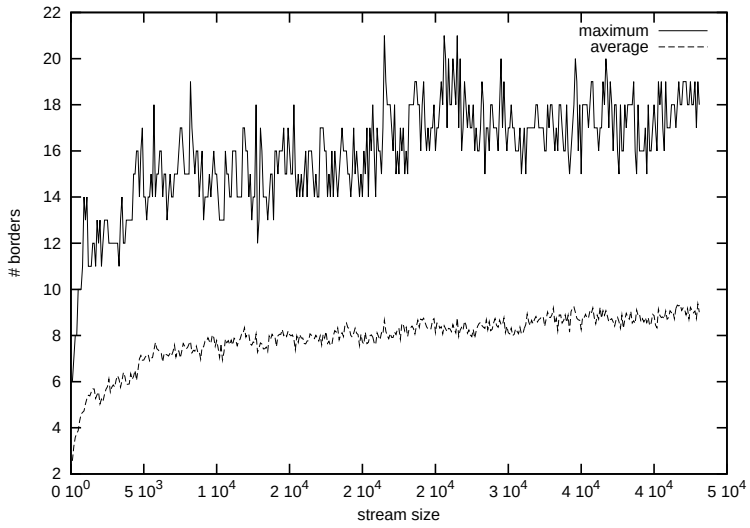


Figure 5.15: Size of the summaries for a real-life dataset

In general, we can conclude that the size of the summary is extremely small w.r.t. the size of the stream. If the probability of the target increases also the size of the summary will increase. When the probability decreases the summary will shrink. This is easily explained by the entries in the summary that need to have increasing frequency.

In Figure 5.15, this experiment is repeated for a real-life dataset. This dataset was obtained by collecting one month of click-stream data of visitors to the website of the University of Antwerp. For every minute, a transaction is generated, consisting of the set of all web pages visited in that minute. For every web page, the max-frequency is monitored with a minimal window length of 60. No minimal frequency threshold was used. At every time point, the maximal and the average number of borders is plotted for all web pages. As can be seen in this real-life experiment, again the sizes of the summaries remain extremely small.

5.7 Conclusion and Future Work

We introduced a new frequency measure for stream mining that does not rely on a fixed window length or a decay factor. We presented an algorithm based on this new measure to efficiently find all frequent sets at any time. The algorithm maintains a summary, that collects all info about the borders and the corresponding frequencies. A theoretical analysis was performed to study the worst case behavior and determine the maximal size of the summary. Experimental results show that the summary remains small in real-life situations.

The most important future work for stream mining is a general statistical analysis of a random stream, to theoretically prove where our method is better than other existing methods. The ultimate goal is to describe and find a real-life dataset such that our method can discover patterns that other methods do not find or have troubles with. Other future research could study the behavior of the combination of our algorithm with one of the incremental algorithms introduced in the literature to find frequent sets within the minimum window, to conclude if optimizations are possible.

Personal Bibliography

[1] T. Calders and N. Dexters. Theoretical bounds on the size of condensed representations. In *Proceedings ECML-PKDD 2004 Workshop Knowledge Discovery in Inductive Databases*, pages 25–36, 2004.

[2] T. Calders, N. Dexters, and B. Goethals. Mining frequent items in a stream using flexible windows. In *ECML/PKDD-2006 International Workshop on Knowledge Discovery from Data Streams (IWKDDS)*, 2006.

[3] T. Calders, N. Dexters, and B. Goethals. A new support measure for items in streams. *Le Monde des Utilisateurs de L'Analyse de Données (La Revue MODULAD), http://www.modulad.fr*, 36:37–41, June 2007.

[4] T. Calders, N. Dexters, and B. Goethals. Mining frequent items in a stream using flexible windows. *Intelligent Data Analysis, Special Issue on Knowledge Discovery from Data Streams*, May 2008.

[5] N. Dexters. Approximating the probability of an itemset being frequent. In *BNCOD22*, pages 1–4, Sunderland, UK, 2005. Poster-paper.

[6] N. Dexters. A probabilistic analysis for frequent itemset mining algorithms. DBDBD '05, October 31 2005, Amsterdam, October 2005. Talk for the Dutch-Belgian Database Day, October 31, Amsterdam.

[7] N. Dexters. Statistics in data mining: Finding frequent patterns. Poster Statistics@Antwerp, January 2007. ISBN 978-90-5728-070-2.

[8] N. Dexters, P. W. Purdom, and D. Van Gucht. Peak-jumping frequent itemset mining algorithms. In *PKDD*, pages 487–494. Springer, 2006.

[9] N. Dexters, P. W. Purdom, and D. Van Gucht. A probability analysis for candidate-based frequent itemset algorithms. volume 1 of 2, pages 541 – 545. Applied Computing 2006 - Proceedings of the 2006 ACM Symposium on Applied Computing, Dijon, France, 2006.

Bibliography

[10] M. Abramowitz and I.A. Stegun. *Handbook of Mathematical Functions, with Formulas, Graphs, and Mathematical Tables*. Dover Publications, New York.

[11] R. Agrawal, T. Imilienski, and A. Swami. Mining association rules between sets of items in large databases. In *Proc. ACM SIGMOD Int. Conf. Management of Data*, pages 207–216, Washington, D.C., 1993.

[12] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. VLDB Int. Conf. Very Large Data Bases*, pages 487–499, Santiago, Chile, 1994.

[13] R. Agrawal and R. Srikant. Mining sequential patterns. In *Proc. IEEE ICDE Int. Conf. on Data Engineering*, pages 3–14, Taipei, Taiwan, 1995.

[14] K. Ali, S. Manganaris, and R. Srikant. Partial classification using association rules. In *Proc. KDD Int. Conf. Knowledge Discovery in Databases*, pages 115–118, 1997.

[15] R. J. Bayardo. Efficiently mining long patterns from databases. In *Proc. ACM SIGMOD Int. Conf. Management of Data*, pages 85–93, Seattle, Washington, 1998.

[16] F. Bodon. Surprising results of trie-based fim algorithms. In *FIMI*, 2004.

[17] F. Bodon and L. Schmidt-Thieme. The relation of closed itemset mining, complete pruning strategies and item ordering in apriori-based fim algorithms. In *PKDD*, pages 437–444, 2005.

[18] S. Brin, R. Motwani, J.D. Ullman, and S. Tsur. Dynamic itemset counting and implication rules for market basket data. In *Proc. ACM SIGMOD Int. Conf. Management of Data*, pages 255–264, Tucson, AZ, 1997.

[19] D. Burdick, M. Calimlim, and J. Gehrke. Mafia: A maximal frequent itemset algorithm for transactional databases. In *ICDE*, pages 443–452, 2001.

[20] E. Robertson C. Giannella, J. Han and C. Liu. Mining frequent itemsets over arbitrary time intervals in data streams. Technical report, Indiana University, Bloomington, USA, 2003.

[21] J. Pei X. Yan C. Giannella, J. Han and P.S. Yu. *Next Generatation Data Mining*, chapter 3, Mining Frequent Patterns in Data Streams at Multiple Time Granularities, pages 191–212. 2003.

