VIETNAM NATIONAL UNIVERSITY, HANOI
UNIVERSITY OF ENGINEERING AND TECHNOLOGY

Le Tuan Anh

OPTIMIZE CNF ENCODING FOR ITEMSET MINING TASKS

Major: Computer Science

HA NOI - 2024

VIETNAM NATIONAL UNIVERSITY, HANOI
UNIVERSITY OF ENGINEERING AND TECHNOLOGY

Le Tuan Anh

OPTIMIZE CNF ENCODING FOR ITEMSET MINING TASKS

Major: Computer Science

Supervisor: Dr. To Van Khanh

HA NOI - 2024

ABSTRACT

Summary: In this thesis, we apply the ”Sequential Encounter Encoding” method to optimize itemset mining tasks. This method has been proven effective in optimizing the search process for itemsets in data. By utilizing ”Sequential Encounter Encoding” we aim to enhance the performance of itemset mining algorithms while minimizing processing time.

We conduct a series of experiments on real-world datasets to evaluate the performance of the applied method. Experimental results demonstrate a significant improvement in accuracy and efficiency compared to traditional methods. The application of this method not only enhances the performance of data mining processes but also opens up potential applications for similar problems in the field of data science and information retrieval.

Keywords: *SAT, SAT Encoding, Sequential Encounter Encoding, Itemset Mining Tasks*

ACKNOWLEDGEMENT

First of all, I would like to express my deepest gratitude to Lecturer, Dr. To Van Khanh, who wholeheartedly guided, guided, encouraged, and helped me throughout the course of this thesis.

I would like to thank the teachers in the Faculty of Information Technology as well as the University of Engineering and Technology - Vietnam National University, Hanoi, for creating conditions for me to study in a good environment and for teaching me the best things to do. Knowledge is especially important for me to continue to study and work in the future.

I especially express my gratitude to my family who have always been a solid support to help and support me in every journey. Finally, I would also like to thank the members of the K65CA-CLC2 class, my friends in the course, and the siblings inside and outside the school who have been with me to study and practice and help each other throughout four years university.

Thank you sincerely!

Le Tuan Anh

AUTHORSHIP

I hereby declare that the thesis ”OPTIMIZE CNF ENCODING FOR ITEMSET MINING TASKS” is done by me and has never been submitted as a report for Graduation Thesis at University of Engineering and Technology - Vietnam National University, Hanoi, or any other university. All content in this thesis is written by me and has not been copied from any source, nor is the work of others used without specific citation. I also warrant that the source code is my development and does not copy the source code of any other person. If wrong, I would like to take full responsibility according to the regulations of University of Engineering and Technology - Vietnam National University, Hanoi.

Ha Noi, May 26 2024

Student

Le Tuan Anh

SUPERVISOR'S APPROVAL

I hereby approve that the thesis in its current form is ready for committee examination as a requirement for the Bachelor of Computer Science degree at the University of Engineering and Technology.

Ha Noi, May 26 2024

Supervisor

Dr. To Van Khanh

Contents

List of Figures

List of Tables

ABBREVIATIONS

FIM	Frequent Itemset Mining
SEE	Sequential Encounter Encoding
SAT	Satisfiability
UNSAT	Unsatisfiability
CNF	Conjunctive Normal Form
ALO	At Least One
AMO	At Most One
ALK	At Least k
AMK	At Most k
FIMI	Frequent Itemset Mining Dataset Repository
CP4IM	Constraint Programming for Itemset Mining

PREFACE

Itemset mining, a fundamental task in data mining, plays a pivotal role in discovering meaningful patterns from large datasets. It involves identifying sets of items that frequently co-occur together within transactions, providing valuable insights into associations and correlations among items.

In this thesis, we delve into the realm of itemset mining and its crucial importance in various domains such as market basket analysis, bioinformatics, and web usage mining. We explore the significance of itemset mining in uncovering hidden patterns, aiding decision-making processes, and enhancing business strategies.

Furthermore, we aim to optimize the itemset mining process by leveraging the Sequential Encounter Encoding method for SAT encoding. This innovative approach offers a novel perspective on encoding itemset mining problems into Boolean satisfiability (SAT) instances, paving the way for efficient and scalable solutions.

The subsequent chapters of this thesis are structured as follows:

Chapter 1: This chapter serves as an introduction to itemset mining tasks and SAT encoding. It elucidates the fundamental concepts, terminology, and real-world applications associated with these fields.

Chapter 2: We delve into the construction of base constraints for itemset mining problems and their encoding into SAT. This includes an exploration of the process of deriving constraints from standard itemset mining algorithms, along with an analysis of their limitations.

Chapter 3: Here, we discuss the Sequential Encounter Encoding method as a novel approach to encoding itemset mining problems into SAT instances. We delve into the intricacies of this encoding technique and highlight its advantages over traditional methods.

Chapter 4: This chapter presents the results of experimental evaluations conducted on both synthetic and real-world datasets. We compare the performance of the Sequential Encounter Encoding method with existing approaches to showcase its efficacy and scalability.

Chapter 5: Finally, we conclude our findings, summarizing the contributions of this research and discussing potential avenues for future exploration in the field of itemset mining and SAT encoding.

Through this thesis, we aim to contribute to the advancement of itemset mining techniques and facilitate the development of more efficient algorithms for pattern discovery in large datasets.

CHAPTER 1

Introduction

This chapter will focuses on introducing the itemset mining tasks and SAT encoding, encompassing their concepts, related terms, and applications.

1.1 Frequent Itemset Mining

Data mining[**survey˙itemset˙mining**] is concerned with either forecasting future trends or deciphering past events. Techniques used for predicting the future, such as neural networks, often function as black-box models because the primary objective is to achieve the highest possible accuracy rather than explainability. On the other hand, various data mining methods aim to uncover patterns in data that are straightforward for humans to interpret.

These methods of pattern discovery can be categorized based on the specific types of patterns they identify, including clusters, itemsets, trends, and outliers. For example, clusters group similar data points together, itemsets identify common associations or groupings in data, trends reveal changes or movements over time, and outliers pinpoint unusual or unexpected data points.

This paper provides a survey that focuses specifically on the discovery of itemsets within databases. Itemset discovery is a popular data mining task, especially when analyzing symbolic data, as it can provide valuable insights into associations and relationships within datasets.

The concept of discovering itemsets in databases was introduced in 1993 by Agrawal and Srikant under the term large itemset mining, which is now known as frequent itemset mining (FIM). The objective of FIM is to identify groups of items (itemsets) that often occur together within customer transactions.

For example, analyzing a customer transaction database may reveal that many customers purchase taco shells along with peppers. Recognizing these associations between items helps to shed light on customer behavior. This knowledge can be invaluable for retail managers, as it enables them to make strategic marketing decisions such as promoting products together or positioning them closer on store shelves. Such strategies can lead to enhanced customer experiences and potentially increased sales.

Frequent itemset mining (FIM) was initially proposed as a method for analyzing customer transaction data, but it has since evolved into a general data mining task that is applicable across various domains. In broader terms, a customer transaction database can be seen as a collection of instances representing objects (transactions), with each object characterized by nominal attribute values (items). As such, FIM can also be understood as the process of identifying attribute values that commonly occur together in a database.

Given that many data types can be represented in the form of transaction databases, FIM finds applications across a diverse range of fields. These include bioinformatics, image classification, network traffic analysis, customer review analysis, activity monitoring, malware detection, and e-learning, among others.

FIM has also been extended in numerous ways to cater to specific requirements and challenges within these domains. For example, extensions of FIM have been developed to discover rare patterns, correlated patterns, patterns in sequences and graphs, and patterns that yield high profit. These adaptations expand the applicability of FIM and demonstrate its versatility and relevance across different areas of data mining.

1.1.1 Concepts

Frequent item sets are a key technique in the realm of data mining[**fim˙geeksforgeeks**], specifically aimed at uncovering relationships among different items. The essence of association rule mining lies in identifying those item relationships that occur frequently together in the dataset.

In simpler terms, imagine a ”frequent itemset” as a group of items that tend to show up in unison across various data entries. We utilize a specific measure called 'support count' to gauge the regularity of these itemset occurrences. The support count essentially quantifies the number of times a particular combination of items appears within the dataset's entries or transactions.

The practical aim here is to pinpoint those itemsets that reach or surpass a predetermined threshold of occurrence, known as the minimum support. By identifying these itemsets, we can infer patterns of frequency within the dataset's transactions or records.

Table 1.1: Example of a dataset of transactions from a retail store

Tid	**Itemsets**
1	apple, banana, cherry
2	apple, mango
3	apple, cherry
4	mango, cherry
5	apple, mango, cherry

In paper A Survey of Itemset Mining[**survey˙itemset˙mining**], the authors have provided a comprehensive survey of frequent itemset mining algorithms. The problem of frequent itemset mining is formally defined as follows. Let there be a set of items (symbols) $I = i_1, i_2, ...i_m$. A transaction database $D = T_1, T_2, ..., T_n$ is a set of transactions such that each transactions $T_q \subseteq I (1 \leq q \leq m)$ is a set of distinct items, and each transaction T_q has a unique identifier q called its TID (Transaction IDentifier). For example, consider the transaction database shown in Table 1.1. This database contains five transactions, represents items bought by customers. For example, the first transaction T_1 represents a customer that has bought the item apple, banana and cherry.

An itemset X is a set of items such that $X \subseteq I$. Let the notation $|X|$ denote the set cardinality or, in other words, the number of items in an itemset X. An itemset X is said to be of length k or a k-itemset if it contains k items ($|X| = k$). The goal of itemset mining is to discover interesting itemsets in a transaction database, that is interesting associations between items. In general, in itemset mining, various measures can be used to assess the interestingness of patterns. In FIM, the interestingness of a given itemset is traditionally defined using a measure called the support. The support (or absolute support) of an itemset X in a database D is denoted as $Supp(X)$ and defined as the number of transactions containing X, that is $Supp(X) = |\{T | X \subseteq T \wedge T \in D\}|$. For example, the support of the itemset

$\{apple, mango\}$ is 2 because this itemset appears in two transactions (T_2 and T_5). Note that some authors prefer to define the support of an itemset X as a ratio. This definition called the relative support is $RelSup(X) = Supp(X)/|D|$. For example, the relative support of the itemset $\{apple, mango\}$ is 0.4.

The task of frequent itemset mining consists of discovering all frequent itemsets in a given transaction database. An itemset X is frequent if it has a support that is no less than a given minimum support threshold minsup set by the user (i.e. $Supp(X) \geq minsup$). For example, if we consider the database shown in Table 1.1 and that the user has set $minsup = 2$, the task of FIM is to discover all groups of items appearing in at least two transactions. In this case, there are exactly 6 frequent itemsets, where the number besides each itemset indicates its support:

- {apple}: 4 (appears in transactions T_1, T_2, T_3 and T_5)
- {mango}: 3 (appears in transactions T_2, T_4 and T_5)
- {cherry}: 4 (appears in transactions T_1, T_3, T_4 and T_5)
- {apple, mango}: 2 (appears in transactions T_2 and T_5)
- {apple, cherry}: 3 (appears in transactions T_1, T_3 and T_5)
- {mango, cherry}: 2 (appears in transactions T_4 and T_5)

1.1.2 Applications

Frequent itemset mining is a powerful analytic process used to examine the relationships between items in large datasets. Taking a practical example from the commercial realm, let's picture a supermarket setting.

