ON MINING MULTI-TIME-INTERVAL SEQUENTIAL PATTERNS

B.Tech CSE - VI Sem

CS351 - Machine Learning

NAME	ROLL NO.	PHONE NO.	EMAIL
Sangeeth S V	181CO246	7907932994	sangeeth.181co246@nitk.edu.in
Rajath C Aralikatti	181CO241	7829158425	rajath.181co241@nitk.edu.in

Abstract

Data mining is the process by which previously unknown, potentially usable data is extracted from databases. It has been established that patterns occur frequently in large databases and several papers have been published about mining such sequential patterns. However, these algorithms for mining sequential patterns cannot establish the timeframe between two sequential events. Therefore, the focus shifted toward mining time-interval sequential patterns. Two approaches were proposed each having their own drawbacks. Setting fixed time frames for each event, which has the drawback of having no exact measure of the time interval. Measuring the exact time interval between successive events, with the drawback of having no way to measure the time between non-successive events.

Therefore, improving upon these approaches became necessary. To that end, Ya-Han Hu et. al. in their paper, "On mining multi-time-interval sequential patterns", they describe two algorithms, the MI-Apriori Algorithm and the MI-PrefixSpan Algorithm for mining multi-time-interval sequential patterns. In this project, we strive to provide implementations for both these algorithms.

Keywords

Multi-Time-Interval Sequential Patterns (MTIS), Sequential Pattern Mining, Sequential Database, MI - Apriori Algorithm, MI - PrefixSpan Algorithm.

Introduction

Mining sequential patterns was first introduced in the mid-1990s, showing that patterns frequently occur in a sequence database. Traditionally mining sequence patterns only involved finding sequences in which events/data items occur with no regard for the time span between them. Learning this additional information on the time span between the events became crucial as it could make decision support much better. So, the focus grew to mining time-interval sequential patterns. However, the first of these algorithms for time-interval sequential patterns made it possible only to have precise time-interval information between successive events. Being able to tell the time intervals between any two events in a sequence could further improve

decision support, and the introduction of Multi-time-interval Sequential (MTIS) patterns made this possible.

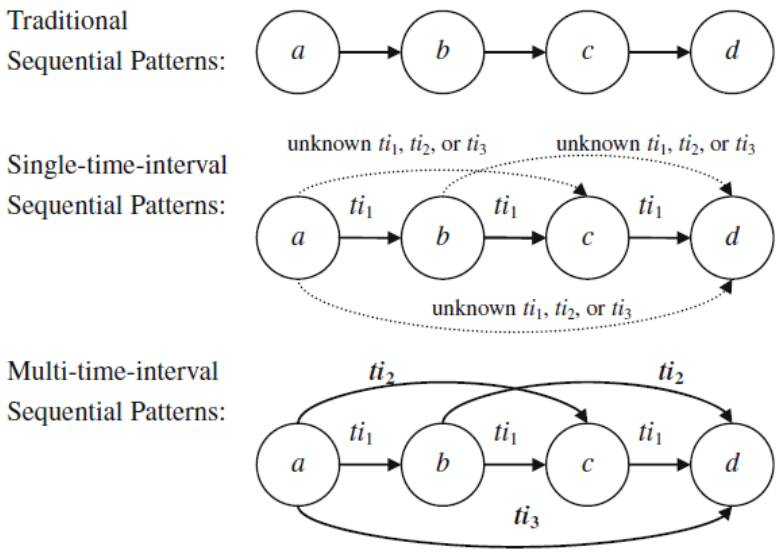


Fig. 1. Different kinds of sequential patterns.

MTIS Patterns are patterns that contain information about the time differences between any two events in a sequential pattern. For example, a person who purchases the first volume of a book series, may be likely to purchase the second one a week after and the third one another 2 weeks later and so on. Knowing about such patterns, will enable the shopkeeper to manage his stocks and provide clubbed offers, etc. Through mining MTIS patterns, we seek to establish such timed relations between sequentially occurring events.

sid	Sequence
10	<(*a*, 1), (*b*, 3), (*c*, 3), (*a*, 5), (*e*, 5), (*c*, 10)>
20	<(*d*, 5), (*a*, 7), (*b*, 7), (*e*, 7), (*d*, 8), (*e*, 8), (*c*, 14), (*d*, 15)>
30	<(*a*, 8), (*b*, 8), (*e*, 11), (*d*, 12), (*b*, 13), (*c*, 13), (*c*, 16)>
40	<(*b*, 15), (*f*, 15), (*e*, 16), (*b*, 17), (*c*, 17)>

Fig. 2. A sequence database.

Fig. 2. shows an example for a Sequential Database with time stamped data. Each item sequence is a list of tuples of the form (item, timestamp). For running our MTIS mining algorithms we must ensure that in each item sequence the items are ordered in the non-decreasing order of their time stamps.