[22] T. Calders. Computational complexity of itemset frequency satisfiability. In *PODS*, pages 143–154, 2004.

[23] J. H. Chang and W. S. Lee. Finding recent frequent itemsets adaptively over online data streams. In *KDD '03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 487–492, New York, NY, USA, 2003. ACM Press.

[24] H. Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on a sum of observations. *Ann. Math. Statist.*, 23:493–507, 1952.

[25] J. H. Conway and R. K. Guy. *Farey Fractions and Ford Circles.* Springer-Verlag, 1996.

[26] E. D. Demaine, A. López-Ortiz, and J. I. Munro. Frequency estimation of internet packet streams with limited space. In *ESA*, pages 348–360, 2002.

[27] U. Egly, R. Pichler, and S. Woltran. On deciding subsumption problems, 2002.

[28] F. Geerts, B. Goethals, and J. Van den Bussche. Tight upper bounds on the number of candidate patterns. *ACM Trans. Database Syst.*, 30(2):333–363, 2005.

[29] B. Goethals. *Efficient Frequent Pattern Mining.* PhD thesis, Transnational University Limburg, Belgium, December 2002.

[30] B. Goethals, S. Nijssen, and M. J. Zaki. Open source data mining: workshop report. *SIGKDD Explorations*, 7(2):143–144, 2005.

[31] L. Golab, D. DeHaan, E. D. Demaine, A. López-Ortiz, and J. I. Munro. Identifying frequent items in sliding windows over on-line packet streams. In *Internet Measurement Comference*, pages 173–178, 2003.

[32] K. Gouda and M. J. Zaki. Genmax: An efficient algorithm for mining maximal frequent itemsets. *Data Min. Knowl. Discov.*, 11(3):223–242, 2005.

[33] D. Gunopulos, R. Khardon, H. Mannila, and H. Toivonen. Data mining, hypergraph transversals, and machine learning. In *PODS*, pages 209–216, 1997.

[34] J. Han and M. Kamber. *Data Mining: Concepts and Techniques.* Morgan Kaufmann, 2000.

[35] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *Proc. ACM SIGMOD Int. Conf. Management of Data*, pages 1–12, Dallas, TX, 2000.

[36] J. Han, J. Pei, Y. Yin, and R. Mao. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data Min. Knowl. Discov.*, 8(1):53–87, 2004.

[37] D. Hand, H. Mannila, and P. Smyth. *Principles of Data Mining.* MIT Press, 2001.

[38] J. Hipp, U. Güntzer, and G. Nakhaeizadeh. Algorithms for association rule mining - a general survey and comparison. *SIGKDD Explorations*, 2(1):58–64, 2000.

[39] http://en.wikipedia.org/wiki/Proofs of totient identities.

[40] R. Jin and G. Agrawal. An algorithm for in-core frequent itemset mining on streaming data. In *ICDM*, pages 210–217, 2005.

[41] R. J. Bayardo Jr., B. Goethals, and M. 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, 2004.

[42] R. M. Karp, S. Shenker, and C. H. Papadimitriou. A simple algorithm for finding frequent elements in streams and bags. *ACM Trans. Database Syst.*, 28:51–55, 2003.

[43] W. A. Kosters and W. Pijls. Apriori, a depth first implementation. In *FIMI*, 2003.

[44] D. Lee and W. Lee. Finding maximal frequent itemsets over online data streams adaptively. In *ICDM*, pages 266–273, 2005.

[45] L. K. Lee and H. F. Ting. A simpler and more efficient deterministic scheme for finding frequent items over sliding windows. In *PODS*, pages 290–297, 2006.

[46] C.-H. Lin, D.-Y. Chiu, Y.-H. Wu, and A. L. P. Chen. Mining frequent itemsets from data streams with a time-sensitive sliding window. In *SDM*, 2005.

[47] G. S. Manku and R. Motwani. Approximate frequency counts over data streams. In *VLDB*, pages 346–357, 2002.

[48] H. Mannila and H. Toivonen. Levelwise search and borders of theories in knowledge discovery. *DMKD*, 1(3):241–258, 1997.

[49] L. Gruenwald M.H. Dunham, Y. Xiao and Z. Hossain. A survey of association rules.

[50] P. W. Purdom, D. Van Gucht, and D. P. Groth. Average-case performance of the apriori algorithm. *SIAM J. Comput.*, 33(5):1223–1260, 2004.

[51] A. Savasere, E. Omiecinski, and S. Navathe. An efficient algorithm for mining association rules in large databases. In *Proc. VLDB Int. Conf. Very Large Data Bases*, pages 432–443, Zürich, Switzerland, 1995.

[52] P.-N. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining*. Addison-Wesley.

[53] H. Toivonen. Discovery of frequent patterns in large data collections, 1996.

[54] A. Veloso, W. Meira Jr., M. de Carvalho, B. Pôssas, S. Parthasarathy, and M. J. Zaki. Mining frequent itemsets in evolving databases. In *SDM*, 2002.

[55] R.C.W. Wong and A.W.C. Fu. Mining top-K frequent itemsets from data streams. *Data Mining and Knowledge Discovery*, 13(2):193–217, 2006.

[56] J. X. Yu, Z. Chong, H. Lu, and A. Zhou. False positive or false negative: Mining frequent itemsets from high speed transactional data streams. In *VLDB*, pages 204–215, 2004.

[57] M. J. Zaki. Scalable algorithms for association mining. *IEEE Trans. Knowl. Data Eng.*, 12(2):372–390, 2000.

[58] M. J. Zaki and K. Gouda. Fast vertical mining using diffsets. In *KDD*, pages 326–335, 2003.

[59] M. J. Zaki and C.-J. Hsiao. Charm: An efficient algorithm for closed itemset mining. In *SDM*, 2002.

[60] M. J. Zaki, S. Parthasarathy, M. Ogihara, and W. Li. New algorithms for fast discovery of association rules. In *KDD*, pages 283–286, 1997.

A

Nederlandse Samenvatting

De resultaten voorgesteld in dit proefschrift situeren zich in het omvangrijke domein van **data mining**, een relatief jonge onderzoeksdiscipline op de grens van databanken, statistiek, machine learning, artificiële intelligentie en probabilistisch modelleren.

Dankzij de ontwikkeling van relationele databanken kunnen bedrijven de laatste jaren gigantische hoeveelheden data efficiënt opslaan. Niet alleen deze enorme hoeveelheden gegevens zelf, maar ook de mogelijkheden om deze te onderzoeken maakt ze waardevol én interessant. Voor een bedrijf is het immers zeer belangrijk om nuttige informatie uit de verzamelde data te halen. Het bestaan van dit soort data en de noodzaak en tevens uitdaging om deze te analyseren, is de belangrijkste motivatie voor data mining: data mining wordt immers gedefineerd als de analyse van gigantische hoeveelheden geobserveerde gegevens, om alzo interessante en onverwachte regelmatigheden te ontdekken en deze te gebruiken om de data samen te vatten zodat deze beter te begrijpen is en daardoor nuttig wordt voor de eigenaar [34].