Through the lens of frequent itemset mining, a supermarket can sift through transactional data to identify combinations of items that customers tend to purchase together regularly. This type of analysis digs deeper than observing mere coincidental purchases; it uncovers patterns that reflect a certain predictability and frequency in customer buying behavior.

For example, a pattern where bread and milk are often purchased together reflects a habitual buying behavior rather than a sporadic trend. These insights are invaluable for retailers, as they allow them to make informed decisions across various aspects of their operations.

Here's how these insights translate into real-world advantages:

Inventory Management: By understanding which itemsets are popular, retailers can better manage their inventory, ensuring that these items are always in stock and accessible to customers. This proactive approach helps avoid stock shortages and enhances the overall shopping experience.

Recommendation Systems: Retailers can implement systems that use frequent itemset data to recommend additional products to customers. For instance, if a customer selects pasta, the system might suggest accompanying it with pasta sauce and grated cheese, based on observed buying patterns. This can lead to greater customer satisfaction and increased sales.

Targeted Marketing: The knowledge of which items are often bought together allows retailers to tailor their marketing efforts. Promotions can be strategized to bundle popular itemsets, attracting customers and encouraging them to buy more.

In essence, frequent itemset mining is a strategic tool in the business intelligence arsenal. It empowers businesses with deep insights into consumer purchasing trends, facilitating data-driven strategies that foster growth and enhance customer engagement.

1.1.3 Challenges

FIM is an enumeration problem. The goal is to enumerate all patterns that meet the minimum support constraint specified by the user. Thus, there is always a single correct answer to a FIM task. FIM is a difficult problem. The naive approach to solve this problem is to consider all possible itemsets to then output only those meeting the minimum support constraint specified by the user. However, such a naive approach is inefficient for the following reason. The number of possible itemsets grows exponentially with the number of items in the database. If there are m distinct items in a transaction database, there are $2^m - 1$ possible itemsets. For example, if the database contains 1000 items, there are $2^{1000} - 1$ possible itemsets, which is clearly unmanageable using a naive approach. It is important to note that the FIM problem can be very difficult even for a small database. For example, a database containing a single transaction with 100 items, with $minsup = 1$ still generates a search space of 2^{100} itemsets.

Thus, the naive approach is not feasible for databases containing a large number of items. The number of itemsets in the search space generally matters more than the size of the data in FIM. But what influences the number of itemsets in the search space? The number of itemsets depends on how similar the transactions are in the database, and also on how low the *minsup* threshold is set by the user.

To discover frequent itemsets efficiently, it is thus necessary to design algorithms that avoid exploring the search space of all possible itemsets and that process each itemset in the search space as efficiently as possible.

1.2 Related Works

Several efficient algorithms have been proposed for FIM. Some of the most famous are Apriori[**hegland**], FP-Growth[**fp˙growth**],... All of these algorithms have the same input and the same output. However, the difference is the strategies and data structures that these algorithms employ to discover frequent itemsets efficiently. More specifically, FIM algorithms differ in whether they use a depth-first or breadth-first search, the type of database representation that they use internally or externally, how they generate or determine the next itemsets to be explored in the search space, and how they count the support of itemsets to determine if they satisfy the minimum support constraint.

1.2.1 Breadth-first search and depth-first search

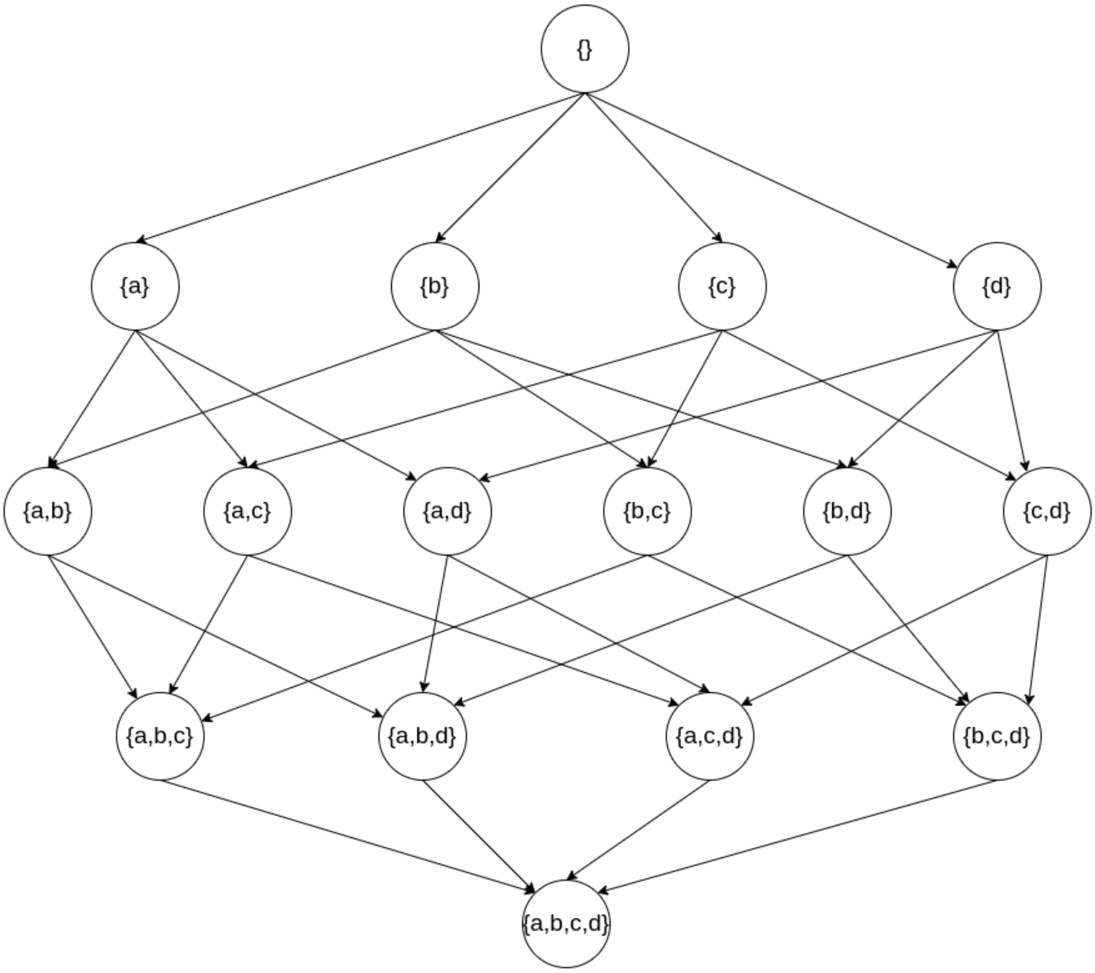
First of all, we can use the letters $a, b, c, ...$ to represent items in Table 1.1. Where *apple* is represented by a, *banana* is represented by b, *cherry* is represented by c and *mango* is represented by d. Then, we can represent it:

Table 1.2: Example of a dataset of transactions

Tid	**Itemsets**
1	a, b, c
2	a, d
3	a, c
4	d, c
5	a, d, c

Most of the existing itemset mining algorithms can be described as either using a breadth-first search or a depth-first search. Assume that there are m items

Figure 1.1: Search space of all possible itemsets for the running example



in a database. A breadth- first search algorithm (also called a level-wise algorithm) such as Apriori[**hegland**] explores the search space of itemsets by first considering 1-itemsets, then 2-itemsets, 3-itemsets . . . , and lastly m-itemsets.

For example, Figure 1.1 depicts the search space of all possible itemsets for the running example. In this figure, the search space is represented as a Hasse diagram[1] A breadth-first search algorithm will first consider 1-itemsets $\{a\}$, $\{b\}$, $\{c\}$, $\{d\}$. Then, it will generate 2-itemsets such as $\{a, b\}$, $\{a, c\}$, $\{a, d\}$, and then 3-itemsets, and so on, until it generates the itemset $\{a, b, c, d\}$ containing all items. On the other hand, depth-first search algorithms such as FPGrowth[**fp˙growth**], H-Mine[**h˙mine**] start from each 1-itemset and then recursively try to append items to the current itemset to generate larger itemsets. For example, in the running example, a typical depth-first search algorithm would explore itemsets in that order: $\{a\}$, $\{a, b\}$, $\{a, b, c\}$, $\{a, b, c, d\}$, $\{a, b, d\}$, $\{a, c\}$, $\{a, c, d\}$, $\{a, d\}$, $\{b\}$,

[1] A Hasse diagram draws an arrow from an itemset X to another itemset Y if and only if $X \subseteq Y$ and $|X| + 1 = Y$.

$\{b, c\}$, $\{b, c, d\}$, $\{b, d\}$, $\{c\}$, $\{c, d\}$, $\{d\}$.

To design an efficient FIM algorithm, it is important that the algorithm avoid exploring the whole search space of itemsets because the search space can be very large. To reduce the search space, search space pruning techniques are used.

1.2.2 Apriori: an horizontal breadth-first search algorithm

Apriori[**hegland**] is a well-known algorithm for mining frequent itemsets. It is the first FIM algorithm. Apriori takes a transaction database and the minsup threshold as input. Apriori uses a standard database representation, as shown in Table 1, also called a horizontal database. The pseudocode of Apriori is given:

Algorithm 1 Apriori algorithm

```
Input: $D$: a horizontal transaction database, $minsup$: a user-specified threshold
Output: the set of frequent itemsets
Scan the database to calculate the support of all items in I;
$F_1 = \{i | i \in I \wedge Supp(\{i\}) \geq minsup\}$;                    ▷ Frequent 1-itemsets
$k = 2$;
while $F_k \neq \emptyset$ do
    $C_k = \{i_1, i_2, ..., i_k | i_1 \in F_1, i_2 \in F_2, ..., i_k \in F_k\}$;      ▷ Generate candidate k-itemsets
    Remove each candidate $X \in C_k$ that contains a $(k-1)$ itemset not in $F_{k-1}$;
    Scan the database to calculate the support of each candidate in $C_k$;
    $F_k = \{X \in C_k | Supp(X) \geq minsup\}$;                      ▷ Frequent k-itemsets
    $k = k + 1$;
end while
return $\bigcup_{k=1...k} F_k$;
```

Apriori first scans the database to calculate the support of each item, i.e. 1-itemset. Then, Apriori uses this information to identify the set of all frequent items, denoted as F_1. Then, Apriori performs a breadth-first search to find larger frequent itemsets. During the search, Apriori uses the frequent itemsets of a given length $k-$ 1 (denoted as F_{k-1}) to generate potentially frequent itemsets of length k (denoted as C_k). This is done by combining pairs of items of length k that share all but one item. For example, if the frequent 1-itemsets are a, b, c, Apriori will combine pairs of these itemsets to obtain the following candidate 2-itemsets: a, b, a, c and b, c. After generating candidates of length k, Apriori checks if the $(k-1)$-subsets of each candidate are frequent. If a candidate itemset X has an infrequent $(k-1)$-subset, X cannot be frequent (it would violate the downward-closure property) and it is thus removed from the set of candidate $k - itemsets$. Then, Apriori scans the database to calculate the support of all remaining candidate itemsets in C_k. Each candidate having a support not less than minsup is added to the set F_k of frequent $k - itemsets$. This process is repeated until no candidates can be generated. The set of all frequent itemsets is then returned to the user.

