



Exploring the trie of rules: a fast data structure for the representation of association rules

Mikhail Kudriavtsev¹ · Vuong M. Ngo⁴ · Mark Roantree³ ·
Marija Bezbradica² · Andrew McCarren³

Received: 28 May 2024 / Revised: 30 September 2024 / Accepted: 1 October 2024 /
Published online: 9 October 2024
© The Author(s) 2024

Abstract

Association rule mining techniques can generate a large volume of sequential data when implemented on transactional databases. Extracting insights from a large set of association rules has been found to be a challenging process. When examining a ruleset, the fundamental question is how to summarise and represent meaningful mined knowledge efficiently. Many algorithms and strategies have been developed to address issue of knowledge extraction; however, the effectiveness of this process can be limited by the data structures. A better data structure can sufficiently affect the speed of the knowledge extraction process. This paper proposes a novel data structure, called the Trie of rules, for storing a ruleset that is generated by association rule mining. The resulting data structure is a *prefix-tree graph structure made of pre-mined rules*. This graph stores the rules as paths within the prefix-tree in a way that similar rules overlay each other. Each node in the tree represents a rule where a consequent is this node, and an antecedent is a path from this node to the root of the tree. The evaluation showed that the proposed representation technique shows significant value. It compresses a ruleset with no data loss and benefits in terms of time for basic operations such as searching for a specific rule, which is the base for many knowledge discovery methods. Moreover, our method demonstrated a significant improvement in graph traversal time compared to traditional data structures.

Keywords Data mining · Association rule mining · Trie of rules · Knowledge extraction · Data representation

1 Introduction

Modern organizations produce, gather, and store substantial volumes of data with the intention of using it to make strategic decisions. The level of complexity in making these decisions can often be exasperated by the depth of understanding required to interpret relationships within the data. Retailers are a typical example, as they utilize regular sequences or patterns in their customer transactions in order to make marketing or supply chain management strategic decisions.

Marija Bezbradica and Andrew McCarren are Joint senior author(s).

Extended author information available on the last page of the article

Association Rule Mining (ARM) is a data mining technique that extracts frequent patterns and relationships among items in a dataset. The process of ARM involves obtaining the dataset, applying an appropriate frequent item mining algorithm to it, and extracting association rules from the frequent itemsets generated by the algorithm. These association rules are stored in a data structure called a *ruleset*, which can be very large and computationally expensive to process. The data structure used for a ruleset is critical, as it determines the efficiency and effectiveness of knowledge extraction methods that are based on traversing the ruleset (Alasow et al, 2020; Bui-Thi et al., 2020; Li & Wu, 2014).

Formally, a rule can be described as an implication $A \rightarrow C$ where A is the antecedent, and C is the consequent, A and C are sets of items from $I = i_1, i_2, \dots, i_m$ where $A \cap C = \emptyset$. Each rule is derived from a database $D = t_1, t_2, \dots, t_n$, where each transaction t_i is a set of items from I .

A rule is usually described by ARM metrics (Geng & Hamilton, 2006; Wu et al., 2010; Luna et al., 2018) such as Support and Confidence, which play a crucial role in knowledge extraction. Support is the proportion of transactions in the database containing both the antecedent and the consequent, while Confidence measures the proportion of transactions containing the antecedent that also contain the consequent (Agrawal et al, 1993; Bayardo & Agrawal, 1999).

Association rule mining *algorithms* were introduced by Agrawal et al (1993) to identify the buying patterns of retail customers. These techniques are based on the exploration of frequently occurring sequences in large-scale databases and have since been widely used in many application areas such as customer behavior analysis, software engineering, medical diagnostics, visual analytics, etc. (Yazgana & Kusakci, 2016; Shaukat Dar & Zaheer, 2015; Ghafari & Tjortjis, 2019; Liu & Shen, 2016; Máša & Rauch, 2024). Generally, the output of an ARM algorithm is a list of rules or *association ruleset* portrayed in a text-based manner, requiring further analysis.

The major difference between an *association rule* and a *frequent sequence* should be emphasized because it is vital for the future explanation of the proposed methodology. A frequent sequence is a list of items that appears as an output of an ARM algorithm. An association rule is a structure made from the frequent sequence by splitting this sequence into an antecedent and consequent. For example, a frequent sequence (a, b, c, d) is mined with an ARM algorithm; rules such as $(a, b) \rightarrow (c, d)$ or $(c, d) \rightarrow (a, b)$ can be constructed using this sequence, where the antecedent and consequent can be any valid combination of items from the sequence. Each rule is derived from the same frequent sequence, so a single frequent sequence can give rise to various association rules without the need for representing the same frequent itemset with multiple sequences. As a result of that difference, evaluation metrics applied to association rules cannot be used for frequent sequences (Agrawal et al, 1993; Brin et al., 1997). The only exception is Support (Agrawal et al, 1993); it reflects the frequency of all items together in a rule or a sequence.

While a considerable amount of attention has been directed toward data structures for *frequent sequence sets* in the context of ARM (Bodon & Rónyai, 2003; Grahne & Zhu, 2003; Coenen et al., 2004), there has been limited focus on identifying a suitable data structure that enables both storage of *sets of association rules* and the efficient extraction of knowledge. As ARM continues to remain a popular field of study, identifying a data structure that can effectively store and represent association rules would improve current knowledge extraction in state-of-the-art approaches.

Contribution. In this paper, we present a novel use of the prefix-tree for storing a ruleset, rather than a frequent sequence set, which is its typical use case. It is essential to emphasize that the primary focus of our work is the representation of the association ruleset obtained

from the mining process. We do not delve into the details of the mining process itself, but rather concentrate on enhancing the efficiency and effectiveness of knowledge extraction methods that rely on traversing the ruleset. Our contributions include:

1. **Trie of Rules Data Structure Development:** The paper introduces the Trie of Rules data structure as a novel approach for efficiently storing and manipulating *association rules*. This data structure aims to enhance the efficiency and comprehensibility of rule-based systems by providing a highly optimized data structure.
2. **Enhancing ruleset traversal time efficiency:** Our exploration of the Trie of Rules data structure in contrast to conventional approaches such as storing rules in pandas DataFrame or NumPy array highlights the pivotal role of specialized data structures in optimizing rule management. This analysis illuminates how adopting tailored data structures can substantially diminish time complexity in knowledge discovery tasks, thus facilitating expedited and more efficient extraction of insights from association rules.
3. **Confidence Calculation for Compound Consequents:** the Trie of Rules data structure introduces a novel property that has the potential to significantly enhance further exploration of knowledge and increase speed efficiency when querying the ruleset: the ability to calculate Confidence for rules with compound consequents directly from the graph structure. This unique feature not only streamlines the process of evaluating complex rules but also facilitates faster traversal and exploration of the ruleset, ultimately improving the efficiency of knowledge extraction in association rule mining

Paper structure The remainder of this paper is structured as follows: Section 2 provides a comprehensive review of existing literature, focusing on the data structures employed for storing transactions, frequent sequences, and association rules, along with their respective advantages and limitations. Section 3 delves into our methodology, offering algorithmic descriptions and an illustrative example to elucidate its workings. In Section 4, we present the results of our evaluation methodology, including comparisons with existing popular solutions. Finally, Section 5 concludes the paper by summarizing key findings and outlining potential avenues for future research. Six detailed algorithms referenced in the main part of the paper are presented in the Appendix.

2 Related works

In Association Rule Mining, the efficient storage and retrieval of both transactional data and association rules are crucial for successful knowledge extraction and decision-making processes. To address this need, various data structures and algorithms have been proposed for storing transactions and association rules. The following sections explore prominent data structures tailored for each purpose. The first subsection focuses on data structures for storing transactions, while the second subsection delves into data structures specifically designed for storing association rules.