Het opsporen van frequente patronen is een fundamenteel probleem in data mining. Ondanks de vele literatuur hierover is dit probleem nog steeds niet opgelost en blijft het een belangrijke uitdaging voor onderzoekers, in het bijzonder wanneer er gezocht moet worden naar patronen in extreem grote databanken of sequenties van gegevens, ook wel gegevensstromen genoemd. De meeste aandacht werd tot nu toe besteed aan het experimenteel en empirisch onderzoek van de gebruikte algoritmes, en is er maar betrekkelijk weinig gedaan vanuit een *theoretisch* oogpunt. Dit werk probeert deze leegte op te vullen.

Dit hoofdstuk is een introductie in de wereld van frequente patronen, waarbij de nadruk ligt op *frequente verzamelingen*, en een samenvatting van de resultaten van het gepresenteerde onderzoek. Sectie A.1 behandelt het traditionele probleem in statische databanken, terwijl Sectie A.2 handelt over dynamische gegevensstromen. In Sectie A.3 wordt een schematisch overzicht van de inhoud en de resultaten van dit proefschrift gegeven.

A.1 Frequent Set Mining

De laatste jaren heeft data mining (in de bredere context van knowledge discovery in databanken - afgekort KDD) een heleboel aandacht gekregen in de databankwereld. Dit werd veroorzaakt door de gigantische hoeveelheden gedigitaliseerde gegevens die bedrijven en organisaties hebben over hun bezigheden. Het doel van data mining is interessante patronen te vinden in databanken, zoals associatieregels, correlaties, sequenties, episodes, clusters en nog veel meer [37, 52]. Het ontdekken van vaak voorkomende of *frequente verzamelingen*, en hierop gebaseerd, *associatieregels* is waarschijnlijk het meest populaire onderzoeksveld binnen data mining [11]. De motivatie voor dit soort onderzoek komt van de noodzaak tot het analyseren van supermarktgegevens om alzo het klantengedrag in functie van de gekochte producten te bestuderen. Associatieregels beschrijven hoe vaak artikelen, ook wel items genoemd, samen gekocht zijn. Bijvoorbeeld "brood $\Rightarrow$ boter (87%)" poneert dat 87% van de klanten die brood kochten, ook boter meenamen. Dit soort ontdekkingen kan zeer nuttig zijn om producten te prijzen, promoties uit te schrijven en de winkel in te richten, met als bedoeling natuurlijk de winst te vergroten. Het zoeken naar vaak voorkomende combinaties van artikelen wordt in het data mining jargon *frequent set mining* genoemd. Het bekleedt een sleutelpositie in het proces om associatieregels te genereren: eens we weten welke artikelen vaak verkocht worden is het relatief eenvoudig om associatieregels te vinden.

In de context van supermarktgegevens is het dus de bedoeling om verbanden te ontdekken tussen artikelen of items die vaak samen verkocht worden. Als we alle artikelen die verkocht worden in de winkel verzamelen in $\mathcal{I} = \{i_1, i_2, \ldots, i_m\}$, dan kunnen we de winkelmandjes modelleren als afzonderlijke verzamelingen van deze artikelen uit het universum van alle mogelijke artikelen. Bijvoorbeeld, het mandje van klant x wordt genoteerd als verzameling I_x, en bestaat uit artikelen uit $\mathcal{I}$. Als we dit doen voor elke klant, dan krijgen we collectie van *verzamelingen* die de verkoop in de winkel beschrijven. Deze worden, samen met een uniek *identificatienummer*, opgeslagen in een *databank*. De combinatie van het identificatienummer en de corresponderende verzameling noemen we een *transactie*.

Voorbeeld A.1.1. *John koopt brood, boter en melk, en Lisa koopt chips, bier en brood. Deze twee verzamelingen, die de winkelmandjes van de twee klanten voorstellen, kunnen opgeslagen worden in databank $\mathcal{D}$.*

$\mathcal{D}$ =

TID	***Artikelen***	
1	*brood, boter, melk*	$\leftarrow$ *transactie*
2	*chips, bier, brood*	

$\uparrow$

transactie-identificatienummer

In de meeste gevallen wordt de informatie in de databank opgeslagen in de vorm van *binaire* vectoren. De aanwezigheid van een zeker artikel uit zich in het gebruik van 1, terwijl de afwezigheid van dat artikel gekarakteriseerd wordt door 0.

Voorbeeld A.1.2. *John koopt brood, boter en melk, en Lisa koopt chips, bier en brood. Deze twee verzamelingen, die de winkelmandjes van deze twee klanten voorstellen, kunnen binair opgeslagen worden in databank $\mathcal{D}$.*

$\mathcal{D}$ =

TID	***brood***	***boter***	***melk***	***chips***	***bier***
1	*1*	*1*	*1*	*0*	*0*
2	*1*	*0*	*0*	*1*	*1*

Definitie A.1.1. *De grootte van de databank $\mathcal{D}$ is het totaal aantal transacties in $\mathcal{D}$, en wordt genoteerd met $|\mathcal{D}| = b$. Het is perfect mogelijk dat sommige transacties meer dan één keer voorkomen in de databank.*

We kunnen dit alles formaliseren in het — in het jargon zo genoemde — *Association Rule Mining probleem*, dat staat voor het zoeken naar associatieregels. Dit probleem overlapt voor een stuk het *Frequent Set Mining probleem*, waarin er gezocht wordt naar vaak voorkomende verzamelingen van items. Vooraleer we deze twee problemen meer formeel kunnen neerschrijven, moeten we eerst een aantal belangrijke concepten definiëren.

Definitie A.1.2. *Om een willekeurige verzameling X netjes te kunnen opschrijven, veronderstellen we een* natuurlijke *orde op de elementen. Bijvoorbeeld, verzameling $X = \{a, b, c, d\}$ wordt genoteerd als* abcd. *De* lengte *van verzameling X is het aantal elementen verzameld in X en wordt genoteerd als $|X|$. Voor verzameling $X = \{a, b, c, d\}$ krijgen we dus $|X| = 4$. Dankzij de orde kunnen we het i-de element uit verzameling X nemen, met $1 \leq i \leq |X|$. Zo kunnen we bijvoorbeeld het 3de element in X vinden als $X[3] = c$.*

Definitie A.1.3. *Gegeven: een verzameling $\mathcal{I} = \{i_1, i_2, \ldots, i_m\}$ en een databank $\mathcal{D}$. De* support *van een willekeurige verzameling $X \subseteq \mathcal{I}$, genoteerd als $support(X)$, is het totaal aantal transacties in $\mathcal{D}$ die alle elementen van X bevatten. Deze elementen worden ook* items *genoemd.*

Definitie A.1.4. *Gegeven: een verzameling $\mathcal{I} = \{i_1, i_2, \ldots, i_m\}$, een databank $\mathcal{D}$ van grootte b en een ondergrens σ. We noemen een verzameling $X \subseteq \mathcal{I}$* frequent *als $support(X) \geq \sigma$. Als $\sigma \leq b$, dan spreekt het voor zich dat de lege verzameling $\emptyset$ frequent is.*

In concreto betekent dit dat we verzameling X frequent noemen als ten minste σ transacties in de databank $\mathcal{D}$ alle elementen van X bevatten. In ons supermarktverhaal betekent dit dat ten minste σ klanten, die verzameld zijn in databank $\mathcal{D}$, alle producten verzameld in X kochten. Voor wat er volgt in de rest van dit proefschrift nemen we aan dat de door de gebruikers gedefinieerde benedengrens $\sigma \leq b$.

