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
SAT	Satisfiability
UNSAT	Unsatisfiability
CNF	Conjunctive Normal Form
ALO	At Least One
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 Itemset Mining Tasks

1.1.1 Overview

Frequent item sets are a key technique in the realm of data mining, 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.

For example, with 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

Table 1.1: Example of a dataset of transactions

With minimum support is 3, we need to find all itemsets appearing in at least 3 transactions and return the following result:

- Itemset 1: {apple, cherry} in transactions [1, 3, 5]
- Itemset 2: {apple} in transactions [1, 2, 3, 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.

1.2 SAT Encoding

1.2.1 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).

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.

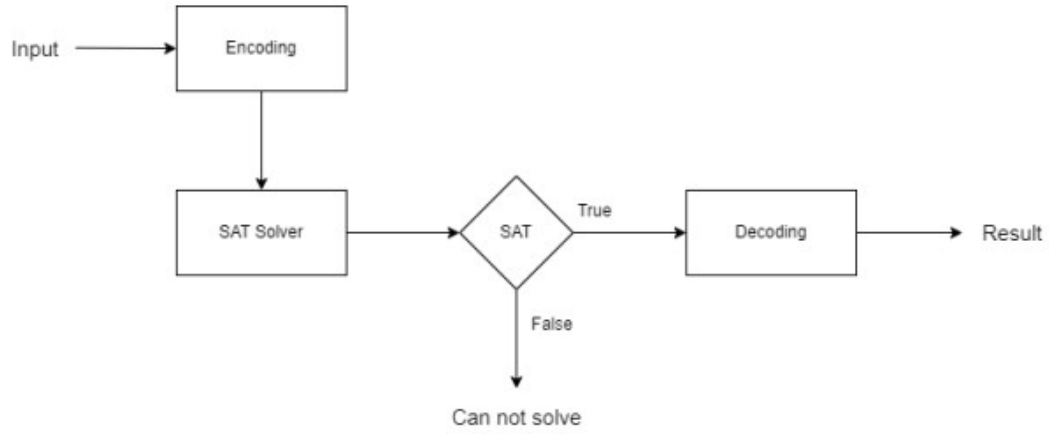

Figure 1.1: SAT Encoding Process

1.2.2 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.

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

1.2.3 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) 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 [2], Lingeling [3], Glucose [4], RSat [5].

SAT is proven to be 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 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.

1.2.4 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.

1.3 Technical background

1.3.1 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 $\neg$ P. The negation is true when P is false.

Truth table:

P	$\neg$ P
T	F
F	T

Table 1.2: Truth table of negation (NOT)

Conjunction

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

Truth table:

P	Q	P $\wedge$ Q
T	T	T
T	F	F
F	T	F
F	F	F

Table 1.3: Truth table of conjunction (AND)

Disjunction

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

Truth table:

P	Q	$P \vee Q$
T	T	T
T	F	T
F	T	T
F	F	F

Table 1.4: Truth table of disjunction (OR)

Implication

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

Truth table:

P	Q	$P \rightarrow Q$
T	T	T
T	F	F
F	T	T
F	F	T

Table 1.5: Truth table of implication

Biconditional

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

Truth table:

P	Q	$P \leftrightarrow Q$
T	T	T
T	F	F
F	T	F
F	F	T

Table 1.6: Truth table of biconditional

1.3.2 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):

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

Table 1.7: Truth table of CNF

1.3.3 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 1.8.

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

Table 1.8: Sample dataset of transactions in binary format

- 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.

CHAPTER 2

SAT-based Encoding of Itemset Mining

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.

2.1 Base constraints

To resolve the itemset mining problem, we using the SAT encoding approach. 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 $D = (1,T_1),...,(m,T_m)$ 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

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

Table 2.1: Example of a dataset of transactions after convert

The itemset mining problem will be defined as:

$$q_1 \leftrightarrow (\neg p_d)$$
$$q_2 \leftrightarrow (\neg p_b \wedge \neg p_c)$$
$$q_3 \leftrightarrow (\neg p_b \wedge \neg p_d)$$
$$q_4 \leftrightarrow (\neg p_a \wedge \neg p_d)$$
$$q_5 \leftrightarrow (\neg p_b)$$
$$q_1 + q_2 + q_3 + q_4 + q_5 \geq \lambda$$

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

2.2 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.

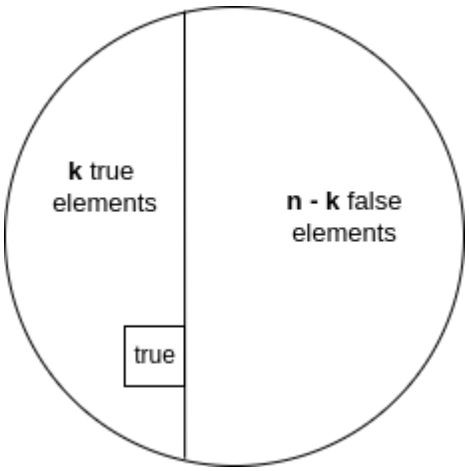

Figure 2.1: Illustration of the standard method C_{n-k+1}

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:

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

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.