Let the MTIS pattern $P = < a, (ti_1), b, (ti_2, ti_1), c, (ti_2, ti_1, ti_1), d >$. This means that:

- The time between a&b is ti_1.
- The time between a&c is ti_2 and b&c is ti_1.
- The time between a&d is ti_2, b&d is ti_1 and c&d is ti_1.

Let us suppose the item sequence $A = \{(a, 1), (b, 3), (c, 5), (d, 6), (e, 8)\}$and the time intervals $T = \{ti_0: [0, 0],\ ti_1: [0, 3],\ ti_2: [3, 6],\ ti_3: [6, \infty]\}$. Here, we can say that the pattern P is contained in A, because the time differences between events in the item sequence match the time intervals defined by the pattern.

Properties of MTIS Patterns

- ➢ Support: Support of an MTIS pattern is defined as the ratio of number of item sequences in which it is contained to the total number of item sequences in the database.
- ➢ Descending Property: Each successive time interval specified in any part of the MTIS pattern is always at most equal to the previous time interval. For example, (ti_3, ti_2, ti_2, ti_1) is valid, whereas, (ti_2, ti_3, ti_2, ti_1) is not valid.
- ➢ Anti-Monotonicity: If a MTIS pattern is frequent, so are all of its subsequences. Accordingly, if a MTIS pattern is not frequent, then its super sequence will not be either. A sequence is said to be frequent if it has more than the minimum support specified for the data.

Algorithms

There are two algorithms mentioned in the paper:

- MI - Apriori Algorithm: This algorithm tries to generate higher order sequences (C_k) from lower order sequences (C_{k-1}). The basic principle behind this algorithm is that any sequence of length k, must have two subsequences of length (k-1). ⇒ Anti-Monotonicity Property.
- MI - PrefixSpan Algorithm: The prefix span algorithm tries to extend existing patterns using the concept of 'projected databases'. The algorithm works recursively, generating all the MTIS patterns that start with a particular prefix.

Dataset Generation

For the purposes of creating and testing both algorithms used in this project, we have used the example dataset from the paper by Ya-Han Hu et. al., with the same item sequences, minimum support and time intervals list. For consequent testing and comparisons, we have used randomly generated databases made using the following algorithm.

- ➢ generate_items(n): returns a list of 'n' items randomly sampled from a predefined list of items.
- ➢ generate_time_list(n): returns a list of 'n' randomly generated timestamps. The times would start from a random timestamp and then be incremented iteratively using a number from a small addend list, say, [1,2,3,4]. So, if the starting timestamp is 1, the next timestamp would at most be 5. This limitation is imposed on the database generation algorithm because we have done all the testing with the differences in time intervals being 3.
- ➢ generate_database(itemList, timeList, minLength, maxLength, dbLength): This method takes as input, a *list of items* (generated using generate_items(n)), a *list of timestamps* (generated using generate_time_list(n)), *minLength* and *maxLength* which indicates the

smallest and greatest number of items possible in each row of the database, and finally, *dbLength* which indicates the required number of rows in the database.

Scope of the Work

In this project, we have successfully implemented and compared the two algorithms, MI-Apriori and MI-PrefixSpan, described in the paper. The runtime complexity and space complexity of both algorithms have also been compared by executing on randomly generated datasets of varying length item sequences. Experimental results from the paper indicate that the MI-PrefixSpan algorithm performs better on small length sequence data when compared to the MI-Apriori algorithm. However, the PrefixSpan algorithm does not scale well with larger item sequences.

We also perform a few optimizations in the runtime and space complexity of both MI-Apriori and MI-PrefixSpan algorithms. The joinCk method in the MI-Apriori method is optimized by combining the candidate generation and the pruning steps. We optimize the MI-PrefixSpan algorithm by replacing the expensive table generation step with a more efficient spanning algorithm.

MI-Apriori Algorithm

The idea of MI - Apriori Algorithm is to generate higher order sequences (Ck) from the previous set of sequences (Ck-1). The fundamental principle behind the apriori algorithm is the anti-monotonicity property of MTIS patterns. So, we can say that every k-sequence must be formed from two (k-1)-subsequences that also satisfy the minimum support requirement.

$L_1 = \text{Find\_1} - \text{frequent} - \text{item}(S);$
For$(k = 2; L_{k-1} \neq null; k++)\{$
$C_k = MI - Apriori\_gen(L_{k-1}, TI);$//generate all possible candidates
For each sequence $c \in C_k$ {
For each sequence $s \in S\{$
If (c contained in s) then{
$c.count++;$
}
}
}
$L_k = \{c \in C_k | c.count \geq min\_sup\}$
}
return $\cup L_k;$

Fig. 3. The MI-Apriori Algorithm.

- The basic join operation seeks to take two k - MTIS patterns as input and combine them to form (k+1) - MTIS patterns. For example consider,
- P_1 = {a, (t_0), b, (t_1, t_1), e} and P_2 = {b, (t_1), e, (t_3, t_2), c}.
- The idea is to check the equality between the latter (k-1) elements of one pattern and the first (k-1) elements of the other pattern. Here, we can see that {b, (t_1), e} forms the first part of P_2 and the last part of P_1.
- Now, the new pattern P'={a, (t_0), b, (t_1, t_1), e, (?, t_3, t_2), c}.
- The question mark (?) is computed using the time-interval information matrix generated using the set of time intervals {t_0, t_1, t_2, t_3}.
- Here, t_3+t_0=t_3 ⇒ P'= {a, (t_0), b, (t_1, t_1), e, (t_3, t_3, t_2), c}.