Definitie A.1.5. *Gegeven: een verzameling van items $\mathcal{I} = \{i_1, i_2, \ldots, i_m\}$, een databank $\mathcal{D}$ en een ondergrens σ. Het* Frequent Set Mining probleem *bepaalt welke deelverzamelingen van $\mathcal{I}$ frequent zijn, dus voorkomen in tenminste σ transacties in $\mathcal{D}$. Deze verzamelingen worden* frequente verzamelingen *genoemd.*

Voorbeeld A.1.3. *Als $\sigma = 2$ in Voorbeeld A.1.1, dan vinden we $support(brood) = 2 \geq \sigma$, dus brood is frequent. We vinden ook $support(chips) = 1 < \sigma$, dus chips is niet frequent.*

De resultaten van de support in bovenstaand voorbeeld worden *absoluut* uitgedrukt, in functie van het aantal transacties in de databank. Andere werkwijzen in data mining werken *relatief*, in procent. Zij drukken de support uit als een percentage van het totale aantal transacties in de databank. In de rest van dit proefschrift zullen we gebruik maken van de absolute schrijfwijze.

Definitie A.1.6. *Gegeven: een verzameling $\mathcal{I} = \{i_1, i_2, \ldots, i_n\}$, een databank $\mathcal{D}$ van grootte b en een ondergrens σ. De* zoekruimte *voor frequente verzamelingen in $\mathcal{D}$ met betrekking tot σ is exponentieel in het aantal items aanwezig in de databank, n. Concreet, de grootte van de zoekruimte is van orde $\mathcal{O}(b\ 2^n)$.*

De hoeveelheid werk gerelateerd aan het frequent set mining probleem is bijgevolg afhankelijk van zowel de grootte van de databank, b, als van het aantal items in de databank, n.

Definitie A.1.7. *Gegeven: een verzameling items* $\mathcal{I} = \{i_1, i_2, \ldots, i_m\}$ *en een databank* $\mathcal{D}$. *Een* associatieregel *is een uitdrukking van de vorm* $X \Rightarrow Y$, *waarin* $X, Y \subseteq \mathcal{I}$ *en* $X \cap Y = \emptyset$. X *wordt de* body *of het* antecedent *genoemd en* Y *de* head *of de* consequent *van de regel.*

Associatieregels worden gebruikt om uit te drukken dat wanneer alle elementen uit verzameling X in een transactie voorkomen, dit normaal gezien ook het geval blijkt te zijn voor alle elementen uit Y. Anders gezegd, wanneer X voorkomt, komt Y waarschijnlijk ook voor. Deze onzekerheid kan bepaald worden aan de hand van de volgende definities.

Definitie A.1.8. *Gegeven: een verzameling items* $\mathcal{I} = \{i_1, i_2, \ldots, i_m\}$ *en een databank* $\mathcal{D}$. *De* support van een associatieregel $X \Rightarrow Y$ *is gedefinieerd als*

$$support(X \Rightarrow Y) := support(X \cup Y) \ .$$

We zijn enkel geïnteresseerd in regels waarvan de support groter is dan een zekere minimale benedengrens. Als de support niet groot genoeg is, betekent dit dat het niet de moeite waard is om deze regel te beschouwen of dat deze regel gewoon minder interessant is.

Definitie A.1.9. *Gegeven: een verzameling items* $\mathcal{I} = \{i_1, i_2, \ldots, i_m\}$, *een databank* $\mathcal{D}$, *en een ondergrens* σ. *Een associatieregel* $X \Rightarrow Y$ *is* frequent *wanneer zijn support de gegeven minimale ondergrens* σ *overschrijdt:*

$$support(X \Rightarrow Y) = support(X \cup Y) \geq \sigma \ .$$

Definitie A.1.10. *Gegeven: een verzameling items* $\mathcal{I} = \{i_1, i_2, \ldots, i_m\}$, *en een databank* $\mathcal{D}$. *De sterkte van een associatieregel* $X \Rightarrow Y$ *noemen we de* betrouwbaarheid. *Dit wordt gedefinieerd als de support van de regel gedeeld door de support van het antecedent van de regel:*

$$betrouwbaarheid(X \Rightarrow Y) := \frac{support(X \cup Y)}{support(X)} \ .$$

De betrouwbaarheid van een associatieregel is in feite de conditionele probabiliteit dat Y voorkomt in een transactie, gegeven dat X voorkomt in die transactie: $P(Y|X)$. Merk op dat er een fundamenteel verschil is tussen de support en de betrouwbaarheid van een associatieregel. De betrouwbaarheid is een maat voor de sterkte van de regel, terwijl de support overeenkomt met statistische significantie.

Definitie A.1.11. *Gegeven: een verzameling items* $\mathcal{I} = \{i_1, i_2, \ldots, i_m\}$, *een databank* $\mathcal{D}$, *en een ondergrens* c. *De associatieregel* $X \Rightarrow Y$ *wordt* betrouwbaar *genoemd wanneer zijn betrouwbaarheid de gegeven minimale betrouwbaarheidsondergrens* c *overschrijdt:*

$$betrouwbaarheid(X \Rightarrow Y) \geq l \ .$$

De door de gebruikers bepaalde minimale betrouwbaarheidsondergrens c dient om regels uit te sluiten die niet interessant genoeg zijn.

Voorbeeld A.1.4. *In ons lopend voorbeeld is de betrouwbaarheid van regel "brood* $\Rightarrow$ *boter"*

$$\frac{support(\{brood, boter\})}{support(brood)} = \frac{1}{2} = 50\% \ .$$

Definitie A.1.12. *Gegeven: een verzameling items* $\mathcal{I} = \{i_1, i_2, \ldots, i_m\}$*, een databank* $\mathcal{D}$*, een benedengrens* σ *voor de support en een benedengrens* c *voor de betrouwbaarheid. Het* Association Rule Mining probleem *heeft als doel het genereren van alle associatieregels* $X \Rightarrow Y$ *die voldoen aan de volgende twee eisen:* $support(X \Rightarrow Y) \geq \sigma$ *en* $betrouwbaarheid(X \Rightarrow Y) \geq c$*.*

Om dit doel te bereiken wordt dit probleem opgesplitst in *twee subproblemen*

1. Vind alle *frequente verzamelingen* $X \subseteq \mathcal{I}$.
2. Vind alle *associatieregels*: Test voor elke frequente verzameling X of voor alle niet lege deelverzamelingen $Y \subset X$ met voldoende betrouwbaarheid geldt dat $X \setminus Y \Rightarrow Y$.