Apriori is an important algorithm as it has inspired many other algorithms. However, it suffers from important limitations. The first one is that because Apriori generates candidates by combining itemsets without looking at the database, it can generate some patterns that do not even appear in the database. Thus, it can spend a huge amount of time processing candidates that do not exist in the database. The second limitation is that Apriori has to repeatedly scan the database to count the support of candidates, which is very costly.

The third limitation is that the breadth-first search approach can be quite costly in terms of memory as it requires at any moment to keep in the worst case all k and $k - 1$ itemsets in memory (for $k > 1$). In terms of complexity, a very detailed complexity analysis of the Apriori algorithm has been done by Hegland[**hegland**]. Briefly, the time complexity is $O(m^2n)$, where m is the number of distinct items and n is the number of transactions.

1.2.3 Pattern-growth algorithms

To overcome the limitations of Apriori, a new class of algorithms called pattern-growth. The main idea of pattern-growth algorithms is to scan a database to find itemsets, and thus avoid generating candidates that do not appear in the database. Furthermore, to reduce the cost of scanning the database, pattern-growth algorithms have introduced the concept of projected database to reduce the size of databases as an algorithm explore larger itemsets with its depth-first search.

Algorithm 2 A pattern-growth algorithm

```
Input: $D$: a horizontal transaction database, $minsup$: a user-specified threshold
Output: the set of frequent itemsets
Scan the database D to find the set Z of all frequent items in D;
for each item z in Z do do
    Output $X \cup \{z\}$;                                   ▷ $X \cup \{z\}$ is a frequent itemset
    $D' = Projecttion(D, z)$;                    ▷ Create the projected database of $X \cup \{z\}$
    $PatternGrowth(D, X \cup \{z\}, minsup)$;           ▷ Recursive call to extend $X \cup \{z\}$
end for
```

The pseudocode of a typical pattern-growth algorithm is shown in Algorithm 2. It takes as input a transaction database D, the empty set, and the *minsup* threshold. Without loss of generality, assume that there exists a total order on items $\prec$ such as the lexicographical order (e.g., $a \prec b \prec c \prec d$). A pattern-growth algorithm explores the search space using a depth-first search by recursively appending items according to the $\prec$ order to frequent itemsets, to obtain larger frequent itemsets. At the beginning, a pattern-growth algorithm considers that the current itemset X is the empty set. A pattern-growth algorithm scans the database D to find the set Z of all frequent items in D. Then for each such item z, the itemset $X \cup \{z\}$ is considered as a frequent itemset. Then, the pattern-growth procedure is called to perform a depth-first search to find larger frequent itemsets that are extensions of $X \cup \{z\}$ in same way. However, it can be observed that not all items in D can be appended to $X \cup \{z\}$ to generate larger itemsets. In fact, the itemset $X \cup \{z\}$ may not even appear in all transactions of the database D. For this reason, a pattern-growth algorithm will create the projected database of the itemset $X \cup \{z\}$ and will use this database to perform the depth-first search. This will allows reducing the cost of scanning the database. After recursively performing the depth-first search for all items, the set of all frequent itemsets will have been output.

Now, let's illustrate these steps in more details with an example. Assume that minsup $= 3$. By scanning the database of Table 1.2, it can be found that the frequent 1-itemsets are $\{a\}$, $\{b\}$, $\{c\}$, $\{d\}$. The algorithm will first consider the item a to try to find larger frequent itemsets starting with the prefix $\{a\}$. The algorithm will thus build the projected database of Table 1.3 The projected database of an item i is defined as the set of transactions where i appears, but where the item i and items preceding i according to the $\prec$ order are removed. Then, to find frequent itemsets starting with $\{a\}$ containing one more item, the algorithm will scan the projected database of $\{a\}$ and count the support of all items appearing in that database.

For example, the support of items in the projected database of $\{a\}$ are: $\{b\}$: 1, $\{c\}$: 2, $\{d\}$: 2. This means that the support of the itemset $\{a, b\}$ is 1, the support of the itemset $\{a, c\}$ is 2, and the support of the itemset $\{a, d\}$ is 2. The algorithm will output the frequent itemsets $\{a, c\}$ and $\{a, d\}$. Then, the algorithm will recursively call the pattern-growth procedure to find larger frequent itemsets starting with $\{a, c\}$ and $\{a, d\}$. Thus, only the itemset $\{a, c\}$ is frequent (recall

Table 1.3: Projected database of itemset {a}

Tid	Itemsets
1	b, c
2	d
3	c
5	d, c

Table 1.4: Projected database of itemset {a, c}

Tid	Itemsets
1	b
5	d

that we assume that minsup = 3 in the running example). Then the algorithm will pursue the depth-first search to find frequent itemsets starting with the prefix $\{a, c\}$. The algorithm will build the projected database of $\{a, c\}$ from the projected database of $\{a\}$. The projected database of $\{a, c\}$ is shown in Table 1.4.

Then, the algorithm will scan the projected database of $\{a, c\}$ to find frequent items in that database. This process will continue until all frequent itemsets have been explored by the depth-first search.

A major advantage of pattern-growth algorithms is that they only explore the frequent itemsets in the search space thus avoiding considering many itemsets not appearing in the database, or infrequent itemsets. Besides, the concept of projected database is also useful to reduce the cost of database scans. A common question about the concept of projected database is: is it costly to create all these copies of the original database? The answer is no if an optimization called pseudo-projection is used, which consists of implementing a projected database as a set of pointers on the original database.

CHAPTER 2

SAT-based Encoding of Itemset Mining

2.1 Technical background

2.1.1 Propositional Expression

Propositional logic formulas or propositional expressions are constructed from variables and logical operators AND (conjunction), OR (disjunction), NOT (negation), and parentheses.

Proposition: Each statement that can be either true or false is called a proposition, denoted by letters such as P, Q, R, ...

Negation

The negation (NOT) of a proposition P is denoted by ¬ P. The negation is true when P is false.

Truth table:

Table 2.1: Truth table of negation (NOT)

P	¬ P
T	F
F	T

Conjunction

The conjunction (AND) of two propositions P and Q is denoted by P ∧ Q. The conjunction is true only when both P and Q are true.

Truth table:

Table 2.2: Truth table of conjunction (AND)

P	Q	P ∧ Q
T	T	T
T	F	F
F	T	F
F	F	F

Disjunction

The disjunction (OR) of two propositions P and Q is denoted by P ∨ Q. The disjunction is true when at least one of P and Q is true.

Truth table:

Table 2.3: Truth table of disjunction (OR)

P	Q	P ∨ Q
T	T	T
T	F	T
F	T	T
F	F	F

Implication

The implication of two propositions P and Q is denoted by P → Q. The implication is false only when P is true and Q is false.

Truth table:

Table 2.4: Truth table of implication

P	Q	P → Q
T	T	T
T	F	F
F	T	T
F	F	T

Biconditional

The biconditional of two propositions P and Q is denoted by P ↔ Q. The biconditional is true when both P and Q have the same truth value.

Truth table:

Table 2.5: Truth table of biconditional

P	Q	P ↔ Q
T	T	T
T	F	F
F	T	F
F	F	T

2.1.2 Conjunction Normal Form (CNF)

A propositional formula is in conjunctive normal form (CNF) if it is a conjunction of clauses, where each clause is a disjunction of literals. A literal is a propositional variable or its negation. The standard form of CNF is

$$(P_1 \vee P_2 \vee ... \vee P_n)_1 \wedge ... \wedge (Q_1 \vee Q_2 \vee ... \vee Q_m)_p \quad n, m, p \geq 1$$

Sample truth table (only for P and Q and R):

Table 2.6: Truth table of CNF

P	Q	R	(P ∧ Q) ∧ R
T	T	T	T
T	T	F	F
T	F	T	F
T	F	F	F
F	T	T	F
F	T	F	F
F	F	T	F
F	F	F	F

2.1.3 Technical background of Itemset Mining

Firstly, we establish several symbols to represent the itemset mining problem. These symbols aid in formalizing the problem and defining key concepts. For instance, we denote:

- Ω: a set of all items
- I: an itemset in Ω, where $I \subseteq \Omega$
- T_i: a transaction identifier. For $T_i = (i, I)$

- D: a transaction database, where D contains a set of transactions, $D = \{T_1, T_2, ..., T_n\}$
- $Supp(I, D)$: the support of itemset I in database D, where $Supp(I, D)$ is the number of transactions in D that contain I

For example, in table 1.1, we can present the dataset as a transaction database D.

Let $a = apple, b = banana, c = cherry, d = mango$. Then we have database transactions in binary format as shown in table 2.7.

Table 2.7: Sample dataset of transactions in binary format

Tid	a	b	c	d
1	1	1	1	0
2	1	0	0	1
3	1	0	1	0
4	0	0	1	1
5	1	0	1	1

- Ω is {apple, banana, cherry, mango}
- I can be {apple}, {apple, mango}, {apple, mango, cherry}, ...
- D = {(1, {apple, banana, cherry}), (2, {apple, mango}), (3, {apple, cherry}), (4, {mango, cherry}), (5, {apple, mango, cherry})}
- T_1 = (1, {apple, banana, cherry}), T_2 = (2, {apple, mango}), T_3 = (3, {apple, cherry}),...
- $Supp(\{apple, cherry\}, D) = 3$, $Supp(\{apple\}, D) = 4$,...

Let λ be the minimum support threshold, the frequent itemset mining problem is to find all itemsets I such that $Supp(I, D) \geq minsup$. In general, it can present by:

$$FIM(D, \lambda) = \{\mathrm{I} \subseteq \Omega \mid Supp(I, D) \geq \lambda\}$$

One of the major challenges in itemset mining is the potential exponential growth of the output, even when using condensed representations of patterns.

2.2 SAT Encoding

2.2.1 Concept

The concept of SAT, also known as the Boolean Satisfiability problem, is a computer science problem aimed at determining the satisfiability of a propositional logic formula.

Input: A propositional logic formula, typically represented in Conjunctive Normal Form[**cnf**] (CNF).

Output:

- SAT (Satisfiable): If there exists a truth value assignment (true/false) to the logical variables that makes the original propositional logic formula evaluate to true.
- UNSAT (Unsatisfiable): If every truth value assignment (true/false) to the logical variables results in the original propositional logic formula evaluating to false.