$ti_0\,(t=0)$	ti_0	-	-	-
$ti_1\,(0<t\leq 3)$	ti_1	ti_1, ti_2	-	-
$ti_2\,(3<t\leq 6)$	ti_2	ti_2, ti_3	ti_3	-
$ti_3\,(6<t\leq \infty)$	ti_3	ti_3	ti_3	ti_3
	$ti_0\,(t=0)$	$ti_1\,(0<t\leq 3)$	$ti_2\,(3<t\leq 6)$	$ti_3\,(6<t\leq \infty)$

Fig. 4. Time-Interval information matrix. It is used to determine the possible time-intervals that can occur when two MTIS patterns are merged.

Numerical Example:

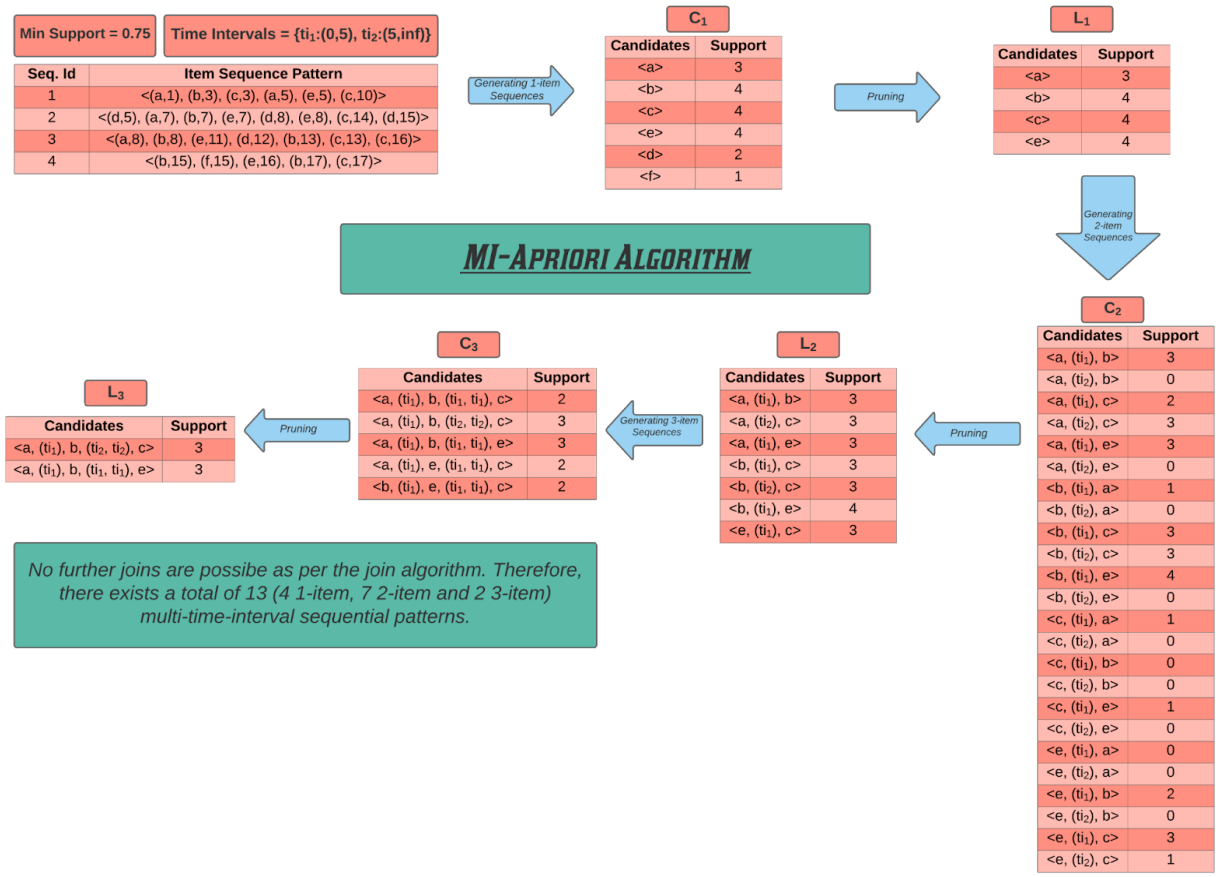


Fig. 5. The working of the MI-Apriori algorithm illustrated with a simple example.

MI-PrefixSpan Algorithm

The idea of MI - PrefixSpan Algorithm is to generate MTIS patterns recursively. In the prefixspan algorithm, we first select a prefix from the list of 1-frequent items. Then the frequent patterns starting with the selected prefix can be obtained from the projected database and recursively performing this algorithm.