Het tweede deel van het probleem kan op een relatief eenvoudige manier opgelost worden, eens alle frequente verzamelingen en hun overeenkomstige frequenties gekend zijn. Het eerste deel, het vinden van alle frequente verzamelingen, is het moeilijkste deel van het probleem. De verdeling in twee subproblemen legt de link tussen het frequent set mining probleem en het association rule mining probleem. Voor de rest van het proefschrift zullen we ons bezighouden met het frequent set mining probleem.

Het frequent set mining probleem [11, 12] is een belangrijk basisprobleem, dat verweven is met in een heleboel andere data mining problemen [11, 12, 13, 14, 22, 38, 41]. Het zoeken naar associatieregels is dus slechts een van de vele onderwerpen. Het is immers zo dat frequente verzamelingen niet alleen gebruikt worden om associatieregels te vinden [11, 12, 38], maar ook voor sequentiële patronen [13], classificatie [14], enz. Sinds de introductie van het frequent set mining probleem zijn er verschillende algoritmes voorgesteld om dit op te lossen [11, 12, 15, 18, 30, 35, 38, 41, 51, 57, 58, 59, 60]. Goethals geeft een gestructureerd overzicht van de meest belangrijke algoritmes [29].

Het Apriori Algoritme, waarop uitgebreid wordt ingegaan in Sectie 2.2, is waarschijnlijk het meest bekende algoritme dat frequente verzamelingen zoekt en vindt [11, 18]. Het is hoofdzakelijk gebaseerd op het volgende, zeer belangrijke principe.

Definitie A.1.13. *Het* Monotoniciteitsprincipe *of* Downward Closure Eigenschap *of* Apriori Eigenschap *[48]. Gegeven: een verzameling items* $\mathcal{I} = \{i_1, i_2, \ldots, i_m\}$*, een databank* $\mathcal{D}$*, en twee verzamelingen* $I_1 \subseteq \mathcal{I}$ *en* $I_2 \subseteq \mathcal{I}$ *zodanig dat* $I_1 \subseteq I_2$*. In databank* $\mathcal{D}$ *zal de support van* I_2 *hoogstens zo groot zijn als de support van* I_1*:*

$$support(I_2) \leq support(I_1) \ .$$

Gevolg A.1.1. *Als een rechtstreeks gevolg van dit Monotoniciteitsprincipe is de betrouwbaarheid van associatieregel* $X \Rightarrow Y$*, gedefinieerd in Definitie A.1.10, altijd kleiner dan of gelijk aan* 1*.*

Een heleboel nieuwe algoritmes zijn voorgesteld nadat Apriori verschenen is. Een beperkte selectie van deze algoritmes zullen we in detail beschouwen in Hoofdstuk 2.

Het zoeken naar alle frequente verzamelingen is een grote uitdaging omdat de zoekruimte exponentieel is m.b.t. het aantal items in de databank. Als er $|\mathcal{I}|$ items in de databank

zijn, dan leidt dit tot $2^{|\mathcal{I}|}$ verzamelingen die mogelijk frequent zijn. De meeste algoritmes die oplossingen genereren van het frequent set mining probleem zijn iteratief en vergen bijgevolg meerdere scans over de databank. Het spreekt vanzelf dat dit een dure aangelegenheid is. De grootte van de supportondergrens σ kan eventueel helpen om de output te beperken. Een efficient frequent set mining algoritme moet immers alle verzamelingen vinden die frequent zijn op basis van ondergrens σ, zónder dat het algoritme alle $2^{|\mathcal{I}|}$ mogelijkheden gaat testen. Vandaar dat de meeste algoritmes pre-testing hebben ingebouwd, om alle verzamelingen te elimineren die niet interessant zijn. Deze pre-testen zijn gewoonlijk gebaseerd op het feit dat frequenties monotoon zijn (Definitie A.1.13).

In dit proefschrift bestuderen we een belangrijke klasse van frequent set mining algoritmes analytisch: *de kandidaat-gebaseerde frequent set mining algoritmes*. Deze algoritmes volgen een strategie van *genereren* en *testen*: ze creëren voortdurend meer specifieke patronen, die kandidaten genoemd worden, en testen in de databank of deze kandidaten effectief frequent zijn. Dit proces wordt herhaald tot er geen nieuwe patronen meer gevonden kunnen worden. Bij deze aanpak is het belangrijk te weten dat het aantal nieuwe kandidaten die in elke stap gemaakt worden behoorlijk beperkt kan worden, door gebruik te maken van resultaten uit vorige iteraties. Afhankelijk van de precieze definitie voor kandidaten en de strategie gevolgd door het algoritme in kwestie, kunnen al dan niet meer of minder patronen buiten beschouwing gelaten worden. Een succesvolle strategie zal veel kandidaten genereren die effectief frequent blijken, en slechts weinig kandidaten die niet frequent blijken.

Een andere belangrijke klasse van algoritmes die we bestuderen in dit proefschrift zijn de *maximal frequent set mining algoritmes*. In dit geval zoeken we naar *grote* frequente verzamelingen. De bedoeling is dat we zo snel mogelijk pieken in de data distributie opsporen en benutten. De inspiratie voor dit onderzoek kan je terugvinden bij het Max-Miner algoritme en meer bepaald bij de vraag waarom dit algoritme zo goed werkt [15].

A.2 Stream Mining

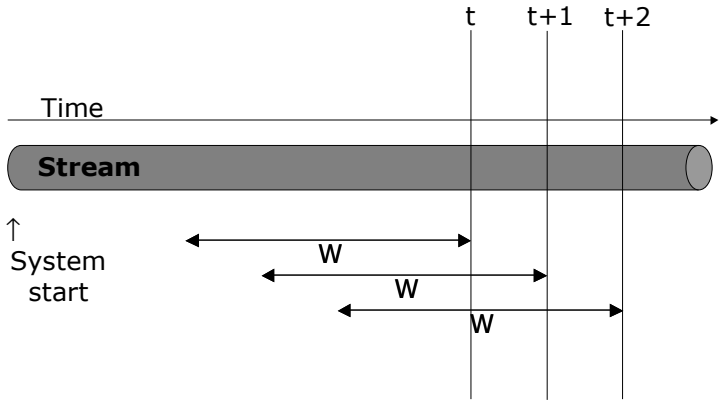
Hoewel we ons in de uiteenzetting tot nu toe vooral geconcentreerd hebben op het *off-line* zoeken naar frequente verzamelingen in *statische* databanken, beperkt de focus van dit proefschrift zich niet enkel tot deze problematiek. Ook het vinden van patronen in *online* sequenties of gegegevensstromen, *stream mining*, is een zeer interessant onderwerp dat vele toepassingen kent. De bedoeling hier is om gegevensstromen te onderzoeken, in de hoop interessante patronen te vinden. Als we deze situatie vergelijken met een statische databank, dan zien we onmiddellijk dat de dynamische gegevensstructuur meer informatie bevat en bijgevolg complexer is om te onderzoeken. Het is een uitdaging met behoorlijk wat toepassingen, gaande van computationele biologie tot network monitoring. Het meten van verbruik, sensor netwerk data analyse, het dynamisch traceren van aandelenfluctuaties op de beurs, enz: ze zijn allemaal gebaseerd op een studie van gegevensstromen. Zelfs onderzoek over klantengedrag kan uitgevoerd worden aan de hand van gegevensstromen. Bij al deze toepassingen kunnen we onderscheid maken tussen gegevensstromen bestaande uit individuele items, zoals bij IP-network monitoring, en gegevensstromen van variabele verzamelingen van items, zoals een sequentie van winkelkarinhouden aan de kassa van een warenhuis.