2.1 Data structures for storing transactions

In the domain of Association Rule Mining, efficient storage and retrieval of transactional data are essential for identifying frequent itemsets, which serve as the basis for generating association rules. Various data structures have been proposed and employed for this purpose, each offering unique advantages and limitations. Here, we explore several prominent data

structures and algorithms tailored for storing transactions in ARM, highlighting their distinct characteristics and applicability.

2.1.1 Linked lists or vertical database

Zaki et al. proposed parallel algorithms for the discovery of association rules, utilizing a vertical database layout (Zaki et al., 1997). This layout organizes transactions such that each item is followed by its transaction ID list (TID-List), containing the list of transaction identifiers containing that item. While efficient for support-based frequent set mining tasks, this data structure is not explicitly designed for managing association rules or incorporating additional metrics beyond Support.

Liu et al. proposed an improved association rules mining method utilizing a trifurcate linked list storage structure for the directed itemsets graph (Liu et al., 2012). This structure organizes data related to frequent itemsets, such as item names, TID-Lists, and Support numbers, to enhance the efficiency of the mining algorithm. However, its focus remains on organizing data for mining frequent itemsets rather than managing association rules directly.

2.1.2 Trees

Coenen et al. proposed the utilization of T-Trees and P-Trees as data structures for storing itemsets in the context of ARM (Coenen et al., 2004). These structures, although not directly employed for storing association rules themselves, play a crucial role in efficiently managing the itemsets necessary for generating association rules. T-Trees and P-Trees are optimized for storing and managing frequent itemsets and are particularly effective in conjunction with ARM metrics such as Support. However, their application is limited solely to the mining process, as they do not manage actual rules or utilize metrics such as Confidence.

Vu and Alagband introduced the FEM algorithm, which primarily focuses on mining frequent patterns from transactional databases (Vu & Alagband, 2011). The algorithm utilizes the FP-tree data structure for mining frequent patterns and TID-Lists for storing transactions. While the FEM algorithm efficiently discovers frequent itemsets and patterns in transactional data, it does not explicitly store association rules and metrics such as Confidence or Lift. Instead, its focus remains on mining frequent itemsets rather than managing association rules.

Shabtay et al. proposed the guided FP-Growth algorithm, incorporating two main data structures to optimize the mining process (Shabtay et al., 2021). These structures, namely the FP-Tree and TIS-Tree (Targeted Item-Set Tree), enable the algorithm to efficiently mine multitude-targeted item-sets and class association rules in imbalanced data. While the guided FP-Growth algorithm focuses on mining specific item-sets of interest, it does not directly store association rule mining (ARM) metrics beyond Support. Instead, it prioritizes the optimization of the mining process to extract valuable insights from transactional data efficiently.

2.1.3 Items graphs

Koh et al. presented the Items Graph, a data structure constructed based on interactions between items in the dataset (Koh et al., 2010). Each node in the graph represents an item, and the edges between nodes indicate the strength of interactions or Support of the itemsets. Yen

and Chen discussed a Graph-Based Approach for Discovering Various Types of Association Rules (Yen & Chen, 2001). Both papers highlight how graphs are fundamental data structures to represent relationships between items and patterns in the dataset. The algorithms construct an association graph where each node represents an item, and edges between nodes indicate associations based on their co-occurrence in transactions. After constructing the association graph, the algorithm traverses it to generate all large itemsets. Furthermore, the graph may capture hierarchical relationships between items or concepts. While effective for itemset-based analysis, these structures are limited to support-based tasks and do not directly handle association rule storage or metrics.

2.1.4 Summary

While these data structures and algorithms excel in efficiently storing and managing transactional data for ARM, their limitations in terms of handling association rules and incorporating additional metrics beyond Support must be stated. They primarily serve the purpose of mining frequent itemsets and patterns, laying the groundwork for subsequent rule generation and knowledge extraction phases of ARM. Understanding their strengths and weaknesses is crucial for selecting the most suitable approach for a given ARM task. The key differentiator of our work lies in addressing these limitations through an approach that targets the efficient storage and manipulation of *association rules*, distinct from transactions or frequent sequences.

2.2 Data structures for storing association rules

In the domain of Association Rule Mining (ARM), efficient storage and retrieval of association rules are essential for knowledge extraction and subsequent decision-making processes. Various data structures and algorithms have been proposed and utilized for storing **association rules**, each with its own set of advantages and limitations. Here, we explore several prominent data structures and algorithms tailored for storing association rules, highlighting their distinct characteristics and applicability.

2.2.1 Data structures in R and python

Hahsler, Grün, and Hornik discussed the *R* *arules* package, a computational environment for mining association rules and frequent itemsets (Hahsler et al., 2005). The package utilizes well-designed *DataFrame* structures to efficiently store association rules, including classes such as transactions, TID-Lists, itemsets, and rules, providing a unified interface for working with sets of itemsets or rules.

Stancin and Jovic provided an overview of free Python libraries for data mining and big data analysis (Stancin & Jovic, 2019). While the paper mentions various data structures such as lists, arrays, dictionaries, and *DataFrames* for storing association rules, it emphasizes the use of *pandas*, a popular library for handling tabular data structures, including *DataFrames*.

Hahsler introduced *ARULESPY*, a Python package for exploring association rules and frequent itemsets (Hahsler, 2023). It provides functionalities to convert data into *pandas DataFrame*s for efficient processing and analysis. In a related paper, Hahsler discussed grouping association rules using Lift (Hahsler, 2016). While the specific data structure used to store the ruleset is not mentioned, the authors highlighted the challenges of dealing with large sets of association rules and the computational complexity involved in clustering strategies.

Given that the ARULESPY library is based on pandas DataFrames, it can be inferred that DataFrames are a common choice for knowledge discovery methods in ARM.

2.2.2 Rules graphs

De Padua et al. conducted an analysis on community detection and clustering algorithms for post-processing association rules (De Padua et al., 2018). While the exact method for storing association rules is not specified, the paper discusses constructing a similarity matrix between nodes to facilitate clustering and community detection techniques. This method involves representing each rule as a node and constructing a graph where each node represents a rule, enabling clustering and community detection based on rule similarity. However, this approach is limited to specific clustering tasks and does not provide insights into the efficiency of the underlying data structure.

Jin et al. proposed the use of a graph data structure in their Bundle Graph Convolutional Network (BGCN) model for multi-behavior recommendation (Jin et al., 2020). The authors construct a heterogeneous graph $G = (V, E)$ where nodes represent users, bundles, and items, and edges capture interactions such as user-bundle, user-item, and bundle-item interactions. This approach allows for the modeling of intricate relationships between users, items, and bundles, facilitating the extraction of valuable association rules for bundle recommendations. However, this approach requires additional domain knowledge about users and more comprehensive data, which may limit its application.

2.2.3 Rules

Berrado and Runger discussed the use of metarules to organize and group discovered association rules (Berrado & Runger, 2007). Although the paper proposes storing metarules as a graph, focusing on organizing and grouping association rules, it does not explicitly discuss the efficiency of the data structure used for storage.

Bui-Thi et al. introduced MoMAC, a multi-objective optimization approach for combining multiple association rules into an interpretable classification model (Bui-Thi et al., 2022). In this approach, association rules are stored in a rule list data structure, providing an ordered set of rules. While efficient for classifier size and prediction accuracy optimization, it primarily focuses on rule combination rather than the storage and representation of individual association rules.

2.2.4 Others

Li and Wu presented a multi-tier granule mining approach for the interpretation of association rules (Li & Wu, 2014). This approach utilizes a multi-tier structure to represent association rules, consisting of sets of granules at different tiers and association mappings that illustrate relationships between these granules. While effective for generating granules and interpreting association rules, it does not explicitly store the original association rules, focusing instead on representing them in terms of granules.

