Università degli Studi di Roma Tre

Dipartimento di Matematica e Fisica

Scuola Dottorale in Scienze Matematiche e Fisiche

XXVIII Ciclo

Ph.D. Thesis

Space-Time Efficient and Secure Algorithms for Data Processing and Transmission

---

Candidate:
Giulio Aliberti

Advisor:
Prof. Roberto Di Pietro

Anno Accademico 2015/2016

“Data is the new science.
Big Data holds the answers.”

PAT GELSINGER

Abstract

Over the past twenty years, the generation of digital data has been increasing due to the widespread use of computers for capturing, analysing, and transmitting different kinds of information. The benefits deriving from the capability of being able to efficiently process large amount of data are countless, ranging from increasing revenues of business companies to accumulating knowledge for research purposes. The main purpose of this thesis is to present original and efficient solutions for the secure processing and transmission of data. More generally, in this manuscript my aim is to discuss relevant topics for this area, including data compression, frequent pattern discovery and retrieving "human-understandable" information from datasets, and secure "light-weighted" protocols for transmitting data ensuring integrity and confidentiality of the message. For each one of these topics, a thorough discussion of the relevant state of the art is provided and at least an original solution is presented and evaluated through both theoretical and experimental analysis.

General Terms: *Theory, Algorithms, Experiments.*

Additional Key Words and Phrases: *Data Science, Bitmap Compression, Data Mining and Knowledge Discovery, Physical Layer Security.*

Acknowledgements

Behind every PhD thesis there is a student, and behind every student there is a supervisor. I am grateful to my supervisor, Professor Roberto Di Pietro, who managed to find such a good balance between the seemingly contradicting requirements of a supervisor: he guided me firmly through the process while still providing me with the liberty to make my own decisions and mistakes. The research in this thesis was done at the Department of Mathematics and Physics, University of Roma Tre and at the Bell Labs (Paris, France). I gratefully acknowledge the financial support from them. However, a place by itself means nothing without the people populating it. And I have been lucky to have such great people around me. I thank all my co-authors for letting me work with you and learning from you. Especially I shall never forget the deep influence Alessandro Colantonio and Stefano Guarino had on me when I was starting my PhD studies. I am grateful to many friends and colleagues of mine for their readiness to exchange ideas and dwell upon random topics: Stefano Marini, Flavio Lombardi and the other colleagues with whom I have shared an office all these years; Leonardo Linguaglossa and Giuseppe Scavo, with whom I have had many pleasant discussions during my visit in Paris; and, fellow summer school, I thank you all. I owe much to my parents who taught me the importance of education and who have always supported me. But above all I must thank my beloved girlfriend Paula.

Contents

List of Figures

List of Tables

List of Algorithms

CHAPTER 1

Introduction

Over the past years, plethora of data have been collected with the help of different instruments including web applications, mobile devices, sensors, and loyalty cards to mention a few. The trend is not going to stop in the years to come since, accordingly to recent estimations, ninety percent of the data in the world was just created in the last five years [31] and it is growing at a rate of forty to sixty percent per year [11]. We are witnesses of the *data explosion* phenomenon, where the volume and the variety of data have already exceeded the storage capacity of conventional databases and far outstripped the potentiality of manual analysis. At the same time, *innovative technologies* are emerging to organize this avalanche of data. From the hardware standpoint, computers became far more powerful, hard drives much more capacious, and networking is ubiquitous; while from a software standpoint, algorithms and platforms have been designed and developed to enable broader and deeper analysis than previously possible increasing the efficiency of the available computational resources.

The convergence of these phenomena is providing to business companies the opportunity to *exploit data* for competitive advantages. For instance, an online retailer can track what customers are buying, what they are looking at, how they are navigating through the site, how much they are influenced by promotions, reviews, and page layouts. In a short time, with the appropriate analysis tools, the retailer can predict what items individual customers are likely to purchase next and can better estimates the returns of investments in advertising campaigns. In conclusion, the online retailer can profit from data by taking smarter decisions with the long term effect of putting competitors out of business. The previous example describes a typical scenario where exploiting data matters, but there is much more. For instance, it is common practice to leverage data in *machine learning* applications, such as artificial intelligence in games, image and speech recognition, and even self-

driving cars. Miscellaneous applications can also be found in *finance*, *human resources*, *health care*, *government policies* and every possible industry where data gets generated [83].

Fostered by economic and social perspectives, the new *data science* field is emerging from the intersection of the different fields of social science, statistics, information, computer science, and design. At a high level, data science is a set of fundamental principles that support and guide the extraction of information and knowledge from data [79]. The most closely related concept to data science is *data mining*, that is the actual process of extracting knowledge from data via algorithms and technologies that implement these principles. However, data mining represents just a sub-field of data science since the latter involves much more than information retrieval algorithms. Indeed, data science is more about dealing with a business problem from a data perspective, while data mining is just a suite of tools for the analysis. In this picture, statistical and social science are necessary knowledges to provide a better understanding of the problem and to drive smarter business decisions.

The common intersection between all the fields that compose data science is, precisely, the increasing volume of data available. In turn, this key point is strictly related with practical and economic issues arising from *storage*, *processing*, and *transmission* of this large quantity data. Indeed, increases in available data means more storage, increased expenditure on overall power consumption, floor space and cooling, while also threatening the feasibility of processing and transmitting the collected data. In order to better sustain high-level activities of data science, new technologies and algorithms to deal efficiently with big volumes of data must be developed.

1.1 Contributions

The contributions provided in this thesis are manifold. Several algorithms for storage, analysis and transmission of data are introduced with the common goal of illustrating how to optimize the available computational resources. Necessary discussions are made to ensure that their connections are clear. From a less general perspective, the thesis also contributes to the scientific literature by bringing in three innovative and top performing algorithms for storage, analysis and secure transmission of data, respectively. A more detailed description of these contributions is provided in the following while the relation to the prior work of the author is discussed in Section 1.2.

Bitmap Compression. The first main contribution is the bitmap compression algorithm CUTSIZE. Bitmaps are data structures widely adopted for imple-

menting databases and, thanks to the compression performed, the presented algorithm is capable of reducing storage requirements for preserving data in several application contexts. The compression is *lossless*, namely without any loss of information, and the memory performance is comparable with that achieved by the best available bitmap compression algorithms in the literature. However, where CUTSIZE really excels is in the management of the compressed data structure. Since decompression is not necessary to perform bitwise operations between bitmaps, the data stored are ready to be processed even after compression. As a result, the data structure obtained with the application of CUTSIZE can be processed with a time save of up to ten times compared to the other data structures obtained with currently adopted bitmap compression solutions.

Frequent Pattern Discovery. The second main contribution is the algorithm for frequent pattern discovery EXPEDITE. One of the main objectives of data analysis is to discover meaningful information within the data and EXPEDITE serves to accomplish this task. Accordingly to a pre-set threshold value that formalizes the idea of frequency, the algorithm retrieves all the frequent patterns within the data — more precisely, *frequent closed itemsets* — behind the rationale that recurring patterns are the most interesting from a human perspective. The literature is plenty of algorithms that perform, in different manners, the same task of extracting frequent patterns. However, compared to the best performing available options, EXPEDITE offers a huge time saving of up to two orders of magnitude which makes it the ideal candidate for processing big amount of data.

Light-Weighted Secure Data Transmission. The third main contribution is the algorithm for light-weighted secure data transmission CRYPTOLESS. Usually, establishing a secure communication between two authenticated parties involves the employment of cryptographic primitives to preserve integrity and confidentiality of transmitted data, although these solutions add substantial delays to the communication due to the cost of cryptographic operations. Less known are the light-weighted solutions — *i.e.* solutions not based on cryptography — for achieving secure data transmission on resource constrained devices. CRYPTOLESS is one of the few available in the literature and it has the additional benefit of providing correcting capabilities to strengthen the reliability of the communication. Despite its promises, CRYPTOLESS cannot be applied regardless to the communication channel and this poses severe limits to applications. Indeed, its security is based on certain physical properties of the channel — *i.e.* the algorithm is designed for the *generalized Ozarow-*

Wyner's wire-tap channel — which must be satisfied in order to provide security. Interestingly, these conditions are satisfiable in a local wireless network and different machines can be securely connected with CRYPTOLESS.

1.2 Organization of the Content

The thesis deals with three relevant topics of data science: storage, analysis and secure transmission of data. To ease exposition, these topics are discussed into three separated chapters. First, Chapter 2 studies the problem of storing data with bitmaps, data structures widely adopted for implementing databases; and, it introduces the innovative bitmap compression algorithm CUTSIZE to improve the storage efficiency. Second, Chapter 3 addresses the problem of analysing data focusing on the extraction of frequent patterns; and, it introduces the innovative algorithm EXPEDITE, the faster solution available in the literature. Third, Chapter 4 deals with the secure data transmission topic focusing on light-weighted solutions; and, it introduces the innovative algorithm CRYPTOLESS, a solution applicable in the generalized Ozarow-Wyner's wire-tap channel. The last chapter, Chapter 5, contains concluding remarks.

Reading this Thesis. The three main chapters of the thesis have a similar structure. In particular, they start with an introductory section that describes and motivates the main problem addressed in the chapter; then, to illustrate currently adopted techniques, a survey on the state of the art follows; they continue with the presentation of the innovative technique proposed by the author; and, finally, possible future directions are highlighted. Due to the adopted structure, the reader interested only in one specific topic can read just the related chapter, although connections with the other chapters could be lost. The reader interested only in innovative algorithms can go directly to Section 2.3, Section 3.3, and Section 4.4 skipping most on non-technical discussions. Moreover, the most technical parts like proofs are highlighted within the text and the reader can easily skip them on a first reading in order to capture a better global picture.

Relation to Prior Work of the Author. This thesis is mainly based on the main three scientific contributions provided by the author listed below. The notation and terminology have been unified, the connections between the articles have been made more clear, and some missing details have been supplied. The three scientific articles on which this thesis is based are:

[5] Giulio Aliberti, Alessandro Colantonio, and Roberto Di Pietro. "CUTSIZE:

a Bitmap Compression Scheme for Fast Bitwise Operations". Technical report.

[6] Giulio Aliberti, Alessandro Colantonio, Roberto Di Pietro, and Riccardo Mariani. "EXPEDITE: EXPress closED ITemset Enumeration". In *Expert Syst. Appl.*, 42, no. 8 (2015): 3933-3944.

[7] Giulio Aliberti, Stefano Guarino, and Roberto Di Pietro. "CRYPTOLESS: Reliable and Perfectly Secret Communication over the Generalized Ozarow-Wyner's Wire-Tap Channel." Submitted to Computer Networks.

In particular, innovative results in Chapter 2 arise from the technical report *"CUTSIZE: a Bitmap Compression Scheme for Fast Bitwise Operations"* [5]; those in Chapter 3 from the publication *"EXPEDITE: EXPress closED ITemset Enumeration"* [6]; and those in Chapter 4 from the submission *"CRYPTOLESS: Reliable and Perfectly Secret Communication over the Generalized Ozarow-Wyner's Wire-Tap Channel"* [7]. The algorithm solvers presented in each one of the three main chapters are new additions in their respective fields of research. Although the problems of storing, analysing and transmitting are being discussed in the literature since the very beginning of the computer science foundation, discursive parts of the thesis on processing and transmitting data are also new and aim to highlight connections between the different articles.

CHAPTER 2

Storing Data with Bitmaps

With the data deluge under way, many business companies and several organizations are facing issues with an infrastructure that is bursting at the seams in an attempt to store the collected data. Solving this storage problem might seem straightforward at a first glance: organizations could just acquire more storage capacity. However, businesses work with budgets and do not have unlimited resources. Thus, the answer is not that simple. A recent solution for reducing costs lies in turning to *storage virtualization* to make better use of thee existing physical infrastructure. This kind of solution offers centralized management of storage without adding more complexity [26]. It means that business companies have to invest less on the hardware, and on the power, the cooling, and the management of the storage. Further, virtualization is just a part of a larger trend that will make storage much more efficient. In particular, it is of key importance for deploying *compression* technologies which are at the heart of making hardware better [31]. Indeed, compression increases storage efficiency by shrinking the data so that a greater volume of information can be stored in the physical disk. Some compression technologies go one step further by operating on data as they are written into disk. This improves also the efficiency of processing the stored data since decompression is no longer required to their elaboration.

There is a big variety of data types in circulation to whom correspond different compression techniques. In the data science context, *bitmaps* are very import type of data. In fact, they are among the most widely adopted data structures for database management due to the very powerful method that they provide for rapidly answering to difficult queries [30]. Over other data structures, bitmaps have the advantage of working with aligned bits which is a property exploitable by a Central Processing Unit (CPU). However, the reward of adopting bitmaps is proportional to the *density* of the data: if used for storing sparse information, bitmaps are indeed a wasteful data structure in

both memory and time. Recent research effort focused on developing *bitmap compression* schemes mainly aiming at reducing wastes of memory. However, most of the adopted solutions are not optimized for improving the CPU performance.

The final goal of this chapter is to present *Compressed Ultra Tiny Sets of Integers by Zeroes Erasure* (CUTSIZE), an innovative bitmap compression algorithm that is optimized for allowing fast bitwise operations between compressed bitmaps. In comparison with *Word Aligned Hybrid* (WAH), that represents the milestone in bitmap compression, the proposed algorithm offers similar compression capabilities but it has the advantage of allowing to perform bitwise operations between compressed bitmaps up to ten times faster. All findings are supported by both theoretical and experimental analyses.

Roadmap of the Chapter. Section 2.1 provides a detailed description of bitmaps and it better introduces the problem of their compression. Section 2.2 describes solutions available in the literature and Section 2.3 provides a detailed presentation of the innovative solution CUTSIZE. Finally, Section 2.4 concludes the chapter discussing possible future directions.

2.1 The Bitmap Compression Problem

This section is divided as follows. First, Section 2.1.1 provides a formal introduction to bitmaps; then, Section 2.1.2 shows a toy example of application; and, finally, Section 2.1.3 introduces the problem of bitmap compression.

2.1.1 The Bitmap Data Structure

Bitmaps [87, 100], also known as *bit arrays* or *bit vectors*, are array data structures that store individual bits and whose length n is a multiple of the word size w of the Central Processor Unit (CPU), namely the amount of information per cycle that the processor can manipulate — typically, $w = 32$ or $w = 64$ in modern architectures. Due to their inherent *data alignment*, bitmaps are ideal for exploiting bit-level parallelism and, consequently, bitwise operations between bitmaps are well supported by computer hardware. Besides being useful to store data, bitmaps allow to interpret queries in database as a composition of bitwise operations. For this reason, bitmaps find natural applications in data warehouse and, more generally, where performing fast queries is necessary [101, 46].

A bitmap of length $n = k \times w$ bits, with w word size of the processor and k length in words of the bitmap, can be associated to a subset of integers

$A \subseteq [n] = \{1, \ldots, n\}$ by mapping the i-th bit of the bitmap to the integer i — in the following example of Section 2.1.2, the integer i corresponds to the row identifier. Hence, a logical "1" in the position i of the bitmap means that the integer i belongs to A while a logical "0" means that i does not belong to A. Through this mapping, bitwise operations between two bitmaps, such as logical AND and logical OR, correspond to set operations, intersection and union respectively. Since the CPU handles w bits per cycle, the computational cost of a bitwise operation between two bitmaps is just k cycles.

2.1.2 Case Example: Storing Data with Bitmaps

To make reading more clear, a toy example is here provided to explain how information is stored within bitmaps. Table 2.1 shows the gender attributes of four individuals and a row identifier associated to their names. Both values of

Name	Gender	RowID
Alice	F	R1
Bob	M	R2
Eve	F	R3
Daniel	M	R4

Table 2.1: Gender attribute of four individuals.

the gender attribute, *i.e.* Male and Female, can be associated to a separated bitmap as shown in Table 2.2. Two different bitmaps of length $n = 4$ are so obtained. Each one has set the entry "1" in column "Rx" if and only if the individual associated to the identifier Rx matches with that attribute value. This structure is very handy for defining queries. For instance, to recover all male individuals it is only needed to finds all the row identifiers where the bit was set to 1, and recovering both male and female individuals is simply possible by performing a logical OR between matches positive to male and female attributes. The same identical procedure can be used to store data from attributes having any integer number of possible values or to store data from multiple attributes. However, it must be notice that, despite this representation is usually very efficient in the context database management, both time and memory performances could drop off rapidly if the attributes can assume more than few hundreds of different values [30].

2.1.3 Bitmap Compression

In Section 2.1.1 has been explained that performing set operations with bitmaps, namely logical bitwise operations, takes k CPU cycles. This method is usually

Gender	R1	R2	R3	R4
Male	0	1	0	1
Female	1	0	1	0

Table 2.2: Bitmap based representation of the gender attribute.

convenient compared with other data structures, such as inter arrays or linked lists. However, this is true only if the two processed bitmaps contain, in average, more than one logical "1" per word. Indeed, this condition means that an average of multiple integers per cycle would be processed. Conversely, if the two bitmaps comprises long sequences of logical "0", several CPU cycles would be wasted without computing any useful information. In addition, using a bitmap to store the value of an attribute that could be also a waste of memory if that attribute is "almost always" unmatched.

To overcame issues related with the density of the integers in the sets to represent, several *bitmap compression* schemes have been proposed over the past years. These algorithms, or schemes, mainly aim to reduce memory wastes when storing integers with bitmaps, but some of them also focus on improving the efficiency of the CPU usage when performing queries. These goals are achieved through compression of long sequences of logical "0", that is indeed an effective method for reducing the waste of memory. However, if not correctly tuned, a compression method could also be detrimental for time performance in data processing applications. This mainly happens when compressed bitmaps must be decompressed before performing set operations with the integers that they are storing.

Due to the importance in practical applications of preserving the time performance of the bitmap data structure, the development and the diffusion of schemes that allow to work directly on the compressed bitmaps has happened. Thus, modern algorithms allow to perform bitwise operations without the need of decompressing the data structure. This feature drastically reduces the execution time of performing bitwise operations with compressed bitmaps, although there is still a unavoidable loss in terms of time in comparison with simple uncompressed bitmaps. In practical applications where both time and memory dimensions matter, such as data warehouse, working with compressed bitmaps is a well accepted compromise. In fact, these schemes offer both the benefits of allowing relatively fast queries and the possibility of storing large datasets in a compressed way.

2.2 State-of-the-Art

The first proposed method for compress bitmaps is the *Run-Length Encoding* (RLE) [43], a simple algorithm that consists in counting and encoding the number of consecutive occurrences of the same logical value — *i.e.* "0" or "1" — within the data. This solutions is not very efficient for arbitrary data since short "runs" of the same logical value lead to an increase of size in stored data. Further, runs must be decompressed in order to query the data. Nevertheless, the run-length encoding is still currently used in few applications mainly due to its simplicity. Over the past years, better solutions have been studied and they are reviewed in the following.

Byte-aligned Bitmap Code (BBC) [9, 8] is the first proposed bitmap compression scheme that aims to preserve the alignment with the data. With BBC, bitmaps are initially aligned in byte blocks. Then, aligned bytes are classified as "gap bytes" if all the bits in that byte store the same logical value, and as "map bytes" otherwise. Adjacent bytes of same class are grouped together and, finally, groups of bytes are losslessly compressed into atomic sequences of bytes. The BBC method achieves remarkable compression ratios but, despite the block alignment, bitwise operations between compress bitmaps are still time consuming to perform.

The *Word-Aligned Hybrid* (WAH) [104, 105, 102, 103] bitmap compression scheme differs from BBC mostly for using words instead of bytes for matching the architecture of the processor. The compression ratio achieved by WAH is a little bit worse compared with BBC. However, the alignment with the CPU word size better supports bitwise operations between compressed bitmaps. This feature significantly improves the runtime of performing bitwise operations and contributed to the widespread use of this scheme. Despite many years have passed, WAH is today the most adopted bitmap compression scheme in applications, and it is still mentioned in the literature as one of the best solutions developed so far.

A more recent method is given by *Position List Word-Aligned Hybrid* PLWAH [35, 76]. This scheme is an extension of WAH whose difference lies in being able to compress also long runs that are exceptionally interrupted by a single bit. This feature is mainly relevant when the data stored are sparse; in this case, the compression ratio achieved by PLWAH is up to two times better compared with WAH. As for the complexity of performing bitwise operations, the runtime achieved by PLWAH and WAH are comparable in both dense and sparse data.

The literature also includes other relevant bitmap compression scheme which are briefly discussed hereafter (see [23] for a recent survey). They

are: *Compressed 'n' Composable Integer Set* (CONCISE) [28], *COMPressed Adaptive indeX format* (COMPAX) [42], *Enhanced Word Aligned Hybrid* (EWAH) [60], and *Variable Length Compression* (VLC) [32]. The CONCISE method is comparable with PLWAH. In fact, both CONCISE and PLWAH are designed to improve the compression ratio when a single bit interrupts a long run. Experimental results show that they achieve similar performances in both time and space [28, 35]. The COMPAX method is also comparable with PLWAH. Similarly, it is designed to compress efficiently long runs interrupted by few "dirty" bytes. In theory, the approach proposed with COMPAX for addressing the compression of dirty data is more general than those implemented in PLWAH and CONCISE. However, in practice, the compression ratios achieved by COMPAX are comparable to those achieved by PLWAH andCONCISE [42]. Another bitmap compression method is EWAH that, differently from others, focuses on improving the runtime of bitwise operations between compressed bitmaps. This feature is obtain starting from the WAH scheme and employing more storage resources for preserving information useful to execute bitwise operations. Despite EWAH succeeds in its goal of reducing the runtime when querying the data, long runs of the same logical value are not well compressed with it. Thus, EWAH noticeably wastes memory resources in comparison with WAH [46, 56] and, overall, it is not worth to choose EWAH over WAH unless it is priorly known that the data have no long runs. Another method for compressing bitmaps is VLC. It uses arbitrary segment lengths for improving compression ratios. However, due to decoding cost overheads, querying data becomes an operation significantly more expensive when compared to the previously described solutions [32]. Finally, *Variable Aligned Length* (VLA) [46] deserves a mention for being a general framework for optimizing the compression of bitmaps by using variable encoding lengths. It has the advantage of maintaining alignment to avoid explicit decompression when performing bitwise operations and it is theoretically adaptable to most of the reviewed bitmap compression methods. So far, it has been applied only to WAH and the achieved results show an improvement in the memory footprint without significant changes in performances when trying to query the compressed data.

2.3 Bitmap Compression with CUTSIZE

This section presents CUTSIZE, an innovative, simple, and efficient bitmap compression scheme. The content is divided as follows. Section 2.3.1 provides a description of CUTSIZE; Section 2.3.2 studies the theoretical behaviour of the proposed scheme; and, Section 2.3.3 illustrates experimental results.

2.3.1 Description of the Algorithm

To easy exposition, we start the presentation of CUTSIZE by discussing the case example of Figure 2.1 which provides a first intuitive explanation of the algorithm. After that, the description of the pseudo-code of CUTSIZE, reported in Algorithm 2.1 and Algorithm 2.2, follows.

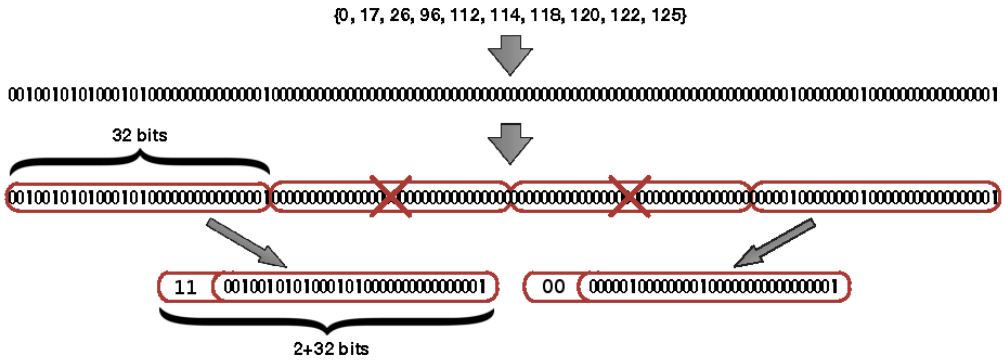


Figure 2.1: Case example of bitmap compression with CUTSIZE. First, the bitmap is subdivided into four aligned and indexed words. Then, the words that contain no integers are deleted.