Het onderzoeken van gegevensstromen is totaal anders dan het traditionele onderzoek van databanken, en dit omwille van een aantal factoren. Een *gegevensstroom* bestaat uit transacties die in serie toekomen. Elke transactie is in feite een verzameling items, en is geassocieerd met een tijdstip. Al deze verzamelingen komen continu voorbij tegen hoge snelheid en de informatie die ze in zich dragen is over het algemeen groot, dus het is echt niet mogelijk om dit snel even op te slaan. Dit betekent dat gegevensstromen slechts *één* keer gelezen kunnen worden. Van zodra een item gepasseerd is, kan het niet meer terug bekeken worden. Gegevensstromen zijn ook onbegrensd, en zoals we reeds aangehaald hebben, tijdsafhankelijk. Al deze eigenschappen tonen dat het niet mogelijk is om de bestaande kennis van databanken klakkeloos over te nemen bij het gegevensstroomonderzoek. Als een duidelijk voorbeeld halen we aan dat het perfect mogelijk is dat een niet-frequente verzameling later in de gegevensstroom wel frequent kan worden, en daarom niet zomaar genegeerd mag worden. Vandaar dus ons idee om enkel cruciale informatie over de gegevensstroom te onthouden, en deze dynamisch aan te passen wanneer we beschikken over nieuwe informatie. Op deze manier kunnen we de evolutie van de gegevensstroom tonen over de tijd. Dit alles laat ons aanvoelen dat het een behoorlijke uitdaging is om het traditionele zoeken naar frequente verzamelingen uit te breiden en aan te passen naar deze dynamische omgeving.

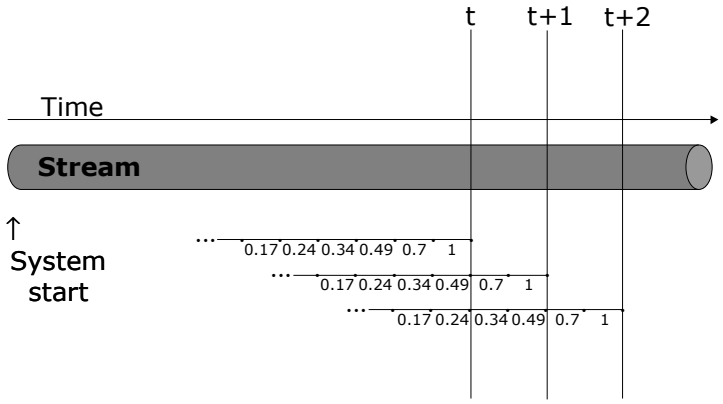
Omwille van de ruime toepasbaarheid van onderzoek in dit gebied, zijn we natuurlijk niet de eersten die gegevensstructuren bestudeerd hebben. Het meeste vroegere werk dat er reeds gebeurd is i.v.m. het zoeken naar vaak voorkomende items of vaak voorkomende verzamelingen in gegevensstructuren focust op 3 verschillende modellen: (1) het sliding window model, (2) het time-fading model, of (3) het landmark model. Het tilted-time venster combineert enkele basisideeën van de drie. Elk van deze modellen specifieert een *vaste* venstergrootte of vertragingsfactor, gegeven door de gebruiker.

In het *sliding window* model is de data die in het glijdende venster van vaste lengte valt het belangrijkst [26, 31, 40, 42, 45, 46, 55]. De bedoeling is hier immers om recente trends in de data terug te vinden: de nadruk ligt dus op de meest recente items in het venster. Dit venster heeft een voorgedefinieerde lengte, die bij het begin van de analyse wordt vastgelegd. De inhoud van het venster verandert als de gegevensstroom langskomt, en de tijd evolueert. Er zijn twee belangrijke manieren om naar dit probleem te kijken: via *count-based* vensters, waarbij het aantal transacties in het venster vast ligt ondanks de snelheid waarmee de gegevensstroom gegenereerd wordt, of via *time-based* vensters, waarbij het aantal tijdseenheden in het venster vastligt wat er voor zorgt dat het aantal transacties bijgevolg telkens kan veranderen. Als een transactie uit het venster glijdt, wordt deze geëlimineerd en stopt deze een bijdrage te leveren aan de frequentie. Figuur A.1(a) illustreert het sliding window model, wanneer we de laatste W transacties beschouwen.

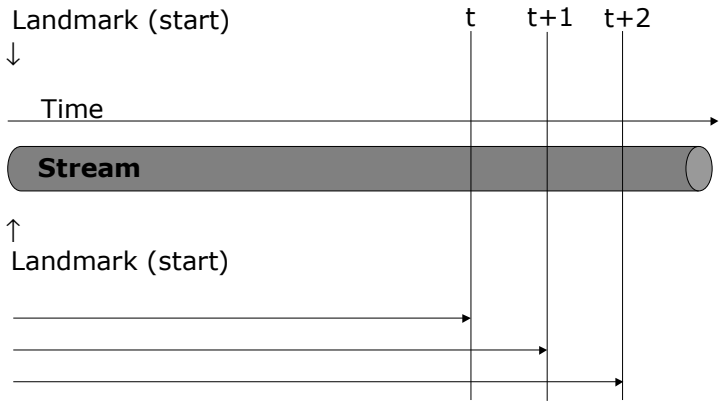
In het *time-fading* model wordt er rekening gehouden met de volledige gegevensstroom om de frequentie van een verzameling te berekenen. Het is echter wel zo dat dit op een tijdsgevoelige manier gebeurt: recente transacties zijn belangrijker dan oudere data. Bijgevolg wegen *recente* transacties *zwaarder* door in de frequentie, in vergelijking met oudere transacties. Dit gebeurt door gebruik te maken van een vertragingsmechanisme, gebaseerd op een vertragingsfactor d [44]. Het gebruik van zo'n factor resulteert in een exponentieel dalend gewicht voor transacties die langer geleden opdoken. Bijvoorbeeld, een transactie die n tijdstippen terug in de gegevensstroom verscheen wordt gewogen met factor d^n. In het algemeen kunnen we



(a) Het *sliding window* model



(b) Het *time-fading* model, met vertragingsfactor $d = 0.7$



(c) Het *landmark* model

Figuur A.1: Illustratie van de verschillende modellen in stream mining

besluiten dat hoe dichter de vertragingsfactor d bij 1 ligt, hoe meer er met de geschiedenis van de gegevensstroom rekening wordt gehouden. Een illustratie van het time-fading model kan je terugvinden in Figuur A.1(b). Elk tijdstip n correspondeert met een zekere transactie, die geassocieerd is met gewicht d^n. De gewichten dalen als de tijd voorbij gaat. Voor deze figuur is $d = 0.7$ gekozen.