2.2.2 Encoding

SAT Encoding is a method in which certain problems can be solved by transforming them into SAT problems: representing problems using propositional logic formulas and applying SAT Solvers to solve these propositional logic formulas[**pham˙ngo**

A problem solved using SAT Encoding follows the following steps: First, identify the input data or the problem's input that needs to be solved. The Encoding module then takes this input data, defines the rules, and encodes these rules into Conjunctive Normal Form[**cnf**] (CNF) logical formulas, producing an output file containing the number of variables, the number of clauses, and the CNF formulas. The SAT Solver takes the output file from the Encoding module as input, processes the expressions, and generates the output result. The output result can be SAT if the SAT Solver finds a dataset satisfying the CNF formulas, or UNSAT if it fails to find any dataset. If the result is SAT, the SAT Solver provides another file containing the corresponding dataset it found. From the discovered

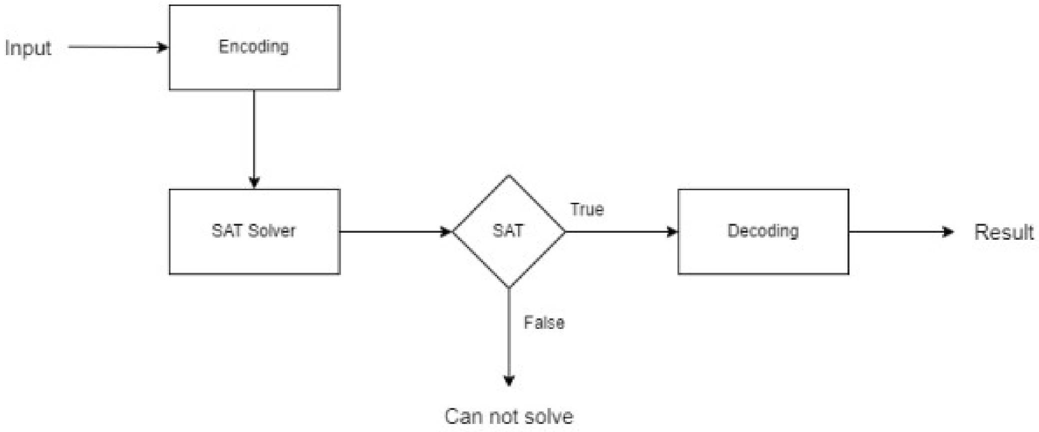


Figure 2.1: SAT Encoding Process

dataset, the solution to the problem can be inferred, resulting in the corresponding answer for the input data.

2.2.3 SAT Solvers

SAT Solver is a tool designed to solve the Boolean Satisfiability (SAT) problem, determining whether a propositional logic formula is satisfiable or not. It utilizes CNF (Conjunctive Normal Form[**cnf**]) formulas.

In the SAT problem, we are given n Boolean variables and m clauses, where each clause is the disjunction of a set of literals, and a literal is a variable or its negation. The aim is to decide whether there is an interpretation of the variables that satisfies all clauses. Currently, numerous SAT- tools can efficiently handle a large number of clauses and variables, providing optimal results. Examples include Minisat[**minisat**], Lingeling[**lingeling**], Glucose[**glucose**], RSat[**rsat**].

SAT is proven to be NP-complete[**sat˙np˙complete**], meaning it has exponential time complexity in the worst case scenario. Despite this, effective and scalable algorithms for SAT have been developed in the 2000s, significantly advancing the automatic resolution of problems with tens of thousands of variables and millions of clauses [6]. One such algorithm used by SAT Solvers is DPLL (Davis-Putnam-Logemann-Loveland), introduced in 1961, relying on a backtracking, exhaustive search approach.

Modern SAT-solving methods introduced in the 2000s have incorporated the Conflict Driven Clause Learning (CDCL) algorithm alongside DPLL, enhancing the efficiency of SAT Solvers in various domains.

Kissat[**kissat˙jku**] is a ”keep it simple and clean bare metal SAT solver” written in C. It is a port of CaDiCaL back to C with improved data structures, better scheduling of inprocessing and optimized algorithms and implementation. The CNF file format used by Kissat consists of the following components:

- The first line specifies the number of variables and the number of clauses in the CNF file.
- The subsequent lines contain the CNF-formatted logical statements, represented as clauses.

```
p cnf [number of variables] [number of clauses]
[clauses]
```

In this format:

- `p cnf` denotes the CNF format.
- `[number of variables]` denotes the count of logical variables used.
- `[number of clauses]` represents the number of logical clauses.
- `[clauses]` refers to the CNF-formatted logical statements.

For example:

```
p cnf 3 2
1 -2 0
2 3 -1 0
```

This CNF file contains 3 variables and 2 clauses. The first clause is `[1 -2 0]`, and the second clause is `[2 3 -1 0]`. The `0` at the end of each clause denotes the end of the clause. It represented as logical formulas:

$$(x_1 \vee \neg x_2) \wedge (x_2 \vee x_3 \vee \neg x_1).$$

The SAT problem can be solved by running the Kissat SAT Solver on the CNF file. The output will indicate whether the logical formula is satisfiable or unsatisfiable and provide the truth value assignments for the logical variables with format:

```
s [status]
v [variable assignments]
```

For this given CNF file, the output will be:

```
s SAT
v 1 -2 -3
```

But it is only one of the possible solutions, there are many other solutions that satisfy the CNF formula. To find all solutions, we need to append the solution found as a clause and run the SAT Solver again. This process is repeated until the SAT Solver returns UNSAT, indicating that all solutions have been found.

2.2.4 Applications

In addition to SAT Encoding, SAT is utilized in various fields of information technology. Notable areas include: In formal methods, SAT is used for hardware model testing, software model testing, and test pattern generation. In the field of artificial intelligence, SAT is employed for planning, knowledge representation problems, and in intelligent games. In the realm of automated design, SAT is applied for equivalence checking, delay computation, error detection, and more. In this chapter, we provide a detailed walkthrough of encoding the itemset mining problem into a SAT problem. We begin by establishing variable conventions and introducing constraints, gradually transitioning to converting these constraints into Conjunctive Normal Form (CNF) using the standard method. Along the way, we discuss the limitations of the standard method, particularly in handling large datasets or high support thresholds. Our goal is to equip readers with a clear understanding of the SAT encoding process for itemset mining and the challenges it entails.

2.3 Base constraints

To resolve the itemset mining problem, we using the SAT encoding approach[**ism˙satapp** In essence, SAT encoding involves the creation of variables and the imposition of constraints to represent the itemset mining problem. These variables serve to denote the presence or absence of items within a candidate itemset and are subjected to linear inequalities to ensure the itemset's support.

In the context of a transaction database $D = (1{,}T_1),...,(\mathrm{m}{,}T_m)$ and a minimum support threshold λ, each item in the candidate itemset X, we denote:

- p_a: is $true$ if the item a is in the itemset X, otherwise $p_a = false$
- q_i: is $true$ if the transaction T_i contains the itemset X, otherwise $q_i = false$

Alongside, a set of constraints is imposed on these variables to establish a one-to-one correspondence between the models of the resulting CNF formula and the set of itemsets.

Firstly, to capture all the transactions where the candidate itemset does not appear, we use following constraint:

$$\bigwedge_{i=1}^{m}(q_i \leftrightarrow \bigwedge_{a \notin T_i} \neg p_a) \tag{2.1}$$

This constraint guarantees that q_i is true if and only if either all items not in T_i are also not in the itemset X, or transaction T_i contains the itemset X.

Constraint 2.1 can be rewritten as follows:

$$\bigwedge_{a \in \Omega} \bigwedge_{a \notin T_i} (\neg p_a \vee \neg q_i) \tag{2.2}$$

$$\bigwedge_{T_i \in D} ((\bigvee_{a \notin T_i} p_a) \vee q_i) \tag{2.3}$$

Finally, the frequency constraint, can be simply expressed as follows:

$$\sum_{i=1}^{m} q_i \geq \lambda \tag{2.4}$$

For example, with a dataset of transactions from a retail store in table 1.1, we mark: a = apple, b = banana, c = cherry, d = mango. Then we have database transactions

Table 2.8: Example of a dataset of transactions after convert

Tid	**a**	**b**	**c**	**d**
1	1	1	1	0
2	1	0	0	1
3	1	0	1	0
4	0	0	1	1
5	1	0	1	1

The itemset mining problem will be defined as:

$$\begin{aligned}
&q_1 \leftrightarrow (\neg p_d)\\
&q_2 \leftrightarrow (\neg p_b \wedge \neg p_c)\\
&q_3 \leftrightarrow (\neg p_b \wedge \neg p_d)\\
&q_4 \leftrightarrow (\neg p_a \wedge \neg p_d)\\
&q_5 \leftrightarrow (\neg p_b)\\
&q_1 + q_2 + q_3 + q_4 + q_5 \geq \lambda
\end{aligned}$$

In the next step, we must encode constraint 2.4 into CNF formula.

2.4 Standard Method in Itemset Mining

After encoding the base constraints, the subsequent step involves applying the standard method to encode formula 2.4.

To solve the problem $q_1 + q_2 + ... + q_n \geq \lambda$, we can use the standard method known as C_{n-k+1}.

The fundamental idea behind this algorithm is as follows: Consider a set of n elements. If there are at least k true elements, it is equivalent to having at most $n - k$ false elements. In simpler terms, when selecting $n - k + 1$ elements, there is a guarantee of having at least one true element among them.

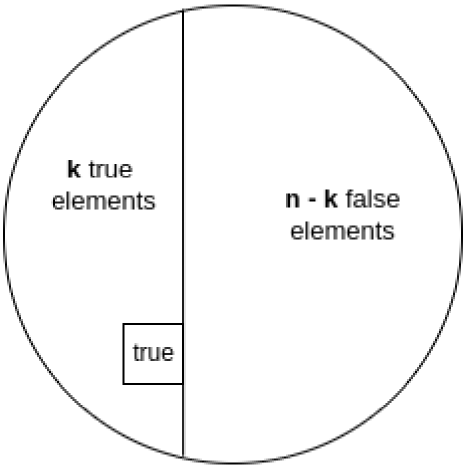


Figure 2.2: Illustration of the standard method C_{n-k+1}

To illustrate this concept further, let's take an example where $n = 5$ and $\lambda = 3$. In this scenario, we can employ the following constraint to depict the standard method

$$q_1 + q_2 + q_3 + q_4 + q_5 \geq 3$$

We can use the constraint below to represent the concept of having at least 3 true elements among 5 elements:

$$\begin{aligned}
&(q_1 \lor q_2 \lor q_3)\\
\land&(q_1 \lor q_2 \lor q_4)\\
\land&(q_1 \lor q_2 \lor q_5)\\
\land&(q_1 \lor q_3 \lor q_4)\\
\land&(q_1 \lor q_3 \lor q_5)\\
\land&(q_1 \lor q_4 \lor q_5)\\
\land&(q_2 \lor q_3 \lor q_4)\\
\land&(q_2 \lor q_3 \lor q_5)\\
\land&(q_2 \lor q_4 \lor q_5)\\
\land&(q_3 \lor q_4 \lor q_5)
\end{aligned} \tag{2.5}$$

Then with n elements and λ, we can present the constraint as:

$$\bigwedge_{i=1}^{n-\lambda+1} \left(\bigvee_{j=i}^{i+\lambda-1} q_j \right) \tag{2.6}$$

In this equation, the outer conjunction $\bigwedge$ iterates from $i = 1$ to $n - \lambda + 1$, representing the range of possible starting positions for the subsequence of length λ. The inner disjunction $\bigvee$ iterates from $j = i$ to $i + \lambda - 1$, representing the elements within each subsequence. By combining these subsequence constraints with the conjunction operator, we ensure that at least one subsequence of length λ contains all true elements.