Case Example Referring to Figure 2.1, it is explained in the following how to compress and decompress with CUTSIZE the bitmap that represents the set of integers $S = \{0, 17, 26, 96, 112, 114, 118, 120, 122, 125\}$ using $w = 32$ as word size. The set S can be naturally associated with the uncompressed bitmap of 128 bits that has a logical value "1" in the i-th position, if and only if the integer i belongs to the set. The resulting bitmap, that consists of four words, is illustrated at the top of the picture — the *least significant bit* is the rightmost one. It is possible to notice from the picture that two words of the bitmap have all bits set to "0", this corresponds to the fact that the intervals 32–63 and 64–95 do not contain any integer. The term *empty* will be used to distinguish this kind of words made by all zeros; thus, the second and the third word are empty with this terminology.

The scheme CUTSIZE is based on the idea that, since empty words do not represent any integer, they can be completely deleted, or "cut", to reduce wastes of both memory and computational resources. Indeed, removing words from bitmaps is effective at compressing the data. In addition, performing bitwise operations becomes faster due to smaller sizes of the bitmaps. However, to remove words from the bitmap without losing information, it is necessary

to index and preserve the positions of the words. To realize this, to each word of the bitmap CUTSIZE attaches an header that contains the position of that word in the bitmap. After that empties words are deleted, the headers of the remaining words provide enough information to rebuild the initial bitmap. At the bottom of Figure 2.1 it is shown the final result that coincides with the compressed bitmap generated with CUTSIZE. Just two of the initial words are remaining, namely the first and the fourth, each one with its own header which represents their position in binary notation.

When required by applications, decompressing a bitmap stored using the CUTSIZE compression is a straightforward operation. Indeed, the procedure consists in copying non-empty words in the right position — which is known due to the headers — in the uncompressed bitmap and padding with "0s" the gaps. The number of bits used to represent each of the two words is $h + w = 2 + 32 = 34$ where $w = 32$ is the word size and $h = 2$ is the chosen length in bits of the header. In particular, h corresponds to the logarithm of the word length of the bitmap — *i.e.* $h = log_2(4)$ in this example.

Algorithm Description The pseudo-code that describes how to compress and decompress using CUTSIZE is reported in Algorithm 2.1 and Algorithm 2.2, respectively. Both procedures, commented in detail hereafter, have a computational complexity that is linear in the bit length n of the uncompressed bitmap. Indeed, the most costly operation of the compressing procedure is a scan of the input bitmap, which happens only once; and, the most costly operation of the decompressing procedure is a copy of the bits in the compressed bitmap (less than n) while filling gaps with zero (the remaining bits up to n).

Algorithm 2.1 Procedure COMPRESS of CUTSIZE.

```
1:  procedure COMPRESS(Bitmap B)
2:      int l ← n/w
3:      Bitmap C ← new Bitmap()
4:      for i = 1,...,l do
5:          Bitmap word ← B.nextWord()
6:          if word.isNotEmpty() then
7:              C.append(i)
8:              C.append(word)
9:          end if
10:     end for
11:     return Bitmap C
12: end procedure
```

The procedure COMPRESS takes in input the initial bitmap B to be com-

pressed, whose length n is a multiple of the word size w of the CPU (Line 1), and it returns in output the compressed bitmap (Line 11). The length in bits of the bitmap n is converted into its length in words l performing the division $l = n/w$ (Line 2), and a new bitmap C for storing the compressed bitmap is created (Line 3). After this initialization step, the words of the uncompressed bitmap B are processed one by one into a logical cycle (Lines 4–10). At each iteration, the i-th word of the bitmap B (Line 5) is temporarily stored into a new bitmap $word$. Note that this iteration can be efficiently implemented by shifting w bits in B. If the current word $word$ is not empty (Line 6), then the header i is appended to the bitmap C (Line 7) — after a binary conversion into h digits — and also the bits in $word$ are appended to C (Line 8). Otherwise, if the current word is empty the cycle continues with the following iteration. After that, all words in B have been processed, the compressed bitmap C is finally returned in output (Line 11).

Algorithm 2.2 Procedure DECOMPRESS of CUTSIZE.

```
1:  procedure DECOMPRESS(Bitmap C)
2:      int l ← n/(w+h)
3:      Bitmap B ← new Bitmap()
4:      int header ← 0
5:      for i = 1,...,l do
6:          int tempHeader ← C.nextHeader()
7:          Bitmap word ← C.nextWord()
8:          if tempHeader = header then
9:              B.append(word)
10:             header ← header + 1
11:         else
12:             for j = 1,...,tempHeader do
13:                 B.appendEmptyWord()
14:             end for
15:             header ← tempHeader
16:         end if
17:     end for
18:     return B
19: end procedure
```

The procedure DECOMPRESS takes in input the compressed bitmap C (Line 1) and returns in output the initial uncompressed bitmap B (Line 18). In the initialization step, the word length of the compressed bitmap C is retrieved by dividing the bit length n of C by the sum of the word size w and the header size h (Line 2); then, and the bitmap B for storing the uncompressed bitmap

is created (Line 3); and, finally, the auxiliary variable $header$ is initialized to zero (Line 4). After that, the words stored into the compressed bitmap C are processed one by one in logical cycle (Lines 5–17). At the i-th iteration, first a number of h bits from C are shifted and moved into a temporary variable $tempHeader$ (Line 6), and then w are shifted and moved into a temporary variable $word$ (Line 7). Thus, these two variable contain the header and the word of the i-word stored into the compressed bitmap C. When the variables $header$ and $tempHeader$ coincide (Line 8), it means that there are no gaps of empty words between two consecutive words stored into C. Thus, the bits in the variable $word$ are appended to the uncompressed bitmap B (Line 9) and the header is increased by one (Line 10). Otherwise, when $header$ and $tempHeader$ differ, it means that there is a gap of empty words that must be covered. The number of empty words is precisely given by the variable $tempHeader$ and a logical cycle is performed to append these empty words to B (Lines 12–14), and the variable header is updated to $tempHeader$ (Line 15). After that, all words in C have been processed, the uncompressed bitmap B is correctly recover and returned in output (Line 18).

2.3.2 Theoretical Analysis

In this section, we provide a theoretical analysis of two fundamental aspects of CutSize: memory footprint and runtime needed to perform bitwise operations.

Memory Footprint The memory footprint of both compressed and uncompressed bitmaps is strongly dependent on the density of the data. To better understand this behaviour, it is possible to consider the case that a bitmap B of length n is used to represent a set $S \in \{1, \ldots, n\}$ of $m = n/w$ integers. On average, each word of the bitmap B would contain only one integer and, thus, approximately m words would be required to store the n/w elements of S. Increasing the number of elements of S to $m > n/w$, the bitmap B begins to be advantageous. Indeed, on average, more than one integer would be stored into the same word. Thus, a number $m > n/w$ of integers could be stored with less than m words. Conversely, when the number m decreases to $m < n/w$, the bitmap B begins to waste memory due to a low average of stored integers per word. The cardinality of the set S, that is the number of integers m to be stored into the bitmap B, is a measure of *density* of the data. It can be normalized as follows:

$$\delta = m/n \in [0, 1]. \tag{2.1}$$

Thus, the density in the critical value $m = n/w$ becomes

$$\delta_c = 1/w. \tag{2.2}$$

To be more precise, previous discussions are only valid when data are uniformly at random distributed — namely, when the set S is uniformly at random generated. Otherwise, the average number of integers per words changes accordingly to the chosen distribution. However, in absence of specific assumptions on the nature data, the uniform distribution seems to be the more neutral for performance evaluation purposes.

When adopting CUTSIZE as data structure, and more generally with any bitmap based compression scheme, the critical value of density δ_c in which the data structure changes its behaviour is still $\delta_c = 1/w$ (for uniformly distributed datasets). In fact, considerations identical to the above case of uncompressed bitmaps can be made. Thus, in low density datasets, formally those that satisfy the condition $\delta < \delta_c$, each word of the compressed bitmap contains, on average, less than one integer. In these datasets, the integer array data structure would uses w bits per integer; hence, the CUTSIZE scheme wastes h bits for any integer to store, also the header of the word representing that integer must be preserved with CUTSIZE. Better said, CUTSIZE achieves the compression ratio of

$$R_1 = \frac{w+h}{w} > 1 \tag{2.3}$$

words per integer in low density datasets that is inefficient since the better compression ratio equal to 1 could be easily achieved using the integer array data structure. However, in high density datasets, namely those satisfying the condition $\delta > \delta_c$, several integers belong to the same word and CUTSIZE effectively compresses the data. In detail, in the limit of $\delta \to 1$, CUTSIZE would use a total of $w+h$ bits to store w integers, achieving a compression ratio of $(w+h)/w$ bits per integer or, equivalently, a compression ratio of

$$R_2 = \frac{w+h}{w^2} < 1 \tag{2.4}$$

words per integer. The integer array data structure performance is not affected by the density of the data and, thus, the compression ratio is still 1 that is significantly worse compared to CUTSIZE. Experimental results in Section 2.3.3 confirm that the evaluation of the compressed ratio R_1 and R_2 are correct, and further comparisons with other scheme in the literature are provided.

Runime of Bitwise Operations The runtime for performing bitwise operations between bitmaps also depends on the density of the data. In fact, during

Algorithm 2.3 Procedure AND of CUTSIZE

```
1:  procedure AND(Bitmap C_1, Bitmap C_2)
2:      int l ← n_1/(w + h_1)
3:      Bitmap C ← new Bitmap()
4:      for i = 1,...,l do
5:          int header_1 ← C_1.nextHeader()
6:          Bitmap word_1 ← C_1.nextWord()
7:          int header_2 ← −1
8:          while header_2 < header_1 do
9:              if C_2.hasNext() then
10:                 int header_2 ← C_2.nextHeader()
11:                 Bitmap word_2 ← C_2.nextWord()
12:             else
13:                 return C
14:             end if
15:             if header_1 = header_2 then
16:                 Bitmap word ← word_1 AND word_2
17:                 if word.isNotEmpty() then
18:                     C.append(header1)
19:                     C.append(word)
20:                 end if
21:             end if
22:         end while
23:     end for
24:     return C
25: end procedure
```

each clock cycle, the CPU processes a word that could contain from 0 to w integers. In sparse datasets, the CPU cannot exploit the typical alignment found in bitmap based data structures, since most of the words are storing none or one integer. Conversely, in very dense datasets, up to w integers are likely to be processed during each clock cycle. The critical value of density where this behaviour changes is still $\delta_c = 1/w$. In fact, for theoretical evaluations, the average number of integers stored in a single word is the core property of the dataset that only matters. To provide more detailed analysis, the pseudocode in Algorithm 2.3 for performing the bitwise operation AND between two compressed bitmaps is reported and commented in the following. Note that there is no significant difference when using a different logical operator, such as OR or XOR. Later in Section 2.3.3, experimental results using uniformly distributed data are reported to better compare time performance with the

currently adopted compression schemes.

The procedure described in Algorithm 2.3 performs a logical AND between two bitmaps C_1, C_2 compressed with CUTSIZE whose lengths after compression are n_1 and n_2, and whose header sizes are h_1 and h_2, respectively. The word size w is equally set for both bitmaps. The two bitmaps C_1 and C_2 are provided in input of the procedure (Line 1) that will return the compressed bitmap C (Line 24) consisting in the intersection of the integers stored in C_1 and C_2. During the initialization step, the minimal word length of C_1 is computed[1] (Line 2) and the bitmap C is created (Line 3). The main operations are performed inside two nested loops. The outer loop processes the words of the bitmap C_1 (Lines 4–23), while the inner loop those of the bitmap C_2 (Lines 8–22). In detail, at each outer iteration reads and temporarily stores the next header (Line 5) and the next word (Line 6) of the compressed bitmap C_1 inside the two variables $header_1$ and $word_1$, respectively. This can be efficiently implemented using shift operations. Then, the temporarily variable $header_2$ is initialized (Line 7) to -1 before entering in the inner loop. Inside this loop, if C_2 has at least more one word to process (Line 9) also the next header (Line 10) and the next word (Line 11) of C_2 are stored inside the two temporarily variables $header_2$ and $word_2$, respectively. Otherwise, if C_2 has no more words to process the procedure ends (Line 13). This step is repeated until the variables $header_1$ and $header_2$ either coincide or happens that $header_2$ crosses $header_1$. In case that, the variables $header_1$ and $header_2$ coincide (Line 15), it means that the compressed bitmaps C_1 and C_2 have a non trivial — *i.e.* not all bits are set to zero — intersection of integers and, thus, the logical AND between words $word_1$ and $word_2$ must be performed (Line 16). If the resulting word is not empty, then it is appended with the header to the compressed bitmap (Lines 17–20). In case that $header_2$ crosses $header_1$, the procure continue with the following iteration of the outer loop. At the end of the procedure the compressed bitmap C representing the intersection between integers stored in C_1 and C_2 is obtained and returned in output (Line 24).

This procedures applies the operator AND between two words at most $\min_{i=1,2}\{n_i/(w+h_i)\}$ times and it reads at most $n_1/(w+h_1)+n_2/(w+h_2)$ words. Compared to the runtime for performing bitwise operations using uncompressed bitmaps, CUTSIZE achieves better CPU performance in low density datasets. In fact, in these datasets, CUTSIZE performs less AND operations between words because empty words are not processed at all. Instead, for high density datasets both schemes achieve approximately the same performance since the number of AND operations between words is comparable. However,

[1] To optimize the procedure, the word length of C_2 can be also computed with the purpose of eventually swapping C_1 C_2 if the length of C_2 is smaller.

compared to uncompressed, there is a little and unavoidable worsening in time performance in dense datasets due to logical operations required to handle the headers. Experimental results in Section 2.3.3 confirm the previous analysis.

2.3.3 Experimental Results

In the following, experiments are performed to confirm the theoretical analysis of CUTSIZE provided in Section 2.3.2. In particular, the main goal is to evaluate both memory footprint and execution time for computing bitwise operations CUTSIZE. The results are compared with some of the most important bitmap compression schemes available in the literature, namely CONCISE and WAH, and they are also compared with some classical data structures commonly used to store integers: *Bitmap*, *Array*, *Linked List*, *Hashtable*, *Balanced Tree*. Experiments are performed on a notebook equipped with an Intel CoreTM 2 Duo CPU P8600 at 2.40 GHz and 3GB of RAM with word size $w = 32$ bits. First, the data used to perform the experiments are described, and then the results are provided.

Data Description The data used to perform experiments consist of synthetic integers sets generated with a uniform distribution. In detail, using the density parameter $\delta \in (0,1]$, a maximum integer number $n_\delta = \lfloor 10^5/\delta \rfloor$ is fixed obtaining the range of integers $[1, n_\delta]$. For different values of δ, several sets of integers of cardinality 10^5 each one are generated by uniformly at random selecting integers from the range $[1, n_\delta]$. For instance, when $\delta = 1$ this procedure leads to 10^5 integers in the range $[1, 10^5]$; this corresponds to all the integers in the range, well representing the meaning of the density parameter δ. Conversely, for $\delta << 1$ the 10^5 integers are selected in a wide range obtaining, thus, a sparse dataset. To improve clearness in the results, the experiments with bitmap-based data structures — *i.e.* Bitmap,CUTSIZE, WAH, CONCISE — are made with bitmaps of length n_δ using $w = 32$ as word size; and, other data structures — *i.e.* Array, Linked List, Hashtable and Balanced Tree — allocate $w = 32$ bits for integer to store independently on the density of the data.

Memory Footprint Figure 2.2a depicts the memory footprint of analysed data structures as a function of the dataset density. On the x-axis (log-scale) is represented the density $\delta \in [0,1]$ and on the y-axis the ratio words per integers average, that is the number of words of allocated memory divided by the number of integers in the dataset. The figure shows that the behaviours

of Array, Linked List, Hashtable and Balanced Tree are independent on the density of the dataset. The reason is that these data structure allocate a fixed amount of memory for every stored integer. For instance, Array uses always one word for each stored integer and, consequently, it achieves a compression ratio of words of allocated memory per integer constantly equal to one. Similarly, for every stored integer, Linked List allocates $w = 32$ bits for the integer and five additional words for storing references to previous and next elements (see [96] for more details). Thus, also the compression ratio of Linked List is a constant that do not depends on the density of the dataset. Among this kind of data structures, Array is the more convenient in terms of allocated memory. Differently, the performance of bitmap based data structures is strongly affected by the density of the dataset. In Section 2.3.2, the critical value for the density, namely $\delta_c = 1/w \approx 0.031$, was derived. The experiments confirm that all these data structure change their behaviour in this point. Indeed, Figure 2.2a shows that the compression ratio of Bitmap crosses the line relative to Array, as predicted by theory, in the critical value. It is worth noticing that the bitmap based compressed data structures begin to outperform Array even for lower values density; for instance, CUTSIZE begins to be more memory efficient than Array for $\delta \approx 0.02$. The reason is that the probability that two integers are stored into the same word for $\delta < \delta_c$ is not zero in uniformly distributed data. Thus, there is a positive probability that CUTSIZE (and other bitmap compression schemes) compresses two or more integers into the same word even for density $\delta < \delta_c$. It is also possible to notice that, for $\delta \to 1$, the limit of the compression ratio of CUTSIZE derived in Eq. (2.4), namely $R \approx 0.044$ words per integer (this value is computed using the header size equal to the binary logarithm of n), is confirmed by the experiments (the last point of the picture corresponds to 0.04722). Bitmap, WAH, CONCISE achieve slightly better ratios; precisely, the last point is 0.039 word per integer for density $\delta = 0.8272$. In fact, in the limit for $\delta \to 1$, they tend to store w integers in each word, achieving a ratio of $1/w \approx 0.031 < R$. Thus, CUTSIZE scheme is (slightly) less efficient in compressing the data. However, the difference is very small and, indeed, CUTSIZE outperformed WAH in low density datasets, *i.e.* for $\delta < \delta_c$. In conclusion, despite not representing the optimum, the compression ratio achieved by CUTSIZE is comparable with those of the best ones available in the literature, for both dense and sparse datasets.

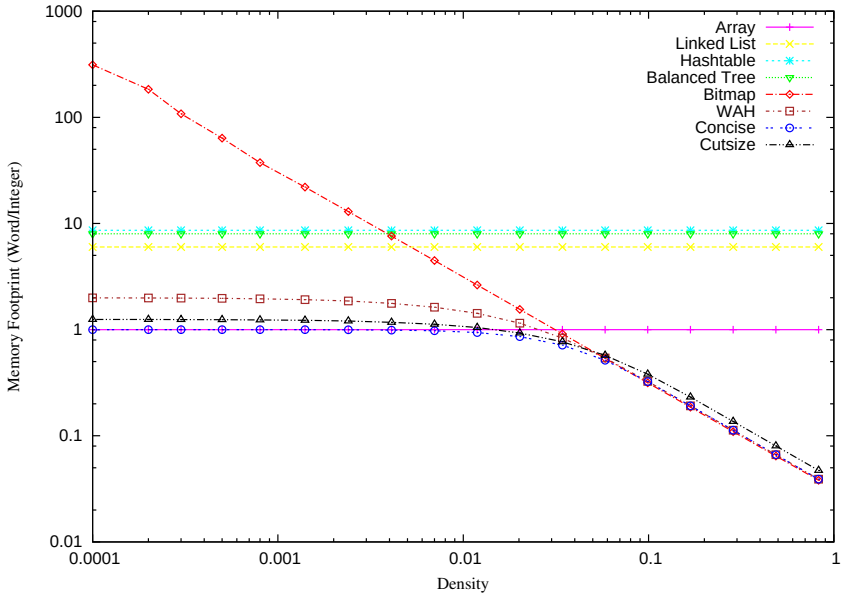
Execution Time Figure 2.2a reports, for each analysed data structure, the execution time for computing the intersection operation between two sets of integers. For bitmap based representations, the intersection corresponds to the bitwise AND operation. On the x-axis (log-scale) is represented the den-

sity $\delta \in [0,1]$ and on the y-axis (log-scale) is represented the time spent to perform the operation. The figure shows, again, that the behaviours of Array, Linked List, Hashtable and Balanced Tree are independent on the density of the dataset. This is intuitive since the intersection operation is not affected by how longs are the gaps between integers when adopting these data structures. Differently, these gaps becomes relevant when adopting a bitmap based data structure. For instance, in low density datasets Bitmap performs very badly due to the presence of a large number of empty words. In fact, these words are processed even if they do not store any integer with the consequence of wasting computational resources. Despite this disadvantageous, Bitmap becomes the fastest data structure when the data become dense. Indeed, compared to other bitmap based data structures Bitmap do not suffer from the overhead costs of managing the headers of the compressed words. Currently adopted methods, such as WAH, CONCISE, mitigate the drawbacks of Bitmap by improving the time efficiency in low density datasets. However, this only happens for a limited range of δ. In fact, the figure shows that Bitmap starts to outperform WAH, CONCISE nearly at $\delta > 10^{-3}$. In comparison,CUTSIZE achieves a much better performance. In fact, CUTSIZE is more time efficient than Bitmap up to a value of density $\delta \approx 10^{-2}$ providing a CPU time saving for a much wider range of density. Furthermore, CUTSIZE outperforms currently adopted schemes in both sparse and dense datasets with consistence. In particular, in low density datasets, CUTSIZE behaves like the Array data structure and provides an improvement of one order of magnitude compared to WAH and CONCISE; and, in high density datasets, it is still twice faster.

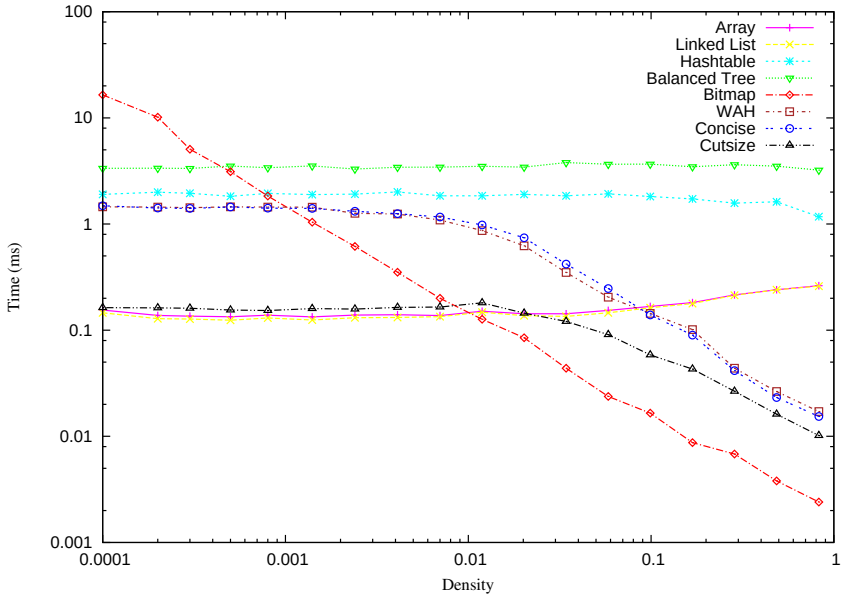
2.4 Future Directions

Being able to exploit the bit level parallelism, computations over bitmaps often outperform those over other data structures such as binary search trees, hash tables and simple arrays. In this chapter, we introduced an innovative bitmap compression algorithm scheme, called CUTSIZE, that provides a significant CPU time saving when performing bitwise operations between compressed bitmaps. Indeed, accordingly to theoretical and experimental analyses, CUTSIZE is up to ten times faster in comparison with some of the most adopted bitmap compression schemes available in the literature, such as WAH and CONCISE. Due to its inherent properties, CUTSIZE is well suited for parallelization over different processors. The reason is that all the procedures used by CUTSIZE, such as compression, decompression, and the bitwise operations, could be naturally subdivided into smaller instances due to presence of the headers in the stored words. Thus, realizing a parallelizable version of CUT-

SIZE is possible in principle and its realization would mainly consist in finding a clean and efficient procedure for distributing the workload accordingly to these headers. This promising direction of research could lead to another significant improvement in time performance with strong consequences for data storage applications.