Jentner and Keim discussed visualization and visual analytic techniques for patterns (Jentner & Keim, 2019). While various visualization approaches were explored, including matrix

representation, graph-based representation, and Fp-tree representation, the underlying data structures used for storing association rules were not analyzed.

Bui-Thi et al. proposed a methodology for clustering association rules to build beliefs and discover unexpected patterns (Bui-Thi et al., 2020). The study utilizes numerical feature vectors to represent association rules, capturing both semantic and lexical relationships between patterns. While this methodology enhances rule clustering, it requires domain knowledge information about items, limiting its application. Additionally, it only allows one application - clustering - as it does not store any ARM metrics or relationships between rules described by a correlation matrix of feature vectors.

Moahmmmed et al. introduced a methodology for clustering association rules in big datasets using Hadoop MapReduce (Moahmmmed et al., 2021). Although the document focuses on the methodology, algorithms, and performance evaluation, it does not explicitly mention the data structure used to store association rules. Common practices suggest that association rules in MapReduce frameworks are stored in key-value pairs or structured formats suitable for distributed computing environments like Hadoop MapReduce.

2.2.5 Summary

In summary, the field of data structures for storing association rules is not extensively explored compared to data structures for storing transactional data. While some methodologies focus on clustering association rules or utilizing multi-tier structures for interpretation, there is a lack of research on efficient data structures specifically tailored for storing and representing association rules. Most existing approaches either focus on rule combination or use generic data structures like DataFrames without optimizing for the unique characteristics of association rules. However, the development of specialized data structures explicitly designed for association rules is imperative. Such structures have the potential to significantly enhance speed, memory efficiency, and uncover implicitly hidden insights. Moreover, a structure that reveals and utilizes the unique structure of association rules can facilitate knowledge extraction and open the field for new methods in the realm of ARM. Further research is needed to develop specialized data structures that enhance the efficiency and effectiveness of knowledge extraction methods in a set of association rules.

3 Methodology

3.1 Trie of rules

Prefix-trees are popular in Association Rule Mining, particularly the FP-tree data structure, which is commonly used for storing frequent sequences and is known for its compactness and efficiency in traversal (Bodon & Rónyai, 2003; Han et al., 2004; Grahne & Zhu, 2003; Shahbazi & Gryz, 2022). However, prefix-trees have not been extensively explored as a data structure for *storing association rules* along with their metrics. Therefore, in this study, we propose a novel use of prefix-trees, also known as tries (Crochemore & Lecroq, 2009), called the "Trie of Rules", for representing a set of association rules that can serve as a viable alternative to a regular DataFrame. Our proposed data structure takes the form of a graph containing all rules and associated metric values. This structure avoids redundancy while also increasing the traverse speed. It is suitable for various graph-based knowledge extraction methods and visualizations, as well as for efficient storage and retrieval of rules.

To ensure that logically equivalent rules are stored consistently in the Trie of Rules, we preprocess the rules by sorting the items within the antecedent and consequent according to

their frequency in the frequent set before inserting them into the Trie. This approach ensures that rules such as $a \rightarrow b$ and $ab \rightarrow c$, as well as $a \rightarrow b$ and $ba \rightarrow c$, are treated identically and stored on the same path in the Trie.

The methodology for using the Trie of Rules can be explained through a three-step process (see Appendix for detailed algorithms):

1. **Step 1:** Apply an ARM algorithm on a transactional dataset to acquire a list of frequent sequences (dictionary of the form {list of items : Support}) and a ruleset (list of Rule objects - Algorithm 2).
2. **Step 2:** Preprocess the ruleset by sorting the items within the antecedent and consequent of each rule based on their frequency in the set of frequent sequences. Using Algorithm 1, create a Trie from the ruleset and frequent set from the previous step to form a trie object. This algorithm builds a trie (Algorithm 4) using the list of frequent sequences from Step 1. Each node (Algorithm 3) in the trie represents a rule. The rule is a path from the root to a desired node, where a consequent is the last node in the path, and an antecedent is all the previous nodes in the same path before the desired node (see Fig. 1).
3. **Step 3:** Label each node with the Support, Confidence, and other metrics relating to the corresponding rule (see Fig. 1) using Algorithm 6.

Algorithm 1 Creating a Trie from ruleset.

Description: This function creates a Trie data structure from a given set of rules and their corresponding frequent sequences.

```

1: function CREATE_TRIE(rule_set, frequent_sequences)  $\rightarrow$  Trie            $\triangleright$  rule_set is a list of rule objects,
   frequent_sequences is a dictionary with sequences of items as keys and their associated Support values
2:   trie  $\leftarrow$  TRIE()                                            $\triangleright$  Initialize an empty trie
3:   for each rule in rule_set do
4:     trie.insert_rule(rule, frequent_sequences)                  $\triangleright$  Insert the rule into the trie
5:   end for
6:   for each child in trie.root.children do
7:     trie.extend_node(child, frequent_sequences)            $\triangleright$  Calculate metrics starting the recursive traversal
   from the root node children, using the frequent sequences
8:   end for
9:   return trie
10: end function

```

To retrieve a certain rule, use Algorithm 7, Searching a Rule in the Trie.

3.2 An illustrative example

Let's consider an example from a simple dataset to illustrate the advantages of using the Trie of rules approach.

1. **Step 1:** In the first step, a ruleset is obtained through an ARM algorithm applied to the dataset. The process of acquiring this ruleset is user-specific and falls outside the scope of this work. The ARM algorithm yields a list of frequent sequences and a set of rules (see Table 1).
2. **Step 2:** The list of rules serves as the source dataset for creating a Trie of rules (see Algorithm 1). The trie is initialized with a root node (Null), followed by the insertion of rules (see Algorithm 5) from Table 1 into the trie, one by one. In this example, the first rule inserted is $(f, c, a \rightarrow m, p)$. The items of the rule are inserted into the trie as nodes. Figure. 2(a) shows the trie after the first rule has been fully traversed. Subsequently, the

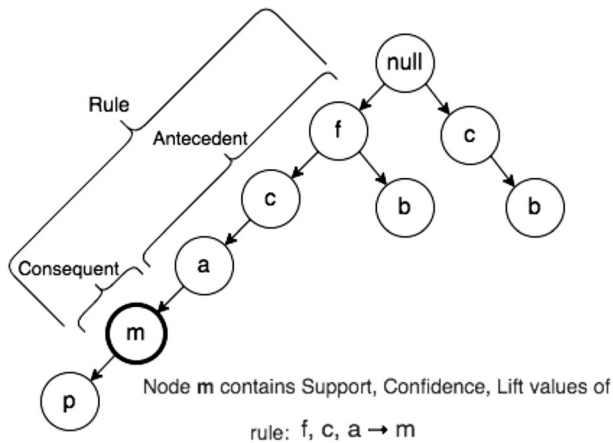


Fig. 1 The structure of a rule in a Trie of rules

second rule ($f \rightarrow b$) is inserted into the trie. Note that the element f has occurred before, as seen in the trie created thus far. Therefore, instead of creating a new branch, the second rule overlays the existing trie and creates an additional branch only when the b item occurs. After traversing the second rule, the trie appears as shown in Fig. 2(b). The last rule to be inserted is ($c \rightarrow b$). Since this rule differs from others in terms of its first item, a new branch is created from the root. Finally, after traversing all the rules, the trie appears as shown in Fig. 2(c).

It is important to note that while a rule, for example, ($f, c, a \rightarrow m, p$) is inserted into the trie, it is treated similarly to a frequent sequence for the purpose of traversal. This means that the antecedent (f, c, a) and the consequent (m, p) are sorted individually based on their frequency and then concatenated for insertion. Therefore, it appears in the trie as a sequence but maintains the distinction of being a rule due to the individual sorting and concatenation process. This approach ensures that the rules are correctly represented and stored within the Trie structure.