This constraint, along with the previously mentioned constraints 2.2, 2.3, and 2.6, can be used to resolve the problem of Itemset Mining.

2.5 Limitation of Standard Method

The standard method C_{n-k+1} is a widely used approach to solve various problems, including itemset mining. However, its major drawback lies in the explosion of combinations, especially when k approaches $n/2 + 1$.

In a simple example, with $n = 30$ and $\lambda = 16$, the number of clauses required to encode C_{15}^{30} is approximately 155 million. In reality, n corresponds to the number of transactions, which can range from thousands to millions. Therefore, the standard method is not feasible for large datasets.

Moreover, solving problems using the standard method can be time-consuming and resource-intensive. The computational complexity increases exponentially with the size of the input. As a result, the standard method has limitations in handling large-scale datasets efficiently.

Additionally, the standard method has a small limit on the size of the input it can handle. When the number of transactions or the dimensionality of the problem exceeds a certain threshold, the standard method becomes impractical and fails to provide accurate results.

These limitations highlight the need for alternative approaches that can overcome the drawbacks of the standard method and efficiently handle large-scale datasets with complex problem structures.

CHAPTER 3

Sequential Encounter Encoding for Optimization

In this chapter, we will introduce a method to optimize CNF encodings, which is called Sequential Encounter Encoding (SEE). Base on the work of Carsten Sinz for problem 'At Most K', we will adapt the SEE method to solve the problem 'at least k' of itemset mining.

3.1 Sequential Encounter Encoding

In a paper published in 2005, Carsten Sinz[**carstensinz**] presented a novel approach to encode constraints within the realm of CNF encodings, which he termed Sequential Encounter Encoding. This technique introduced a method to count occurrences of boolean variables, particularly useful for encoding problems where the number of true variables must not exceed a certain threshold k in the series of variables $x_1, x_2, ..., x_n$. It is denote as $\leq_k (x_1, ..., x_n)$, where the goal is to ensure that at most k of the variables $x_1, x_2, ..., x_n$ are true.

The approach is to encode a circuit that sequentially counts the number of x_i that are true. For each $1 \leq i \leq n$, a register is introduced to store the count of $x_1, x_2, ..., x_i$ that are true. Each register maintains its count in base one, using k bits to count to k. Thus, the encoding introduces new variables r_{ij}, $1 \leq i \leq n$, $1 \leq j \leq k$, where each r_{ij} represents the j^{th} bit of register i.

In section 2, we have encoded the problem of item set mining in terms of constraints 2.2, 2.3 and 2.4. With constraint 2.4, we ensure that the number of transactions q_i that contain the itemset X is at least λ.

Building on Sinz's foundational work, this paper adapts the SEE technique for an **'at least k'** problem context, applying it specifically to CNF encoding in itemset mining. Instead of ensuring that at most k of the variables are true, the goal is to ensure that at least k of the variables are true.

Within the scope of itemset mining, this means that each transaction q_i is represented by a boolean variable, true if the transaction contains the itemset X and false otherwise. The goal is to ensure that at least λ (equivalent to k) of these transactions are true within a selected subset of transactions $q_1, q_2, ..., q_n$.

To approach this, each transaction is processed one by one, updating a register r which holds the running true transactions up to that point. If r_{ij} is true, it indicates that by the time the sequence reaches transaction q_i, there have been j transactions from q_1 to q_i are true.

In adapting Sinz's method for this new application, this paper sheds light on the potential versatility of Sequential Encounter Encoding in addressing a broader range of logical constraints, extending its utility beyond the original 'at most k' problems.

3.2 Optimization CNF Encoding using Sequential Encounter Encoding

We divide the process into two parts. The first part involves constructing the variables r_{ij}, and the second part involves adding the requirement that at least λ transactions must be true.

3.2.1 Construct register

First of all, we introduce the following constraint to establish the fundamental relationship between q and r:

$$\bigwedge_{i=1}^{n} (q_i \rightarrow r_{i1}) \tag{3.1}$$

This constraint ensures that if q_i is true, then the first bit of register i will also be true. In other words, when processing q_i and q_i is true, we can guarantee that at least one transaction q from q_1 to q_i is true. While there may be more than one transaction that containt the expected itemset, we only ensuring a minimum of one transaction is true.

When $i < \lambda$, if q_i is false, bit i of register i will be false. Suppose all transactions from q_1 to q_{i-1} are true, but q_i is false, then $r_{i,i}$ will be false. This constraint is expressed as:

$$\bigwedge_{i=1}^{k} (\neg q_i \rightarrow \neg r_{ii}) \tag{3.2}$$

Nextly, we have a constraint that allows us to disregard cases where the number of processed transactions is less than λ:

$$\bigwedge_{i=1}^{\lambda-1} \bigwedge_{j=i+1}^{\lambda} (\neg r_{ij}) \tag{3.3}$$

This constraint ensures that if i is less than λ, all bits after i in register i will be false. In practice, it has been observed that in order to have at least λ transactions that are true from q_1 to q_i, we need to process at least λ transactions. Therefore, we can disregard cases where the number of processed transactions is less than λ.

Additionally, we have a constraint that allows us to clone the previous result of the register to the next register:

$$\bigwedge_{i=2}^{n} \bigwedge_{j=1}^{\lambda} (r_{i-1,j} \rightarrow r_{ij}) \tag{3.4}$$

This constraint demonstrates that if there are at least j transactions that are true from q_1 to q_{i-1}, it is a fact that there are at least j transactions that are true from q_1 to q_i, regardless of the value of q_i.

Next, as we process the transaction q_i and consider its value, we need to take into account the previous result that does not include the transaction q_i. This leads us to the following constraint:

$$\bigwedge_{i=2}^{n} \bigwedge_{j=2}^{\lambda} (q_i \wedge r_{i-1,j-1} \rightarrow r_{ij}) \tag{3.5}$$

This constraint establishes the relationship between the value of q_i and the previous result when processing up to $i - 1$. Whether q_i is true or false determines whether the result of register i is true or false. Additionally, the previous result at bit i influences the result of register i at bit j. In other words, if there are $j - 1$ transactions that are true from q_1 to q_{i-1}, and q_i is true, then there are j transactions that are true from q_1 to q_i.

If both $r_{i-1,j}$ and q_i are false, we guarantee that r_{ij} will also be false:

$$\bigwedge_{i=2}^{n} \bigwedge_{j=1}^{\lambda} (\neg q_i \wedge \neg r_{i-1,j} \rightarrow \neg r_{ij}) \tag{3.6}$$

When q_i is false and there are not j transactions that are true from q_1 to q_{i-1}, then there will also not be j transactions that are true from q_1 to q_i. In other words, the value of q_i being false does not affect the result of register i.

To enforce the sequential order of bits in register i, make sure not have bit 0 before bit 1 in register i, we can use the following constraint:

$$\bigwedge_{i=2}^{n} \bigwedge_{j=2}^{\lambda} (\neg r_{i-1,j} \wedge \neg r_{i-1,j-1} \rightarrow \neg r_{ij}) \tag{3.7}$$

This constraint ensures that if both bit j and bit $j-1$ of register $i-1$ are false, then bit j of register i will also be false. In other words, if transaction q_{i-1} is false and there are not having j true transactions from q_1 to q_{i-1}, then when processing the next transaction q_i, there will also not be j true transactions from q_1 to q_i. The best case scenario is that there can only be $j-1$ true transactions.

3.2.2 At least λ required

Firstly, to ensure having extract λ transactions contains item set X from q_1 to q_n, we can impose the following constraint:

$$r_{n-1,\lambda} \vee (q_n \wedge r_{n-1,\lambda-1}) \tag{3.8}$$

This ensures that when processing q_{n-1}, there are either at least λ transactions that are true from q_1 to q_{n-1}, or there are $\lambda - 1$ transactions that are true from q_1 to q_{n-1} and q_n is true.

Secondly, if the value of q_n is false, to having at least λ true transactions, we need $r_{n-1,\lambda}$ must be true. This ensures that there are at least λ transactions that are true from q_1 to q_{n-1}.

$$\neg q_n \rightarrow r_{n-1,\lambda} \tag{3.9}$$

Finally, we have the constraint for all $i > \lambda$

$$\bigwedge_{i=\lambda+1}^{n} (\neg q_i \wedge r_{i,\lambda} \rightarrow r_{i-1,\lambda}) \tag{3.10}$$

It ensures sure that if q_i is false but when process to it, there are already having λ transactions is true from q_1 to q_i. This is only because of the previous transactions from q_1 to q_{i-1} has λ transactions that are contain item set X.

We have completed the process of defining the constraints for solving the "at least k" problem in itemset mining, replacing equation 2.4. By combining all the constraints together, including equations 2.2 and 2.3 from section 2, we obtain the final CNF encoding for the itemset mining problem.

3.3 Implement Sequential Encounter Encoding

The sequential encounter encoding method is implemented in Python. You can found source code in tanhtanh1505/ism-kissat[1]. The implementation is based on the SAT-based encoding of itemset mining problem. The main difference is that the sequential encounter encoding method encodes the constraints for the sequential encounter of items in the transactions. The implementation is provided in the `sequential_encounter.py` file.

For constraint at least one bit of register i is true when q_i is true (3.1) and bit i of register i is false when q_i is false (3.2).

```
Input: number of transactions $n$, min support $\lambda$
Output: constraints c
c = []
for $i = 1$ to $n$ do
    c.append($\neg q_i \lor r_{i,1}$)
    if $i \leq \lambda$ then
        c.append($q_i \lor r_{i,1}$)
    end if
end for
return c
```

[1] https://github.com/tanhtanh1505/ism-kissat

Next step, we implement code to disregard some bits of register when $i \leq \lambda$ (3.3)

```
**Input**: number of transactions $n$, min support $\lambda$
**Output**: constraints c
c = []
for $i = 1$ to $\lambda$ do
    for $j = i + 1$ to $\lambda + 1$ do
        c.append($\neg r_{i,j}$)
    end for
end for
return c
```

We add the following code for relationship between the value of q_i and the previous result when processing up to $i - 1$ (3.4)

```
**Input**: number of transactions $n$, min support $\lambda$
**Output**: constraints c
c = []
for $i = 2$ to $n + 1$ do
    for $j = 1$ to $\lambda + 1$ do
        c.append($\neg r_{i-1,j} \vee r_{i,j}$)
    end for
end for
return c
```

And 3 constraints

- q_i is true add one true transaction to the previous result when processing up to $i - 1$ (3.5)
- q_i being false does not affect the result of register i (3.6)
- the sequential order of bits in register i (3.7)

```
Input: number of transactions $n$, min support $\lambda$
Output: constraints c
c = []
for $i = 2$ to $n + 1$ do
    for $j = 1$ to $\lambda + 1$ do
        if $j > 1$ then
            c.append($\neg q_i \vee \neg r_{i-1,j-1} \vee r_{ij}$)
            c.append($r_{i-1,j} \vee r_{i-1,j-1} \vee \neg r_{ij}$)
        end if
        c.append($q_i \vee r_{i-1,j} \vee \neg r_{ij}$)
    end for
end for
return c
```

Finally, we use below code for adding rule require at least k (constraints 3.8, 3.9, 3.10)

```
Input: number of transactions $n$, min support $\lambda$
Output: constraints c
c = []
c.append($r_{n-1,\lambda} \vee q_n$)
c.append($r_{n-1,\lambda} \vee r_{n-1,\lambda-1}$)
c.append($q_n \vee r_{n-1,\lambda}$)
for $i = k + 1$ to $n + 1$ do
    c.append($q_i \vee \neg r_{i,k} \vee r_{i-1,k}$)
end for
return c
```

CHAPTER 4

Experiments

In this chapter, the focus lies on showcasing the setup process, resource utilization, dataset selection, and experimental comparison between two methods: sequential encounter encoding and the standard method for solving the itemset mining problem. The chapter will provide detailed instructions on how to install the necessary tools and libraries, specify the hardware and software resources utilized, describe the datasets employed for experimentation, and present the results of the comparative analysis between the sequential encounter encoding method and the standard method.