(a) Memory Footprint



(b) Runtime

Figure 2.2: Performance evaluation of several data structures for storing integers. Fig. 2.2a compares the memory footprint by reporting the ratio of words used per integer stored. Fig. 2.2b compares the runtime of the intersection between two sets of integers.

CHAPTER 3

Frequent Pattern Discovery

Due to the upward trend in collecting data for business and social purposes over the past years, nowadays vast amounts of data are available to analysts who are responsible for discovering meaningful information hidden inside all the stored data. This work requires to sift through an immense amount of material and to intelligently probe it to find exactly where the value resides. Due to the large size of the datasets to process, these activities are often infeasible by human means and powerful computers are a basic need perform analysis. This motivates the development of algorithms explicitly designed to support data analysis activities fuelling an active area of research with plentiful applications, often referred to as *data mining and knowledge discovery* in datasets [40].

The field of data mining includes at least four different topics corresponding to *clustering*, *classification*, *outlier analysis*, and *frequent itemset mining* [2]. Just to provide a rough idea, clustering is the process of making a group of abstract objects into classes of similar objects; classification is the process of predicting categorical class labels; outlier analysis is the process of identifying abstract objects that do not comply with the general behaviour of the data; and, frequent itemset mining, is the process of discovering those patterns that occur frequently within the data. Compared to the other three problems, frequent itemset mining is relatively recent but, in spite of its shorter history, it is considered the marquee problem of data mining and it has now a special place in the data mining research community.

The field of frequent pattern mining is considered a mature one and the development of innovative algorithms has slowed down due to the increasing difficulties in outperforming the performance achieved by existing solutions. Nevertheless, there is a high demand for more efficient frequent itemset mining algorithms. This is proven by the fact that recent research effort focused on developing parallelizable versions of the best available solutions [17, 72].

Improving existing frequent mining algorithms is still possible even without resorting to parallel computations. As a proof of concept, this chapter presents *EXPress closED ITemset Enumeration* (EXPEDITE), that is an innovative frequent itemset miner algorithm designed to speed up the process of extracting frequent patterns from the stored data. Compared to the state of the art, EXPEDITE provides a CPU time saving of up to two orders of magnitude without compromising other dimensions of performance (*e.g.* memory). The findings are analytically motivated first, and then experimentally supported by extensive tests on real datasets.

Roadmap of the Chapter. Section 3.1 introduces the frequent itemset mining problem. Section 3.2 describes solutions available in the literature and Section 3.3 provides a detailed presentation of the innovative solution EXPEDITE. Section 3.4 complements the description of EXPEDITE discussing possible variations of the algorithm and Section 3.5 extensively discusses experimental results. Finally, Section 3.6 concludes the chapter discussing possible future directions.

3.1 Frequent Itemset Mining

This section is dived as follows. First, Section 3.1.1 introduces the frequent itemset mining problem; then, Section 3.1.2 provides the formal notation required for the reminder of the chapter; and, finally, Section 3.1.3 shows a toy example of application.

3.1.1 The Frequent Itemset Mining Problem

Since its introduction, the *Frequent Itemset Mining* (FIM) problem [3] has been one of the most studied problems in data mining and there have been hundreds of follow-up publications due to its importance in applications. To name a few, pattern matching is used to analyse the behaviour of individuals in webshops, the proteins interactions in bioinformatics, and the frequent symptoms of a disease in the health-care domain [47, 57].

FIM instantiates over a dataset of *transactions*, where each transaction is made up of *items*. The goal is to find *frequent itemsets* (FIs), namely sets of items occurring in at least a user-specified percentage of transactions. The formal terminology comes from the original dataset used to introduce FIM, which contains purchases of customers in a supermarket: each transaction corresponds to a set of products (*i.e.* items) purchased by a customer and the

goal is to understand the behaviour of buyers by discovering which products are often bought together, namely finding the FIs. The described scenario is known in the literature as *market basket analysis* [3] and toy example of dataset is discussed later in Section 3.1.3. However, as already mentioned, there are many other real scenarios of application since the only requirement is dataset of binary attributes which is very common to find.

In the early days of FIM, most approaches were focused on enumerating all FIs as efficiently as possible. However, the number of FIs that can be extracted from real-world data is often so huge that their storage and post-processing operations require unrealistic resources. As a consequence, several condensed representations for FIs have been proposed to drastically reduce the number of patterns to extract [22]. The most relevant representations were based on closed sets [75], on disjunction-free sets [21] and on δ-free sets [19]. The one based on closed sets, commonly known as *frequent closed itemsets* (FCIs), or also as *closed frequent itemsets* (CFIs), grasped particular attention within the data mining community. This representation is a lossless, sound and informative compression of the larger set of all the FIs [59]. This means that the set of FCIs allows to derive any FI, do not include redundant elements and provides information on the data even without the need of decompressing the representation. These appealing features led the researchers to further develop FCIs miners rather than FIs miners.

3.1.2 Basic Definitions and Notation

This section formally defines the idea of *frequent* and *closed* itemsets while also providing required notation for explaining the algorithm EXPEDITE.

Definition 3.1 (FIM)**.** *FIM can be modelled with a triplet $\langle \mathscr{T}, \mathscr{I}, \mathscr{D} \rangle$, where $\mathscr{T}$ denotes the set of* transactions*, $\mathscr{I}$ the set of* items *and $\mathscr{D} \subseteq \mathscr{T} \times \mathscr{I}$ the set of binary relations between transactions and items. In particular, $\langle t, i \rangle \in \mathscr{D}$ means that the transaction $t \in \mathscr{T}$ contains the item $i \in \mathscr{I}$. To each transaction, is associated a unique identifier, commonly known as* tid*, and a set of transactions $T \subseteq \mathscr{T}$ is often called* tidset*. Instead, a set of items $I \subseteq \mathscr{I}$ is called* itemset *and a k-*itemset *is an itemset $I \subseteq \mathscr{I}$ made up of k items, that is $|I| = k$.*

The goal of FIM is to detect the subset of all possible itemsets such that they occur frequently within the data. The idea is that it is more interesting to know which are the items occurring in a large number of transactions rather then those occurring occasionally together. The following definition formalizes the meaning of frequent itemset.

Definition 3.2 (Frequent Itemset)**.** *The* support *of an itemset $I \subseteq \mathscr{I}$ is the number of transactions that contain I. Formally, it is defined as the function* $\text{support}\colon \wp(\mathscr{I}) \to \mathbb{N},\ I \mapsto |\{t \in \mathscr{T} : \langle i, t\rangle \in \mathscr{D}, \forall i \in I\}|$ *where $\wp()$ denotes the power set and $||$ the cardinality function. An itemset I is said to be* frequent *(FI for short) when its support is greater or equal than a user-specified minimum support threshold $\delta \geq 0$, namely when* $\text{support}(I) \geq \delta$.

When increasing the threshold, the number of itemset to mine drastically decreases. At the same time, enumerated itemsets are less descriptive of the given dataset. The main benefit of introducing the minimum support is the speed improvement of mining algorithms. However, when facing a big dataset, for acceptable values of δ the number of itemset to mine is still so large that their storage and their post-processing operations require unrealistic resources. To overcome this issue, closed itemsets have been proposed.

Definition 3.3 (Closed Itemset)**.** *The function* $\text{tidset}\colon \mathscr{I} \to \wp(\mathscr{T})$ *identifies the set of all the transactions that contain a given item, formally* $\text{tidset}(i) = \{t \in \mathscr{T} \mid \langle i, t\rangle \in \mathscr{D}\}$. *Given an itemset $I \subseteq \mathscr{I}$, with abuse of notation, we refer to* $\text{tidset}(I)$ *as the transactions that contain the given itemset I, namely* $\text{tidset}(I) = \bigcap_{i \in I} \text{tidset}(i) = |\{t \in \mathscr{T} : \langle i, t\rangle \in \mathscr{D}, \forall i \in I\}|$. *Analogously,* $\text{itemset}\colon \mathscr{T} \to \wp(\mathscr{I})$ *identifies all items contained within a transaction, formally* $\text{itemset}(t) = \{i \in \mathscr{I} \mid \langle i, t\rangle \in \mathscr{D}\}$*, and the largest itemset shared between a given set of transaction is indicated with* $\text{itemset}(T) = \bigcap_{t \in T} \text{itemset}(t)$. *The composite function* $\text{closure}\colon \wp(\mathscr{I}) \to \wp(\mathscr{I}),\ I \mapsto \text{itemset}(\text{tidset}(I))$ *is called* closure operator. *An itemset $I \subseteq \mathscr{I}$ is said to be* closed *(CI for short) when* $\text{closure}(I) = I$ *or* frequent closed *(FCI) if it is also frequent. Finally, a* cluster *denotes a pair $\langle I, T\rangle$, where $T \subseteq \mathscr{T}$ is a tidset and $I \subseteq \mathscr{I}$ an itemset such that* $T = \text{tidset}(I)$.

FCIs can be thought as a compressed representation of FIs. In fact, it is easy to derive the set of FIs once that FCIs are known. However, the number of FCIs can be exponentially smaller than that of FIs [108] and, thus, there is a noticeable advantage in mining FCIs rather than FIs.

3.1.3 Case Example: Market Basket Analysis

To facilitate reading, a toy example is here provided to illustrate in practice what are frequent itemset and frequent closed itemsets. Table 3.1 shows a toy dataset of transactions that is typical of the market basket analysis. Each transaction is associated with a tid identifier and it consists in a list of different items. This kind of data can be conveniently represented using the bitmap data structure, as shown in Table 3.2 — and possibly after the application of a bitmap compression scheme, such as CutSize, for optimizing memory

Tid	Item1	Item2	Item3	Item4
T_1	Bread	Milk	-	-
T_2	Bread	Diaper	Beer	Eggs
T_3	Milk	Diaper	Beer	Coke
T_4	Bread	Milk	Diaper	Beer
T_5	Bread	Milk	Diaper	Coke

Table 3.1: A dataset of transactions.

Tid	Beer	Bread	Milk	Diaper	Eggs	Coke
T_1	0	1	1	0	0	0
T_2	1	1	0	1	1	0
T_3	1	0	1	1	0	1
T_4	1	1	1	1	0	0
T_5	0	1	1	1	0	1

Table 3.2: Bitmap based representation of the dataset of transactions.

resources. The definition of frequent itemset (Definition 3.2) is based on a user-defined threshold value $\delta \geq 0$. In the following, this value is fixed to $\delta = 2$ so that an itemset I (*i.e.* a set of items) is said to be frequent if there are at least $\delta = 2$ transactions that contain all the items belonging to I (*i.e.* support$(I) \geq 2$). The goal of frequent itemset mining is to discover all the FIs. Accordingly to the dataset and using $\delta = 2$, the solution consists in 17 FIs. All of them are grouped by cardinality and listed in Table 3.3. The number

Cardinality	Itemsets	Total
1	{beer},{bread},{milk},{diaper},{coke}	5
2	{beer,bread},{beer,milk},{beer,diaper},{bread,milk}, {bread,diaper},{milk,diaper},{milk,coke},{diaper,coke}	8
3	{beer,bread,diaper},{beer,milk,diaper},{bread,milk,diaper}, {milk,diaper,coke}	4

Table 3.3: List of frequent itemsets with $\delta = 2$.

of discovered FIs is relatively large compared to the size of the dataset and, in practical applications, this becomes a relevant issue due to the big size of datasets. Although the number of FIs can be reduced by increasing the threshold value δ, this is not a practicable solution since it costs the loss of much information contained in the set of discovered FIs. The best solution for addressing this problem is given by the frequent closed itemset representation. Reminding that a FCI is an FI that it not a subset of any other FI with the same

support, the list of FCIs (for $\delta = 2$) is reported in Table 3.4 grouping the itemsets by their cardinality. The solution consists in just 11 FCIs and there is no

Cardinality	Itemsets	Total
1	{bread},{milk},{diaper}	3
2	{beer,diaper},{bread,milk},{bread,diaper},{milk,diaper}	4
3	{beer,bread,diaper},{beer,milk,diaper},{bread,milk,diaper}, {milk,diaper,coke}	4

Table 3.4: List of frequent closed itemsets with $\delta = 2$.

loss of information. For instance, the FIs but not FCIs {beer} and {beer,bread} could be retrieved by computing the subsets of the FCI {beer,bread,diaper}. In addition, the supports of the two FIs (*i.e.* 3 *and* 2*, respectively*) can be also retrieved by computing the minimum supports of the FCIs that contain {bread} (*i.e.* the support of {beer,diaper}) and {beer,bread} (*i.e.* the support of {beer,bread,diaper}), respectively.

3.2 State-of-the-Art

Simultaneously with the development of algorithms for extracting FCIs, considerable effort on comparing the performances of these algorithms has been made and several surveys on this topic have been published [53, 1, 107, 47, 85, 18, 78]. Using the classification criteria adopted by [107], the main FCI miners in the current literature can be classified into three categories:

1. **"Test-and-generate"**: This category contains the algorithms based on the downward closure property that characterizes APRIORI [4]: an itemset is frequent only if all of its sub-itemsets are also frequent. Exploiting this property, FIs can be mined by first scanning the dataset to find FIs of cardinality one and then using the found FIs to generate other FIs of bigger cardinality. Other representative algorithms of this category are ECLAT [111], CLOSE [75] and TITANIC [88].

2. **"Divide-and-conquer"**: This category contains the algorithms based on methods similar to the FP-growth [48], that is the core engine of CLOSET [77]. The FP-growth method consists in to organize the original data into a compacted data structure, called Frequent Pattern tree, that is then used for splitting the instance of the problem into smaller ones. Other representative algorithms of this category are CLOSET+ [97], AFOPT [63] and FP-CLOSE [44].

3. **"Hybrid"**: This category contains the algorithms based on both the two previously mentioned strategies and the most representative of them is CHARM [110] which relies on a particular data structure, called Itemset-Tidset tree. This algorithm explores the search space in a depth-first manner, like algorithms of the divide-and-conquer category, but without splitting the main instance into several ones. Instead, it iteratively operates on the data structure using a test-and-generate approach to generate FCIs at each step. Other representative algorithm of this category are CLOSEMINER [86], PGMINER [70], DCI-CLOSED [65], LCM [91, 92, 93], DBV-MINER [94] and FCP-MINER [57].

The third proposed version of LCM [93] won the best implementation award of Frequent Itemset Mining Implementations workshop (FIMI) [41] and, nowadays, it is still the reference algorithm for efficiency of mining [73]. However, it must be pointed out that the performances of all these FCI mining algorithms are highly dependent on the nature of the data and, consequently, empirical comparisons do not reveal an unique best algorithm for all the datasets used for the evaluation. In particular, other experimental work [107, 85, 78] showed that DCI-CLOSED [65] outperforms LCM in several datasets confirming that DCI-CLOSED can be considered at least as good as the top performing FCI miner in the literature.

In the last decade, research efforts in the field of FCI mining have shifted towards different topics, such as *parallel computing*, *distributed system* and top-k FI mining. As far the parallel FCI mining, the state of the art is represented by MT-CLOSED [17] and PLCM [72] that are parallel implementations of two of the fastest algorithms according to the FIMI workshop, namely DCI-CLOSED and LCM. A different approach is also the one provided by PARAMINER [73] that is less performing but with the advantage of being applicable for more generic data mining tasks. The research line addressing implementation on distributed systems, is more recent and less investigated but, even in this field, one main goal is to achieve distributed versions of the algorithms that competed in the FIMI workshop. For instance, DIST-ECLAT and BIGFIM [69] are implementations based on ECLAT and APRIORI that are both deployed for the *MapReduce* [34] platform. Finally, as far the problem of mining only the k most representative FI [49, 39], efficient solutions have been recently proposed [36, 54].

3.3 Frequent Closed Itemset Mining with EXPEDITE

This section presents EXPEDITE, an innovative and very time efficient FCI miner algorithm. The content is divided as follows. Section 3.3.1 provides a description of the algorithm; Section 3.3.2 proves its correctness; then, Section 3.3.3 compares EXPEDITE with DCI-CLOSED; and, finally, Section 3.3.4.

3.3.1 Description of the Algorithm

The algorithm EXPEDITE consists in a recursive procedure, precisely named EXPEDITE, whose pseudo-code is reported in Algorithm 3.1. Its operating principles can be organized into four main steps:

1. *Initial Data Preparation (Line 1)*. Four input parameters must be provided to call the procedureEXPEDITE():

   δ: A threshold value $\delta \geq 0$ that specifies the minimum accepted support. Itemsets with support lesser than δ are ignored by the algorithm and their closure is not computed.

   $\mathscr{P}$: A *prefix* $\mathscr{P}$ that is a set of *clusters*. In turn, a cluster is a pair $\langle I, T \rangle$ consisting of an itemset I with its corresponding tidset $T = \text{tidset}(I)$. When explicitly declared, the clusters of a prefix are sorted according to the lexicographical order denoted by the $\prec$ symbol. Precisely, the lexicographical order on sets is defined as $A = \{a_1, \ldots a_n\} \prec B = \{b_1, \ldots, b_m\}$ if and only if $n < m$ or $\exists i$ such that $a_j = b_j \ \forall j < i$ and $a_i < b_i$ while the lexicographical order on clusters as $\langle I, T \rangle \prec \langle I', T' \rangle$ if and only if $T \prec T'$ or $T = T' \wedge I \prec I'$. In particular, $(\mathscr{P}, \prec)$ is a totally ordered set. The first time we call the procedure EXPEDITE, $\mathscr{P}$ is the set of clusters such that their itemsets are frequent 1-itemsets. Formally, $\mathscr{P} = \mathscr{P}_{\text{init}} = \big\{\langle I, T \rangle \in \wp(\mathscr{I}) \times \wp(\mathscr{T}) : |I| = 1 \wedge T = \text{tidset}(I) \wedge |T| \geq \delta\big\}$.

   $\mathscr{L}$: A stack of pending prefixes $\mathscr{L}$ which is helpful for keeping track of the recursion path. The first time we call the procedure EXPEDITE, the stack $\mathscr{L}$ is empty.

   $\mathscr{S}$: A map $\mathscr{S} : T \mapsto \mathscr{S}[T]$ containing the supersets of each tidsets belonging to a cluster of $\mathscr{P}$ discovered during the algorithm's execution. We refer to these supersets as super-tidsets. For each cluster $\langle I, T \rangle \in \mathscr{P}$ thus, $\mathscr{S}[T]$ stores the known tidsets existing in $\mathscr{P}$ that are proper supersets of T. Formally, $\mathscr{S}[T] \subseteq \{T' \subseteq \mathscr{T} : \exists \langle I', T' \rangle \in$

$\mathscr{P}$, $T \subsetneq T' \wedge I' \subseteq \mathscr{I}\}$. This map is helpful to avoid several computations of the same closed itemsets. The first time we call the procedure EXPEDITE, $\mathscr{S}[T]$ is empty for all the tidsets T found in $\mathscr{P}_{\text{init}}$.

2. *Itemset Closure (Lines 2–21)*. For each cluster in $\mathscr{P}$, EXPEDITE performs itemset closure by replacing every cluster $\langle I, T\rangle \in \mathscr{P}$ with the cluster $\langle \text{closure}(I), T\rangle$. While doing so, it updates both the map of known supertidsets $\mathscr{S}$ and the map of clusters $\mathscr{C} : T \mapsto \mathscr{C}[T]$. Both these maps are useful to avoid redundant computations during the new prefixes generation in Step 3.

   In the following, further details on the closure step are provided by commenting the related Lines 2–21. There are two nested cycles iterating on the clusters $\langle I, T\rangle$ of $\mathscr{P}$ and both use the lexicographical order on $\mathscr{P}$. The outer one (Line 2) follows a descending order (cluster with lower $|T|$ are chosen first), while the inner one (Line 4) an ascending order (clusters with lower $|T|$ are chosen last). Each time a pair of clusters is found, say $\langle I, T\rangle$ and $\langle I', T'\rangle$, with the property that T' is a potential undiscovered superset of T (Line 5), the following operations are performed to ensure the itemset closure. First, the sub-procedure PARTIALINTERSECTIONT, T' is called to save the output in the variable Σ (Line 6). This PARTIALINTERSECTION sub-procedure checks, one by one, if elements in T are also in T'. In positive cases, the elements are added to Σ but at the first negative case the sub-procedure stops (the code is omitted). If T has not been scanned till the end in this way, then T' is not a superset of T and the procedure can skip to the next element because it means that T and T' belong to two different closed itemsets. However, the map $\mathscr{C}$ is update (Line 17) since $T \cap T'$ could generate new CIs. If T has been scanned until the end, instead, the following operations are made (Lines 7–16). If $T = T'$, then it means that itemsets I and I' have the same closure and only one of them needs to be processed. Thus, EXPEDITE deletes the cluster $\langle I, T\rangle$, merges items of I and I' into I' and goes back to the next iteration of the outer cycle (note that the cluster $\langle I', T'\rangle$ must follow the cluster $\langle I, T\rangle$ in the lexicographical order, otherwise $\langle I, T\rangle$ would have been deleted before). Instead, if $T \neq T'$ EXPEDITE simply merges items of I and I' into I and updates the map $\mathscr{S}$ since $T \subset T'$. It can be directly verified that due to the merging operations, at the end of both cycles, the prefix $\mathscr{P}$ will contain only distinct clusters whose itemsets are closed.

3. *Generation of New Prefixes (Lines 23–34)*. Step 2 produces a set of dis-

tinct closed itemsets but, generally, there are others to be discovered. To continue the extraction of CIs, the algorithm generates new prefixes which are provided as input to recursive calls of EXPEDITE. Doing this in the right way, all the FCIs can be generated (see Section 3.3.2 for a proof of correctness). The generation of new prefixes is based on the map $\mathscr{C}$ that contains clusters with the potential of generating new FCIs.

Referring to the code, the generation step happens in Lines 23–34. Using the ascending lexicographical order, EXPEDITE first selects and pushes an existing cluster $\langle I, T\rangle \in \mathscr{P}$ into the stack $\mathscr{L}$, meanwhile initializing a new prefix $\mathscr{P}_{\langle I,T\rangle}$ with its map $\mathscr{S}_{\langle I,T\rangle}$ (Lines 23–25). Then, a cycle on clusters $\langle I', T'\rangle$ of $C[T]$ using the ascending order follows (Lines 26–34), to perform the following operations. The sub-procedure COMPLETEINTERSECTION(T, T', Σ) is called to save into Λ the output intersection $T \cap T'$ (Line 27). Note that Σ is used to speed-up the computation by recovering the final state of PARTIALINTERSECTION(T, T'). A check is made to verify if Λ is a potential support of a new closed itemset (Line 28), which happens if the cardinality of Λ is above the threshold value δ and if Λ is not contained into a tidset of another already generated prefix. This last control is performed through the procedure ISDUPLICATE$\Lambda, \mathscr{L}, \mathscr{H}$ whose pseudo-code is reported in Algorithm 3.2. Then, if Λ is a potential support for a new CI, the cluster $\langle I \cup I', \Lambda\rangle$ is added to the new prefix $\mathscr{P}_{\langle I,T\rangle}$ updating the map $\mathscr{S}_{\langle I,T\rangle}$ with all clusters containing a tidset superset of T'; otherwise, the cluster $\langle I', T'\rangle$ and all clusters whose tidset is contained in T' are discarded.