2.3 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

3.1 Sequential Encounter Encoding

3.2 Optimization CNF Encoding using Sequential Encounter Encoding

First, we add following constraint for relationship between q and r:

$$\begin{aligned} q_i &\rightarrow r_{i1} \quad \forall i \in [1, n] \\ \neg q_i &\rightarrow \neg r_{ii} \quad \forall i \in [1, n] \end{aligned} \tag{3.1}$$

This constraint guarantees that if q_i is true, then r_{i1} must be true. And if q_i is false, r_{ii} must be false.

Secondly, we encode

$$\neg r_{ij} \quad \forall i \in [1, \lambda - 1], j \in [i + 1, \lambda] \tag{3.2}$$

Thirdly, we encode

$$\neg r_{i-1,j} \rightarrow r_{ij} \quad \forall i \in [2, n], j \in [1, \lambda] \tag{3.3}$$

Fourthly, we encode

$$\begin{aligned} q_i \wedge r_{i-1,j-1} &\rightarrow r_{ij} \quad \forall i \in [2, n], j \in [2, \lambda] \\ \neg q_i \wedge \neg r_{i-1,j} &\rightarrow \neg r_{ij} \quad \forall i \in [2, n], j \in [1, \lambda] \\ \neg r_{i-1,j} \wedge \neg r_{i-1,j-1} &\rightarrow \neg r_{ij} \quad \forall i \in [2, n], j \in [2, \lambda] \end{aligned} \tag{3.4}$$

Finally, we encode

$$\begin{gathered} r_{n-1,\lambda} \vee (q_\lambda \wedge r_{n-1,\lambda-1}) \\ \neg q_n \rightarrow r_{n-1,\lambda} \\ \neg q_i \wedge r_{j,\lambda} \rightarrow r_{i-1,\lambda} \quad \forall i \in [\lambda+1, n] \end{gathered} \tag{3.5}$$

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`.

Parameters: The generator takes the following parameters:

- `transactions`: the number of transactions in the dataset
- `items`: the number of items in the dataset
- `min_support`: the minimum support threshold for frequent itemsets
- `output_file`: the path of the output file

Output: The generator outputs a dataset in the annotated transaction format. Each line represents a transaction, with items separated by spaces.

How to run: Dataset can be generated by manual script or during benchmarking.

4.1.3 Real-World Datasets

The datasets used in the experiments are coming from FIMI[1] and CP4IM[2]. The characteristics of the considered datasets are given in Table 4.1.

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

Table 4.1: Characteristics of the considered datasets

The FIMI 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 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).

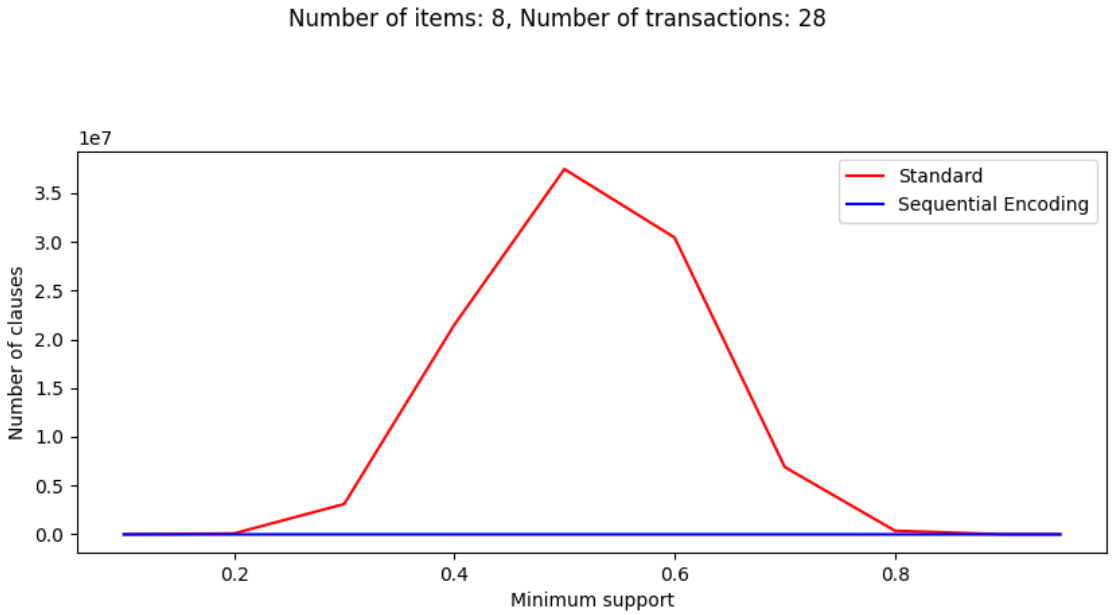

Figure 4.1: Comparison of the number of clauses in the CNF encoding of the optimization problem using the sequential encounter encoding and the standard encoding.