4.1 Setup and Datasets

4.1.1 Setup

The experiments were conducted on a machine with the following specifications:

- Processor: Intel Core i7-7700HQ CPU @ 2.80GHz
- Memory: 16GB DDR4 RAM
- Operating System: Ubuntu 20.04 LTS

For the setup, the experiments are conducted using Python 3.8 as main programming language, and the following libraries are used such as NumPy, Pandas, and Matplotlib for data manipulation and visualization.

The SAT Solver used in the experiments is Kissat. Kissat is a ”keep it simple and clean bare metal SAT solver” written in C. It is a port of CaDiCaL back to

C with improved data structures, better scheduling of inprocessing and optimized algorithms and implementation. The source code for Kissat can be found at `https://github.com/arminbiere/kissat`.

After cloning the repository, the solver can be built and run by executing a command in the command line using the subprocess module in Python. This allows for seamless integration of the solver into existing Python scripts or workflows. The subprocess module provides a way to spawn new processes, connect to their input/output/error pipes, and obtain their return codes. By utilizing this module, the solver can be invoked and its output can be captured and processed programmatically. This provides flexibility and automation in running the solver and analyzing its results.

But Kissat is find only the first solution, so we need to modify the source code to find all solutions. By adding solved result to the list and continue to find the next solution until there is no solution left.

4.1.2 Generated Datasets

To compare the performance of the sequential encounter encoding method with the standard method. In this repository, it provides a synthetic dataset generator that can generate a dataset with a specified number of transactions, items and minimun support. The generator is written in Python and can be found in the `generator.py` file in folder `input`.

Algorithm 3 Dataset Generation Algorithm

```
Input: transactions, items, output_folder
Output: dataset saved in the output_folder
dataset = []
for transaction in transactions do
    transaction = []
    for item in items do
        item = random.choice([0, 1])
        transaction.append(item)
    end for
    dataset.append(transaction)
end for
write_to_file(dataset, output_file)
```

4.1.3 Real-World Datasets

The datasets used in the experiments are coming from FIMI[**fimi**] and CP4IM[**cp4im**]. The characteristics of the considered datasets are given in Table 4.1.

Table 4.1: Characteristics of the considered datasets

Dataset	Transactions	Items
zoo-1	101	36
primary-tumor	336	32
vote	435	48
soybean	630	50
chess	3196	75
mushroom	8124	119

The FIMI[1] datasets are commonly utilized in the data mining community to assess the performance of frequent itemset mining algorithms. These datasets are designed to evaluate the ability of algorithms to discover frequent itemsets efficiently. On the other hand, the CP4IM[2] datasets are specifically created for evaluating constraint programming-based itemset mining algorithms. These datasets are formatted in ARFF, which is a file format commonly used by the Weka data mining software. To ensure compatibility with both the sequential encounter encoding method and the standard method, the datasets have been preprocessed and converted into a suitable format.

Format: The datasets are in annotated transaction format with labels: every line is one transaction. A transaction is a space-separated list of item identifiers (offset 0), the last item is either 1 or 0 and represents the class label. The meaning of every label is given in the header of the file: $@ < nr >$: ... lines describe item number $< nr >, @class$: ... describes the two classes. To parse the files correctly, all lines starting with @, with % and empty lines should be ignored. (the format is a combination of the FIMI format with annotations like the ARFF format).

Preprocessing: The datasets are preprocessed to remove any unnecessary information and ensure that they are in the correct format.

- Remove all tag lines starting with @, with % and empty lines.
- Remove class labels from the transactions.

[1] `http://fimi.uantwerpen.be/data/` [2] `https://dtai.cs.kuleuven.be/CP4IM/datasets/`

- Run the script `convert_real_world.py` to preprocess the datasets into readable format.
- The preprocessed datasets are saved in the `input` folder.

4.2 Results and Analysis

This section presents the results of the experiments conducted to evaluate the performance of the sequential encounter encoding and the direct encoding in solving the optimization problem. Because the number of variables and clauses in the CNF encoding when using standard method is too large, we benchmarked the performance of the two encoding methods on a smaller generated dataset.

4.2.1 Comparison of Sequential Encounter Encoding and Standard Encoding

The dataset was generated using the `input/generate.py` script, which is included in the source code repository. It is stored in the `input` directory in the repository. The dataset used in the experiments was generated using the following parameters:

- Number of items: 8
- Number of transactions: 28

After generating the dataset, the optimization problem was encoded into CNF format using the sequential encounter encoding and the standard encoding. With timeout 900ms and 1GB memory limit, we ran the Kissat SAT Solver on the CNF file to find all solutions. The number of clauses and variables in the CNF encoding for each encoding method is shown in Figure 4.1 when Minimum Support is increased from 0.1 to 0.9 (10% to 90% of the total transactions).

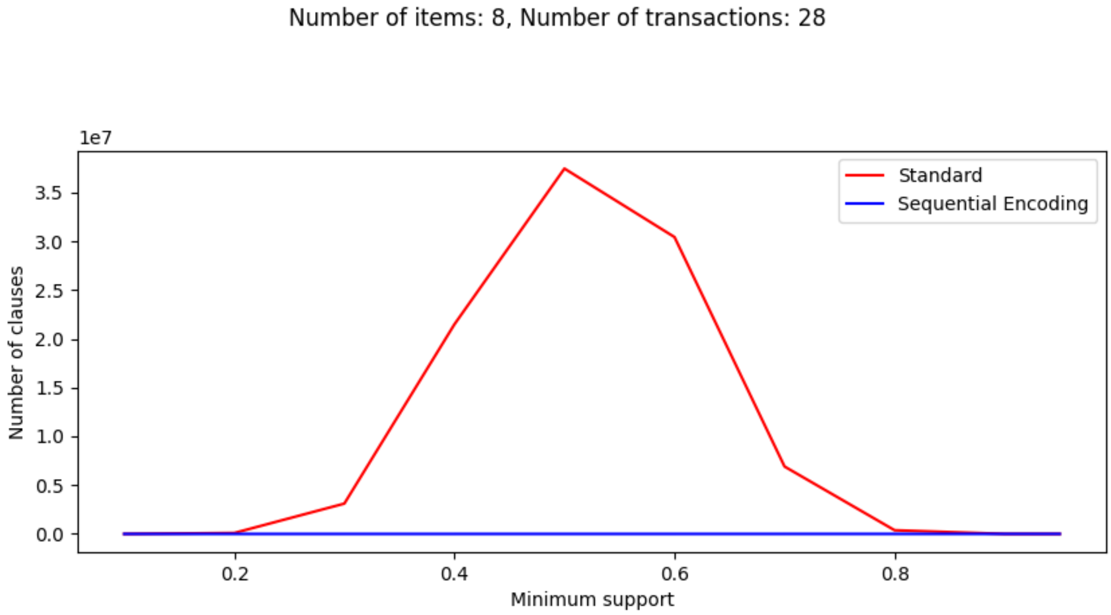


Figure 4.1: Comparison of the number of clauses in the CNF encoding of the optimization problem using the sequential encounter encoding and the standard encoding.

This graph illustrates a notable difference between the number of clauses in the CNF encoding utilizing the sequential encounter encoding compared to the standard encoding method. Particularly, it demonstrates that the sequential encounter encoding results in a significantly smaller number of clauses and variables.

Of special interest is the observation that the number of clauses generated by the standard encoding reaches a peak value, approximately 35 million clauses, as the minimum support approaches the midpoint threshold. This suggests a critical point where the standard encoding method generates an overwhelming number of clauses, highlighting the potential inefficiency and scalability issues associated with this approach.

Additionally, the number of clauses generated by the sequential encounter encoding method appears to remain consistently low, maintaining stability across various levels of minimum support. This suggests that the sequential encounter encoding method is capable of producing fewer clauses while still ensuring the efficiency of the encoding process, thus keeping the data analysis process manageable even as the constraints grow larger.

In addition to comparing the number of clauses, we also compared the time taken to find all solutions using both methods. The results are shown in Figure

4.2.

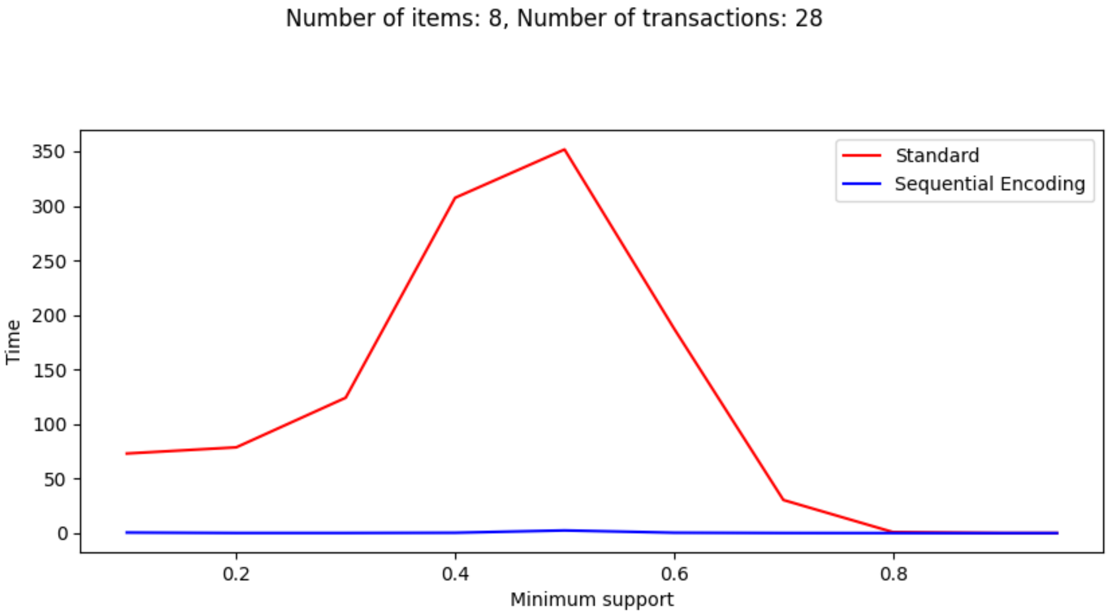


Figure 4.2: Comparison of the time taken to find all solutions using the sequential encounter encoding and the standard encoding

The graph indicates that the sequential encounter encoding method consistently outperforms the standard encoding method in terms of time efficiency. Even as the complexity of the problem increases, the sequential encounter encoding method demonstrates faster solution-finding times, highlighting its superiority not only in minimizing the number of clauses but also in accelerating the solution discovery process.