In het *landmark* model wordt er *een zekere tijdsperiode* vastgelegd, vertrekkende op een zeker tijdstip, de zogenoemde landmark, die de start van het systeem markeert, tot aan de huidige tijd [40, 42, 56]. Gewoonlijk wordt er een selectie van landmarkpunten vastgelegd. De analyse van het systeem gebeurt dan enkel voor het deel van de gegevensstroom tussen het landmarkpunt en het huidige tijdstip, zoals je kan zien in Figuur A.1(c). Het belangrijkste nadeel van deze methode is dat de grootte van het venster verandert: het start met grootte nul en groeit tot de volgende landmark, waar het terug op nul wordt gezet.

Het *tilted-time* venster is geïntroduceerd als een *combinatie* van al deze methodes [20, 21]. De techniek haalde zijn inspiratie hoofdzakelijk bij de time-fading opvattingen. Dit kan je zien aan de veranderende tijdsschalen voor de vensters als de tijd verstrijkt. Recente data wordt bekeken met een fijne tijdsschaal, terwijl data van langer geleden onderzocht wordt met een ruwere tijdsschaal. Voor een verzameling worden de frequenties berekend voor de meest recente vensters, met lengtes w, $2w$, $4w$, ..., enz. Door dit te doen wordt het mogelijk om op een efficiënte manier vragen te beantwoorden over de geschiedenis van de gegevensstroom, terwijl er toch rekening gehouden wordt met het feit dat recente gegevens belangrijker zijn.

Al deze methodes tonen aan dat het meeste reeds geleverde werk i.v.m. het zoeken naar vaak voorkomende verzamelingen over gegevensstromen gewoonlijk werkt met een venster waarvan de lengte op voorhand wordt *vast*gelegd. In veel toepassingen is het echter *niet* mogelijk om op voorhand reeds te weten welke de beste venstergrootte is of welke vertragingsfactor het meest optimaal is voor elke verzameling op elk moment in de groeiende gegevensstroom. Om dit probleem te verhelpen, stellen wij in dit proefschrift een nieuwe *frequentiemaat* voor, gebaseerd op een flexibele venstergrootte. Deze nieuwe maat voor het frequent voorkomen van verzamelingen in gegevensstromen is totaal verschillend van de maten die reeds in de literatuur zijn voorgesteld. Voor zover wij weten zijn we de eerste die een aanpak voorstellen waarbij er geen vast venster of een vaste vertragingsfactor wordt gebruikt. Meer nog, het blijkt dat in alle gerelateerd werk de analyses bij benadering gebeuren, dus er een fout wordt toegelaten op de resultaten. Dit is niet zo in ons werk. Onze werkwijze levert *exacte* resultaten op, in vergelijking met de benaderingen in ander werk.

We hebben ook een *algoritme* ontwikkeld, gebaseerd op deze nieuwe maat, dat een korte *samenvatting* van alle relevante informatie gebaseerd op de geschiedenis van de gegevensstroom bijhoudt. Deze samenvatting laat ons toe de huidige frequentie van een specifieke verzameling onmiddellijk op te vragen. Belangrijk voor onze techniek is de plaats die de samenvatting inneemt in het geheugen van de computer. We berekenen een *theoretische bovengrens* op de grootte van de samenvatting, voor een bepaalde soort gegevensstromen, de *Farey* stromen. *Experimenten* tonen dat deze bovengrens voor realistische data distributies zeer klein is, en dit zelfs terwijl deze in het slechtste geval theoretisch gezien afhankelijk is van de lengte van de gegevensstroom. Het spreekt voor zich dat zo'n kleine samenvatting absoluut te wensen is, omdat we zo onze nieuwe maat zeer snel en efficiënt kunnen verkrijgen.

A.3 Structuur van het proefschrift

A.3.1 Frequent Set Mining

In hoofdstuk 2 geven we een overzicht van een selectie van algoritmes uit een belangrijke klasse van algoritmes voor het vinden van vaak voorkomende verzamelingen: de *kandidaat-gebaseerde* frequent set mining algoritmes. Deze algoritmes worden getypeerd door het volgen van een strategie van *genereren* en *testen* van kandidaat verzamelingen. We stellen een beperkte selectie van deze algoritmes voor, waarmee we zullen verder werken in de rest van dit proefschrift. We focussen in eerste instantie op de *kandidaatstest* op basis waarvan elk algoritme kandidaten genereert. In de volgende hoofdstukken zullen we dan een gedetailleerde analyse uitvoeren van deze beperkte selectie van belangrijke algoritmes.

Zo kan de lezer in hoofdstuk 3 een theoretische, probabilistische analyse terugvinden van de gebeurtenissen *kandidaat*, *succes* en *mislukking*, en dit voor verschillende data distributies, in de offline context van statische databanken. Als eerste stap focussen we op de constructie van kandidaten, aangezien dit een zeer belangrijke stap is in de werking van frequent set mining algoritmes. Naast de kandidaatsprobabiliteit bestuderen we ook de probabiliteiten van de concepten succes en mislukking, en dit voor een zeer eenvoudig data model. Hiertoe nemen we aan dat alle transacties in de databank onafhankelijk zijn en dat elke combinatie van items zijn eigen probabiliteit heeft, dus dat eender welke vorm van correlatie tussen de items mogelijk is. De resultaten van deze basisanalyse worden dan gebruikt om de verschillende algoritmes te vergelijken over verschillende types van datasets.

Hoofdstuk 4 gaat dan verder met het bestuderen van het gedrag van maximal frequent set mining algoritmes. Vanaf dit hoofdstuk gaan we dus niet meer rechtstreeks op zoek naar alle frequente verzamelingen, maar enkel naar de maximale frequente verzamelingen. Eens we deze verzamelingen gevonden hebben, kunnen op basis hiervan alle frequente verzamelingen gereconstrueerd worden. Datasets die zulke grote frequente verzamelingen bevatten, vertonen pieken in hun distributie. Onze bedoeling is nu om tijdens onze zoektocht in zulke datasets zo snel mogelijk te *springen* naar deze verzamelingen, om alzo in het begin van de analyse reeds deze grote frequente verzamelingen te vinden. Op deze manier kunnen we de bijbehorende kennis uitbuiten met als doel uiteindelijk minder werk te doen. We stellen een collectie van algoritmes voor die springen, en dit voor een eenvoudig data model. Onze analyse verklaart waarom het springen soms werkt, en we geven de kenmerken waaraan een dataset moet voldoen opdat dit het geval zou zijn. Uiteindelijk vinden we Max-Miner terug, een belangrijk en reeds bekend algoritme uit de literatuur [15]. Terzelfdertijd leggen we dus in feite uit waarom Max-Miner zo goed werkt.