Subroutine MI - PrefixSpan(α, k, $S|_\alpha$)

Paramenter : α : a multi - time - interval sequential pattern

k : the length of α

γ : a frequent item

Methods :

Scan $S|_\alpha$ one time

→ If $k = 0$, then find all frequent items in $S|_\alpha$.

For every frequent item γ, append γ to α as α'.

Output all α'.

→ If $k > 0$, then construct the *Table* for α.

⇒ Find all frequent items γ in $S|_\alpha$

⇒ Generate all kinds of possible multi - time - intervals $\&_{k+1}$.

For every cell $Table(\&_{k+1}, \gamma) \geq min\_sup$, append $(\&_{k+1}, \gamma)$ to α as α'.

Output all α'.

For each α', construct α' – projected database, and call MI - PrefixSpan (α', $k+1$, $S|_{\alpha'}$).

Fig. 6. The MI-PrefixSpan Algorithm.

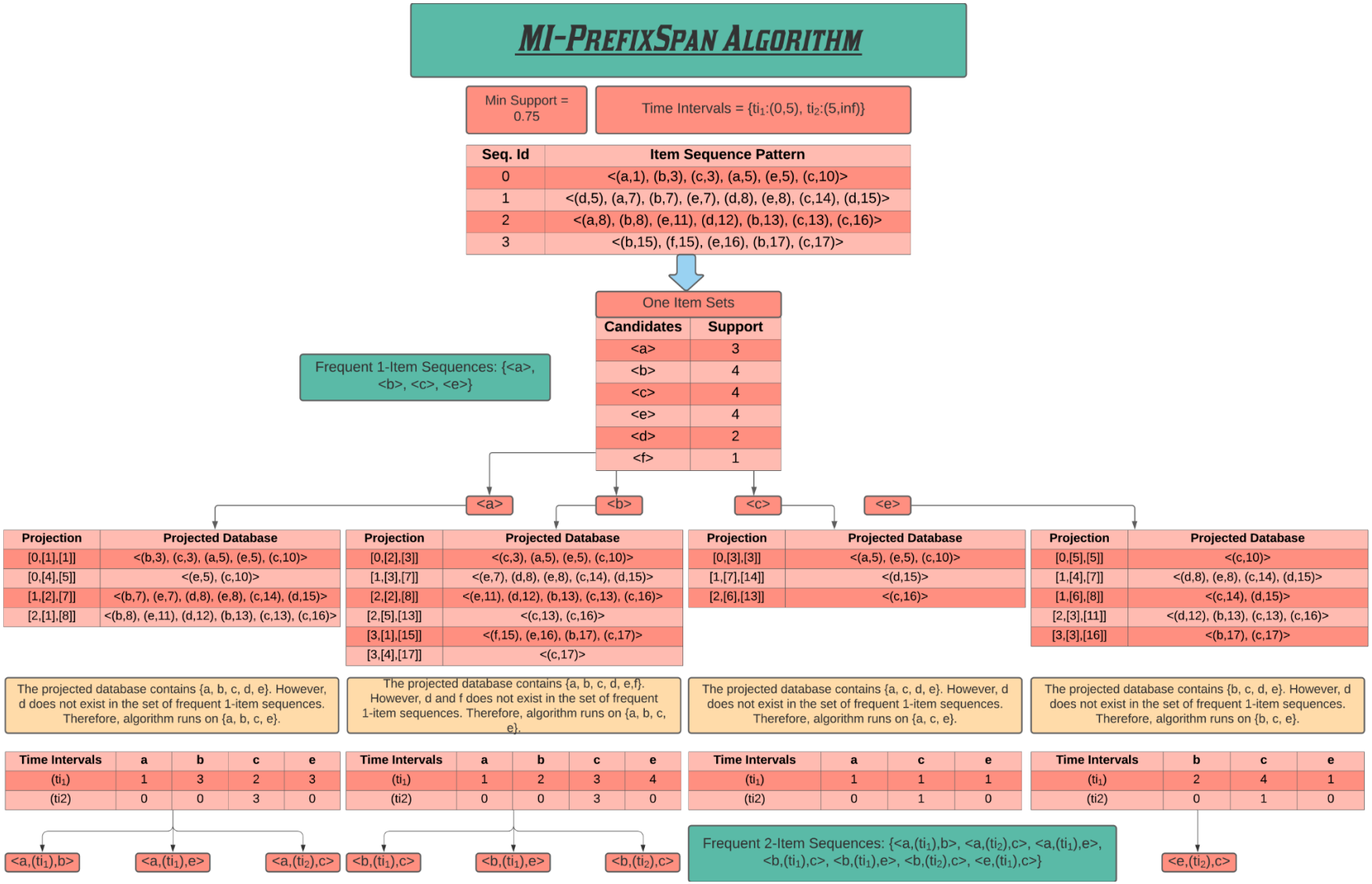
The projected database for a prefix is denoted by a 3-tuple notation for each row in the projected database.