Table 1 Ruleset and frequent sequences

Ruleset		Frequent sequences		
No	Rule	No	Frequent sequence	Support
1	$f, c, a \rightarrow m, p$	1	f, c, a, m, p	0.1
2	$f \rightarrow b$	2	f, c, a, m	0.15
3	$c \rightarrow b$	3	f, c, a	0.2
		4	f, c	0.25
		5	f, b	0.2
		7	c, b	0.2
		6	f	0.5
		8	c	0.3
		9	b	0.25
		10	a	0.2
		11	m	0.15
		12	p	0.1

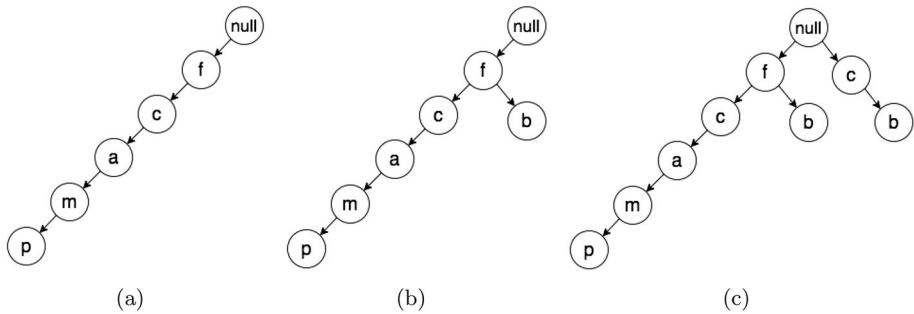


Fig. 2 Step 2 process. The Trie of rules after: the first (a), second (b), and third (c) rules are inserted

- Step 3:** Each node in the Trie of rules is extended with metrics such as Support, Confidence, and any others corresponding to the rule that the node represents (see Algorithm 6). Refer to Fig. 3 for illustration.

3.3 Confidence for compound consequents

Luxemburger (1991) and Kryszkiewicz (2002) discussed the Confidence Transitivity Property, which allows for the calculation of Confidence for compound consequents. This property states that the Confidence of a compound-consequent rule can be calculated as the multiplication of Confidence values of the nodes in the consequent. The following equations illustrate this property:

$$Conf(a, b \rightarrow c, d) = \frac{Sup(a, b, c, d)}{Sup(a, b)} \quad (1)$$

$$Conf(a, b \rightarrow c) = \frac{Sup(a, b, c)}{Sup(a, b)} \quad (2)$$

$$Conf(a, b, c \rightarrow d) = \frac{Sup(a, b, c, d)}{Sup(a, b, c)} \quad (3)$$

To prove the point, (1) can be derived by combining (2) and (3).

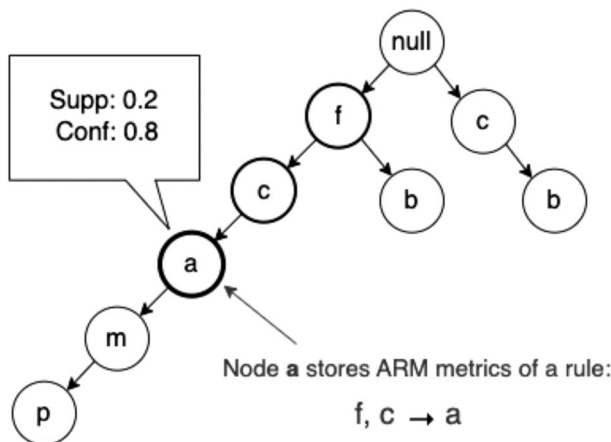


Fig. 3 Step 3. ARM metrics of node *a*

$$\begin{aligned}
 \text{Conf}(a, b \rightarrow c) \times \text{Conf}(a, b, c \rightarrow d) &= \frac{\text{Sup}(a, b, c)}{\text{Sup}(a, b)} \times \frac{\text{Sup}(a, b, c, d)}{\text{Sup}(a, b, c)} \\
 &= \frac{\text{Sup}(a, b, c, d)}{\text{Sup}(a, b)} \\
 &= \text{Conf}(a, b \rightarrow c, d)
 \end{aligned} \quad (4)$$

Our structure naturally employs this rule for calculating the Confidence of compound consequents, allowing faster retrieval through the dataset. In a Trie of rules, each node shows Confidence only for a rule with a single-item consequent; however, the proposed representation model can be used to derive the value of Confidence for more complex rules directly from the graph. This calculation is shown in Algorithm 7, specifically in line 17. Figure 4 illustrates this concept.

This feature is possible because of the specifics of a Trie of rules. In order to calculate Confidence of a rule, two values are used: Support of the antecedent and Support of the entire rule. As mentioned, the Trie of rules is based on the prefix-tree structure. Hence, every path starting from the root is unique because identical sequences will overlay each other in one path. Consequently, when picking a node and observing a Support value in it, one can be sure that this value represents true Support for the sequence equal to the path to this node. Therefore, the calculation of Confidence does not require any information from other branches. All this allows us to multiply Confidence values for a sequence of nodes, which further allows us to evaluate rules with compound consequents in a Trie of rules.

However, this rule is limited to Confidence calculation. For other metrics, different methods need to be employed, which we consider as future work.

The next section presents an evaluation of the proposed data structure.

4 Evaluation

The evaluation of the proposed data structure is crucial for assessing its effectiveness in advancing knowledge discovery methods. In this section, we perform the following analyses:

- Analysis of search time for finding a rule in a ruleset.
- Assessment of how search time varies with different ruleset sizes.
- Evaluation of the memory efficiency of the data structure.

As many knowledge extraction algorithms involve traversing a dataset and searching for certain rules, improving the speed of these functions can enhance the efficiency of related tasks. To benchmark the performance of the Trie of Rules data structure, we compare it with

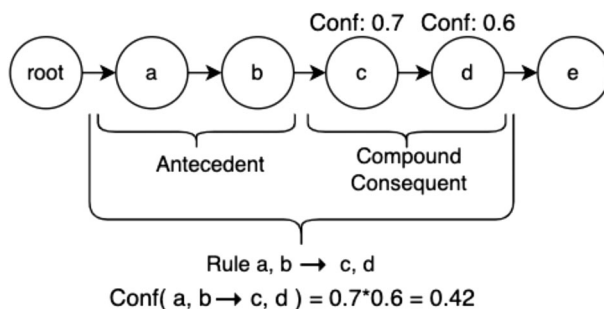


Fig. 4 A rule with a compound consequent

two widely used alternatives: the Pandas DataFrame and NumPy. These comparisons allow us to assess the effectiveness and efficiency of our proposed approach for storing association rules.

All experiments were conducted on the same machine within the same environment. The machine used was a MacBook Air (2019) with a 1.6 GHz dual-core Intel Core i5 processor, 8 GB of LPDDR3 RAM, a 256 GB SSD, and macOS Sonoma (version 14.5). The experiments were conducted within the JupyterLab environment, using Python 3.11, pandas version 2.1.4, and NumPy version 1.26.2.