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.

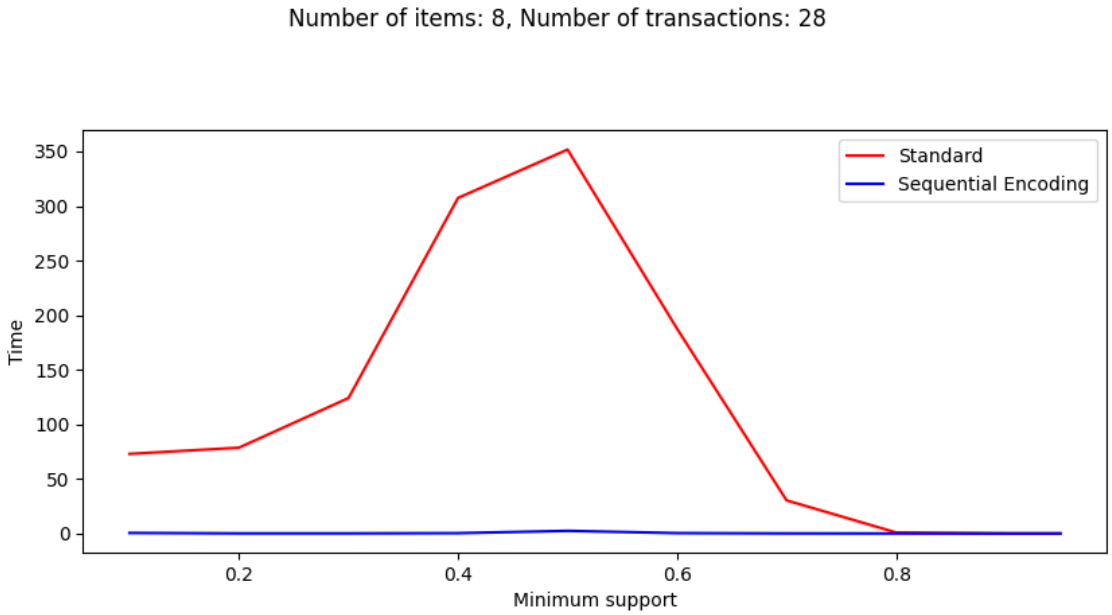

Figure 4.2: Comparison of the time taken to find all solutions using the sequential encounter encoding and the standard encoding

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.

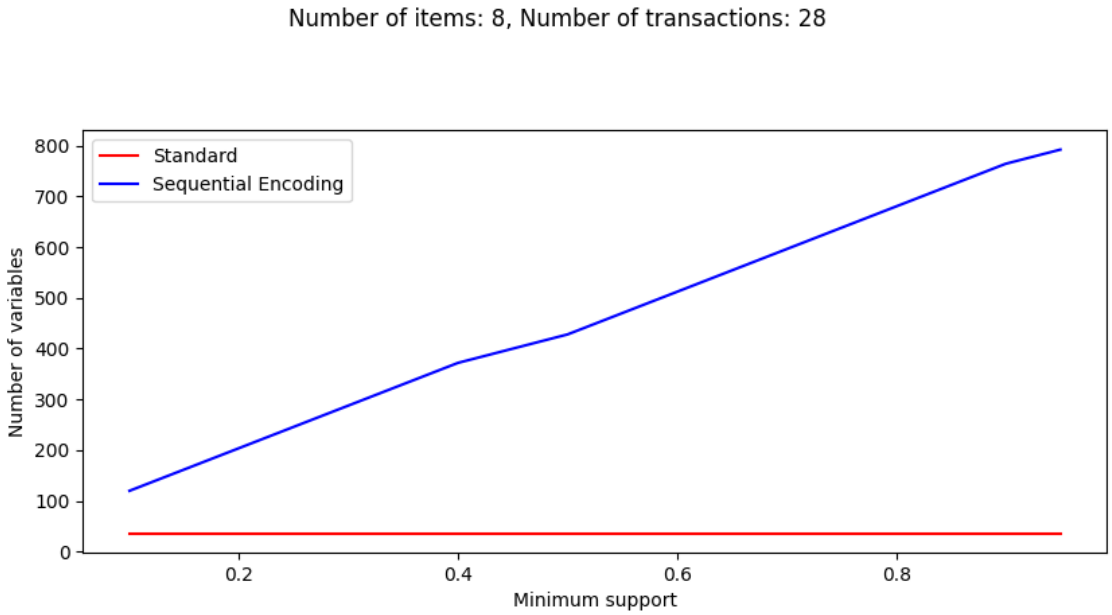

Figure 4.3: Comparison of the number of variables in the CNF encoding of the optimization problem using the sequential encounter encoding and the standard encoding.

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.

Trans	Min Supp	Standard Encoding				Sequential Encounter			
		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

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

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 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.

Dataset	Items	Trans	Min Supp	Standard 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

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

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.

Conclusions