- The first part denotes the sequence id in the database.
- The following list in the second part denotes the indices of the items in the prefix.
- The last part denotes the timestamps of each of the items in the prefix.

Consider the following:

- Item sequence: A = {(a, 1), (b, 3), (c, 3), (a, 5), (e, 5), (c, 10)} and id=10.
- Prefix: P = {a, (ti1), c}, where ti1 = (0, 3)
- The projection notation and the corresponding projected database would look like: [10,[1,3],[1,3]] = {(a, 5), (e, 5), (c, 10)}

Numerical Example:



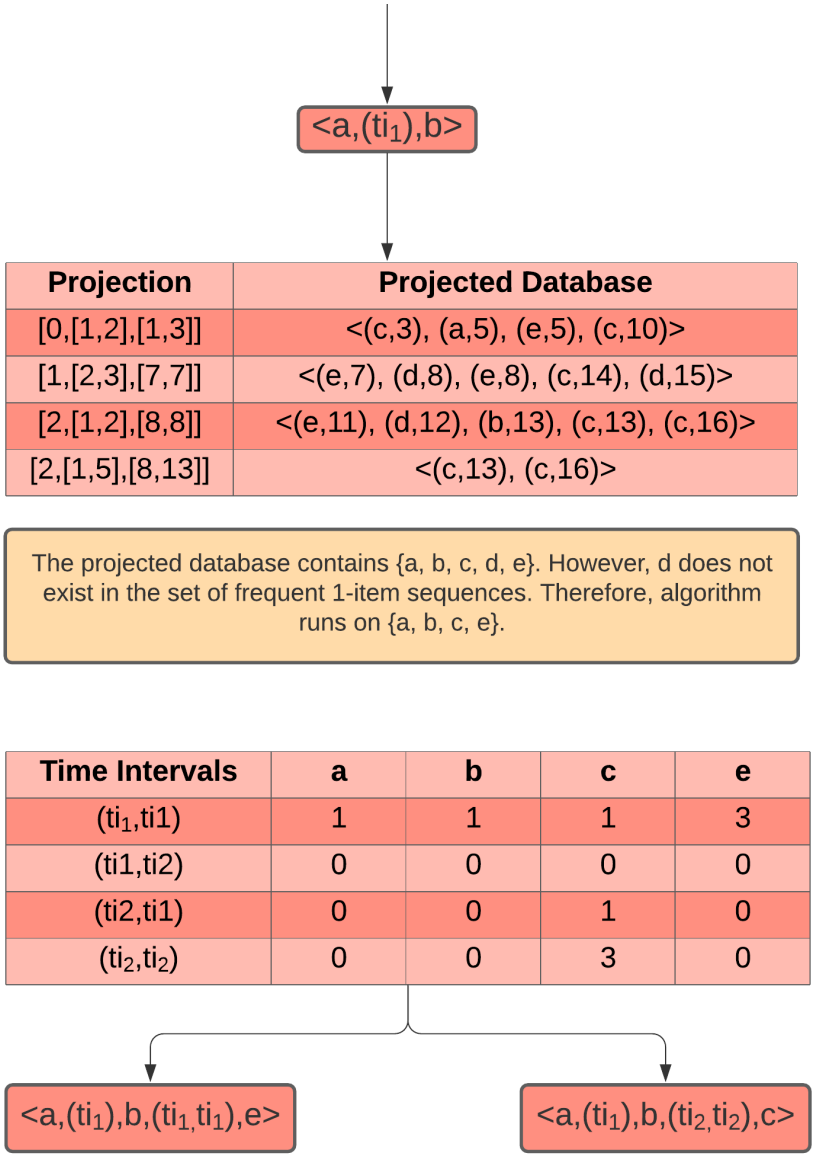


Fig. 7. The working of the MI-PrefixSpan algorithm illustrated with a simple example.

Here, the MI-PrefixSpan Algorithm uses the table generation algorithm for extending the selected prefix. The algorithm generates all possible sets of time intervals from the given time interval list and also generates a list of possible candidates (γ) for the extension step. This leads to the generation of a table with $|T|^{k-1} * d$ where T is the time interval list, k is the length of the extended MTIS pattern that is to be formed and d is the number of distinct elements in the projected database for the prefix in question.

Improvements

We have made a few runtime optimizations to the two algorithms. The goal of these optimizations is to produce slight changes in the implementation of the algorithms which can drastically affect the runtime and memory utilization of the algorithms as the problem is scaled.

MI - Apriori Algorithm: The intended optimization to the MI - Apriori Algorithm is performed by modifying the joinCk method (combines two sequences from C_{k-1} to obtain a candidate for C_k). The approach is to combine the pruning and candidate generation steps, i.e, as each candidate c_i is generated, the minimum support requirement is checked immediately. Only if this requirement is satisfied, the candidate c_i is stored.