4.1 Pandas dataframe

The pandas library provides the DataFrame data structure (The Pandas development team, 2024; Wes McKinney, 2010), designed for two-dimensional tabular data handling, akin to spreadsheets or SQL tables. Internally, pandas uses hash tables for indexing, enabling fast lookups and efficient data operations. With hash table indexing, pandas ensures swift data retrieval based on labels or position, facilitating operations such as filtering, aggregation, and transformation. This makes it a popular choice for storing association rules, as modern

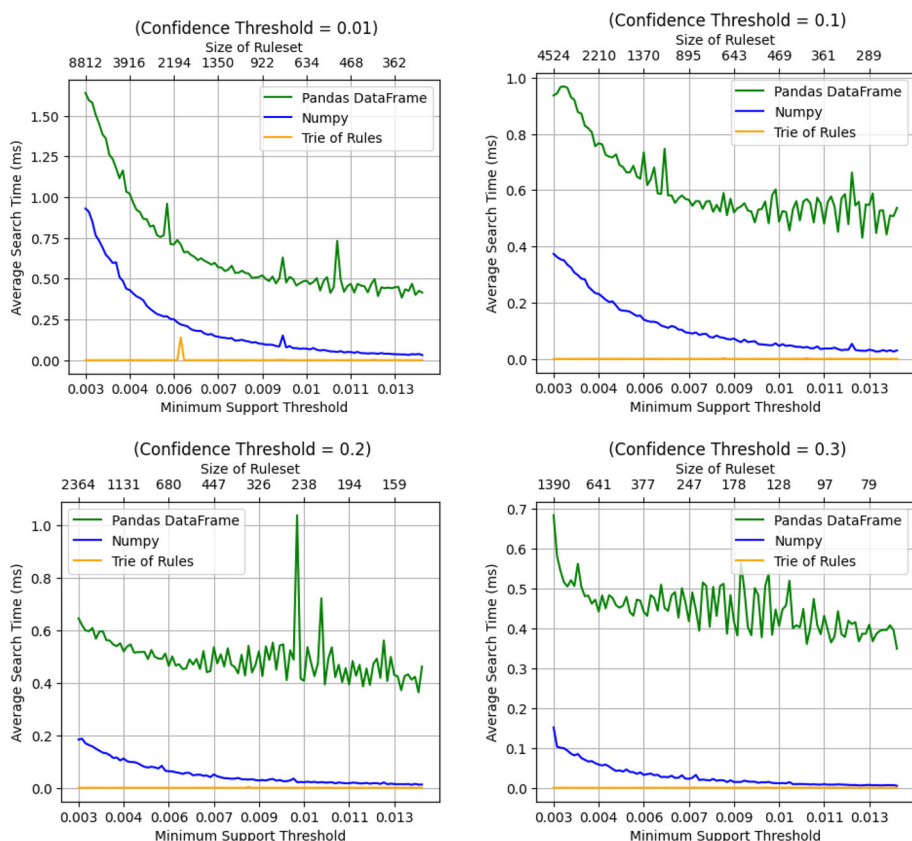


Fig. 5 Evaluation of average search time for rules in data structures created with different support and confidence thresholds

Python libraries for Association Rule Mining often default to pandas DataFrame for rule storage.

4.2 NumPy

NumPy (Harris et al., 2020) provides Support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays.

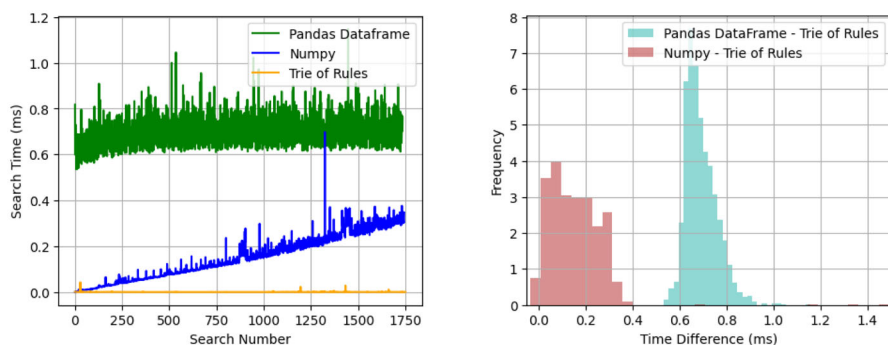
NumPy's ndarray (n-dimensional array) allows for efficient storage and manipulation of large datasets with its Support for multiple dimensions and various data types. It provides a versatile and efficient foundation for numerical computation and data manipulation tasks in Python.

The ndarray can serve as a data structure for storing association rules in a 2D matrix format with columns representing antecedent, consequent, Support, and Confidence.

4.3 Experiment on grocery dataset

To assess the proposed data structure, we utilized a grocery dataset sourced from the "arules" package within the R Project for Statistical Computing (Hahsler et al., 2006). This dataset, originally introduced in the context of association rule mining, comprises transactional data collected from a local grocery outlet over a one-month period. With a total of 9,834 transactions and 169 unique items, this dataset provides a rich source of information for studying market basket analysis and deriving meaningful insights into consumer buying patterns.

Experiments were conducted to examine the average search time for finding a rule in a ruleset constructed with different pairs of Support and Confidence thresholds. All generated rules were used to construct the data structures. Using Algorithm 7, which finds a rule and returns its metrics, all the rules were searched. This operation is commonly needed in knowledge discovery methods and for typical user queries to retrieve rule metrics. As shown in Fig. 5, the average search time was compared across various Confidence and Support values, demonstrating how the size of the ruleset affects the search time. A range of 100 different minimum Support thresholds between 0.003 and 0.0135, along with four different Confidence values: 0.01, 0.1, 0.2, and 0.3, was chosen to cover a broad spectrum of ruleset sizes, ensuring both sparse and dense rulesets were included in the analysis.



(a) Comparison of search time of rules in a ruleset between Trie of Rules, Pandas DataFrame, and NumPy data structures.

(b) Distribution of differences between search time in Trie of Rules and Pandas DataFrame and NumPy.

Fig. 6 Search time analysis and comparative statistics for Trie of Rules, Pandas DataFrame, and NumPy

Table 2 Summary of evaluation on grocery dataset (1,752 rules, minimum support = 0.005, minimum Confidence = 0.1)

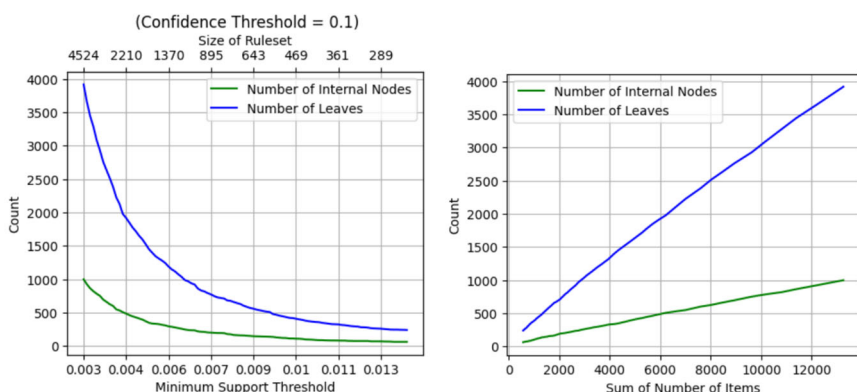
	Trie of rules	Pandas dataframe	NumPy
Total traversal time (sec)	0.00287	1.361	0.320
Average search time (μ s)	1.63	770	183
Size (megabytes)	0.917	1.996	0.0564

Based on this analysis, a minimum Support threshold of 0.005 and a Confidence threshold of 0.1 were selected for a more detailed evaluation. These values provided a balance between having a manageable number of rules and sufficient Confidence for meaningful insights. The specific values allowed for generating a comprehensive set of rules while avoiding the extreme cases of very sparse or excessively dense rulesets. The ARM algorithm generated 1,001 frequent sequences and 1,752 association rules. All generated rules were used for the Trie of Rules.