4. *Cluster Saving, Memory Deallocation, and Recursion (Lines 22–38)*. Once a cluster $\langle I, T\rangle \in \mathscr{P}$ has generated its own prefix $\mathscr{P}_{\langle I,T\rangle}$, it can be removed from the memory space and saved into the device used to store the wanted FCIs (Line 35). In fact, keeping track of it in memory is not actually needed to carry on the execution of the algorithm and Step 2 ensures that it is already a FCI. It is worth noticing that, while storing clusters, EXPEDITE keeps in memory also their tidsets. After the write operation, previous steps are applied through recursion (Line 36) on the new generated prefix $\mathscr{P}_{\langle I,T\rangle}$. Finally, $\langle I, T\rangle$ is popped out from the stack $\mathscr{L}$ (Line 37). The algorithm ends after that it generates and recursively analyses the last generated prefix of the last cluster of $\mathscr{P}_{\text{init}}$.

Algorithm 3.1 Main procedure of EXPEDITE

```
1: procedure EXPEDITE(𝒫, ℒ, 𝒮, δ)
2:     for each ⟨I, T⟩ ∈ (𝒫, ≺), descending do
3:         𝒞[T] ← ∅
4:         for each ⟨I′, T′⟩ ∈ (𝒫, ≺) : ⟨I, T⟩ ≺ ⟨I′, T′⟩, ascending do
5:             if T′ ∉ 𝒮[T] then
6:                 Σ ← PARTIALINTERSECTION(T, T′)
7:                 if Σ is s.t. T has been scanned till the end then
8:                     if |T| = |T′| then
9:                         𝒫 ← 𝒫 \ {⟨I, T⟩}
10:                        I′ ← I′ ∪ I
11:                        break loop
12:                    else
13:                        I ← I ∪ I′
14:                        𝒮[T] ← 𝒮[T] ∪ {T′} ∪ 𝒮[T′]
15:                    end if
16:                else
17:                    𝒞[T] ← 𝒞[T] ∪ {⟨I′, T′, Σ⟩}
18:                end if
19:            end if
20:        end for
21:    end for
22:    for each ⟨I, T⟩ ∈ (𝒫, ≺), ascending do
23:        𝒫_⟨I,T⟩ ← ∅
24:        𝒮_⟨I,T⟩ ← ∅
25:        ℒ ← ℒ ∪ ⟨𝒫, I, T⟩
26:        for each ⟨I′, T′, Σ⟩ ∈ 𝒞[T], ascending on (𝒞[T], ≺) do
27:            Λ ← COMPLETEINTERSECTION(T, T′, Σ)
28:            if |Λ| ≥ δ ∧ ¬ ISDUPLICATE(Λ, ℒ, ℋ) then
29:                𝒫_⟨I,T⟩ ← 𝒫_⟨I,T⟩ ∪ {⟨I ∪ I′, Λ⟩}
30:                𝒮_⟨I,T⟩[Λ] ← {⟨Î, T̂⟩ ∈ 𝒫_⟨I,T⟩ | ∃T″ ∈ 𝒮[T′], T̂ = T ∩ T″}
31:            else
32:                𝒞[T] ← {⟨Î, T̂⟩ ∈ 𝒞[T] | T′ ∉ 𝒮[T̂]}
33:            end if
34:        end for
35:        WRITE(I, T)
36:        EXPEDITE𝒫_⟨I,T⟩, ℒ, 𝒮_⟨I,T⟩, δ
37:        ℒ ← ℒ \ ⟨𝒫, I, T⟩
38:    end for
39: end procedure
```

Algorithm 3.2 Procedure IsDuplicate of Expedite

```
1: procedure IsDuplicate(Λ, 𝓛)
2:     for each ⟨𝒫, I, T⟩ ∈ 𝓛 do
3:         for each ⟨I′, T′⟩ ∈ (𝒫, ≺) : ⟨I′, T′⟩ ≺ ⟨I, T⟩, ascending do
4:             if Λ ⊆ T′ then
5:                 return true
6:             end if
7:         end for
8:     end for
9:     return false
10: end procedure
```

3.3.2 Proof of Correctness

The correctness of Expedite can be proved by showing that the following two claims are true:

1. If the output of IsDuplicate is *always* false, then Expedite writes all and only the frequent closed itemsets;

2. The output of IsDuplicate is true only if the current cluster cannot generate *new* frequent closed itemsets.

In fact, by combining these two claims can be deduced that Expedite writes each frequent closed itemset at least once. The proof starts with the following lemma.

Lemma 3.1. *The computation of all the intersections between two or more tidsets of the clusters belonging to the initial prefix $\mathscr{P}_{\text{init}}$ provides the tidset of each frequent closed itemset.*

Proof. First, can be we notice that $\mathscr{P}_{\text{init}}$ contains by definition all and only clusters $\langle I_i, T_i \rangle$ whose itemsets I_i are frequent 1-itemsets. Then, assume that $I = \{i_1, i_2\}$ is any frequent 2-itemset (not necessarily closed). Both items in I must have support greater or equal than δ (otherwise it would not be a FI) and, consequently, both its items must be 1-itemsets belonging to two clusters of $\mathscr{P}_{\text{init}}$ that are denoted with $\langle \{i_1\}, T_1 \rangle, \langle \{i_1\}, T_2 \rangle$. Then, the tidset of the 2-frequent itemset $I = \{i_1, i_2\}$ can be obtained by intersecting their tidsets, namely $T_1 \cap T_2 = \text{tidset}(I)$. Thus, for each frequent 2-itemset it is possible to generate its tidset by computing pairwise intersections of the tidsets of clusters belonging to $\mathscr{P}_{\text{init}}$. Generalizing to more than two items, it is possible to generate the tidset of all the frequent k-itemsets, for each $k \geq 1$, by considering the intersections of k tidsets of clusters belonging to $\mathscr{P}_{\text{init}}$. To conclude,

is sufficient to note that FCIs are a subset of FIs and, thus, the tidset of each frequent closed itemset is obtained. □

Lemma 3.1 states that for each FCIs $\bar{I}$, through the computation of these intersections, it is possible to obtain a cluster $\langle I, T\rangle$ such that tidset$(\bar{I})$ = tidset$(I) = T$. In particular, by computing also the closure operator of I, namely closure$(I) = \bar{I}$, a procedure for computing all the FCIs is obtained. Once proved that EXPEDITE correctly perform this, its correctness will be achieved.

Theorem 3.1. *Assume that the output of* ISDUPLICATE *is always false, then* EXPEDITE *writes all and only the frequent closed itemsets.*

Proof. EXPEDITE *writes only FCIs.* The writing operation of any cluster $\langle I, T\rangle$ occurs at Line 35 after the closure step at Lines 2–21. This means that EXPEDITE writes only CIs. Then, the initial prefix contains only frequent itemsets, the closure step does not decrease their cardinality and new generated prefixes contain only frequent itemsets due to the control at Line 28. Thus, the CIs wrote by EXPEDITE must be also frequent.

EXPEDITE *writes all the FCIs.* It is enough to show that EXPEDITE computes all the intersections of tidsets contained in clusters of $\mathscr{P}_{\text{init}}$ thanks to Lemma 3.1. To each prefix can be associated a level number that counts the nested recursive calls of EXPEDITE: $\mathscr{P}_{\text{init}}$ has level zero, for each $\langle I, T\rangle \in \mathscr{P}_{\text{init}}$ $\mathscr{P}_{\langle I,T\rangle}$ has level one and so forth. At the first call EXPEDITE$(\mathscr{P}_{\text{init}}, \ldots)$, the cycle at Line 22 generates a new prefix $\mathscr{P}_{\langle I,T\rangle}$ of level one for each cluster $\langle I, T\rangle \in \mathscr{P}_{\text{init}}$. Then, for each cluster $\langle I', T'\rangle \in \mathscr{P}_{\text{init}}$ such that $\langle I, T\rangle \prec \langle I', T'\rangle$, the intersection $\Lambda = T \cap T'$ between their two tidsets is computed. This means that all pairwise intersections between tidsets of clusters in $\mathscr{P}_{\text{init}}$ are computed during the first level of recursion. Since ISDUPLICATE is always false by hypothesis, each new generated cluster (*e.g.* $\langle I \cup I', \Lambda\rangle$) is added to the prefix of level one (respectively, $\mathscr{P}_{\langle I,T\rangle}$) if and only if its itemset is frequent ($\Lambda \geq \delta$). For each of the new prefixes, a recursive call (Line 36) is made and the cycle at Line 22 generates again new prefixes of level two made by only clusters obtained as pairwise intersections (such as $\langle I \cup I', \Lambda\rangle$). Better said, at level two, EXPEDITE generates all the pairwise intersections between pairwise intersections of tidsets in $\mathscr{P}_{\text{init}}$. Consequently, this means that, at level two, EXPEDITE generates all the intersections of three or four tidsets of $\mathscr{P}_{\text{init}}$. In general at level k of recursion EXPEDITE generates all the intersections of up to 2^k tidsets of $\mathscr{P}_{\text{init}}$. Eventually, all the intersections of tidsets contained in clusters of $\mathscr{P}_{\text{init}}$ are computed, concluding the proof. □

Theorem 3.1 provides the correctness of EXPEDITE when there is no use of the sub-procedure ISDUPLICATE for checking duplicate tidsets. This subprocedure significantly improves the time efficiency by reducing the number

of intersections to perform but its correctness must be also proved. This is done in the following final theorem.

Theorem 3.2. *The output of* IsDuplicate *is true only if the current cluster cannot generate a frequent closed itemset that would not be generated otherwise.*

Proof. The goal is to show that if Expedite does not insert a cluster into a prefix, which happens when IsDuplicate$\Lambda, \mathscr{L}$ is true, then Expedite is still generating all the frequent closed itemsets according to Theorem 3.1. This statement is proved by induction. The assumption is that the prefixes $\mathscr{P}_{\text{init}} = \mathscr{P}_0, \mathscr{P}_1, \ldots, \mathscr{P}_{t-1}$ have been already generated by Expedite and that IsDuplicate did not compromise the correctness for these prefixes. The goal is to prove that the procedure IsDuplicate do not compromise the correctness of the algorithm during the generation of the subsequent prefix $\mathscr{P}_t$. The base case of the induction $\mathscr{P}_{\text{init}} = \mathscr{P}_0$ is trivial since IsDuplicate is not applied during the generation of $\mathscr{P}_{\text{init}}$. Hence, the induction hypothesis is that $\mathscr{P}_t = \mathscr{P}_{\langle I,T\rangle}$ is generated by a cluster $\langle I, T\rangle \in \mathscr{P}_{t'<t}$ and that, for some different cluster $\langle I', T'\rangle \in \mathscr{P}_{t'}$, IsDuplicate$T \cap T', \mathscr{L}$ is true. This means that $T \cap T' \subseteq T^*$, where T^* belongs to a cluster $\langle I^*, T^*\rangle \in \mathscr{P}_{t*\leq t'}$. All the FCIs that would derive from $\mathscr{P}_{\langle J, T\cap T'\rangle}$, for any itemset J, have been already generated from $\mathscr{P}_{\langle I^*,T^*\rangle}$ whose index is strictly lesser than t. In fact, the inductive hypothesis guarantees that $\mathscr{P}_{\langle I^*,T^*\rangle}$ correctly generates FCIs even using IsDuplicate, and $T \cap T' \subseteq T^*$ ensures that all the FCIs deriving from $T \cap T'$ are also derivable from T^*. This complete the proof since IsDuplicate preserves the correctness for the prefix $\mathscr{P}_t$. □

3.3.3 Expedite vs. DCI-Closed

The presented algorithm Expedite is a FCI miner that has not yet been optimized neither for parallel computations nor for distributed systems. For this reason, in the following the algorithm is compared with DCI-Closed discussing in detail their differences. In fact, beside being one of the top performing algorithms available in the literature, DCI-Closed is also the FCI mining algorithm more similar to Expedite. Other details are in Section 3.3.4 that shows a case example for illustrating both Expedite and DCI-Closed and in Section 3.5) that includes their experimental comparison.

The pseudo-code of DCI-Closed is reported in Algorithm 3.3. Its working principles are similar to those of Expedite. In fact, the adopted strategy is based on performing all the intersections among frequent closed 1-itemsets according to Lemma 3.1. To easily compare the codes, also DCI-Closed description is separated into the same four steps used to describe Expedite:

1. *Initial Data Preparation (Line 1)*. Five parameters must be provided to call the procedure DCI-CLOSED:

   δ: A threshold value δ that specifies the minimum accepted support, as in EXPEDITE.

   $\mathscr{P}$: A prefix $\mathscr{P}$ whose clusters, differently from EXPEDITE, are pairs of the form $\langle i, \Sigma \rangle$ where i is an item and Σ is the inner state of the sub-procedure PARTIALINTERSECTION$(T, \mathscr{L})$. The first time that the procedure DCI-CLOSED is called, $\mathscr{P}$ is initialized as $\mathscr{P} = \mathscr{P}_{\text{init}} = \left\{ \langle i, \emptyset \rangle \;\middle|\; i \in \wp(\mathscr{I}) \right\}$.

   $\mathscr{L}$: A stack of pending prefixes $\mathscr{L}$ with functionalities analogous to that of EXPEDITE.

   I: An itemset I that is a parameter needed for the recursion. The first time that the procedure DCI-CLOSED is called, I is an empty set.

   T: A set of transaction T that is a parameter needed for the recursion. The first that the procedure DCI-CLOSED is called, T is the set containing all the transactions.

2. *Itemset Closure (Line 10)*. Closure of itemsets is performed through the cycle at Lines 7–14 that is similar to that of EXPEDITE (the cycle at Line 4 of Algorithm 3.1). The main difference is that DCI-CLOSED computes the closure by adding single items rather entire itemsets: this is a consequence of using clusters made by single items. Another difference is that EXPEDITE updates the maps $\mathscr{S}$ and $\mathscr{C}$ to avoid redundant computations. Indeed, differently from DCI-CLOSED, EXPEDITE generates new prefixes into a different cycle and, thus, it needs to store these map to resume the computations of the former cycle.

3. *Generation of New Prefixes (Line 12)*. The generation of new prefixes is also very similar to that of EXPEDITE although the order of operations is a bit different. The main differences are in the sub-procedure ISDUPLICATE. In particular, the version used by DCI-CLOSED is more time-efficient since it only checks whether or not the questioned tidset is contained into the tidset of any frequent closed 1-itemset (which can be pre-calculated).

4. *Cluster Saving, Memory Deallocation, and Recursion (Lines 15,16)*. Exactly like EXPEDITE, DCI-CLOSED makes use of recursive calls (Line 16) and allows to store apart the extracted frequent closed itemsets (Line 15). However, what is really different is how DCI-CLOSED makes use of the recursion: DCI-CLOSED recursively computes all the possible intersections among the tidset of frequent closed 1-itemsets. It must do so because its

prefixes contain single items. Instead EXPEDITE, whose prefixes contain entire itemsets, computes the intersections among the tidsets of these itemsets. The result is that requires to compute a smaller number of intersections to be in range of Lemma 3.1.

Summing up, the fundamental difference between EXPEDITE and DCI-CLOSED is that the clusters belonging to the prefixes of EXPEDITE contain entire itemsets, while those of DCI-CLOSED only single items. Thus, EXPEDITE needs to compute a smaller number of intersections than DCI-CLOSED, consequently reducing the number of duplicate and infrequent itemsets extracted.

3.3.4 Case Example

In the following, a case example is described in detail to show the operations performed by EXPEDITE and DCI-CLOSED. The toy dataset used for this example is depicted in Figure 3.1d (crosses represent the binary relations between items and transactions). The value of threshold is set to zero for both algorithms.

EXPEDITE. Figure 3.1a depicts the operations performed by EXPEDITE. The first row of the topmost rectangle is the initial prefix $\mathscr{P}_{\text{init}}$. Each of its cluster $\langle I_i, T_i \rangle$ has the itemset I_i, which contains only the single i-th item, and the transactions set T_i equals to tidset(I_i). In practice, each column of the dataset is converted into a cluster. Below, always in the first rectangle, is shown $\mathscr{P}_{\text{init}}$ after the closure step: it contains five FCIs. Note that the clusters $\langle \{C\}, \{a, e, f\} \rangle$ and $\langle \{D\}, \{a, e, f\} \rangle$ share the same tidset and are thus merged together, according to the control at Line 8. Further, the cluster $\langle \{F\}, \{b, g, h\} \rangle$ is expanded with the item E due to the closure operation made according to Line 13. The tidsets of the remaining five clusters are then intersected to generate new prefixes recursively (the arrows in the figure represent the intersections computed by the algorithm). To provide all the details, the values of maps $\mathscr{C}$ and $\mathscr{S}$ are computed, in order, and up to this step (some punctuations and the inner state Σ of PARTIALINTERSECTION are omitted for higher readability). The results are reported in Table 3.5.

The first new generated prefix is $\mathscr{P}_{\langle \{A\}, \{a,b,c\} \rangle}$ and it is shown in the rectangle number two (rectangles are enumerated to display the recursion path). Since $\mathscr{C}[abc]$ has four elements and none of them is a duplicate, namely IS-DUPLICATE is false, $\mathscr{P}_{\langle \{A\}, \{a,b,c\} \rangle}$ has four clusters obtained through intersections of the tidset $\{a, b, c\}$ with those of the clusters in $\mathscr{P}_{\text{init}}$. The prefix $\mathscr{P}_{\langle \{B\}, \{a,b,d\} \rangle}$, instead, could have three clusters ($\mathscr{C}[abd]$ has three elements) but all the intersections fail the check for duplicates. In fact, $abd \cap bcgh = b$ is a tidset already generated by $\mathscr{P}_{\langle \{A\}, \{a,b,c\} \rangle}$, $abd \cap bgh$ is not computed since it is use-

Algorithm 3.3 The algorithm DCI-CLOSED

```
1:  procedure DCI-CLOSED(𝒫, ℒ, I, T, δ)
2:      for each ⟨i, Σ⟩ ∈ 𝒫, sorted by ascending i do
3:          Λ ← COMPLETEINTERSECTION(T, tidset(i), Σ)
4:          if |Λ| ≥ δ ∧ ¬ ISDUPLICATE(Λ, ℒ) then
5:              I_Λ ← I ∪ {i}
6:              𝒫_Λ ← ∅
7:              for each ⟨i′, Σ′⟩ ∈ 𝒫 : i ≺ i′, sorted by ascending i′ do
8:                  Σ ← PARTIALINTERSECTION(Λ, tidset(i′))
9:                  if Σ is s.t. Λ has been scanned till the end then
10:                     I_Λ ← I_Λ ∪ {i′}
11:                 else
12:                     𝒫_Λ ← 𝒫_Λ ∪ {⟨i′, Σ⟩}
13:                 end if
14:             end for
15:             WRITE(I_Λ, Λ)
16:             DCI-CLOSED(𝒫_Λ, ℒ, I_Λ, Λ, δ)
17:             ℒ ← ℒ ∪ {i}
18:         end if
19:     end for
20: end procedure

21: procedure ISDUPLICATE(Λ, ℒ)
22:     for each i ∈ ℒ do
23:         if Λ ⊆ tidset(i) then
24:             return true
25:         end if
26:     end for
27:     return false
28: end procedure
```

Map entry	Value
$\mathscr{C}[bcgh]$	$\emptyset$
$\mathscr{S}[bcgh]$	$\emptyset$
$\mathscr{C}[bgh]$	$\emptyset$
$\mathscr{S}[bgh]$	$\{bcgh\}$
$\mathscr{C}[aef]$	$\{\langle FE, bgh\rangle, \langle E, bcgh\rangle\}$
$\mathscr{S}[aef]$	$\emptyset$
$\mathscr{C}[abd]$	$\{\langle CD, aef\rangle, \langle FE, bgh\rangle, \langle E, bcgh\rangle\}$
$\mathscr{S}[abd]$	$\emptyset$
$\mathscr{C}[abc]$	$\{\langle B, abd\rangle, \langle CD, aef\rangle, \langle FE, bgh\rangle, \langle E, bcgh\rangle\}$
$\mathscr{S}[abc]$	$\emptyset$

Table 3.5: Values of maps $\mathscr{C}$ and $\mathscr{S}$ after the first round.

less to intersect abd with a subset of $bcgh$ (here the map $\mathscr{S}$ helps to avoid computations) and $abd \cap aef = a$ is also already generated by $\mathscr{P}_{\langle\{A\},\{a,b,c\}\rangle}$. Thus, $\mathscr{P}_{\langle\{B\},\{a,b,d\}\rangle} = \emptyset$. Then, $\mathscr{P}_{\langle\{CD\},\{a,e,f\}\rangle}$ is empty since it could have two new clusters, but $aef \cap bcgh = \emptyset$ and so are the remaining two prefixes since the corresponding entries of the map $\mathscr{C}$ are also empty. Hence, $\langle\{A\},\{a,b,c\}\rangle$ is the only cluster in $\mathscr{P}_{\text{init}}$ that generates not empty prefixes. The four obtained new FCIs are shown at the bottom of the rectangle number two of the picture, after the closure step. The computations of the maps $\mathscr{C}$ and $\mathscr{S}$ are reported in Table 3.6 Finally, it can be verified that the FCIs in the rectangle number two

Map entry	Value
$\mathscr{C}[bc]$	$\emptyset$
$\mathscr{S}[bc]$	$\emptyset$
$\mathscr{C}[ab]$	$\{\langle AE, bc\rangle\}$
$\mathscr{S}[bc]$	$\emptyset$
$\mathscr{C}[b]$	$\emptyset$
$\mathscr{S}[b]$	$\{ab, bc\}$
$\mathscr{C}[a]$	$\{\langle AFE, b\rangle, \langle AE, bc\rangle\}$
$\mathscr{S}[a]$	$\{ab\}$

Table 3.6: Values of maps $\mathscr{C}$ and $\mathscr{S}$ after the second round.

generate only empty prefixes and, consequently, the algorithm ends.