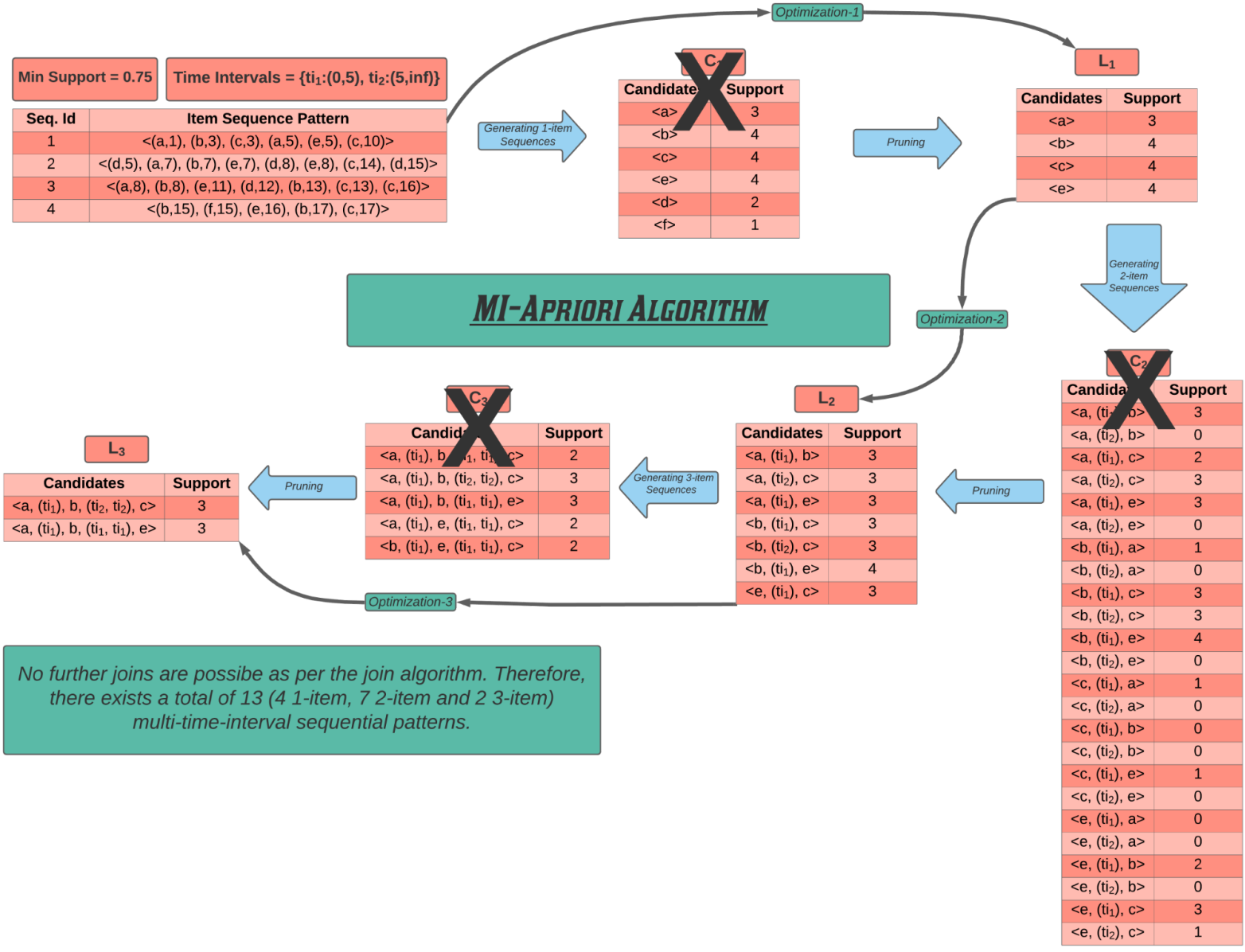


Fig. 8. The MI-Apriori Algorithm with pruning and candidate generation steps combined.

The potential candidate list (C_k) is usually pruned to obtain the list of candidates which satisfy the minimum support requirement. The memory used by this step is of the order $O(|C_k| + |L_k|)$. By combining the candidate generation with the pruning step, this space complexity will go down to $O(L_k)$. The resulting reduction in loop iterations and the number of loads and stores consequently leads to an improvement in the runtime of the algorithm as well.