Figure 6(a) demonstrates the results of searching all rules in each data structure, with a summary of the results provided in Table 2. The average search time for the Trie of Rules was 1.63 microseconds, significantly lower than that of Pandas DataFrame, which was 770 microseconds, and NumPy, which was 183 microseconds. The total time taken to search all rules was 0.00287 seconds for the Trie of Rules, 1.361 seconds for Pandas DataFrame, and 0.320 seconds for NumPy.

Pairwise t-tests confirmed the significance of these differences (Fig. 6(b)). The null hypothesis, stating that the difference in search times between the Trie of Rules and each of the other data structures is zero, was rejected with a p-value < 0.05, indicating that the differences are statistically significant.

The Trie of Rules utilizes a compact representation. The number of nodes is always approximately equal to the number of rules as indicated by Fig. 7(a). The number of internal nodes and leaves grows linearly with the total number of items in all rules, as shown in Fig. 7(b).



(a) Comparison of the number of nodes with the size of the ruleset. (b) Comparison of the number of nodes with the total number of items in rules.

Fig. 7 Analysis of node count relative to ruleset size and total item count

Table 3 Summary of evaluation on retail dataset (381,912 rules, minimum support = 0.02, minimum confidence = 0.2)

	Trie of rules	Pandas dataframe	NumPy
Total traversal time (sec)	9.4	27,766	18,675
Average search time (ms)	0.024	72	48.9
Size (megabytes)	188	452	12

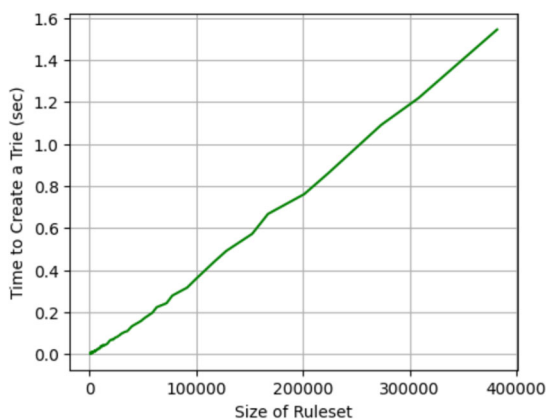
4.4 Experiment on retail dataset

To further evaluate the scalability of the proposed data structure, a larger dataset was used. The Retail dataset (Chen, 2015) contains information related to online retail customers, capturing their transactional behavior over a given period. It consists of approximately 18,000 transactions and 3,600 different items. A minimum Support threshold of 0.002 and a minimum Confidence threshold of 0.2 were chosen, resulting in 45,362 frequent sequences and 381,912 association rules. These thresholds were selected to produce a larger set of rules, which is ideal for thoroughly examining the effectiveness and efficiency of the Trie of Rules data structure. The summary of the evaluation is presented in Table 3.

Traversing through all rules in the Trie of Rules took 9.4 seconds with an average time of 24 microseconds. For NumPy, the total time was 5 hours with an average time of 48.9 milliseconds, while for Pandas DataFrame, it took more than 7 hours to traverse through the whole dataset with an average time of 72 milliseconds.

The Trie of Rules boasted impressive space efficiency, as evidenced by its compact representation. The tree has a height of 9, comprising 248,864 leaf nodes and 141,053 internal nodes, occupying 188 megabytes of memory. Meanwhile, NumPy and DataFrame both contained 381,912 rows, with NumPy consuming only 12 megabytes of memory and DataFrame 452 megabytes. These results underscore the 'Trie of Rules' capacity for space optimization, further emphasizing its suitability for handling large-scale datasets.

On this larger dataset, the time to create a Trie of Rules was also measured. As shown in Fig. 8, the creation time is linearly dependent on the size of the ruleset. The plot indicates that the slope is approximately 0.4 seconds per 100,000 rules. This linear relationship suggests that the Trie of Rules scales efficiently with increasing ruleset sizes, maintaining a predictable and manageable increase in creation time as the number of rules grows. This characteristic

Fig. 8 Time to create a trie of rules (sec) depending on the size of the ruleset

is particularly advantageous for large-scale datasets, ensuring that the structure can be built within a reasonable timeframe even as the dataset size expands.

In conclusion, the Trie of Rules outperforms both Pandas DataFrame and NumPy in terms of search time and space efficiency across different datasets. This faster search time can be attributed to the Trie of Rules naturally clustering rules, narrowing the search space with each additional item as it traverses deeper into the tree. Its structural efficiency and fast traversal make it a promising candidate for efficiently handling large-scale datasets. Moreover, its unique properties, such as a breadth-first search strategy and optimized space usage, contribute to its effectiveness in knowledge extraction tasks.

5 Conclusion and future work

Association rule mining is a popular method used for knowledge discovery in various fields. However, little attention has been given to the data structure used to store the resulting ruleset. In this paper, we proposed the Trie of Rules data structure for storing association rules, aiming to facilitate efficiency in further knowledge extraction by increasing the speed of traversal through a dataset and providing a graph representation of rules, allowing the use of graph-based knowledge extraction methods.

The Trie of Rules organizes rules into a prefix-tree graph structure, enabling faster traversal through the ruleset and reducing the time complexity of knowledge discovery methods. Comparing this method with existing methods such as storing rules as a plain table or hash-tables Supported data frames showed superior efficiency in terms of traversal time. This novel data structure provides a valuable tool for knowledge extraction and has the potential to contribute to the development of more efficient and effective approaches for rule exploration and visualization.

Further investigation is needed to determine the optimal order in which items should be placed in the Trie, considering factors such as their Support or other relevant parameters. This optimization could lead to better performance and scalability of the data structure. Additionally, exploring the possible use of the Trie of Rules data structure for visualization purposes of association rules could enhance the interpretability of ARM results. Visualizations can provide insights into the relationships between items and help users understand complex patterns more intuitively.

Appendix A Algorithms

Algorithm 2 Rule object definition with constructor procedure.

class Rule:

attributes:

- *antecedent*: list[str]

▷ Sequence of items in the rule antecedent

- *consequent*: list[str]

▷ Sequence of items in the rule consequent

- *metrics*: dictionary[str, float]

▷ Dictionary to store metrics

procedure RULE():

self.antecedent ← []

self.consequent ← []

self.metrics ← {'Support': None, 'Confidence': None }

Algorithm 3 Trie node structure with constructor procedure.

```

1: class TrieNode:
2:   attributes:
3:   - children: dictionary[str, TrieNode]           ▷ dictionary of child nodes (item name : node)
4:   - predecessor: TrieNode                         ▷ The parent node
5:   - value: str                                     ▷ Name of the item associated with the node
6:   - metrics: dictionary[str, float]             ▷ Dictionary to store metrics
7:
8:   procedure TRIENODE(predecessor, value):
9:     self.children ← {}                             ▷ Initially, no children; dictionary is empty
10:    self.predecessor ← predecessor
11:    self.value ← value
12:    self.metrics ← {'Support': None, 'Confidence': None }

```

Algorithm 4 Trie structure with constructor procedure.

```

1: class Trie:
2:   attributes:
3:   - root: TrieNode                                ▷ Represents the root node of the trie
4:   - item_frequency: dictionary[str, int]         ▷ Frequency of individual items
5:   methods:
6:   - insert_rule(self, rule): void                ▷ Inserts a new rule into the trie
7:   - search_rule(self, rule): Rule or None          ▷ Searches for a rule in the trie
8:   - extend_node(self, node, frequent_sequences): void ▷ Extend nodes with ARM metrics
9:
10:  procedure TRIE():
11:    self.root ← TRIENODE(predecessor=None, value=None)
12:    self.root.metrics ← {'Support': 1.0, 'Confidence': 1.0 }
13:    self.item_frequency ← {}

```

Algorithm 5 Inserting a rule in the Trie - Trie class method.