DCI-Closed. Figure 3.1b depicts the operations performed by DCI-Closed. The first row contains the clusters in $\mathscr{P}_{\text{init}}$ (the grey ones are FCIs), the second represents those generated at the first level of the recursion. Similarly, the third and the fourth lines represent those generated at the second and the

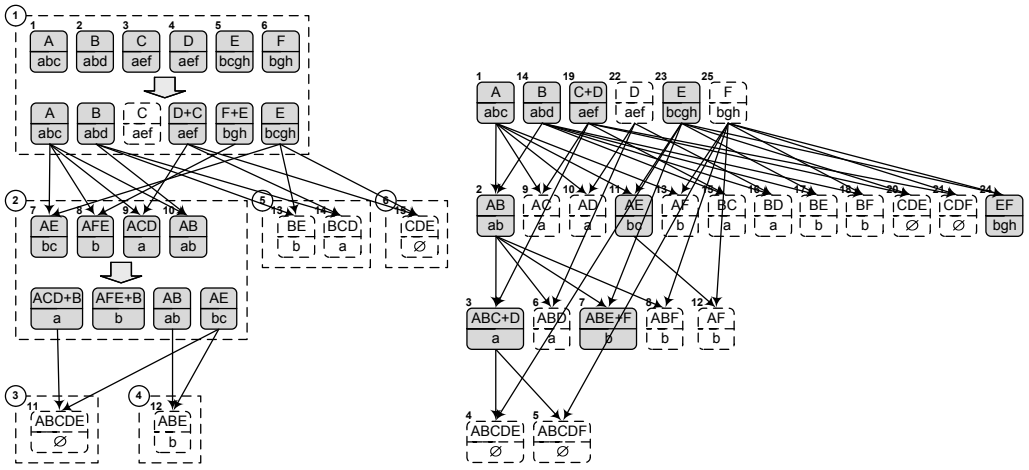
third level of the recursion. In detail, the algorithm starts to analyse the first element A of the initial prefix $\mathscr{P}_{\text{init}} = \{A, B, C, D, E, F\}$ (we omit the inner state Σ and some punctuations for more readability). It computes the closure of A obtaining the FCI $\langle I_\Lambda, \Lambda\rangle = \langle A, abc\rangle$. Then, a recursive call is made on the prefix $\mathscr{P}_\Lambda = P_{abc} = \{B, C, D, E, F\}$. Again, the algorithm analyses $B \in \mathscr{P}_{abc}$, computes the FCI $\langle I_\Lambda, \Lambda\rangle = \langle AB, ab\rangle$ and goes to the second level of recursion whose prefix is $P_{ab} = \{C, D, E, F\}$. In this level, DCI-CLOSED computes the intersection $ab \cap tidset(C) = a$ and the closure of a finding the FCI $\langle ABCD, a\rangle$ and, then, it goes to the next level of recursion whose prefix is $\mathscr{P}_a = \{E, F\}$. However, $\mathscr{P}_a$ does not generate new FCIs since $a \cap tidset(E) = a \cap tidset(F) = \emptyset$ and, thus, the deeper recursive call ends going back of one level. In the figure, is displayed the order of itemsets generation and all the performed intersections (arrows represent intersections). In particular, comparing Figure 3.1b and Figure 3.1a, it is possible to notice that EXPEDITE requires less intersections than DCI-CLOSED to complete the process of FCIs extraction.

3.4 Improving the Performances of EXPEDITE

In this section, two variants of EXPEDITE are proposed with the goal of further improving both the time efficiency and the memory footprint. The first variant uses hashcodes for checking the duplicate itemsets. When it is known that the data to process contain many FCIs of large size, this variant highly reduces the running time of EXPEDITE. In fact, the issue with sub-procedure ISDUPLICATE described in Algorithm 3.1 is that is becomes very computational expensive when the sizes of the FCIs increase; instead, just checking if the hashcode of a FCI was already generated is way faster. However, when the FCIs within the data are mostly of small sizes, duplicates are uncommon and to generate hashcodes is a waste of computational resources. The second variant of EXPEDITE here proposed adopts the diffset representation [109] to improve the memory footprint. This representation performs well when there is high density of binary relations between items and transactions in the whole dataset The first variant is described in Section 3.4.1 while the second in Section 3.4.2.

3.4.1 Using Hashcodes for Checking Duplicates

The sub-procedure ISDUPLICATE has a computational cost that, in the worst case, is cubic in the size of the supports of FCIs. This means that the larger the size of the FCIs within the data, the more the sub-procedure is inefficient. To overcome this issue, it is possible to generate and store hashcodes of any found FCI. This allows to check in constant time if a FCI has been already generated,



(a) EXPEDITE (b) DCI-CLOSE

Symbol	*Meaning*
	Generated cluster. Capital letters before the "+" sign (e.g., "F") are items inherited from parent clusters; capital letters after the "+" sign (e.g., "E") are items added after closure operation; small letters (e.g., "bgh") are transactions; and, the number (e.g., "5") indicates the generation order.
	Discarded cluster due to low support or duplication.
	Indicates the pair of clusters which generate the new one
	Prefix and generation order (EXPEDITE only).

(c) Legend

transactions	*items* A	B	C	D	E	F
a	×	×	×	×		
b	×	×			×	×
c	×				×	
d		×				
e			×	×		
f			×	×		
g					×	×
h					×	×

(d) Dataset

Figure 3.1: Case example of EXPEDITE and DCI-CLOSED. Fig. 3.1a and Fig. 3.1b illustrate the operation performed by the algorithms. Fig. 3.1c is the legend of symbols and Fig. 3.1d is the dataset used.

removing the dependence on the size of the FCIs in the computational cost of the procedure.

The pseudo-code of the hash-based implementation of IsDuplicate is reported in Algorithm 3.4. It differs from the original only by Line 2. As mentioned, the idea is to check if the hash of the tidset Λ has been already generated and stored into the map $\mathscr{H}$; if not, then Λ cannot be a duplicate and the sub-procedure ends without scanning any tidset. The map $\mathscr{H}$, in turn, can be implemented in two ways. The first one consists in mapping the known hashcodes of the tidsets, namely $\mathscr{H} : h(T) \mapsto \{0,1\}$ where h is an arbitrarily chosen hash function. Initially, all the values are set to zero and each time that Expedite generates a new tidset T, the hash code $h(T)$ is computed and $\mathscr{H}(h(T))$ is set to one. The negative aspect of this method is that it is possible to lose some FCIs due to collisions of the hash function. Hence, requiring Lines 5–10. Alternatively, it is possible to use hashtables to track also tidsets with the same digest. Formally, using a map $\mathscr{H} : h(T) \mapsto \wp(\mathscr{T})$. In this case, Lines 5–10 would not be necessary.

It is important to choose a very fast hash function to improve the time-efficiency of IsDuplicate. One possible option to do this is to compute digests of tidsets [55], as described in the following. First, each transaction $t_i \in \mathscr{T}$ is assigned to a unique non negative integer identifier $x_i \in \mathbb{N}$. For instance, to the transactions $a, b, c, \ldots$ of the example in Section 3.3.4 can be assigned the numbers $1, 2, 3, \ldots$. Then, for each tidset $T = \{t_1 \ldots t_n : t_i \in \mathscr{T}\}$, the hash value is defined as $h_p(T) = \sum_{i=1}^{n} p^{n-i} * x_i \ mod \ 2^{32}$ where p is any fixed prime and 32 represents the processor's word length in bits. This is the hash function adopted for our experiments later in Section 3.5. It is time efficient and works well for the intended purposes; however, different functions could be implemented as well.

3.4.2 Diffset Representation

The diffset representation [109] of FCIs can drastically cut down the memory footprint of FCIs miners by exploiting repetitions of transactions belonging to itemsets of the same generation pattern. This representation is based on the equivalence between storing a tidset and tracking, instead, differences in the tidsets through its generation pattern.

To show in detail how this representation works, it is possible to consider a cluster $\langle I', T' \rangle$ of a given prefix $\mathscr{P}_{\langle I,T \rangle}$. It shares common transactions with the cluster $\langle I, T \rangle$ since the tidset T' is obtained through the intersection of T with another tidset. Thus, if storing T and the commons items between T and T', namely $T \cap T'$, it is possible to recover T'. In particular, only $|T| + |T \cap T'|$ (instead of $|T| + |T'|$) memory cells are required to store the two tidsets. In

Algorithm 3.4 Hash-based procedure IsDuplicate

```
1: procedure IsDiplicate(Λ, 𝓛)
2:     if HashCode(Λ) ∉ 𝓗 then
3:         return false
4:     end if
5:     for each ⟨𝒫, I, T⟩ ∈ 𝓛 do
6:         for each ⟨I′, T′⟩ ∈ (𝒫, ≺) : ⟨I′, T′⟩ ≺ ⟨I, T⟩, ascending do
7:             if Λ ⊆ T′ then
8:                 return true
9:             end if
10:        end for
11:        return false
12:    end for
13: end procedure
```

addition, the cluster $\langle I', T' \rangle$, generates its own prefix $\mathscr{P}_{\langle I', T' \rangle}$ whose elements, in turn, share transactions with the initial cluster $\langle I, T \rangle$. Thus, all the clusters that derive from a same prefix potentially share several transactions, and the diffset representation allows to reduce memory resources by storing only the elements in the intersections of their tidsets. The implementation into Expedite can be obtained by replacing the sub-procedures PartialIntersection and CompleteIntersection, respectively, with similar sub-procedure DiffsetPartialIntersection and DiffesetCompleteIntersection that compute different transactions between the two tidsets in input instead of the common ones.

3.5 Experimental Results

The goal of this section is to provide an empirical evidence that it is always more advantageous to extract FCIs using Expedite rather than DCI-Closed. Both the algorithms are tested on the twelve datasets described in Table 3.7, whereof eight of them derive from the FIMI repository[1] [41] and the others are access control systems datasets[2] made available by the HP laboratory[3] [38]. For each dataset, the table includes: the number of items, transactions and binary relations; the density of the dataset, that is the ratio between existing binary relations, and the product of items and transactions; and, last, the threshold value used for experiments (except those of Figure 3.2 where

[1] http://fimi.cs.helsinki.fi/data/

[2] Users are mapped into items and permissions are mapped into transactions.

[3] http://www.hpl.hp.com/personal/Robert_Schreiber/data/sacmat%20relations.zip

Table 3.7: Datasets used to evaluate the performance of EXPEDITE.

	Dataset	*Threshold %*	*Items*	*Transactions*	*Relations*	*Density*
FIMI	*1.* chess	30.000	50	3 196	110 593	0.69
	2. connect	30.000	46	67 557	2 425 605	0.78
	3. gaz	0.002	423	59 568	149 299	0.01
	4. mushroom	0.000	119	8 124	186 852	0.19
	5. pumsb	60.000	39	49 046	1 642 346	0.86
	6. pumsb_star	20.000	86	49 046	1 827 509	0.43
	7. T10I4D100K	0.004	844	100 000	1 009 645	0.01
	8. T40I10D100K	0.080	783	100 000	3 896 458	0.05
HP	*9.* americas_large	0.000	10 127	3 485	185 294	0.01
	10. americas_small	0.000	1 587	3 477	105 205	0.02
	11. apj	0.000	1 164	2 044	6 841	0.00
	12. customer	0.000	277	10 021	45 427	0.01

threshold is used as a variable). Experiments are all performed using the C++ programming language on a PC with an Intel Core 2 Duo CPU 3GHz with 3GB RAM.

Figure 3.2 depicts the running times of both EXPEDITE (the standard version described in Algorithm 3.1) and DCI-CLOSED (Algorithm 3.3) to extract all the FCIs from the datasets as a function of the threshold value. For both algorithms, of course, the lower the threshold, the higher is the runtime. The figure shows that, in every datasets, EXPEDITE outperforms DCI-CLOSED from one to two orders of magnitude independently on the threshold. As discussed in Section 3.3.3, the time improvement is mostly due to the reduced number of intersections computed by EXPEDITE. To support this latter statement, in Figure 3.3 it is plotted an histogram that illustrates the number of the duplicate or infrequent itemsets generated by both algorithms. In each dataset, EXPEDITE generates fewer unwanted itemsets, consequently reducing the operations (intersections) and justifying the shortest runtime.

In Section 3.4 two variants of EXPEDITE were proposed to improve its performance. In particular, Section 3.4.1 discussed the implementation of hashcodes for checking duplicates. Figure 3.4a depicts the percentage variation of the runtime with respect to the standard version: grey bars are associated to the simple hashcodes method, while cross-hatched bars to the hashtables one. Results are different for each dataset, varying from an improvement higher than 80% to a worsening of 40% of total runtime of the standard implementation. Both the hash-based implementations, however, performs consistently better when the data are "well-clustered", namely when many itemsets within the data have large support. The most relevant example of a cluster dataset is Americas Large, as shown in a work [29] that provides a visual analysis of this dataset and further confirmed by the *minability index* [27] that is a metric for estimating how well the data are clustered. In particular, the minability index could be computed to priorly estimate whether it is advantageous to use

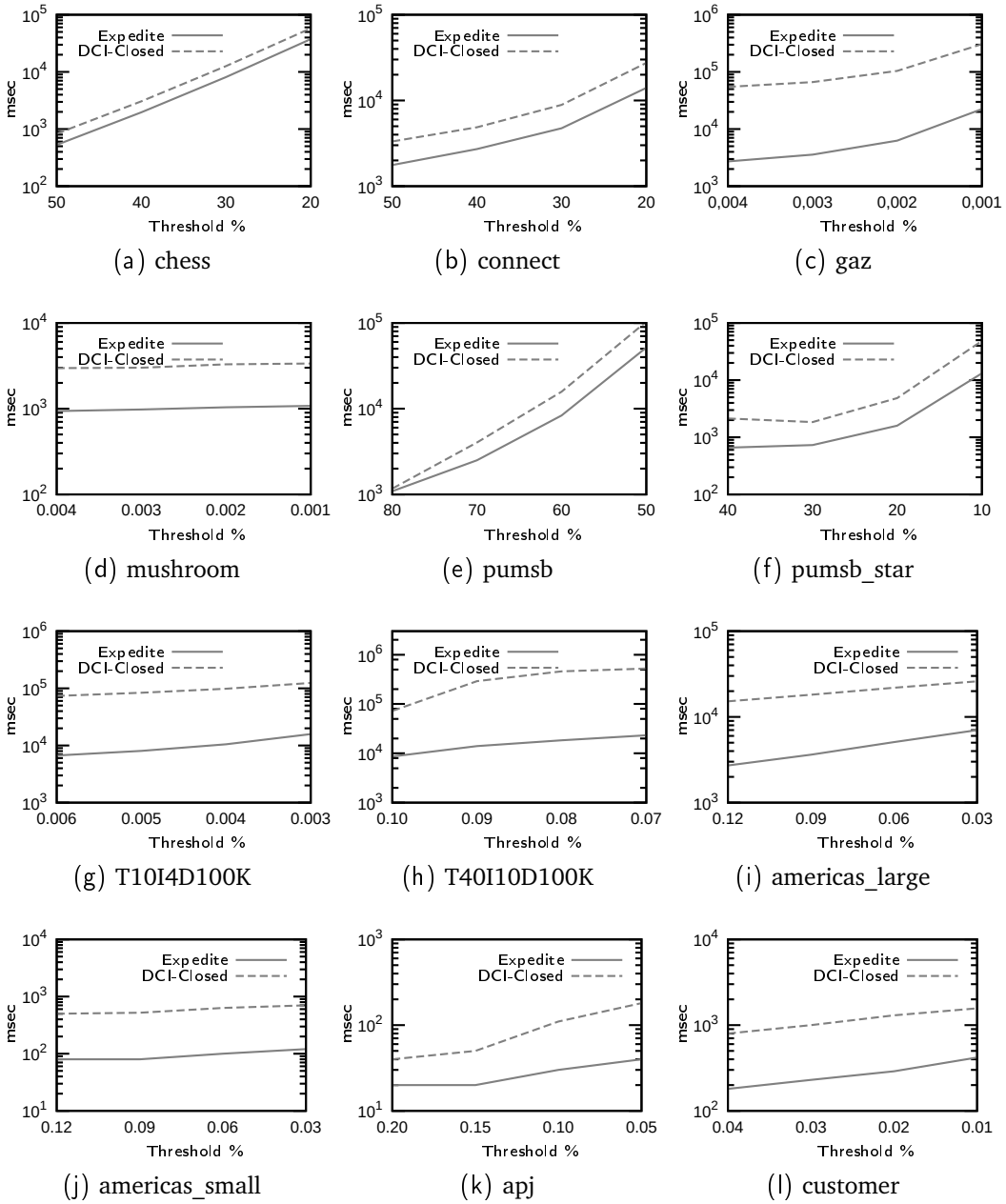


Figure 3.2: Runtime comparison between EXPEDITE and DCI-CLOSED using different values of threshold.

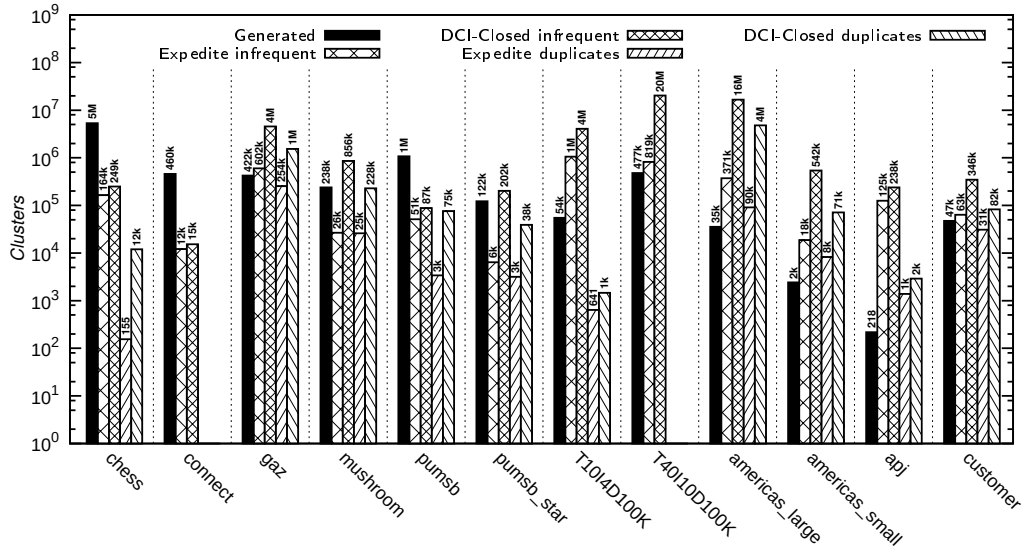
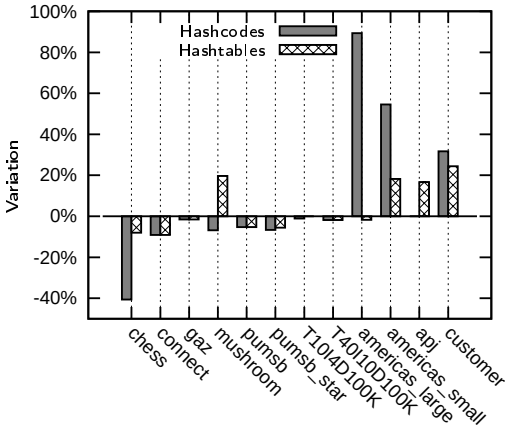


Figure 3.3: Generated clusters by EXPEDITE and DCI-CLOSED. For each dataset and from left to right, the figure depicts: FCIs generated (equal number for both algorithms), generated infrequent itemsets by EXPEDITE, generated infrequent itemsets by DCI-CLOSED, duplicate FCIs by EXPEDITE and duplicate FCIs by DCI-CLOSED.

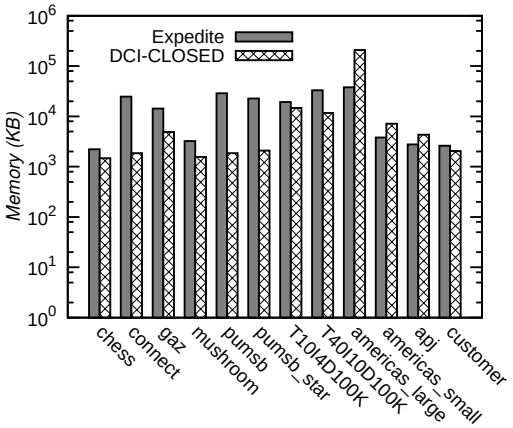
hashmaps for checking duplicates. Finally, Section 3.4.2 introduced the diffset representation with the goal of improving the memory footprint of EXPEDITE. Figure 3.4b depicts the memory footprint of both DCI-CLOSED and EXPEDITE. In average, DCI-CLOSED is slightly more efficient since the structure of cluster used by EXPEDITE is a bit more complex than the one used by DCI-CLOSED. The diffset implementation of EXPEDITE overcomes to this minor drawback. As shown in Figure 3.4c, in ten cases out of twelve, the diffset representation significantly improves the memory footprint of EXPEDITE. This improvement is proportional to the density of the dataset. It is intuitive, indeed, that when the density of the dataset is high the number of FIs comparable by inclusion increases and, thus, the diffset representation becomes advantageous.

3.6 Future Directions

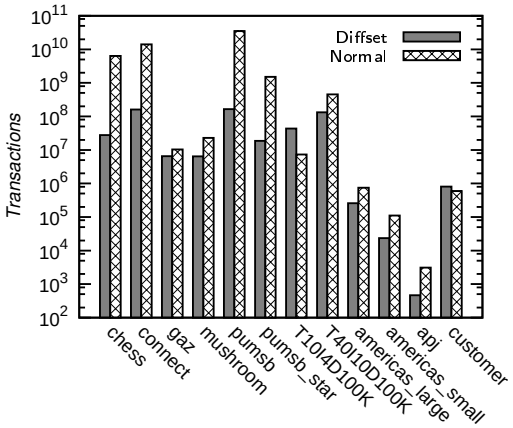
This chapter presented EXPEDITE, a new algorithm for mining frequent closed itemsets. It significantly reduces computation time by avoiding to mine infrequent and duplicate itemsets at intermediate steps of the algorithm. This improvement can be used to speed up computations and to reduce compu-



(a) Percentage varation of Expedite runtime when implementing hashcodes and hashtables for checking duplicates.



(b) Comparison of the memory resources required by EXPEDITE and DCI-CLOSED.



(c) Comparison of the memory resources required by the basic implementation and the diffset implementation of EXPEDITE.

Figure 3.4: Figure 3.4a shows the percentage variation of EXPEDITE's runtime with respect to that of Figure 3.2 after the introduction of hashcodes and hashtables for checking duplicates; Figure 3.4b shows the memory required in kilobytes for running EXPEDITE and DCI-CLOSED; Figure 3.4c shows the number of transactions required to store all the FCIs found by EXPEDITE with and without the implementation of diffsets.

tational requirements in several applications ranging from market analysis, fraud detection, and customer retention, to production control and science exploration. To further boost algorithm performance, also two variations were proposed and analysed. The first one uses hash codes to check duplicate itemsets. This technique can reduce the running time of the algorithm by a huge is specific circumstances. The second variation uses the diffset representation of frequent closed itemsets to reduce the memory footprint. All the findings, other than being corroborated by sound theory, were supported by extensive experiments on the standard datasets used by FIMI workshops.

Compared with DCI-CLOSED, one of the best algorithms presented at FIMI workshops, the basic version of EXPEDITE is up to hundred times faster. Despite algorithms presented at FIMI workshop still are the references for efficiency of mining, recently FCP-MINER [57] has been proposed. Similar to us, the authors of FCP-MINER have chosen DCI-CLOSED for comparing and evaluating the performances of their approach finding that also their algorithm is orders of magnitude faster than DCI-CLOSED. However, it was proved theoretically and experimentally that EXPEDITE is strictly more time efficient than DCI-CLOSED; instead, FCP-MINER can be up to ten times slower than DCI-CLOSED according to empirical results provided by [57]. Thus, a clear advantage of adopting EXPEDITE over FCP-MINER lies in the consistency of results achieved by our algorithm that always guarantees an improvement over DCI-CLOSED.

As for future directions, it is possible to develop new implementations of EXPEDITE to address current big data challenges. Since other similar algorithm have been already ported to multi-core processors [73] and to the MapReduce platform [69], parallel and distributed implementations of EXPEDITE are main future research lines. However, also adapting EXPEDITE to compute only the k most representative frequent itemsets is a promising approach. Indeed, this similar mining task consumes less resources [36] and in some practical applications, for instance in recommendation systems, velocity is a better quality than precision for a mining algorithm.

CHAPTER 4

Secure and Light-Weighted Data Transmission

The advent of the *Internet of Things* (IoT) [10] is leading to a scenario where a multitude of objects, with constrained computational capabilities, are equipped with identification systems, such as RFID tags [95]. In this context, only *light-weighted* protocols are practical for applications, that is to say only those protocols that do not rely on the computational capabilities of the devices where they are running. Light-weighted protocols also have the advantageous of minimizing time delay and energy consumption of devices, and they can be potentially designed for secure data transmission purposes, allowing to protect data carried by tagged objects, denying unauthorized and potentially harmful readings.