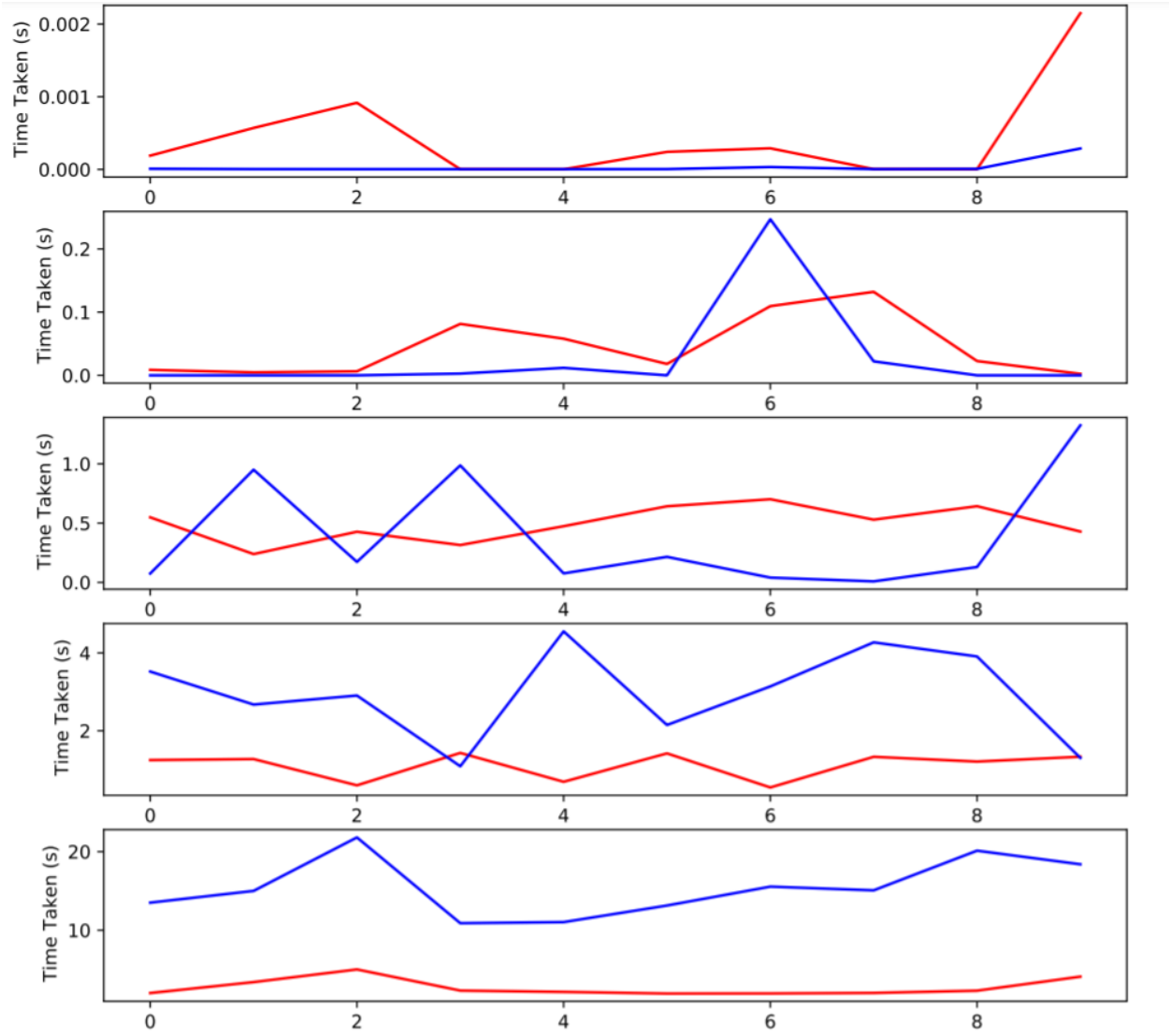


Fig. 9. Graphs of runtime (y-axis) vs. ID of the sequence database (x-axis). The length of sequences in the databases used are (1,10), (10,20), (20,30), (30,40) and (40,50) respectively from the first to the last graph. The red line denotes the MI-Apriori Algorithm that combines pruning and candidate generation and the blue line denotes the original implementation.

From the graphs in **Fig. 9.**, we can clearly see that while there is not much change in the runtimes for executing the smaller sequences, there is a drastic difference in the runtime as the sequence length increases. The difference in runtime is tangible just from the small datasets it has been run on.

MI - PrefixSpan Algorithm: The intended optimization to the MI - PrefixSpan Algorithm is performed on the time interval table generation step (generates all the time interval possibilities for expanding the prefix at the time). As all the time interval possibilities are being generated in the table, there is no possibility for error in generating MTIS patterns.