However, it's worth noting that while the number of variables may increase, this increment is negligible compared to the significant reduction in the number of clauses.

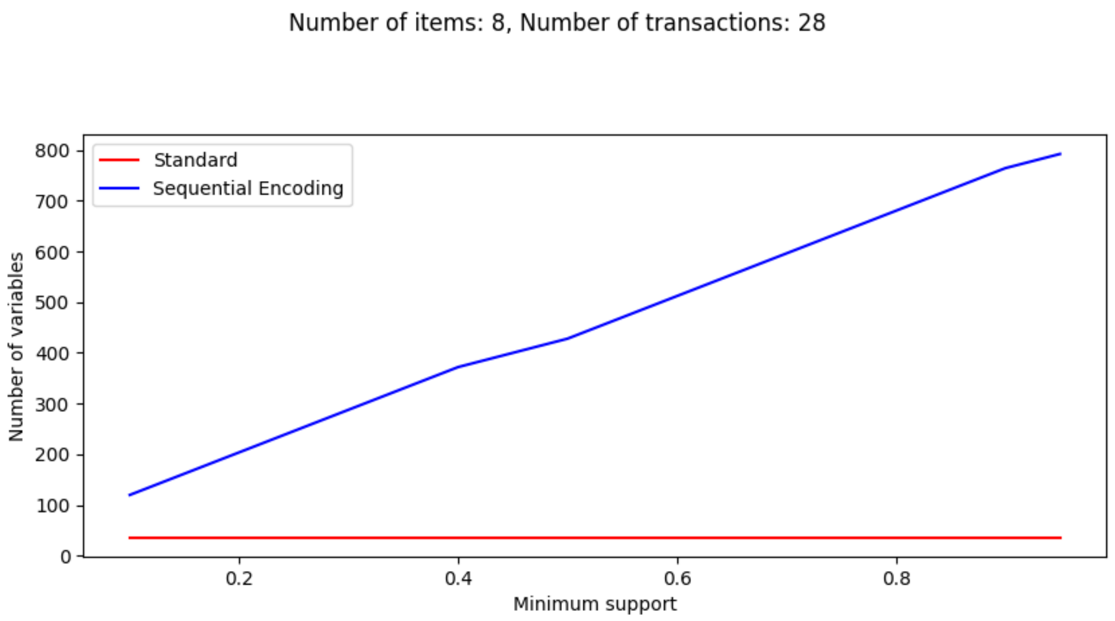


Figure 4.3: Comparison of the number of variables in the CNF encoding of the optimization problem using the sequential encounter encoding and the standard encoding.

To further test the performance, we conducted additional experiments. First, we increased the number of transactions from 25 to 28 and compared the number of variables in the CNF encoding of the optimization problem using the sequential encounter encoding and the standard encoding. The results are shown in Figure 4.4.

Next, we kept the number of transactions constant and varied the minimum support from 0.3 to 0.6 (step 0.1). We then compared the number of variables in the CNF encoding using both encoding methods. The results are shown in Figure 4.5.

These additional tests provide a more comprehensive evaluation of the sequential encounter encoding method and the standard encoding method, allowing us to assess their performance under different scenarios.

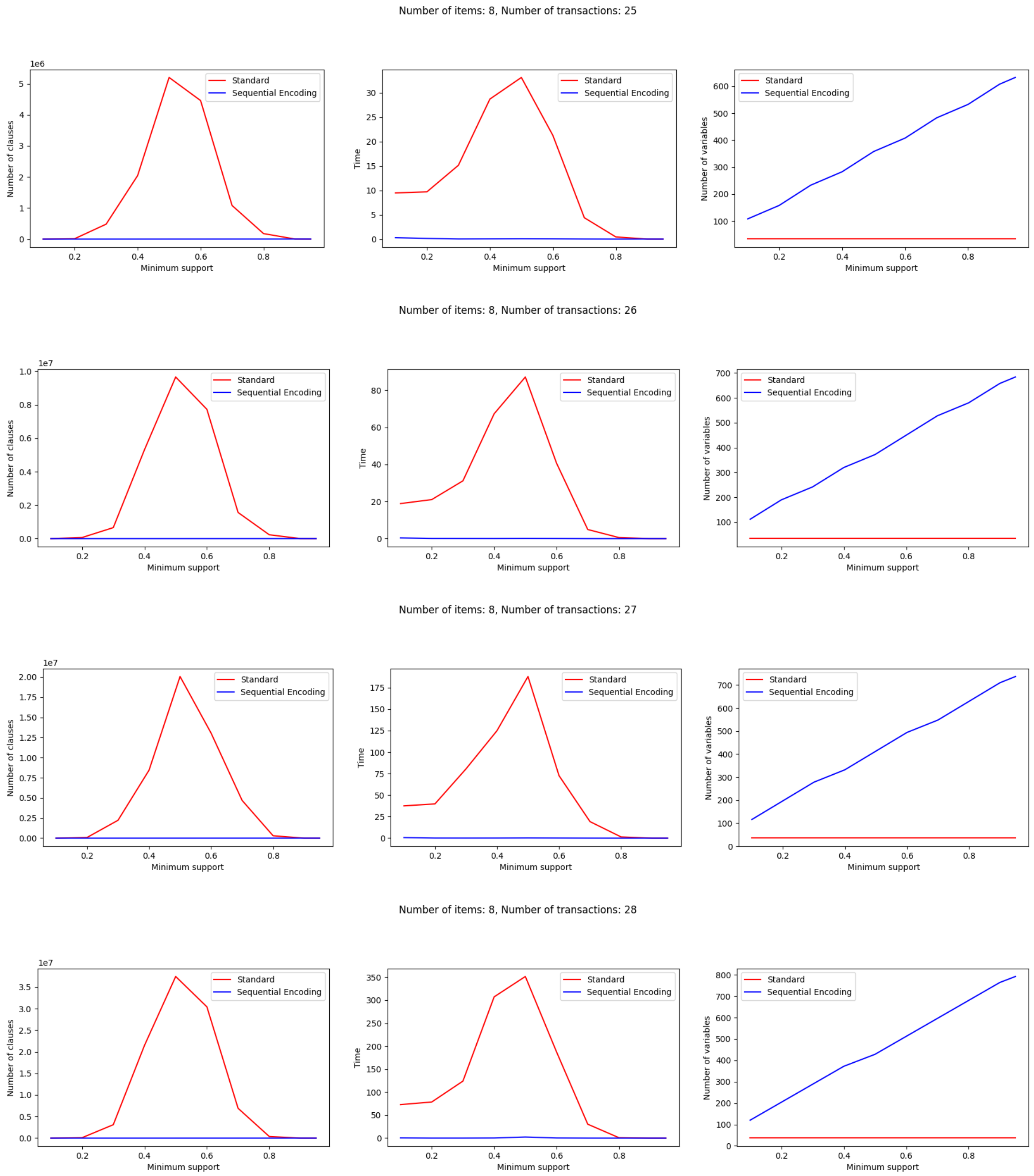


Figure 4.4: Comparison of the number of variables in the CNF encoding of the optimization problem using the sequential encounter encoding and the standard encoding with other number transactions.

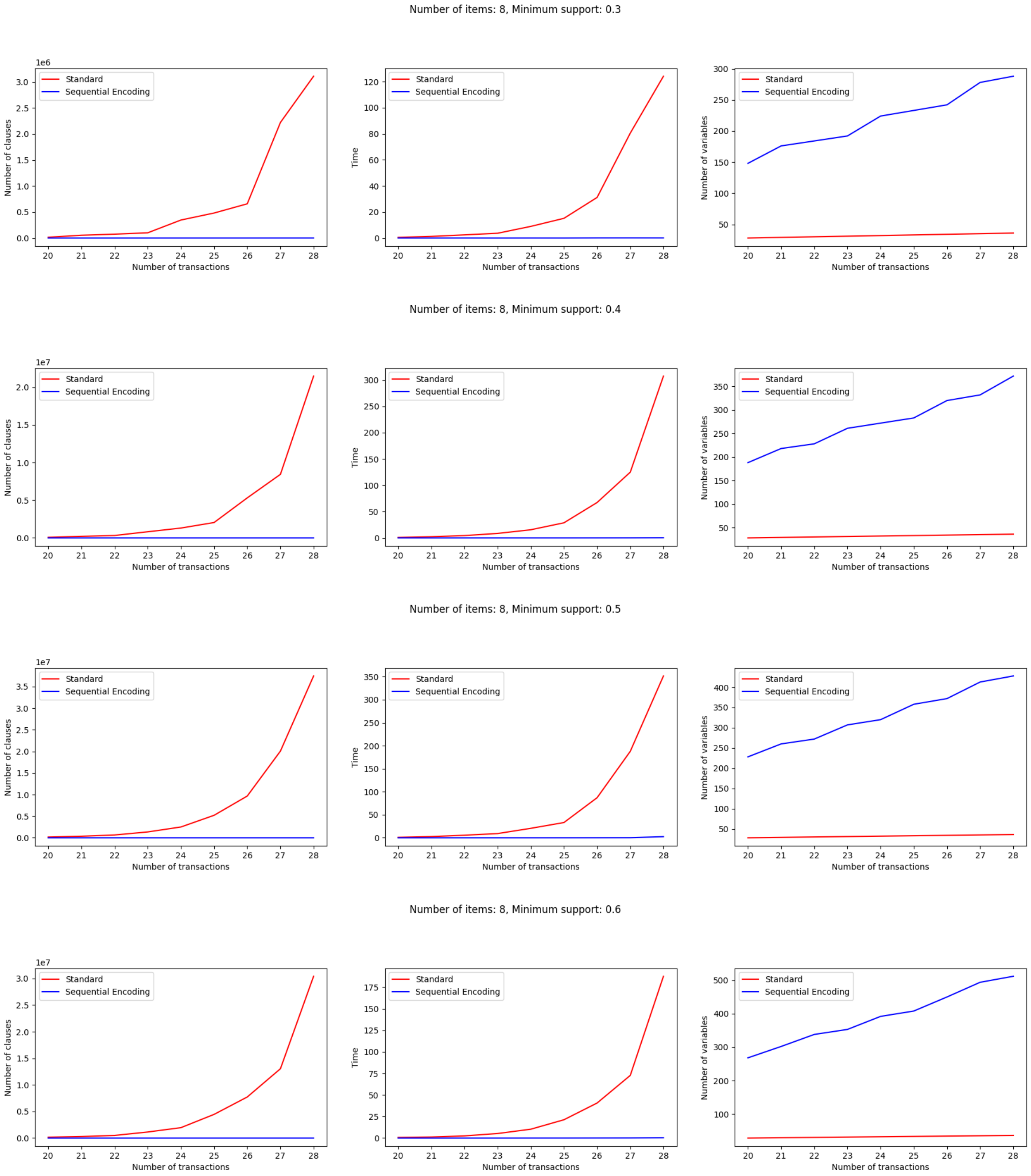


Figure 4.5: Comparison of the number of variables in the CNF encoding of the optimization problem using the sequential encounter encoding and the standard encoding with other minsupp in the same number of transactions.