In a typical secure communication system, messages undergo two different encodings: an *error-correcting code* is applied at the physical layer to ensure correct reception by the addressee (*i.e.* integrity), while at an upper protocol layer cryptography is leveraged to enforce secrecy with respect to eavesdroppers (*i.e.* confidentiality). However, over the past decades, the research effort focused on developing techniques for implementing at the physical layer also the encoding used to enforce secrecy, rather than using cryptography in the higher layers. This shift makes theoretically possible to obtain light-weighted protocols for secure data transmission, which is the main objective of *physical layer security*.

All constructive solutions proposed so far, to concurrently achieve both integrity and confidentiality at the physical layer in a given channel model, aim — with different degrees of success — at meeting the *secrecy capacity*, namely at maximizing the rate of the code while guaranteeing an asymptotically small information leakage. The main goal of this chapter is to propose CRYPTOLESS, a viable encoding scheme that, for the first time, guarantees both *perfect secrecy* (*i.e.*, no information leakage) and reliable communication over a channel model where the eavesdropper can only leak a limited portion of the transmit-

ted message, precisely over the *generalized Ozarow-Wyner's wire-tap* channel. The provided solution, other than being supported by thorough analysis, is practicable to secure communication systems where the involved devices are resource constrained.

Roadmap of the Chapter. Section 4.1 better introduces physical layer security and provides a description of the channel model. Section 4.2 reviews the state-of-the-art and Section 4.3 provides theoretical results concerning the security of linear codes. Section 4.4 describes the innovative solution CRYPTO-LESS that combines *secret sharing* and linear error-correcting codes for obtaining the desired combination of reliable and perfectly secret communication. Finally, Section 4.5 concludes the chapter discussing possible future directions.

4.1 Physical Layer Security

This section is divided as follows. Section 4.1.1 provides a brief introduction to physical layer security underlying its main objectives and, then, Section 4.1.2 formally describes the communication channel model introducing the necessary notation for the reminder of this chapter.

4.1.1 Introduction to Physical Layer Security

Secure communication requires two equally important conditions being concurrently satisfied: *integrity*, *i.e* correct reception of the message by the intended recipient; and, *confidentiality*, *i.e.* only authorized users should be able to access the content of the message. The integrity of the message received by the addressee may be voluntarily endangered by an adversary, through jamming for instance, or disturbed by natural phenomena such as noise, distortion, and fading. Even if the adversary is not able (or not intending) to modify the message, she can easily eavesdrop on the transmissions whenever the communication channel is insecure, as often happens in wireless communications. Regardless of the origin of the noise, reliable communication over *noisy* channels is made possible by adding redundancy to the data transmitted through *Error-Correcting Codes* (ECC), whereas cryptography is the standard mean to enforce data confidentiality and integrity under active attacks [52].

In many circumstances, the adversary can access or modify only a limited amount of information with respect to the intended recipient. To describe a similar scenario, Wyner introduced a model for physical layer security, called *wire-tap channel* [106], in which the message travels over two different channels: the main channel, accessible to the addressee, and the eavesdropped

channel, suffering from superior noise. The model was later simplified by Ozarow and Wyner with the introduction of the *wire-tap channel II* [74], also known as *Ozarow-Wyner's wire-tap channel*. Here, the main channel is noiseless and the concept of eavesdropped channel is substituted by the assumption that the adversary can choose any subset of $l \leq n$ noiseless digits, where n is the message length. The *Generalized Ozarow-Wyner's wire-tap* (GOW) channel [71] combines the wide applicability of the original wire-tap channel with the precisely defined eavesdropper of the wire-tap II, assuming that the main channel is noisy, and that the adversary can eavesdrop on a subset of l codeword digits of her choice.

For traditional channels, Shannon proved that through ECCs it is possible to reliably communicate at rates arbitrarily close to the capacity of the channel, provided that codewords are sufficiently long. Similarly, Wyner proved that it is possible to reliably and securely communicate — *i.e.* achieving *perfect secrecy* — over the wire-tap channel at rates arbitrarily close to what he called the *secrecy capacity* of the channel. Wyner did not propose any practical construction for a perfectly secret and reliable code, but recent work showed how the secrecy capacity of the channel can be actually achieved with advanced coding schemes [37, 25]. Unfortunately, all similar results consider the asymptotic behaviour of the code and, thus, perfect secrecy is only guaranteed when the message becomes "infinitely long". Traditional ECCs that achieve some level of secrecy exist [15], and secret sharing [84] or similar techniques can provide perfect secrecy over the wire-tap channel II, but none of them alone can provide both security requirements over the GOW channel.

While trying to maximize the rate of secure communications is extremely fascinating, it is likewise important to understand whether current protocols, that do not require cryptography or unrealistically long codewords, can concurrently guarantee perfect secrecy and resilience to transmission errors, and what is the related overhead. Later in this chapter, it is shown how to combine ECCs and secret sharing to achieve perfect secrecy while enforcing arbitrary error correction capabilities in the GOW wire-tap channel model.

4.1.2 The Communication Channel Model

A description of both the wire-tap channel and the generalized wire-tap channel is provided in the following. Hereinafter, $\mathbf{F}_q$ will denote the finite field of order q, where $q = p^\nu$ is a prime power.

Wire-Tap Channel. The wire-tap channel model, depicted in Figure 4.1, describes a scenario where two parties, Alice and Bob, want to communicate

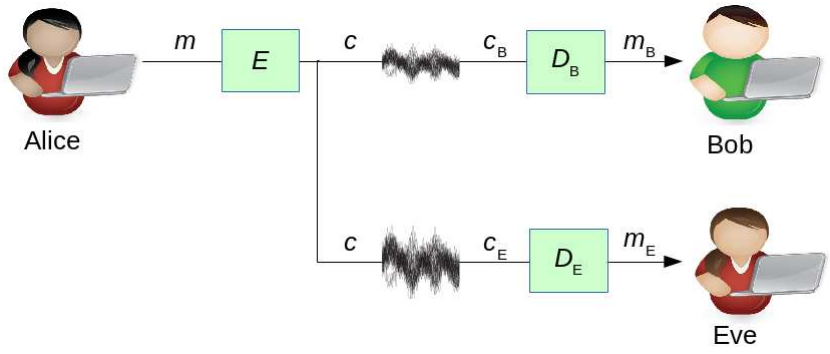


Figure 4.1: A graphical representation of the wire-tap channel.

over a noisy channel, but an adversary Eve tries to eavesdrop the communications. Since the channel between Alice and Bob, referred to as *main* channel, is noisy, Alice uses an encoder E to obtain from the original message $m \in \mathbf{F}_q^k$ a codeword $c \in \mathbf{F}_q^n$. Bob receives a noisy version c_B of c and uses a decoder D_B to remove the noise and obtain a message m_B. The communication is successful if $m_B = m$. To model Eve having limited eavesdropping capacity, in Wyner's model she is assumed to have physical access to a channel than noisier than that of Bob, called *eavesdropped* channel, over which the same codeword c is sent. Eve receives a different noisy version c_E of c, and tries to decode it with her own decoder D_E, obtaining a message m_E.

Generalized Ozarow-Wyner's Channel. The wire-tap channel is a generic model, whose performance is considerably dependent on the type of noise experienced by the recipient Bob and the adversary Eve. To simplify the analysis, the Ozarow-Wyner's model (OW) was proposed as a special case of the wire-tap, where the main channel is noiseless, and Eve is capable to gain access to a subset of $l \leq n$ noiseless digits of c of her choice, where n is the codeword length. A middle ground between such two configurations is the Generalized Ozarow-Wyner's (GOW) model, in which the main channel is noisy, but the eavesdropped channel is modelled as in the OW. From a security standpoint, without specific assumptions on the type of noise affecting the main channel, and by letting l vary from 0 to n, the GOW model covers all possible scenarios, from the best to the worst case. Assuming that Eve is able to extract l noiseless digits allows to neglect both the noise affecting the adversary and the unknown quantity of information she intercepts, while only focusing on what is ultimately relevant. Besides, application settings where a similar eavesdropper is realistic do exist: if the codewords are split into several sub-codewords,

each one transmitted over a different (noiseless) physical layer, the assumption that the adversary cannot earn more than l digits can be replaced with the assumption that the adversary cannot eavesdrop from more than l physical links. Other common settings could be find in wireless communications, whereas the signal degrades with the distance and the assumption that the adversary eavesdrop l noiseless digits can be used as a bound to evaluate the communication security in worst case scenarios.

4.2 State-of-the-Art

Whenever communication occurs over an insecure channel, it is fundamental to concurrently ensure integrity and confidentiality of the transmitted data. In particular, the recent rise of wireless transmissions drew the attention to physical layer security as a promising paradigm to protect communications against eavesdropping attacks by exploiting the physical characteristics of the channel [112]. The fundamentals for physical layer security [15] were laid in the early seventies with the introduction and elaboration of Wyner's wire-tap channel [106]. Since then, several extensions of Wyner's channel model have been considered: for instance, the Broadcast Channel with Confidential messages (BCC) [33] in which, similarly to the wire-tap channel, a message intended for one of the receivers is confidential; the Gaussian channel [61], that is the Wyner's wire-tap channel when Additive White Gaussian Noise (AWGN) is assumed as model for data transmission errors; and, channels that impose a combinatorial constrain, rather than probabilistic, to the adversary [74, 25].

To discuss the security of the wire-tap channel, Wyner introduced the notions of *reliability condition* and *security condition* [106]. The reliability condition is verified if $\lim_{k\to+\infty} \Pr[m' \neq m] = 0$, namely if the error probability approaches 0 as the size of the message grows. The security condition, instead, is verified if $\lim_{k\to+\infty} I(m, y)k^{-1} = 0$, namely if the normalized mutual information between the eavesdropped data and the message is 0. Based on such two requirements, Wyner also introduced the concept of *secrecy capacity* of the channel, that is the maximum rate at which information can be transmitted over the channel with the reliability and security conditions holding. Wyner proved that, when the channel of the receiver is subject to less noise than the channel of the wire-tapping opponent, the secrecy capacity is positive and, thus, it is possible to communicate over that channel without violating none of the two conditions. Several papers [68, 67] (even very recently [51, 16, 13, 81, 50]) followed Wyner's work, focusing on the concept of secrecy capacity and the security properties of error-control coding techniques.

The work of Wyner had two main limitations. On the one hand, its security

condition was too weak, as highlighted by Maurer [68], who suggested to replace Wyner's security condition with the requirement that the mutual entropy approaches to zero as the size of the message grows; that is: $\lim_{k\to+\infty} I(m,y) = 0$. However, even this definition responds to an idea of asymptotic security, while perfect secrecy actually means to leak no information at all. On the other hand, Wyner did not provide any constructive indication for designing codes approaching the secrecy capacity. Several researchers tried to fill in this gap, but the best results were obtained under precise assumptions on the channel model (*e.g.*, Binary Erasure Channel [89], Binary Symmetric Channel [66], combinatorial constrained model [25], Gaussian wire-tap channel [58, 64], compound wire-tap channel [20], broadcast channel with confidential messages [45]). In general, what emerges is that LDPC codes [89, 58, 20] and Polar Codes [66, 45, 64] seem the most promising solutions.

Wrapping up, past research concerning the wire-tap channel mostly focused on understanding if and how it is possible to communicate at rates approaching the secrecy capacity of the channel, only trying to guarantee (to some extent) asymptotic secrecy. Conversely, this chapter aims at providing a constructive solution, that is CRYPTOLESS, for obtaining perfect secrecy with practical encoding and decoding algorithms. This goal is achieved by applying secret sharing to any generic error-correcting encoder. The joint use of ECCs and some sort of secret sharing is not new in the literature. Solutions based on Rivest's *All-Or-Nothing Transform* (AONT) [82], a primitive assimilable to (i,j) secret sharing[1], have been suggested [37, 25], but AONTs frequently make use of symmetric ciphers and do not offer resilience to data loss. Reliability to transmission errors could in principle be obtained combining AONTs with error-correction codes, as successfully proposed for data security in dispersed storage systems [80]. However, analogous solutions for the wire-tap channel have never been investigated and they would rely on a cryptographic construction.

4.3 Security of Linear Codes

This section provides fundamental results helpful to determine the level of security achieved by linear ECCs when adopted as encoders in the generalized Ozarow-Wyner's model. To this end, Section 4.3.1 reminds the definition of linear ECCs and introduces the notion of uncertainty rate, a measure to capture in which extent a code leaks information; then, Section 4.3.2 provides

[1] A more typical notation is (k,n) secret sharing. However, k and n are reserved for denoting dimension and length of a code, respectively.

two practical formulas binding the uncertainty rate of the code to its parameters and to the code rate, respectively; and, finally, these results are discussed in Section 4.3.3, and compared with the state-of-the-art in Section 4.3.4.

4.3.1 Linear ECCs and Uncertainty Rate

In the following, the definition of linear ECCs is reviewed.

Definition 4.1 (Linear Error-Correcting Code)**.** *A* linear Error-Correcting Code (ECC) *is a map E from a set of messages $M = \mathbf{F}_q^k$ into a set of codewords $C \subset \mathbf{F}_q^n$, such that, for each $m \in M$, the digits of $c = E(m) \in C$ are obtained as n linear combinations of the digits of m. The set C is a linear subspace of $\mathbf{F}_q^n$ of dimension k, and it uniquely determines the code. The code is usually defined by either means of its $n \times k$ matrix G, called* generating matrix *of the code, such that $c = G \cdot m$, or by its $(n-k) \times n$* parity-check matrix *H, such that $H \cdot c = 0$ if and only if $c \in C$. Each codeword of length n conveys k information digits and the ratio $r = k/n$ is called* code rate*.The parameters k and n are called the* dimension *and the* length *of the code, respectively.*

As better discussed in Section 4.2, Wyner [106] proposed a definition of security for the wire-tap channel based on two desiderata, that he defined reliability and security conditions. Based on such requirements, he introduced the notion of secrecy capacity of a channel that, intuitively, corresponds to the maximum rate at which information can be securely transmitted over that channel. However, this security measures relies on the intrinsic and asymptotic properties of the channel without considering the code used for data transmission. Instead, a different approach consists in measuring the amount of leaked information by as a function of both the specific threat model (*i.e.* the communication channel) and coding scheme adopted (not necessarily ECCs). This approach led to define the *equivocation rate* [62] as a secrecy metric that has been recently validated [99, 12]. The main idea was to use the entropy of the code as security metric. In the following, a new secrecy measure is introduced: the *uncertainty rate*, which can be seen as the application of the equivocation rate to the ECCs codes. The result is that a very practical secrecy metric that directly uses the parameters of an ECC to quantify the information leakage. This measure is introduced in the following.

The assumption is that Alice and Bob are communicating over a generic wire-tap channel using a linear ECC of dimension k and length n, and that the adversary Eve eavesdrops the transmission of a codeword c, obtaining a noisy version c_E of c.

Definition 4.2 (Dimension of Uncertainty)**.** *Based on c_E and leveraging on the linear equations binding the digits of c, Eve is capable of reducing the space where c varies to a set of q^s equally likely codewords. The parameter $s \leq k$, which depends on both the system and the threat model, is called the* dimension of uncertainty *of the adversary.*

Note that, since the total number of admitted codewords is q^k, the ratio between the two dimensions $\epsilon = s/k \in [0,1]$ is a normalized measure of the adversary uncertainty. Thus, the uncertainty rate can be defined as follows.

Definition 4.3 (Uncertainty Rate)**.** *Let $s \leq k$ be the dimension of uncertainty of the adversary, and let $\epsilon = s/k \in [0,1]$. The parameter ϵ is called* uncertainty rate *of the adversary.*

To be equally uncertain among q^s codewords means to be equally[2] uncertain among q^s possible alternatives for the original message m, since messages and codewords are in one-to-one correspondence. If the dimension of uncertainty is s, it is possible therefore to assume that the adversary recovered

$$k - s = \lceil (1-\epsilon)k \rceil \tag{4.1}$$

digits of the original message m.

The dimension of uncertainty s coincides with the Shannon's entropy of the codeword c, conditioned to the intercepted word c_E. In fact, if knowing c_E allows Eve to infer that the codeword c is uniformly distributed in a set of size q^s, then

$$\mathscr{H}(c|c_E) = -\sum_{i=1}^{q^s} \frac{1}{q^s} log_q \left(\frac{1}{q^s}\right) = s$$

where $\mathscr{H}$ denotes the entropy function[3].

The uncertainty rate is a normalized metric that depends on the dimension k of the code, and the higher it is, the lesser is the information leakage of the code. Ideally, the objective would be to determine a code with uncertainty rate $\epsilon = 1$ since this would guarantee zero leakage. Unfortunately (yet intuitively), Section 4.3.2 proves that no code achieves $\epsilon = 1$ under the GOW model with positive parameter $l > 0$.

[2]The two statements are equivalent from a computational security standpoint.

[3]The uncertainty rate coincides with the equivocation rate of the message in the traditional Wyner's model (except for base q logarithm, as digits are in the set $\mathbf{F}_q$), but it becomes a more direct measure of the level of uncertainty of the adversary when focusing on the GOW channel.

4.3.2 Measuring Security Through the Uncertainty Rate

In the GOW model, the adversary Eve eavesdrops l noiseless digits of her choice from the transmitted codeword c. The working assumption is that the specifications of the code used are known to Eve and, thus, she knows the linear parity-check equations that bind the digits of c. Based on the l digits available to her and on such equations, Eve can infer information on the original message m. Relying on the definition of uncertainty rate, Theorem 4.1 and Corollary 4.1 establish to which extent this happens. To enhance readability, all proofs are presented at the end of this section.

Theorem 4.1. *Assume that a linear ECC code of dimension k and length n is used as an encoder in the GOW wire-tap channel model, in which the adversary has access to l noiseless digits of the transmitted codeword c. The dimension of uncertainty of the code is 0, and so is the uncertainty rate, if and only if $l \geq k$. For all $l < k$, the dimension of uncertainty of the code is $s = k - l$, and the uncertainty rate is $\epsilon = \frac{s}{k} = 1 - \frac{l}{k}$. In particular, $\epsilon = 1$ if and only if $l = 0$. Hence, a linear code alone cannot guarantee perfect secrecy for any $l > 0$.*

Observe that Theorem 4.1 could be equivalently stated in terms of the rank of the parity check matrix H of the code, recalling that such rank is $n-k$. This may turn especially useful since some powerful families of linear codes (*e.g.*, LDPC codes), are usually described and generated by means of the matrix H.

While Theorem 4.1 binds the uncertainty rate of the code to its parameters and to the number l of intercepted digits, the following Corollary 4.1 expresses the same results in terms of rates of information transmitted, and eavesdropped.

Corollary 4.1. *Assume that a linear ECC code of dimension k and length n is used as an encoder in the GOW wire-tap channel model, in which the adversary has access to l noiseless digits of the transmitted codeword c. Let $\rho = \frac{k}{n}$ denote the code rate, and let $\lambda = \frac{l}{n}$ denote the eavesdropping rate of the adversary. The uncertainty rate of the code is 0 if and only if $\rho \leq \lambda$. For all $\rho > \lambda$, the uncertainty rate is $\epsilon = 1 - \frac{\lambda}{\rho}$. In particular, $\epsilon = 1$ if and only if $\lambda = 0$.*

When the code length n grows, the number l of digits accessible to the adversary can be reasonably expected to grow proportionally, exactly as the code dimension k. The eavesdropping rate λ is exactly the proportionality constant between l and n, similarly to the code rate ρ for k and n. Corollary 4.1 shows how ϵ depends on ρ and λ, capturing the idea that the uncertainty rate does not really depend on the code dimension and length, but rather on the rates to which information is transmitted and eavesdropped. The corollary shows

that there exists a critical value for the code rate under which the code becomes completely unreliable for security purposes under the GOW channel model, and that such a critical value is exactly the eavesdropping rate of the adversary.

Proofs. The remainder of this section focuses on providing the proof of the above results.

Definition 4.4 (Row-Column Permutation)**.** *A row-column permutation of a $t \times u$ matrix H is a map $\sigma : \mathscr{M}_{t\times u} \longrightarrow \mathscr{M}_{t\times u}$ defined by $(h_{ij}) \mapsto (h_{\sigma_t(i)\sigma_u(j)})$ where σ_t and σ_u are permutations of the sets $\{1,\dots,t\}$ and $\{1,\dots,u\}$, respectively.*

A row-column permutation σ of a parity-check matrix H defines a new matrix $\tilde{H} = \sigma(H)$ that shares the same size of H and whose codewords are a permutation of those defined by H. In fact, $\sigma(H)\sigma_u(c)^T = 0$ is satisfied if and only if $Hc^T = 0$ (note that, since c is a vector of dimension u, it can be though as a matrix $1 \times u$ and, thus, $\sigma(c)$ permutes its elements accordingly to $\sigma_u(c)$).

Lemma 4.1. *Let H be the $(n-k)\times n$ parity-check matrix of a linear code used as an encoder in the OW wire-tap channel model, and let c be the transmitted codeword. The adversary can recover the whole codeword c if and only if there is a row-column permutation $\sigma : H \mapsto \sigma(H) = \tilde{H}$ such that $\tilde{H} = [\tilde{H}_1\ \tilde{H}_2]$ is a two blocks matrix where $\tilde{H}_1$ is an $(n-k)\times l$ sub-matrix, and $\tilde{H}_2$ an $(n-k)\times(n-l)$ sub-matrix having* $\mathrm{rk}(\tilde{H}_2) = n-l$*. Further, the dimension of uncertainty of the LDPC is*

$$s = \min_{\sigma}\{n - l - \mathrm{rk}(\tilde{H}_2)\}. \tag{4.2}$$

Proof. ($\Longleftarrow$) Let us assume that there is a row-column permutation $\sigma : H \mapsto \sigma(H) = \tilde{H} = [\tilde{H}_1\ \tilde{H}_2]$ such that $\tilde{H}_1$ is an $(n-k)\times l$ sub-matrix and $\tilde{H}_2$ is an $(n-k)\times(n-l)$ sub-matrix with $\mathrm{rk}(\tilde{H}_2) = n-l$. We want to show that the adversary can recover any codeword c transmitted over the channel. Let us denote with $\tilde{c} = \sigma_n(c)$ the application of this permutation to a generic codeword c, namely $\tilde{c} = [c_{\sigma_n(1)}\ \dots\ c_{\sigma_n(l)}\ c_{\sigma_n(l+1)}\ \dots\ c_{\sigma_n(n)}]$. To easy notation, we write $\tilde{c} = [\tilde{x}\ \tilde{y}]$ where $\tilde{x}$ represents the first l elements of $\tilde{c}$ and $\tilde{y}$ the remaining $n-l$. Accordingly to the OW model, the adversary can choose to eavesdrop the l bits composing $\tilde{x}$. Noticing that $\tilde{H}\tilde{c}^T = 0$ is equivalent to

$$\tilde{H}_1\tilde{x}^T = \tilde{H}_2\tilde{y}^T \tag{4.3}$$

and that $\tilde{H}_1\tilde{x}^T$ is a vector known to the adversary, she can retrieve the missing $n-l$ components of $\tilde{y}$ by solving the system defined in Eq. (4.3). In fact, it is made of $\mathrm{rk}(\tilde{H}_2) = n-l$ independent linear equations and it has, thus, a

single solution. Finally, she can retrieve the original codeword c by applying the inverse of the row column permutation, namely $c = \sigma_n^{-1}$.