Description: This procedure inserts a rule into a Trie object after sorting the antecedent and consequent items based on their frequency.

```

1: procedure TRIE.INSERT_RULE(self, rule, frequent_sequences)
2:   current_node ← self.root
3:
4:   for each item in rule.antecedent + rule.consequent do           ▷ Update item frequencies
5:     if item not in self.item_frequency then                       ▷ Concatenate two lists
6:       self.item_frequency[item] ← frequent_sequences[item]
7:     end if
8:   end for
9:
10:  sorted_antecedent ← sort(rule.antecedent,  $\lambda(x, y) \rightarrow \text{self.item\_frequency}[x] >$ 
    self.item_frequency[y])
11:
12:  sorted_consequent ← sort(rule.consequent,  $\lambda(x, y) \rightarrow \text{self.item\_frequency}[x] >$ 
    self.item_frequency[y])           ▷ Sort the consequent items by their frequency (descending)
13:  rule_itemlist ← sorted_antecedent + sorted_consequent           ▷ Concatenate two lists
14:  for each item in rule_itemlist do
15:    if item not in current_node.children then
16:      current_node.children[item] ← TRIENODE(predecessor = current_node, value = item)
17:    end if
18:    current_node ← current_node.children[item]
19:  end for
20: end procedure

```

Algorithm 6 Extend Trie with ARM metrics - Trie class method.**Description:** This procedure adds Support and Confidence metrics to the node and its children recursively.

```

1: procedure TRIE.EXTEND_NODE(self, node, frequent_sequences)    ▷ frequent_sequences is a dictionary
   with sequences of items as keys and their associated Support values
2:   item_list ← []
3:   current_node ← node
4:   while current_node ≠ self.root do
5:     item_list ← item_list + current_node.value                ▷ Append items in the path to the root to the list
6:     current_node ← current_node.predecessor
7:   end while
8:   node.metrics['Support'] ← frequent_sequences[item_list]    ▷ Set Support metric from the frequent
   sequences
9:                                                         ▷ Calculate Confidence for the node:
10:  node.metrics['Confidence'] ← node.metrics['Support'] / node.predecessor.metrics['Support']
11:  for each child in node.children do
12:    self.extend_node(child, frequent_sequences)              ▷ Recursively traverse child nodes
13:  end for
14: end procedure

```

Algorithm 7 Searching a rule in the Trie - Trie class method.**Description:** This function searches for a rule in the Trie and returns its metrics if found.

```

1: function TRIE.SEARCH_RULE(self, rule)
2:   current_node ← self.root
3:   sorted_antecedent ← sort(rule.antecedent,  $\lambda(x, y)$  → self.item_frequency[x] >
   self.item_frequency[y])
4:   sorted_consequent ← sort(rule.consequent,  $\lambda(x, y)$  → self.item_frequency[x] >
   self.item_frequency[y])
5:   for each item in sorted_antecedent do
6:     if item not in current_node.children then
7:       return None                                             ▷ Rule not found
8:     end if
9:     current_node ← current_node.children[item]
10:  end for
11:  rule_confidence ← 1.0
12:  for each item in sorted_consequent do
13:    if item not in current_node.children then
14:      return None                                             ▷ Rule not found
15:    end if
16:    current_node ← current_node.children[item]
17:    rule_confidence ← rule_confidence × current_node.metrics['Confidence']
18:  end for
19:  rule.metrics ← {'Support': current_node.metrics['Support'], 'Confidence': rule_confidence}
20:  return rule.metrics
21: end function

```

Author Contributions M.K. developed the methodology and concept, conducted the literature review, and wrote the manuscript. A.M. and M.B. supervised the project, helped develop the concept, and reviewed all aspects of the work. M.R. reviewed the paper and contributed to the formalization of the concept. V. M. N. assisted with writing, literature review, and the formalization of the concept. All authors participated in drafting and revising the manuscript and have approved the final version.

Funding This research was funded by Science Foundation Ireland under Grant numbers 18/CRT/6223 and SFI/12/RC/2289_2 as part of the Centre for Research Training in Artificial Intelligence (CRT-AI) program.

Data Availability The datasets used in this study are publicly available. The Groceries dataset can be downloaded at <https://github.com/mhahsler/arules/blob/master/data/Groceries.rda>. The Retail dataset can be downloaded at <https://archive.ics.uci.edu/dataset/352/online+retail>.

Declarations

Competing Interests The authors declare no competing interests.

Ethical Approval The authors state that this research complies with ethical standards. This research does not involve either human participants or animals.

Open Access This article is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License, which permits any non-commercial use, sharing, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if you modified the licensed material. You do not have permission under this licence to share adapted material derived from this article or parts of it. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by-nc-nd/4.0/>.

References






- Agrawal, R., Imieliński, T., & Swami, A. (1993). Mining association rules between sets of items in large databases. *ACM SIGMOD Record*, 22(2), 207–216. <https://doi.org/10.1145/170036.170072>
- Alasow, M.A., Mohammed, S.A., El-Alfy, E.S.M. (2020) Parallel association rules pruning algorithm on Hadoop MapReduce, Springer Singapore, Singapore, pp 117–130. https://doi.org/10.1007/978-981-15-3852-0_8
- Bayardo, R.J., Agrawal, R. (1999) Mining the most interesting rules. In: *Proceedings of the fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '99*. ACM Press, New York, New York, USA, (pp 145–154). <https://doi.org/10.1145/312129.312219>
- Berrado, A., & Runger, G. C. (2007). Using metarules to organize and group discovered association rules. *Data Mining and Knowledge Discovery*, 14(3), 409–431. <https://doi.org/10.1007/s10618-006-0062-6>
- Bodon, F., Rónyai, L. (2003) Trie: an alternative data structure for data mining algorithms. In: *Mathematical and Computer Modelling*, pp. 739–751. [https://doi.org/10.1016/0895-7177\(03\)90058-6](https://doi.org/10.1016/0895-7177(03)90058-6)
- Brin, S., Motwani, R., Ullman, J. D., et al. (1997). Dynamic itemset counting and implication rules for market basket data. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 26(2), 255–264. <https://doi.org/10.1145/253262.253325>
- Bui-Thi, D., Meysman, P., & Laukens, K. (2020). Clustering association rules to build beliefs and discover unexpected patterns. *Applied Intelligence*, 50(6), 1943–195. <https://doi.org/10.1007/s10489-020-01651-1>
- Bui-Thi, D., Meysman, P., & Laukens, K. (2022). Momac: multi-objective optimization to combine multiple association rules into an interpretable classification. *Applied Intelligence*. <https://doi.org/10.1007/s10489-021-02595-w>
- Chen, D. (2015) Online retail. *UCI Machine Learning Repository*, <https://doi.org/10.24432/C5BW33>
- Coenen, F., Leng, P., & Ahmed, S. (2004). Data structure for association rule mining: T-trees and P-trees. *IEEE Transactions on Knowledge and Data Engineering*, 16(6), 774–778. <https://doi.org/10.1109/TKDE.2004.8>
- Crochemore, M., Lecroq, T. (2009). Trie, Springer US, Boston, MA, pp 3179–3182. https://doi.org/10.1007/978-0-387-39940-9_1143
- De Padua, R., Carmo, L.P.D., Rezende, S.O., et al. (2018). An analysis on community detection and clustering algorithms on the post-processing of association rules. *Proceedings of the International Joint Conference on Neural Networks 2018*-July. <https://doi.org/10.1109/IJCNN.2018.8489603>
- Geng, L., & Hamilton, H. J. (2006). Interestingness measures for data mining: a survey. *ACM Comput Surv*, 38(3), 9–es. <https://doi.org/10.1145/1132960.1132963>
- Ghafari, S. M., & Tjortjij, C. (2019). A survey on association rules mining using heuristics. *WIREs Data Mining and Knowledge Discovery*, 9(4), 1307. <https://doi.org/10.1002/widm.1307>
- Grahne, G., Zhu, J. (2003). Efficiently using prefix-trees in mining frequent itemsets. *Proc of the 1st IEEE ICDM Workshop on Frequent Itemset Mining Implementations* pp. 236–245
- Hahsler, M. (2016). Grouping association rules using Lift. *Proceedings of the 11th INFORMS Workshop on Data Mining and Decision Analytics*. <http://cran.r-project.org/>