A.3.2 Stream Mining

Vanaf hoofdstuk 5 verandert de klemtoon van het proefschrift naar Stream Mining. Omdat het, zoals reeds eerder aangehaald, niet eenvoudig is om een vaste venstergrootte of vertragingsfactor te bepalen aan het begin van de analyse van de groeiende gegevensstroom, die dan werkt voor alle verzamelingen op alle tijdstippen, stellen we voor om voor elke verzameling het venster te beschouwen waarin deze verzameling de hoogste frequentie heeft. Alle details over deze nieuwe definitie van frequentie worden gegeven, en dit zorgt voor een compleet nieuwe visie op gegevensstromen. We zijn immers, voor zover wij weten, de eerste die een maat

voorstellen op basis van een flexibele venstergrootte. We gaan natuurlijk in op belangrijke eigenschappen van deze nieuwe maat. We stellen ook een incrementeel algoritme voor dat ons toelaat de huidige frequentie van een verzameling onmiddellijk op elk moment te geven. Het algoritme houdt een kleine samenvatting bij waarin bepaalde cruciale informatie uit de gegevensstroom is opgeslagen. Telkens wanneer er een nieuwe transactie in de gegevensstroom binnenkomt, wordt deze samenvatting aangepast. Een aantal interessante resultaten volgt automatisch. Het spreekt voor zich dat het belangrijk is dat deze samenvatting niet te groot wordt. Een analyse van de slechtst mogelijke situatie gebeurt m.b.v. een theoretische bovengrens op de grootte van de samenvatting, voor een specifiek type van gegevensstromen, de Farey gegevensstromen. Experimenten tonen aan dat voor realistische data distributies de grootte van de samenvatting klein blijft, en dit ondanks het feit dat de theoretische analyse aantoont dat de grootte van de samenvatting in het slechtste geval afhankelijk is van de lengte van de gegevensstroom.

Index

Personal Bibliography - Addendum

[61] T. Calders and N. Dexters. Theoretical bounds on the size of condensed representations. In *Proceedings ECML-PKDD 2004 Workshop Knowledge Discovery in Inductive Databases*, pages 25–36, 2004.

[62] T. Calders, N. Dexters, and B. Goethals. Mining frequent items in a stream using flexible windows. In *ECML/PKDD-2006 International Workshop on Knowledge Discovery from Data Streams (IWKDDS)*, 2006.

[63] T. Calders, N. Dexters, and B. Goethals. Mining frequent itemsets in a stream. Proc. IEEE Int. Conf. on Data Mining, 2007.

[64] T. Calders, N. Dexters, and B. Goethals. A new support measure for items in streams. *Le Monde des Utilisateurs de L'Analyse de Données (La Revue MODULAD), http://www.modulad.fr*, 36:37–41, June 2007.

[65] T. Calders, N. Dexters, and B. Goethals. Mining frequent items in a stream using flexible windows. *Intelligent Data Analysis, Special Issue on Knowledge Discovery from Data Streams*, May 2008.

[66] N. Dexters. Approximating the probability of an itemset being frequent. In *BNCOD22*, pages 1–4, Sunderland, UK, 2005. Poster-paper.

[67] N. Dexters. A probabilistic analysis for frequent itemset mining algorithms. DBDBD '05, October 31 2005, Amsterdam, October 2005. Talk for the Dutch-Belgian Database Day, October 31, Amsterdam.

[68] N. Dexters. Statistics in data mining: Finding frequent patterns. Poster Statistics@Antwerp, January 2007. ISBN 978-90-5728-070-2.

[69] N. Dexters, P. W. Purdom, and D. Van Gucht. Peak-jumping frequent itemset mining algorithms. In *PKDD*, pages 487–494. Springer, 2006.

[70] N. Dexters, P. W. Purdom, and D. Van Gucht. A probability analysis for candidate-based frequent itemset algorithms. volume 1 of 2, pages 541 – 545. Applied Computing 2006 - Proceedings of the 2006 ACM Symposium on Applied Computing, Dijon, France, 2006.

Erratum

Unfortunately, there are some typos in the proof of **Theorem 5.3.2** on page 109.

Theorem 5.3.2. Let $\mathbb{S}$ be a stream, and let $1 \leq q \leq |\mathbb{S}|$. Position q is a *border* for target set A in $\mathbb{S}$ *if and only if* for all indices j, k with $1 \leq j < q$ and $q \leq k \leq |\mathbb{S}|$, it holds that

$$\mathit{freq}(A, \mathbb{S}[j, q-1]) \;<\; \mathit{freq}(A, \mathbb{S}[q, k]) \;.$$

Proof.

$\Rightarrow$ This follows directly from Theorem 5.3.1, as illustrated in the original proof (page 109).

$\Leftarrow$ We need to show that there exists a continuation $\mathbb{S}'$ of stream $\mathbb{S}$, resulting in stream $\mathbb{S} \cdot \mathbb{S}'$, in which q is the starting point of the maximal window. We consider *two* cases: either q is the rightmost border in $\mathbb{S}$, or not.

If q is the rightmost border, then q is the maximal border in $\mathbb{S}$, because for any other border $1 \leq p < q$, $\mathit{freq}(A, \mathbb{S}[p, q-1]) < \mathit{freq}(A, \mathbb{S}[q, |\mathbb{S}|])$ which implies $\mathit{freq}(A, \mathbb{S}[p, |\mathbb{S}|]) < \mathit{freq}(A, \mathbb{S}[q, |\mathbb{S}|])$, and hence the Theorem holds.

In the other case, we have shown in the rest of the proof that it is always possible to continue $\mathbb{S}$ in such a way that the rightmost border disappears, while all other borders remain and no new borders are introduced. By consecutively applying this procedure, any border will eventually become the rightmost border at a certain point, and hence become the starting point of the maximal window.

Let $q < q'$ be the two largest borders in $\mathbb{S}$. Because of the only-if part, we have

$$\mathit{freq}(A, \mathbb{S}[q, q'-1]) \;<\; \mathit{freq}(A, \mathbb{S}[q', |\mathbb{S}|]) \;=\; \frac{\mathit{count}(A, \mathbb{S}[q', |\mathbb{S}|])}{|\mathbb{S}| - q' + 1} \;. \tag{A.1}$$

Because of (A.1), we can always find positive integers $x \leq y$ such that

$$\mathit{freq}(A, \mathbb{S}[q, q'-1]) = \frac{\mathit{count}(A, \mathbb{S}[q', |\mathbb{S}|]) + x}{|\mathbb{S}| - q' + 1 + y} \;. \tag{A.2}$$

Based on parameters x and y, we construct $\mathbb{S}'$, the continuation of $\mathbb{S}$, as follows:

$$\langle \overbrace{\boldsymbol{A}\ \boldsymbol{A}\ \cdots\ \boldsymbol{A}}^{x} \overbrace{\emptyset\ \emptyset\ \cdots\ \emptyset}^{y-x} \rangle \;.$$

It is clear that the translation of condition (5.13) has to be

$$\frac{a+u}{b+v} \;\geq\; \frac{a+u+\boldsymbol{n}+x}{b+v+m+y} \;\Leftrightarrow\; \frac{a+u}{b+v} \;\geq\; \frac{\boldsymbol{n}+x}{m+y} \;.$$

The resulting stream $\mathbb{S} \cdot \mathbb{S}'$ has exactly the same borders as $\mathbb{S}$, except for q', which is no longer a border. $\square$