($\Longrightarrow$) If the adversary can retrieve a codeword c by exploiting the linear equations defined by $Hc^T = 0$ and using only l bits, then it means that she has at least $n-l$ independent linear equations to work with; namely, $\text{rk}(H) \geq n-l$. Then, a row-column permutation $\sigma(H) = \tilde{H} = [\tilde{H}_1\ \tilde{H}_2]$ with $\text{rk}(\tilde{H}_2) = n-l$ must exists because $\text{rk}(H) \geq n-l$ implies that H has a sub-matrix M with rank $n-l$ (the elements of M can be arbitrarily moved to match $\tilde{H}_2$ using an appropriate permutation σ).

The goal is to show that the dimension of uncertainty is $s = \min_\sigma\{n - l - \text{rk}(\tilde{H}_2)\}$. As already noticed, the recovering of a codeword is bind to the solution of the Eq.(4.3). Hence, the adversary must recover $n-l$ bits using $\text{rk}(\tilde{H}_2)$ independent linear equations. Taking the minimum of this value among all the possible row-column permutations σ, the wanted equation is obtained. □

The following statement is equivalent to Lemma 4.1 but easier to apply.

Theorem 4.2. *Let H be the $(n-k)\times n$ parity-check matrix of a linear code used as an encoder in the OW wire-tap channel model and let c be the transmitted codeword. The adversary can recover the whole codeword c if and only if* $\text{rk}(H) \geq n-l$. *If the adversary cannot recover the whole codeword, then the dimension of uncertainty is* $n-l-\text{rk}(H)$.

Proof. ($\Longleftrightarrow$) Due to Lemma 4.1, it is only necessary to prove that there is a row-column permutation $\sigma : H \mapsto \sigma(H) = \tilde{H} = [\tilde{H}_1\ \tilde{H}_2]$ such that $\tilde{H}_1$ is an $(n-k)\times l$ sub-matrix and that $\tilde{H}_2$ is an $(n-k)\times(n-l)$ sub-matrix with $\text{rk}(\tilde{H}_2) = n-l$ if and only if $\text{rk}(H) \geq n-l$. The condition $\text{rk}(H) \geq n-l$ is equivalent to say that H has a $(n-k)\times(n-l)$ sub-matrix M with rank $\text{rk}(M) = n-l$ obtained by removing l columns from H. Thus, the corollary is proved by picking σ as the rows column permutation that moves the elements of M to the sub-matrix $\tilde{H}_2$. □

Theorem 4.2 directly proves Theorem 4.1 reminding that, for each linear code, the rank of the parity-check matrix H is $n-k$. Instead, Corollary 4.1 immediately follows reminding that the code rate is defined as $\rho = \frac{k}{n}$.

4.3.3 Discussion

The uncertainty rate is remarkably suitable to measure the level of security guaranteed by a linear code under the GOW model. Corollary 4.1 is particularly interesting, relating the uncertainty rate with the code rate ρ and the

eavesdropping rate λ. The code rate, that is, the ratio between the dimension k of the code and its length n, measures how much information a code conveys. The eavesdropping rate, that is, the ratio of code digits available to the adversary, measures the amount of information leaked. The smaller is the code rate, the larger is the redundancy introduced by the code, improving the error correcting capabilities of the code, but concurrently facilitating the attack. Secure communications when $\rho \leq \lambda$ are impossible, and, in general, the security depends of the ratio $\frac{\lambda}{\rho}$.

A graphical representation is provided in Figure 4.2 and Figure 4.3 that show achievable values of uncertainty rate for linear codes with fixed code length, by varying the rank of the parity-check matrix and the eavesdropping ability of the adversary. In particular, Figure 4.2 shows achievable uncertainty rates $\epsilon \in [0,1]$ for a linear code with $n = 20$ as a function of both the rank of the parity-check matrix $\mathrm{rk}(H)$ and the number l of digits eavesdropped by the adversary. The larger is l, the lower must be the rank of the parity-check matrix to ensure a positive value of uncertainty. The special cases $l = 5, 10, 15$ are further depicted in Figure 4.3; for instance, the maximum value of uncertainty rate achievable when the adversary gets $l = 10 = n/2$ digits of the codeword is $\epsilon = 0.5$.

4.3.4 Comparison with Similar Results in the Literature

Ozarow and Wyner [74] supported their OW model describing its mathematical properties. To achieve their main results they did not make any particular assumption on the encoder of the wire-tap channel. Thus, they have found general properties applicable to any kind of encoder proving non-constructive existence theorems to show that encoders with given security parameters must exist. In a similar way, the recent work of Cheng *et al.* [24] provides theoretical results for characterizing the achievable values of code rate in the OW model. Instead, this work focused on linear codes to obtain more specific and practical results concerning the secrecy of the less restrictive GOW model. In particular, Theorem 4.1 and Corollary 4.1 provide precise formulae to compute the uncertainty rate of the adversary, which allowed to exactly compute — and plot in Figure 4.2 — the uncertainty rates achievable by a linear code of a fixed length. In their work, Ozarow and Wyner also found interesting properties related to the uncertainty achieved by linear codes that have been further refined by Wei [98]. However, linear codes were not explicitly considered and there is not any result similar to the uncertainty rate formula here provided. Applications of some families of linear codes (*e.g.*, LDPC codes) to the wire-tap channel model have been considered [90, 51] but their applica-

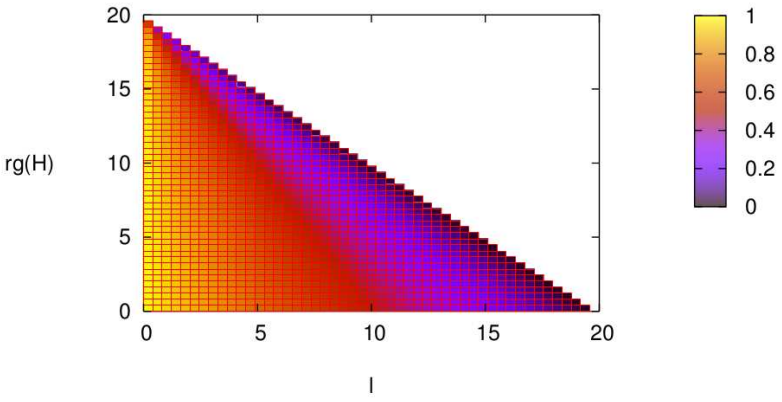


Figure 4.2: Values for the uncertainty rate $\epsilon = s/k \in [0,1]$ achievable by an ECC code with length $n = 20$. The colour matches with the value of ϵ (darker is lower) that is depicted as a function of both the rank of the parity-check matrix rk(H) and the number of digits eavesdropped by the adversary l.

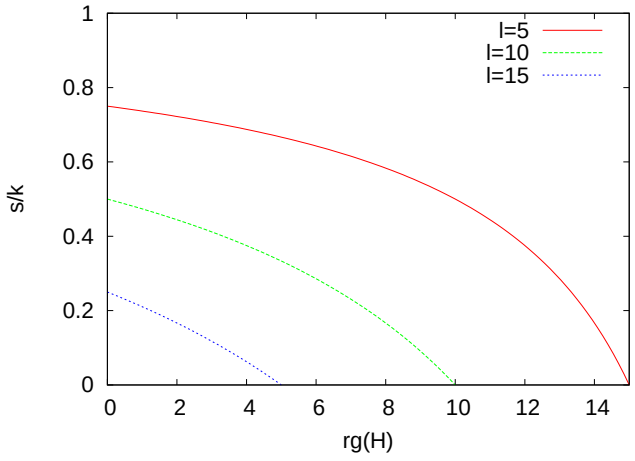


Figure 4.3: Values for the uncertainty $\epsilon = s/k \in [0,1]$ achievable by an ECC code with length $n = 20$ for different values of $l = 5, 10, 15$ and for different ranks of the parity-check matrix.

tion on the GOW model was not previously investigated.

4.4 Secure Communication with CRYPTOLESS

Section 4.3 shows that ECCs cannot offer perfect secrecy, and that the error correcting capability of a code is proportional to the information leakage it causes. A possible approach to obtain perfect secrecy while allowing reliable communications is given by the adoption *secret sharing* techniques, as described in this Section. Precisely, Section 4.4.1 reviews how secret sharing works; then, Section 4.4.2 shows how to combine ECCs codes with secret sharing to obtain a reliable and secure encoder, that is CRYPTOLESS; and, finally, Section 4.4.3 discusses a toy example where a (j, j) secret sharing scheme is combined with a $(l + 1)$ repetition code (a special case of linear ECC).

4.4.1 Secret Sharing

Assuming that a user u knows a secret S, a (i, j)-threshold *secret sharing* scheme allows u to choose two positive integers j and $i \leq j$ and to generate j pieces of information, such that any i out of them are *necessary* and sufficient to recover S. The parameter i is the threshold that determines the amount of information needed to reconstruct S and, thus, perfect secrecy is guaranteed provided that no more than $i-1$ shares are leaked; further, correct reception is ensured when no more than $j-i$ shares are lost during transmission. The most known construction of secret sharing schemes relies on polynomial interpolation,[4] leveraging on the fact that any point of the curve defined by a polynomial of degree $i-1$ determines a linear equation satisfied by the i coefficients of the polynomial. Formally, if $S \in \mathbf{F}_q$, a random polynomial $f(x) \in \mathbf{F}_q[x]$ with free term S is chosen, and j pieces of information $d_1, \ldots, d_j \in \mathbf{F}_q$, denoted *shares*, are generated as $d_t = f(t) \mod \mathbf{F}_q$, for $t = 1, \ldots, j$. Anyone with access to i or more shares can recover $f(x)$, and thus S, but with less than i shares anyone of the possible q values for S is exactly equally likely, and no information about S is leaked.

Notwithstanding their interesting properties, secret sharing schemes were not designed to allow communication over a noisy channel. This is particularly evident when considering a channel model where transmissions may be subject to a combination of erasures and errors. In fact, a secret sharing based encoder would be resilient to data loss — since it would be tolerant to a loss of

[4] Sharing schemes were formally introduced independently by Shamir [84] and Blakley [14]. The two schemes are *de facto* equivalent, but Shamir's definition, based on polynomial interpolation, is the most renowned.

up to $j-i$ shares — but not to data corruption — since corrupted data would be indistinguishable from the recipient perspective and secret sharing. In coding theory terminology, this means that secret sharing can be used to tackles erasures, not errors. While potentially suitable for the OW channel model (where the main channel is noiseless), the applicability of secret sharing alone to the GOW model is severely limited.

4.4.2 CRYPTOLESS:Combining Secret Sharing and ECC

An ECC can guarantee reliable communication over a noisy channel, but it cannot provide perfect secrecy in the GOW channel model, as it has uncertainty rate $\epsilon = 1 - \frac{l}{k} < 1$ for each $l > 0$. On the opposite side of the spectrum, a (i, j) secret sharing based encoding scheme can guarantee perfect secrecy if $l < i$, but cannot provide suitable error correcting capabilities on noisy channels. Therefore, in the following the two primitives are combined to obtain a coding technique that concurrently achieves both requirements.

The proposed scheme, called CRYPTOLESS and described in Algorithm 4.4.2, returns a codeword $c \in \mathbf{F}_q^n$ from a single digit message $m \in \mathbf{F}_q$. The encoding consists of the following steps: (i) $i-1$ coefficients are randomly picked in $\mathbf{F}_q$ (Lines 2–4), and they are used together with m to define the polynomial $f(x) \in \mathbf{F}_q[X]$ (Line 5); (ii) $f(x)$ is used to implement a (i, j) secret sharing scheme, obtaining the j shares $d_1, \ldots, d_j \in \mathbf{F}_q$ (Lines 6–8); and, finally, (iii) the word $(d_1, \ldots, d_j) \in \mathbf{F}_q^j$ is encoded with a liner ECC E to obtain the codeword $c \in \mathbf{F}_q^n$ (Line 9).

Algorithm 4.1 The main procedure of CRYPTOLESS.

1: **procedure** CRYPTOLESS(Message $m \in \mathbf{F}_q$, Linear ECC Encoder $E : \mathbf{F}_q^j \to \mathbf{F}_q^n$)
2: **for** $u = 1, \ldots, i-1$ **do**
3: $\alpha_u \xleftarrow{R} \mathbf{F}_q$
4: **end for**
5: $f(X) \in \mathbf{F}_q[X] \leftarrow m + \alpha_1 X + \cdots + \alpha_{i-1} X^{i-1}$
6: **for** $u = 1, \ldots, j$ **do**
7: $d_u = f(u) \mod \mathbf{F}_q$
8: **end for**
9: **return** $c \leftarrow E(d_1, \ldots, d_j) \in \mathbf{F}_q^n$
10: **end procedure**

Theorem 4.3 and Corollary 4.2 provide precise results concerning the security properties of the proposed scheme, assuming that the codeword c is sent over a GOW channel and where the adversary Eve is able to eavesdrop l noiseless digits.

Theorem 4.3. *Under the GOW channel model, the proposed scheme guarantees perfect secrecy if $l < i$, while it allows perfect recovery of the original message if $l \geq i$.*

Proof. Theorem 4.1 establishes that the uncertainty rate of the considered model is $\epsilon = 1 - \frac{l}{j}$. As stated in Eq. (4.1), the number of digits of the word $(d_1, \ldots, d_j)$ that the adversary can reconstruct is $(1-\epsilon)j$. Putting things together, the adversary can reconstruct $(1-\epsilon)j = \frac{l}{j}j = l$ shares of the secret m, and the thesis follows from the properties of (i, j) secret sharing schemes. □

Given Theorem 4.3, the following corollary is straightforward.

Corollary 4.2. *Under the GOW channel model, the proposed scheme guarantees perfect secrecy* and *reliable communication if the adversary can access a number of noiseless digits $l < i$, while the recipient can recover a number of noiseless digits $l \geq i$.*

The condition for reliable communication expressed by Corollary 4.2 is only a sufficient one. More generally, according to the error correcting capabilities of the linear ECC used, whenever the corresponding decoder allows the recipient Bob to reconstruct at least i digits of the word $(d_1, \ldots, d_j)$, Bob can recover $f(X)$ by interpolation and obtain the constant term m.

4.4.3 A Toy Example

This section discusses a toy example of code obtained by combining a (j, j) secret sharing scheme (*i.e.*, all j shares are necessary to recover the secret S) with a $(l+1)$ repetition code. Precisely, first a brief description of repetition codes and their properties is provided and, then, it is shown that we show that the defined code achieves both perfect secrecy and error correcting capabilities.

Repetition Codes. Repetition codes are a special family of linear ECCs. As the name suggests, in a r repetition code each digit of the original message is simply repeated r times. That digit can be recovered by majority if at least half of the r copies are correctly received, regardless of the fact that the other copies are erased or corrupted. Formally, k message digits $(x_1, \ldots, x_k) \in \mathbf{F}_q^k$ are encoded into $n = rk$ code digits $(x_1, \ldots, x_1, \ldots, x_k, \ldots, x_k) \in \mathbf{F}_q^n$, where each digit x_i, $i = 1, \ldots, k$ is replicated $r \in \mathbb{N}$ times. The code rate is $\rho = \frac{1}{r}$. Thanks to the results of Section 4.3, the following corollary holds.

Corollary 4.3. *Under the GOW channel model, if the adversary eavesdrops l digits, the uncertainty rate of a r repetition code is $\epsilon = 0$ if $l \geq k$, while it is $\epsilon = 1 - \frac{l}{k}$ otherwise, regardless of r. Equivalently, if the eavesdropping rate of the adversary is λ, the uncertainty rate is $\epsilon = 0$ if $\lambda \geq \frac{1}{r}$, while it is $\epsilon = 1 - r\lambda$ otherwise.*

Corollary 4.3 proves that a repetition code guarantees positive uncertainty provided that the adversary eavesdrops $l < k$ digits since, in her best-case scenario, Eve eavesdrops just one copy each of distinct digits of the original message. However, the uncertainty rate is pretty low, especially if compared to the level of reliability provided by the code: to be sure to correctly recover all the original message, the intended recipient Bob needs at least $\frac{r}{2}$ copies of *all* message digits.

As a special case, consider for instance a scenario where Alice wants to transmit to Bob a single digit $x \in \mathbf{F}_q$, *i.e.*, $k = 1$. Alice applies a r repetition code to get the codeword $(x, \ldots, x)$ composed of r copies of x. What Theorem 4.3 says is that, no matter how large r is, if Eve intercepts $l \geq 1$ digits of the codeword the uncertainty rate is $\epsilon = 0$. Indeed, all digits of the codeword coincide with x, so to intercept any one of them means to get to know x. Conversely, if $l < k$, *i.e.*, if $l = 0$, the uncertainty is clearly $\epsilon = 1$.

Analysis of the Proposed Toy Example To amplify the uncertainty rate provided by an r repetition code, it can be combined with a preliminary step consisting of a (j, j) secret sharing scheme. To easy exposition, here is considered the special case where Alice wants to send to Bob a single digit $m \in \mathbf{F}_q$. It is, then, proved that such a composed scheme achieves perfect secrecy for up to a suitable l, depending on the choice of r and j. The proposed toy example is formally described in Algorithm 4.4.3.

From a single message digit m, the encoder of Algorithm 4.4.3 produces a codeword of length $n = jr$ as follows: (i) $j - 1$ digits $x_1, \ldots, x_{j-1}$ are picked uniformly at random in $\mathbf{F}_q$ (Lines 2–4); (ii) one further digit is computed as $x_j = m + x_1 + \cdots + x_{j-1} \mod \mathbf{F}_q$ (Line 5); and, finally, (iii) all j digits $x_1, \ldots, x_j$ are replicated r times to produce the code digits $c_{1,1}, \ldots, c_{j,r}$ (Lines 6–10). Steps (i) and (ii) implement a (j, j) secret sharing scheme.[5] Step (iii) is a simple r replication scheme, where each x_i is replicated r times.

The threat model allows the adversary Eve to eavesdrop any l noiseless digits. Eve can successfully recover m if and only if she gets access to all the shares $x_1, \ldots, x_j$ due to the secret sharing properties. Thus, if $l \geq j$, Eve might

[5] When $i = j$, a (i, j) secret sharing scheme can be implemented as shown, without recurring to polynomial interpolation.

Algorithm 4.2 The main procedure of CRYPTOLESS applied to a repetition code.

procedure CRYPTOLESS-REPETITIONCODE(Message $m \in \mathbf{F}_q$)
  for $i = 1, \dots, j-1$ **do**
    $x_i \xleftarrow{R} \mathbf{F}_q$
  end for
  $x_j \leftarrow m + x_1 + \cdots + x_{j-1} \mod \mathbf{F}_q$
  for $i = 1, \dots, j$ **do**
    for $t = 1, \dots, r$ **do**
      $c_{i,t} \leftarrow x_i$
    end for
  end for
  return $c \leftarrow (c_{1,1}, \dots, c_{1,r}, c_{2,1}, \dots, c_{j,r}) \in \mathbf{F}_q^{jr}$
end procedure

be able to pick such l digits so as to have at least one copy of all such digits. However, if $l < j$, there is no way for Eve to get all the digits $x_1, \dots, x_j$, and it is impossible for her to recover any information on m. Consequently, the uncertainty is $\epsilon = 0$ if $l \geq j$, while it is $\epsilon = 1$ if $l < j$. The same concepts can be stated using the notion of uncertainty rate, together with the results of Section 4.3. Focusing on the repetition code, it is used to encode the word $(x_1, \dots, x_j)$ of length j, so it produces an uncertainty rate $\epsilon = 1 - \frac{l}{j}$ over the word $(x_1, \dots, x_j)$. This means that Eve can recover $(1-\epsilon)j = l$ digits of the set $(x_1, \dots, x_j)$. Once again, this yields perfect secrecy if $l < j$ thanks to the properties of secret sharing, while zero uncertainty if $l \geq j$.

For what concerns the impact of the proposed scheme on the reliability of the transmission, it is easy to realize that using only the first step of the example (*i.e.*, secret sharing) it is possible to get even better secrecy: if Alice directly transmits the word $(x_1, \dots, x_j)$ without replication, perfect secrecy is guaranteed as long as Eve eavesdrops $l < j$ digits over a shorter message. This means that it is possible to obtain the same level of secrecy for a much larger eavesdropping rate $\left(\frac{l}{j} \text{ instead of } \frac{l}{jr}\right)$. However, directly transmitting $(x_1, \dots, x_j)$ provides *no* correction capabilities to Bob in the presence of errors or erasures: if even a single digit x_i is not correctly received, it is impossible to recover m. Conversely, the toy example allows perfect reception of m, provided that at least $\frac{r}{2}$ of the r copies of each digit x_i are correctly received. Finally, it is worth noticing that both the general scheme and the toy example can be easily extended to any message of length $k > 1$, and they still guarantee perfect secrecy whenever it is true that the adversary can eavesdrop no more than $j-1$ noiseless digits for each codeword sent.

4.5 Future Directions

This chapter proposes, for the first time in the literature, a constructive solution that combine secret sharing and linear error-correcting codes to overcome the presence of transmission errors, while guaranteeing perfect security on the generalized Ozarow-Wyner's wire-tap channel model. This work significantly deviates from the research trend in the area. In fact, most of prior work was focusing on the secrecy capacity, studying limiting behaviours of the model when the size of the message approaches infinity. Taking a different approach, it was possible to show that reliable and perfectly secret data transmission is possible in practice, at the cost of a (slightly) lower communication rate.

While the proposed formalization and theoretical contributions stand on their own, they have also a wealth of practical applications, for instance in contexts where the amount of transmitted data is limited, or where key management and costly cryptographic algorithms are hard to implement — such as in many distributed and unattended application settings — or, finally, where perfect security is at premium. As for possible future directions, the most promising is a thorough analysis of the performance of different type of ECCs. Indeed, when adopting the proposed scheme the major constrain is the reduced communication rate and finding which are the codes better suited for optimizing the performance would be of utmost importance in practical applications. Other possible research lines, include the study of different channel models that might be more appropriate for specific practical applications or an experimental study for evaluating the impact of light-weighted security solutions in daily life applications.

CHAPTER 5

Conclusion

This thesis covered various topics of data science proposing innovative solutions for common issues that arise from data management of large datasets: storage, analytics, and secure transmission of data. For each one of these macro-problems, an algorithm for helping at optimizing available computing and memory resources is proposed, evaluated both theoretically and experimentally, and compared with other methods currently adopted in practical applications. In detail, these algorithms are: *(i)* CUTSIZE, a bitmap compression scheme for data storage that mainly serves to reduce space resources; *(ii)* EXPEDITE, a very time efficient algorithm for data analysis that discovers frequent patterns within data; and, *(iii)* CRYPTOLESS, an innovate solution for transmitting data that guarantees integrity and confidentiality of the communication applicable on resource constrained devices. In their respective fields of research, each one of the three proposed inventions is a relevant contribution on its own. In fact, at the time of writing, they represent (some of) the best solutions available in the literature in terms of space and time performance.

As for further work, several research lines are practicable. In both fields of data storage and data analysis, promising research directions lie in the development of implementations of the proposed algorithms, namely CUTSIZE and EXPEDITE, for supporting parallel and distributed computing. In fact, over the last years, the most relevant improvements in there area were achieved by parallel implementations of older algorithms or by adapting them to platforms for distributed computing. The two algorithms proposed in this thesis are parallelizable with relatively little effort: EXPEDITE is in many ways similar to DCI-CLOSED for which a version for parallel computing already exists, and CUTSIZE works with aligned data making it ideal for parallel computing. Instead, in the field of secure and light-weighted data transmission, promising research directions lie in achieving results similar to those achieved in this the-

sis, but using different communication channel models. In fact, CRYPTOLESS guarantees security in the GOW model, and being able to guarantee similar security properties in different models could play a main role in the spread on physical layer security applications.