This observation underscores the efficiency of the sequential encounter encoding method in striking a balance between the complexity of the problem and the computational resources required. Despite the slight increase in variables, the overall computational overhead remains substantially lower, making the sequential encounter encoding method a promising approach for tackling large-scale constraint satisfaction problems.

In addition, the experiments were conducted using various combinations of parameters to evaluate the performance of the sequential encounter encoding and the standard encoding in solving the optimization problem. By varying the number of transactions from 26 to 28 and the minimum support from 0.1 to 0.9, we aimed to assess the scalability and efficiency of both encoding methods across different problem sizes and constraints.

- Number of items: 8
- Number of transactions: 26 to 28
- Minimum Support: 0.1 to 0.9

Table 4.2 presents the results obtained from these experiments. It provides a comprehensive comparison of the number of variables, clauses, solutions, and time taken for each combination of parameters using both the standard encoding and the sequential encounter encoding.

Table 4.2: Comparison of the number of variables, clauses, solutions, and time taken to find all solutions using the sequential encounter encoding and the standard encoding

Trans	**Min Supp**	**Standard Encoding**				**Sequential Counter**			
		Vars	**Clauses**	**Sols**	**Time**	**Vars**	**Clauses**	**Sols**	**Time**
26	0.20	34	65,935	27	21.03	190	**774**	27	**0.11**
26	0.30	34	657,955	17	31.15	242	**987**	17	**0.10**
26	0.40	34	5,311,890	7	67.28	320	**1,314**	7	**0.08**
26	0.50	34	9,657,855	6	87.12	372	**1,537**	6	**0.13**
26	0.60	34	7,726,315	2	40.67	450	**1,879**	2	**0.09**
26	0.70	34	1,562,430	1	4.92	528	**2,230**	1	**0.02**
26	0.80	34	230,385	1	0.57	580	**2,469**	1	**0.01**
26	0.90	34	2,755	1	0.01	658	2,835	1	0.01
26	0.95	34	480	1	0.00	684	**2,959**	1	**0.01**
27	0.10	35	499	89	37.55	116	**467**	89	**0.68**
27	0.20	35	80,878	35	39.81	197	**791**	35	**0.13**
27	0.30	35	2,220,223	18	80.47	278	**1,124**	18	**0.11**
27	0.40	35	8,436,433	8	124.90	332	**1,351**	8	**0.13**
27	0.50	35	20,058,448	7	187.88	413	**1,699**	7	**0.24**
27	0.60	35	13,038,043	3	72.68	494	**2,056**	3	**0.14**
27	0.70	35	4,686,973	1	19.21	548	**2,299**	1	**0.05**
27	0.80	35	296,158	1	1.43	629	**2,671**	1	**0.01**
27	0.90	35	3,073	1	0.01	710	**3,052**	1	**0.01**
27	0.95	35	499	1	0.00	737	3,181	1	0.01
28	0.10	36	547	58	72.94	120	**500**	58	**0.49**
28	0.20	36	98,449	29	78.61	204	**836**	29	**0.08**
28	0.30	36	3,108,274	13	124.10	288	**1,181**	13	**0.09**
28	0.40	36	21,474,349	9	307.41	372	**1,535**	9	**0.30**
28	0.50	36	37,442,329	6	351.73	428	**1,776**	6	**2.41**
28	0.60	36	30,421,924	3	187.64	512	**2,145**	3	**0.34**
28	0.70	36	6,907,069	1	30.33	596	**2,523**	1	**0.07**
28	0.80	36	376,909	1	0.91	680	**2,910**	1	**0.01**
28	0.90	36	3,445	1	0.01	764	**3,306**	1	**0.01**
28	0.95	36	547	1	0.00	792	3,440	1	0.01

From the table 4.2, we can observe that as the minimum support increases, the number of clauses and variables in the CNF encoding generally tends to increase for both encoding methods. However, the sequential encounter encoding consistently generates a significantly smaller number of clauses and variables compared to the standard encoding. This reduction in the size of the CNF encoding demonstrates the efficiency and effectiveness of the sequential encounter encoding method in representing the optimization problem.

Furthermore, the table also shows the time taken to find all solutions using each encoding method. It is evident that the sequential encounter encoding outperforms the standard encoding in terms of time efficiency. Even as the complexity of the problem increases with higher minimum support values, the sequential encounter encoding method demonstrates faster solution-finding times. This highlights the advantage of using the sequential encounter encoding method for solving large-scale constraint satisfaction problems.

Overall, the experimental results support the superiority of the sequential encounter encoding method over the standard encoding method in terms of both the size of the CNF encoding and the time efficiency of finding solutions. These findings validate the effectiveness and scalability of the sequential encounter encoding approach in solving optimization problems with varying problem sizes and constraints.

4.2.2 Sequential Encounter Encoding on Real-World Dataset

We delve into the empirical evaluation of the sequential encounter encoding technique within the domain of frequent itemset mining. To ascertain the effectiveness and efficiency of this method, a series of comprehensive experiments were conducted using authentic datasets procured from the Frequent Itemset Mining Implementations (FIMI) and Constraint Programming[**constraint˙programming**] for Itemset Mining (CP4IM) repositories.

The initial step involved an extensive preprocessing phase, wherein the datasets were meticulously formatted to ensure compatibility with the encoding algorithms. Subsequent to this preprocessing, the datasets were encoded using only the sequential encounter encoding because the standard encoding method is not feasible due to the large number of variables and clauses generated.

The datasets selected for the experimental study are succinctly summarized in Table 4.3. The table provides an insightful juxtaposition of key metrics such as the number of variables, clauses, and the computational time expended for each dataset.

Table 4.3: Comparison of the number of variables, clauses, solutions, and time taken using the sequential encounter encoding and the standard encoding

Dataset	Items	Trans	Min Supp	Sequential Counter Encoding		
				Vars	Clauses	Time
zoo-1	36	101	0.10	1,245	5,489	0.06
primary-tumor	31	336	0.10	11,790	50,304	0.10
vote	48	435	0.10	19,614	81,411	0.17
soybean	50	630	0.10	40,360	165,745	0.36
chess	75	3,196	0.10	1,025,990	4,212,750	9.45
chess	75	3,196	0.20	2,048,710	8,455,790	18.16
chess	75	3,196	0.30	3,068,234	12,787,491	43.26
chess	75	3,196	0.40	4,090,954	17,235,011	38.59
chess	75	3,196	0.50	5,110,478	21,770,553	52.63
mushroom	119	8,124	0.10	6,613,018	26,853,806	65.64
mushroom	119	8,124	0.20	13,209,706	54,226,732	187.69
mushroom	119	8,124	0.30	19,814,518	82,293,931	798.24

Observing Table 4.3, it becomes evident that the sequential encounter encoding method demonstrates varied performance benefits across different datasets. For instance, the 'zoo-1' dataset, characterized by 36 items and 101 transactions, was encoded with significantly fewer variables and clauses, resulting in an impressively swift computation time of only 0.06 seconds. This marked efficiency showcases the potential of the sequential encounter encoding in dealing with smaller datasets.

Conversely, as we scrutinize datasets with a larger number of transactions and items, such as 'chess' and 'mushroom', we notice a discernible trend of increased complexity. The 'chess' dataset at a 0.10 minimum support level demanded over a million variables in the standard encoding approach, with a consequent computational time of approximately 9.45 seconds. The escalation in complexity is palpable when the minimum support is altered, impacting the number of solutions and, inevitably, the time required for computation.

The 'mushroom' dataset further elucidates the scalability challenges, where the number of variables peaks for the standard encoding at a 0.30 minimum support, with an accompanying computational time that extends to a substantial 798.24 seconds. This starkly contrasts with the lesser demanding 'zoo-1' and 'primary-tumor' datasets, underscoring the necessity of an encoding method that can adeptly adapt to varying dataset sizes and complexities.

The table emphatically highlights the nuanced relationship between dataset characteristics and encoding performance. The sequential encounter encoding method's capability to effectively manage the number of variables and clauses without compromising on the discovery of solutions is a testament to its robustness. Additionally, the encoding method's impact on computational time is of paramount importance, as it directly correlates with the practicality of the approach in real-world applications.

In summary, the experimental evaluation substantiates the hypothesis that sequential encounter encoding can be a potent alternative to standard encoding in itemset mining, particularly when tailored to the dataset at hand. These findings advocate for a discerning application of encoding techniques in itemset mining, with the sequential encounter encoding method emerging as a significant contender in the field.

CHAPTER 5

Conclusions

5.1 Conclusion

In conclusion, this study has provided a comprehensive exploration of itemset mining tasks and introduced innovative techniques for optimizing the encoding process. We began by elucidating the fundamentals of itemset mining, highlighting its significance in various domains such as retail, healthcare, and bioinformatics. The encoding of itemset mining problems into SAT instances was meticulously examined, showcasing the sequential encounter encoding approach as a promising alternative to traditional methods. By leveraging the sequential nature of transactional data, sequential encounter encoding offers a more efficient and scalable solution for itemset mining tasks.

Furthermore, we delved into the application of the standard method C_{n-k+1} for encoding itemset mining constraints, shedding light on its limitations and challenges, particularly in handling large datasets. Through a comparative analysis, we demonstrated the advantages of sequential encounter encoding in reducing computational complexity and improving mining efficiency.

Moving forward, there are several avenues for future research. Expanding the application of sequential encounter encoding to different problem domains and refining the encoding process to accommodate diverse datasets are promising areas for exploration. Additionally, investigating hybrid approaches that combine sequential encounter encoding with other optimization techniques could yield further improvements in mining performance.

Overall, this study contributes to the advancement of itemset mining methodologies and lays the groundwork for future research endeavors aimed at enhancing the efficiency and effectiveness of mining algorithms. By embracing innovative approaches such as sequential encounter encoding, we can unlock new insights from large-scale datasets and drive progress in data mining and analytics.

5.2 Future Work

In future work, it would be valuable to explore the following areas:

Refinement of Encoding Methods: Further refinement and exploration of encoding techniques, particularly sequential encounter encoding, to improve efficiency and scalability.

Scalability Challenges: Addressing scalability challenges associated with large datasets by developing encoding methods optimized for handling massive data volumes.

Domain-Specific Applications: Investigating the application of encoding techniques in specific domains to uncover domain-specific insights and optimizations.

Integration with Machine Learning: Exploring the integration of encoding techniques with machine learning algorithms to enhance the accuracy and efficiency of data mining tasks.

Benchmarking and Evaluation: Conducting benchmarking studies and evaluations to compare the performance of different encoding methods across various datasets and problem domains.