- Hahsler, M. (2023). ARULESPY: exploring association rules and frequent itemsets in Python. [arXiv:2305.15263](https://arxiv.org/abs/2305.15263)
- Hahsler, M., Grun, B., Hornik, K. (2005). Arules - a computational environment for mining association rules and frequent itemsets. *Journal of Statistical Software* 14(15). <https://doi.org/10.18637/jss.v014.i15>
- Hahsler, M., Hornik, K., Reutterer, T. (2006). Implications of probabilistic data modeling for mining association rules. *From Data and Information Analysis to Knowledge Engineering* pp. 598–605. https://doi.org/10.1007/3-540-31314-1_73
- Han, J., Pei, J., Yin, Y., et al. (2004). Mining frequent patterns without candidate generation: a frequent-pattern tree approach. *Data Mining and Knowledge Discovery* 8(1) 53–87. <https://doi.org/10.1023/B:DAMI.0000005258.31418.83>
- Harris, C. R., Millman, K. J., van der Walt, S. J., et al. (2020). Array programming with NumPy. *Nature*, 585(7825), 357–36. <https://doi.org/10.1038/s41586-020-2649-2>
- Jentner, W., & Keim, D. A. (2019). Visualization and visual analytic techniques for patterns. *Studies in Big Data*, 51, 303–337. https://doi.org/10.1007/978-3-030-04921-8_12
- Jin, B., Gao, C., He, X. et al. (2020). Multi-behavior recommendation with graph convolutional networks. *SIGIR 2020 - Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval* pp. 659–668. <https://doi.org/10.1145/3397271.3401072>, [arXiv:arXiv:2005.03475v1](https://arxiv.org/abs/2005.03475v1)
- Koh, YS., Pears, R., Yeap, W. (2010). Valency based weighted association rule mining. pp 274–285. https://doi.org/10.1007/978-3-642-13657-3_31
- Kryszkiewicz, M. (2002). Concise representations of association rules. In D. J. Hand, N. M. Adams, & R. J. Bolton (Eds.), *Pattern Detection and Discovery* (pp. 92–109). Berlin Heidelberg, Berlin, Heidelberg: Springer.
- Li, Y., & Wu, J. (2014). Interpretation of association rules in multi-tier structures. *International Journal of Approximate Reasoning*, 55(6), 1439–1457. <https://doi.org/10.1016/j.ijar.2014.04.015>
- Liu, X., & Shen, H. W. (2016). Association analysis for visual exploration of multivariate scientific data sets. *IEEE Transactions on Visualization and Computer Graphics*, 22(1), 955–964. <https://doi.org/10.1109/TVCG.2015.2467431>
- Liu, X., Zhai, K., & Pedrycz, W. (2012). An improved association rules mining method. *Expert Systems with Applications*, 39(1), 1362–1374. <https://doi.org/10.1016/j.eswa.2011.08.018>
- Luna, J. M., Ondra, M., Fardoun, H. M., et al. (2018). Optimization of quality measures in association rule mining: an empirical study. *International Journal of Computational Intelligence Systems*, 12, 59–78. <https://doi.org/10.2991/ijcis.2018.25905182>
- Luxemburger, M. (1991). Implications partielles dans un contexte. *Mathématiques Informatique et Sciences Humaines* 11335–55. http://www.numdam.org/item/MSH_1991__113__35_0/
- Máša, P., & Rauch, J. (2024). A novel algorithm for mining couples of enhanced association rules based on the number of output couples and its application. *Journal of Intelligent Information Systems*, 62(2), 431–458. <https://doi.org/10.1007/s10844-023-00820-1>
- Wes McKinney. (2010). Data structures for statistical computing in python. In: Stéfan van der Walt, Jarrod Millman (eds) *Proceedings of the 9th Python in Science Conference*, pp. 56 – 61. <https://doi.org/10.25080/Majora-92bf1922-00a>
- Moahammed, SA., Alasow, MA., El-Alfy, ESM. (2021). Clustering of association rules for big datasets using hadoop mapreduce. *International Journal of Advanced Computer Science and Applications* 12(3). <https://doi.org/10.14569/IJACSA.2021.0120364>
- Shabtay, L., Fournier-Viger, P., Yaari, R., et al. (2021). A guided fp-growth algorithm for mining multitude-targeted item-sets and class association rules in imbalanced data. *Information Sciences*, 553, 353–375. <https://doi.org/10.1016/j.ins.2020.10.020>
- Shahbazi, N., & Gryz, J. (2022). Upper bounds for can-tree and FP-tree. *Journal of Intelligent Information Systems*, 58(1), 197–222. <https://doi.org/10.1007/s10844-021-00673-6>
- Shaukat Dar, K., & Zaheer, S. (2015). Association rule mining: an application perspective. *International Journal of Computer Science and Innovation*, 1, 29–38.
- Stancin, I., Jovic, A. (2019). An overview and comparison of free Python libraries for data mining and big data analysis. 2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics, MIPRO 2019 - Proceedings pp 977–982. <https://doi.org/10.23919/MIPRO.2019.8757088>
- The Pandas development team. (2024). Pandas-dev/pandas: Pandas. <https://doi.org/10.5281/zenodo.10957263>
- Vu, L., Alaghhband, G. (2011). A fast algorithm combining fp-tree and tid-list for frequent pattern mining. In: *Proceedings of information and knowledge engineering*, pp. 472–477

- Wu, T., Chen, Y., & Han, J. (2010). Re-examination of interestingness measures in pattern mining: a unified framework. *Data Mining and Knowledge Discovery*, 21(3), 371–39. <https://doi.org/10.1007/s10618-009-0161-2>
- Yazgana, P., Kusakci, AO. (2016) A literature survey on association rule mining algorithms. *Southeast Europe Journal of Soft Computing* 5(1) 5–14. <https://doi.org/10.21533/scjournal.v5i1.102> <https://doi.org/10.21533/scjournal.v5i1.102>
- Yen, S. J., & Chen, A. (2001). A graph-based approach for discovering various types of association rules. *IEEE Transactions on Knowledge and Data Engineering*, 13(5), 839–845. <https://doi.org/10.1109/69.956106>
- Zaki, M. J., Parthasarathy, S., Ogihara, M., et al. (1997). Parallel algorithms for discovery of association rules. *Data Mining and Knowledge Discovery*, 1(4), 343–373. <https://doi.org/10.1023/A:1009773317876>

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Authors and Affiliations

Mikhail Kudriavtsev¹  · Vuong M. Ngo⁴  · Mark Roantree³  ·
Marija Bezbradica²  · Andrew McCarren³ 

✉ Mikhail Kudriavtsev
mikhail.kudriavtsev2@mail.dcu.ie

Vuong M. Ngo
vuong.nm@ou.edu.vn

Mark Roantree
mark.roantree@dcu.ie

Marija Bezbradica
marija.bezbradica@dcu.ie

Andrew McCarren
andrew.mccarren@dcu.ie

- ¹ School of Computing, Dublin City University, Dublin, Ireland
- ² Adapt Research Centre, Dublin City University, Dublin, Ireland
- ³ Insight Centre for Data Analytics, Dublin City University, Dublin, Ireland
- ⁴ Ho Chi Minh City Open University, Ho Chi Minh City, Vietnam