Bibliography

[1] Charu C. Aggarwal. Towards long pattern generation in dense databases. *SIGKDD Explorations*, 3(1):20–26, 2001.

[2] Charu C. Aggarwal and Jiawei Han. *Frequent Pattern Mining*. Springer International Publishing, 2014.

[3] Rakesh Agrawal, Tomasz Imielinski, and Arun Swami. Mining association rules between sets of items in large databases. In *SIGMOD '93: Proceedings of the 1993 ACM SIGMOD international conference on Management of data*, pages 207–216. ACM Press, 1993.

[4] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB '94, pages 487–499, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc.

[5] Giulio Aliberti, Alessandro Colantonio, and Roberto Di Pietro. CUTSIZE: a Bitmap Compression Scheme for Fast Bitwise Operations. *Technical Report*, 2016.

[6] Giulio Aliberti, Alessandro Colantonio, Roberto Di Pietro, and Riccardo Mariani. EXPEDITE: EXPress closED ITemset Enumeration. *Expert Syst. Appl.*, 42(8):3933–3944, 2015.

[7] Giulio Aliberti, Stefano Guarino, and Roberto Di Pietro. CRYPTOLESS: Reliable and Perfectly Secret Communication over the Generalized Ozarow-Wyner's Wire-Tap Channel. *Submitted to Computer Networks*, 2016.

[8] G. Antoshenkov. Byte aligned data compression, November 8 1994. US Patent 5,363,098.

[9] G. Antoshenkov. Byte-aligned bitmap compression. In *Proceedings of the Conference on Data Compression*, DCC '95, page 476, Washington, DC, USA, 1995. IEEE Computer Society.

[10] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The Internet of Things: A survey. *Computer Networks*, 54(15):2787–2805, oct 2010.

[11] Ashar Baig. Rethinking the enterprise data archive for big data analytics and regulatory compliance. http://rainstor.com/2013_new/wp-content/uploads/2014/11/WP_Gigaom_Rethinking_the_Enterprise_Data-Archive.pdf, 2014. [Online; accessed 24-February-2016].

[12] Marco Baldi, Giacomo Ricciutelli, Nicola Maturo, and Franco Chiaraluce. Performance assessment and design of finite length LDPC codes for the gaussian wiretap channel. *CoRR*, abs/1506.01966, 2015.

[13] Meryem Benammar and Pablo Piantanida. Secrecy capacity region of some classes of wiretap broadcast channels. *CoRR*, abs/1407.5572, 2014.

[14] G.R. Blakley. Safeguarding cryptographic keys. In *Proceedings of the 1979 AFIPS National Computer Conference*, pages 313–317, Monval, NJ, USA, 1979. AFIPS Press.

[15] M. Bloch and J. Barros. *Physical-Layer Security: From Information Theory to Security Engineering*. Cambridge University Press, 2011.

[16] Holger Boche, Rafael F. Schaefer, and H. Vincent Poor. On the continuity of the secrecy capacity of compound and arbitrarily varying wiretap channels. *CoRR*, abs/1409.4752, 2014.

[17] Francesco Bonchi and Claudio Lucchese. Extending the state-of-the-art of constraint-based pattern discovery. *Data & Knowledge Engineering*, 60(2):377–399, 2007.

[18] Christian Borgelt. Frequent item set mining. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2(6):437–456, 2012.

[19] Jean-François Boulicaut, Artur Bykowski, and Christophe Rigotti. Free-sets: a condensed representation of boolean data for the approximation of frequency queries. *Data Mining and Knowledge Discovery*, 7(1):5–22, 2003.

[20] J.J. Boutros, V. Dedeoglu, and M. Bloch. *The Anti-Diversity Concept for Secure Communication on a Two-Link Compound Channel*. ETH-Zürich, 2014.

[21] Artur Bykowski and Christophe Rigotti. A condensed representation to find frequent patterns. In *Proceedings of the Twentieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '01, pages 267–273, New York, NY, USA, 2001. ACM.

[22] Toon Calders, Christophe Rigotti, and Jean-François Boulicaut. A survey on condensed representations for frequent sets. In *Constraint-Based Mining and Inductive Databases*, volume 3848 of *Lecture Notes in Computer Science*, pages 64–80. Springer, 2006.

[23] Zhen Chen, Yuhao Wen, Junwei Cao, Wenxun Zheng, Jiahui Chang, Yinjun Wu, Ge Ma, Mourad Hakmaoui, and Guodong Peng. A survey of bitmap index compression algorithms for big data. *Tsinghua Science and Technology*, 20(1):100–115, Feb 2015.

[24] Fan Cheng, Raymond W Yeung, and Kenneth W Shum. Imperfect secrecy in wiretap channel ii. *Information Theory, IEEE Transactions on*, 61(1):628–636, 2015.

[25] M. Cheraghchi, F. Didier, and A. Shokrollahi. Invertible extractors and wiretap protocols. *Information Theory, IEEE Transactions on*, 58(2):1254–1274, Feb 2012.

[26] Tom Clark. *Storage Virtualization: Technologies for Simplifying Data Storage and Management*. Addison-Wesley Professional, 2005.

[27] Alessandro Colantonio, Roberto Di Pietro, Alberto Ocello, and Nino Vincenzo Verde. A new role mining framework to elicit business roles and to mitigate enterprise risk. *Decision Support Systems*, 50(4):715–731, 2011.

[28] Alessandro Colantonio and Roberto Di Pietro. Concise: Compressed 'n' composable integer set. *Inf. Process. Lett.*, 110(16):644–650, 2010.

[29] Alessandro Colantonio, Roberto Di Pietro, Alberto Ocello, and Nino Vincenzo Verde. Visual role mining: A picture is worth a thousand roles. *IEEE Transactions on Knowledge and Data Engineering*, 24(6):1120–1133, 2012.

[30] Burleson Consulting. Oracle bitmap index techniques. http://www.dba-oracle.com/oracle_tips_bitmapped_indexes.htm/, 2014. [Online; accessed 24-February-2016].

[31] IBM Corporation. Storage catches up to the data explosion. http://www.ibm.com/midmarket/us/en/att/pdf/Feat_2_WOI.pdf?ca=fv1306&me=feature2&re=usartpdf?, 2013. [Online; accessed 24-February-2016].

[32] Fabian Corrales, David Chiu, and Jason Sawin. Variable length compression for bitmap indices. In Abdelkader Hameurlain, Stephen W. Liddle, Klaus-Dieter Schewe, and Xiaofang Zhou, editors, *DEXA (2)*, volume 6861 of *Lecture Notes in Computer Science*, pages 381–395. Springer, 2011.

[33] Imre Csiszár and János Körner. Broadcast channels with confidential messages. *IEEE Transactions on Information Theory*, 24(3):339–348, 1978.

[34] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. In *OSDI*, pages 137–150. google labs, 2004.

[35] François Deliège and Torben Bach Pedersen. Position list word aligned hybrid: optimizing space and performance for compressed bitmaps. In Ioana Manolescu, Stefano Spaccapietra, Jens Teubner, Masaru Kitsuregawa, Alain Lèger, Felix Naumann, Anastasia Ailamaki, and Fatma Özcan, editors, *EDBT*, volume 426 of *ACM International Conference Proceeding Series*, pages 228–239. ACM, 2010.

[36] Zhi-Hong Deng. Fast mining top-rank-k frequent patterns by using node-lists. *Expert Systems with Applications*, 41(4, Part 2):1763 – 1768, 2014.

[37] Yevgeniy Dodis, Amit Sahai, and Adam Smith. On perfect and adaptive security in exposure-resilient cryptography. In Birgit Pfitzmann, editor, *EUROCRYPT*, volume 2045 of *Lecture Notes in Computer Science*, pages 301–324. Springer, 2001.

[38] Alina Ene, William Horne, Nikola Milosavljevic, Prasad Rao, Robert Schreiber, and Robert E. Tarjan. Fast exact and heuristic methods for role minimization problems. In *Proceedings of the 13th ACM Symposium on Access Control Models and Technologies*, SACMAT '08, pages 1–10, New York, NY, USA, 2008. ACM.

[39] Guo-Dong Fang and Zhi-Hong Deng. Vtk: Vertical mining of top-rank-k frequent patterns. In *FSKD (2)*, pages 620–624. IEEE Computer Society, 2008.

[40] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. From data mining to knowledge discovery in databases. *Ai Magazine*, 17:37–54, 1996.

[41] FIMI. FIMI '04, proceedings of the ieee icdm workshop on frequent itemset mining implementations, brighton, uk, november 1, 2004. In Roberto J. Bayardo Jr., Bart Goethals, and Mohammed Javeed Zaki, editors, *FIMI*, volume 126 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2004.

[42] Francesco Fusco, Marc Ph. Stoecklin, and Michail Vlachos. Net-fli: On-the-fly compression, archiving and indexing of streaming network traffic. *Proc. VLDB Endow.*, 3(1-2):1382–1393, September 2010.

[43] Solomon W. Golomb. Run-length encodings. In *IEEE Transactions on Information Theory*, pages 399–401, 1966.

[44] Gosta Grahne and Jianfei Zhu. Fast algorithms for frequent itemset mining using FP-Trees. *IEEE Transactions on Knowledge and Data Engineering*, 17(10):1347–1362, 2005.

[45] Talha Cihad Gulcu and Alexander Barg. Achieving secrecy capacity of the wiretap channel and broadcast channel with a confidential component. *CoRR*, abs/1410.3422, 2014.

[46] G. Guzun, G. Canahuate, D. Chiu, and J. Sawin. A tunable compression framework for bitmap indices. In *Proceedings of the IEEE International Conference on Data Engineering*, 2014.

[47] Jiawei Han, Hong Cheng, Dong Xin, and Xifeng Yan. Frequent pattern mining: Current status and future directions. *Data Mining and Knowledge Discovery*, 15(1):55–86, 2007.

[48] Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, SIGMOD '00, pages 1–12, New York, NY, USA, 2000. ACM.

[49] Jiawei Han, Jianyong Wang, Ying Lu, and Petre Tzvetkov. Mining top-k frequent closed patterns without minimum support. In *International Conference on Data Mining*, pages 211–218. IEEE Computer Society, 2002.

[50] Te Sun Han, H. Endo, and M. Sasaki. Reliability and secrecy functions of the wiretap channel under cost constraint. *Information Theory, IEEE Transactions on*, 60(11):6819–6843, Nov 2014.

[51] Willie K. Harrison, João Almeida, Matthieu R. Bloch, Steven W. McLaughlin, and João Barros. Coding for secrecy: An overview of error-control coding techniques for physical-layer security. *IEEE Signal Process. Mag.*, 30(5):41–50, 2013.

[52] W.K. Harrison and S.W. McLaughlin. Physical-layer security: Combining error control coding and cryptography. In *Communications, 2009. ICC '09. IEEE International Conference on*, pages 1–5, June 2009.

[53] Jochen Hipp, Ulrich Güntzer, and Gholamreza Nakhaeizadeh. Algorithms for association rule mining – general survey and comparison. *SIGKDD Explorations*, 2(1):58–64, 2000.

[54] Quyen Huynh-Thi-Le, Tuong Le, Bay Vo, and Bac Le. An efficient and effective algorithm for mining top-rank-k frequent patterns. *Expert Systems with Applications*, 42(1):156 – 164, 2015.

[55] Owen Kaser and Daniel Lemire. Strongly universal string hashing is fast. *CoRR*, abs/1202.4961, 2012.

[56] Owen Kaser and Daniel Lemire. Compressed bitmap indexes: beyond unions and intersections. *CoRR*, abs/1402.4466, 2014.

[57] András Király, Asta Laiho, János Abonyi, and Attila Gyenesei. Novel techniques and an efficient algorithm for closed pattern mining. *Expert Systems with Applications*, 41(11):5105 – 5114, 2014.

[58] Demijan Klinc, Jeongseok Ha, Steven W. McLaughlin, JoÃ£o Barros, and Byung-Jae Kwak. LDPC codes for the gaussian wiretap channel. *IEEE Transactions on Information Forensics and Security*, 6(3-1):532–540, 2011.

[59] Marzena Kryszkiewicz. Concise representations of association rules. In *Pattern Detection and Discovery*, pages 92–109, 2002.

[60] Daniel Lemire, Owen Kaser, and Kamel Aouiche. Sorting improves word-aligned bitmap indexes. *CoRR*, abs/0901.3751, 2009.

[61] S. Leung-Yan-Cheong and M.E. Hellman. The gaussian wire-tap channel. *Information Theory, IEEE Transactions on*, 24(4):451–456, Jul 1978.

[62] Yingbin Liang and H. Vincent Poor. Generalized multiple access channels with confidential messages. *CoRR*, abs/cs/0605014, 2006.

[63] Guimei Liu, Hongjun Lu, Jeffrey Xu Yu, Wei Wang 0011, and Xiangye Xiao. AFOPT: An efficient implementation of pattern growth approach. In *FIMI*, volume 90 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2003.

[64] Ling Liu, Yanfei Yan, and Cong Ling. Achieving secrecy capacity of the gaussian wiretap channel with polar lattices. *CoRR*, abs/1503.02313, 2015.

[65] C. Lucchese, S. Orlando, and R. Perego. Fast and memory efficient mining of frequent closed itemsets. *IEEE Transactions on Knowledge and Data Engineering*, 18(1):21–36, 2006.

[66] H. Mahdavifar and A. Vardy. Achieving the secrecy capacity of wiretap channels using polar codes. *Information Theory, IEEE Transactions on*, 57(10):6428–6443, Oct 2011.

[67] Ueli Maurer and Stefan Wolf. Information-theoretic key agreement: From weak to strong secrecy for free. In Bart Preneel, editor, *Advances in Cryptology - EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 351–368. Springer Berlin Heidelberg, 2000.

[68] UeliM. Maurer. The strong secret key rate of discrete random triples. In Richard E. Blahut, Jr. Costello, Daniel J., Ueli Maurer, and Thomas Mittelholzer, editors, *Communications and Cryptography*, volume 276 of *The Springer International Series in Engineering and Computer Science*, pages 271–285. Springer US, 1994.

[69] Sandy Moens, Emin Aksehirli, and Bart Goethals. Frequent itemset mining for big data. In *BigData Conference*, pages 111–118. IEEE Computer Society, 2013.

[70] H. D. K. Moonesinghe, Samah Jamal Fodeh, and Pang-Ning Tan. Frequent closed itemset mining using prefix graphs with an efficient flow-based pruning strategy. In *International Conference on Data Mining*, pages 426–435. IEEE Computer Society, 2006.

[71] Mohamed Nafea and Aylin Yener. Wiretap channel ii with a noisy main channel. In *Information Theory (ISIT), 2015 IEEE International Symposium on*, pages 1159–1163. IEEE, 2015.

[72] Benjamin Negrevergne, Alexandre Termier, Jean-François Méhaut, and Takeaki Uno. Discovering closed frequent itemsets on multicore: Parallelizing computations and optimizing memory accesses. In *Interna-*

tional Symposium on High Performance Computing Systems, pages 521–528. IEEE Computer Society, 2010.

[73] Benjamin Negrevergne, Alexandre Termier, Marie-Christine Rousset, and Jean-François Méhaut. Para miner: a generic pattern mining algorithm for multi-core architectures. *Data Mining and Knowledge Discovery*, 28(3):593–633, 2014.

[74] L.H. Ozarow and A.D. Wyner. Wire-tap channel II. In Thomas Beth, Norbert Cot, and Ingemar Ingemarsson, editors, *Advances in Cryptology*, volume 209 of *Lecture Notes in Computer Science*, pages 33–50. Springer Berlin Heidelberg, 1985.

[75] Nicolas Pasquier, Yves Bastide, Rafik Taouil, and Lotfi Lakhal. Efficient mining of association rules using closed itemset lattices. *Information Systems*, 24(1):25 – 46, 1999.

[76] T.B. Pedersen and F. Deliege. Compression of bitmaps and values, January 17 2013. US Patent App. 13/389,399.

[77] Jian Pei, Jiawei Han, and Runying Mao. CLOSET: An Efficient Algorithm for Mining Frequent Closed Itemsets. In *ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, pages 21–30, 2000.

[78] Lenka Piskovà and Tomàs Horvàth. Comparing performance of formal concept analysis and closed frequent itemset mining algorithms on real data. In *Concept Lattices and Their Applications*, volume 1062 of *CEUR Workshop Proceedings*, pages 299–304. CEUR-WS.org, 2013.

[79] Foster Provost and Tom Fawcett. Data Science and its Relationship to Big Data and Data-Driven Decision Making. *Big Data*, 1(1):51–59, March 2013.

[80] Jason K. Resch and James S. Plank. Aont-rs: Blending security and performance in dispersed storage systems. In *Proceedings of the 9th USENIX Conference on File and Stroage Technologies*, FAST'11, pages 14–14, Berkeley, CA, USA, 2011. USENIX Association.

[81] Z. Rezki, A. Khisti, and M.-S. Alouini. On the secrecy capacity of the wiretap channel with imperfect main channel estimation. *Communications, IEEE Transactions on*, 62(10):3652–3664, Oct 2014.

[82] Ronald L. Rivest. All-or-nothing encryption and the package transform. In Eli Biham, editor, *FSE*, volume 1267 of *Lecture Notes in Computer Science*, pages 210–218. Springer, 1997.

[83] Manish Saraswat. 13 amazing applications / uses of data science today. http://www.analyticsvidhya.com/blog/2015/09/applications-data-science/, 2015. [Online; accessed 24-February-2016].

[84] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, November 1979.

[85] Maryam Shekofteh. A survey of algorithms in fcim. In *Data Storage and Data Engineering*, pages 29–33. IEEE Computer Society, 2010.

[86] Ningthoujam Gourakishwar Singh, Sanasam Ranbir Singh, and Anjana K. Mahanta. Closeminer: Discovering frequent closed itemsets using frequent closed tidsets. In *International Conference on Data Mining*, pages 633–636. IEEE Computer Society, 2005.

[87] Israel Spiegler and Rafi Maayan. Storage and retrieval considerations of binary data bases. *Inf. Process. Manage.*, 21(3):233–254, 1985.

[88] Gerd Stumme, Rafik Taouil, Yves Bastide, Nicolas Pasquier, and Lotfi Lakhal. Computing iceberg concept lattices with titanic. *Data & Knowledge Engineering*, 42(2):189–222, 2002.

[89] Arunkumar Subramanian, Andrew Thangaraj, Matthieu R. Bloch, and Steven W. McLaughlin. Strong secrecy on the binary erasure wiretap channel using large-girth LDPC codes. *IEEE Transactions on Information Forensics and Security*, 6(3-1):585–594, 2011.

[90] Andrew Thangaraj, Souvik Dihidar, A. Robert Calderbank, Steven W. McLaughlin, and Jean-Marc Merolla. Applications of LDPC codes to the wiretap channel. *IEEE Transactions on Information Theory*, 53(8):2933–2945, 2007.

[91] Takeaki Uno, Tatsuya Asai, Yuzo Uchida, and Hiroki Arimura. An efficient algorithm for enumerating closed patterns in transaction databases. In *Discovery Science*, volume 3245 of *Lecture Notes in Computer Science*, pages 16–31. Springer, 2004.

[92] Takeaki Uno, Masashi Kiyomi, and Hiroki Arimura. Lcm ver. 2: Efficient mining algorithms for frequent/closed/maximal itemsets. In *FIMI*, volume 126 of *CEUR Workshop Proceedings*, 2004.

[93] Takeaki Uno, Masashi Kiyomi, and Hiroki Arimura. Lcm ver.3: Collaboration of array, bitmap and prefix tree for frequent itemset mining. In *Proceedings of the 1st International Workshop on Open Source Data Mining: Frequent Pattern Mining Implementations*, pages 77–86, New York, NY, USA, 2005. ACM.

[94] Bay Vo, Tzung-Pei Hong, and Bac Le. Dbv-miner: A dynamic bit-vector approach for fast mining frequent closed itemsets. *Expert Systems with Applications*, 39(8):7196–7206, June 2012.

[95] Harald Vogt. Efficient object identification with passive RFID tags. In *Proceedings of the First International Conference on Pervasive Computing*, volume 2414 of *Lecture Notes in Computer Science*, pages 98–113, Zurich, August 2002. Springer-Verlag.

[96] Mikhail Vorontsov. Java Performance Tuning Guide. http://java-performance.info/, 2015. [Online; accessed 24-February-2016].

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

[98] V.K. Wei. Generalized hamming weights for linear codes. *Information Theory, IEEE Transactions on*, 37(5):1412–1418, Sep 1991.

[99] Chan Wong Wong, Tan F. Wong, and John M. Shea. LDPC code design for the bpsk-constrained gaussian wiretap channel. *CoRR*, abs/1103.3915, 2011.

[100] Harry K. T. Wong, Hsiu-Fen Liu, Frank Olken, Doron Rotem, and Linda Wong. Bit transposed files. In *Proceedings of the 11th International Conference on Very Large Data Bases - Volume 11*, VLDB '85, pages 448–457. VLDB Endowment, 1985.

[101] R. Wrembel. *Data Warehouses and OLAP: Concepts, Architectures and Solutions: Concepts, Architectures and Solutions*. Gale virtual reference library. IRM Press, 2006.

[102] K. Wu, S. Ahern, E. W. Bethel, J. Chen, H. Childs, E. Cormier-Michel, C. Geddes, J. Gu, H. Hagen, B. Hamann, W .Koegler, J. Lauret, J. Meredith, P. Messmer, E. Otoo, V. Perevoztchikov, A. Poskanzer, Prabhat, O. Rübel, A. Shoshani, A. Sim, K. Stockinger, G. Weber, and W.-M.

Zhang. Fastbit: interactively searching massive data. *Journal of Physics: Conference Series*, 180(1):012053, 2009.

[103] K. Wu, A. Shoshani, and E. Otoo. Word aligned bitmap compression method, data structure, and apparatus, December 14 2004. US Patent 6,831,575.

[104] Kesheng Wu, Ekow J. Otoo, and Arie Shoshani. Compressing bitmap indexes for faster search operations. In *SSDBM*, pages 99–108. IEEE Computer Society, 2002.

[105] Kesheng Wu, Ekow J. Otoo, and Arie Shoshani. Optimizing bitmap indices with efficient compression. *ACM Trans. Database Syst.*, 31(1):1–38, March 2006.

[106] Aaron D. Wyner. The Wire-tap Channel. *Bell Systems Technical Journal*, 54(8):1355–1387, January 1975.

[107] S. Ben Yahia, T. Hamrouni, and E. Mephu Nguifo. Frequent closed itemset based algorithms: A thorough structural and analytical survey. *SIGKDD Explorations*, 8:93–104, 2006.

[108] Guizhen Yang. The complexity of mining maximal frequent itemsets and maximal frequent patterns. In *Proceedings of the tenth ACM SIGKDD International Conference on Knowledge Discovery and Data mining*, pages 344–353. ACM Press, 2004.

[109] Mohammed J. Zaki and Karam Gouda. Fast vertical mining using diffsets. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 326–335, 2003.

[110] Mohammed J. Zaki and Ching-Jui Hsiao. Efficient algorithms for mining closed itemsets and their lattice structure. *IEEE Transactions on Knowledge and Data Engineering*, 17(4):462–478, 2005.

[111] Mohammed Javeed Zaki, Srinivasan Parthasarathy, Mitsunori Ogihara, and Wei Li. Parallel algorithms for discovery of association rules. *Data Mining and Knowledge Discovery*, 1(4):343–373, 1997.

[112] Yulong Zou, Jia Zhu, Xianbin Wang, and V. Leung. Improving physical-layer security in wireless communications using diversity techniques. *Network, IEEE*, 29(1):42–48, Jan 2015.