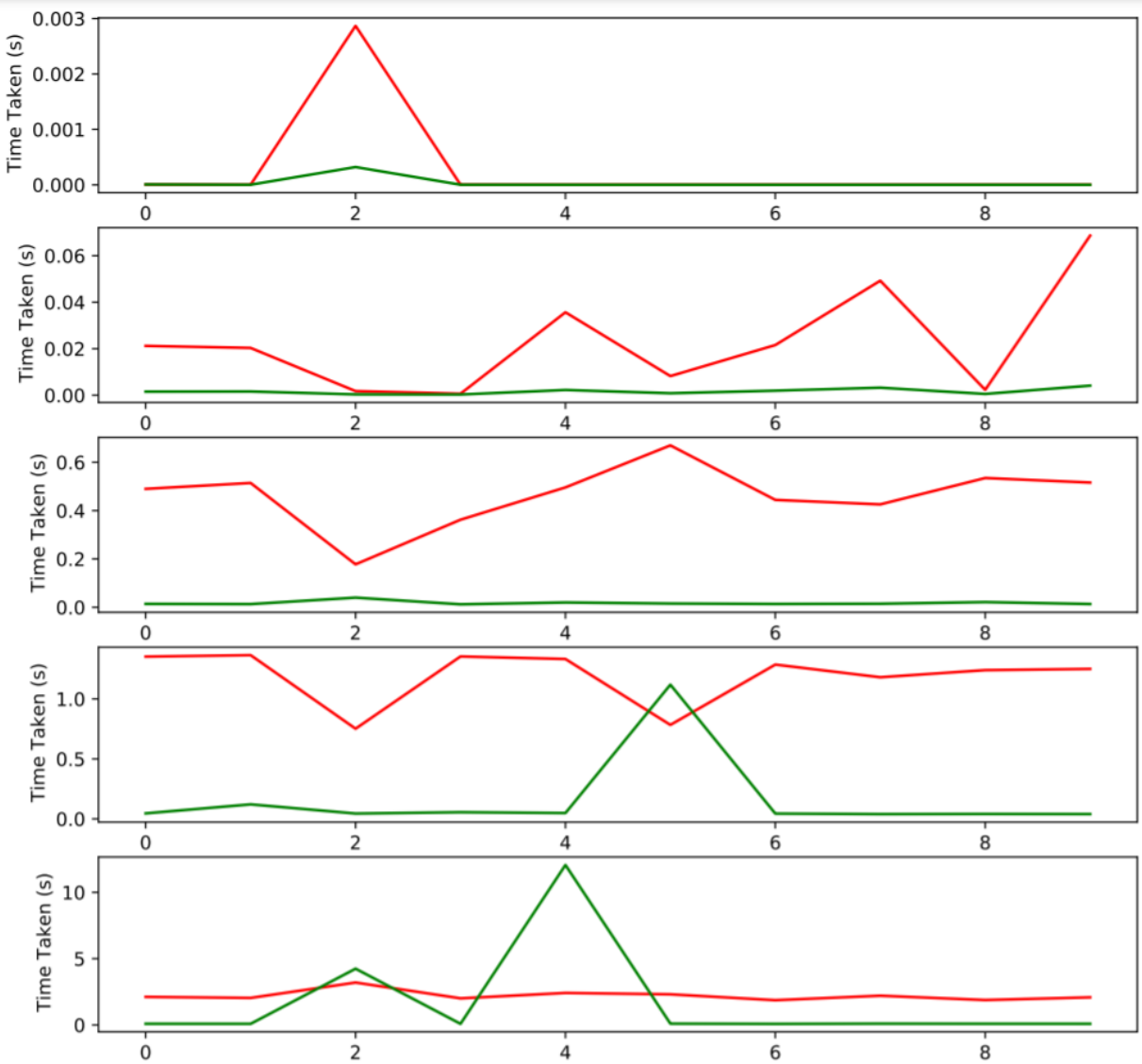


Fig. 10. Graphs of runtime (y-axis) vs. ID of the sequence database (x-axis). The length of sequences in the databases used are (1,10), (10,20), (20,30), (30,40) and (40,50) respectively from the first to the last graph. The red line denotes the faster MI-Apriori Algorithm implementation and the green line denotes the MI-PrefixSpan Algorithm implementation.

However, the memory utilization of the table generation step is of the order of $O(|T|^{k-1} * d)$, where $|T|$ denotes the cardinality of the time interval list ($|T| = 4$ for the sample time list). Here, 'k' denotes the length of the MTIS pattern (new prefix) we're trying to obtain and 'd' is the number of 1-frequent items.

In terms of scalability, the time complexity doesn't fare any better. In addition to the exponentially increasing table generation step, the support of each such generated pattern has to be computed. The time complexity of the table generation step as well as computing the support is of the order of $O(|T|^{k-1} * d * s)$, where d is the number of distinct frequent items in the projected database and s is the time complexity for calculating the support.The alternative proposed is, to simply iterate through the projected database and produce a frequency map for each item in the projected database. Using this frequency map, calculating the support is trivial and does not take any additional time. The worst case time complexity and space complexity is $O(L)$ where L denotes the number of items in the projected database for a given prefix. As the length of the pattern increases, this could be a better alternative to the exponential time complexity of the table generation step.

From the graphs in **Fig. 10.**, we can clearly see that the prefixspan algorithm consistently outperforms even the optimized apriori algorithm. It even scales well with increasing sequence lengths even though experimental results to the contrary are mentioned in the paper.

PrefixSpan Algorithm without the Table Generation Step

As mentioned above, in this algorithm the computationally expensive table generation step is replaced with the more efficient database spanning step. Instead of generating all possible time intervals and their supports, we simply iterate through the projected database while maintaining a count of the number of occurrences of each pattern observed. At the end of this iteration, it is trivial to find the support of these patterns, as the number of occurrences have already been recorded. When compared with the exponentially increasing complexity involved in the table generation algorithm, this seems to be computationally faster and more memory efficient.

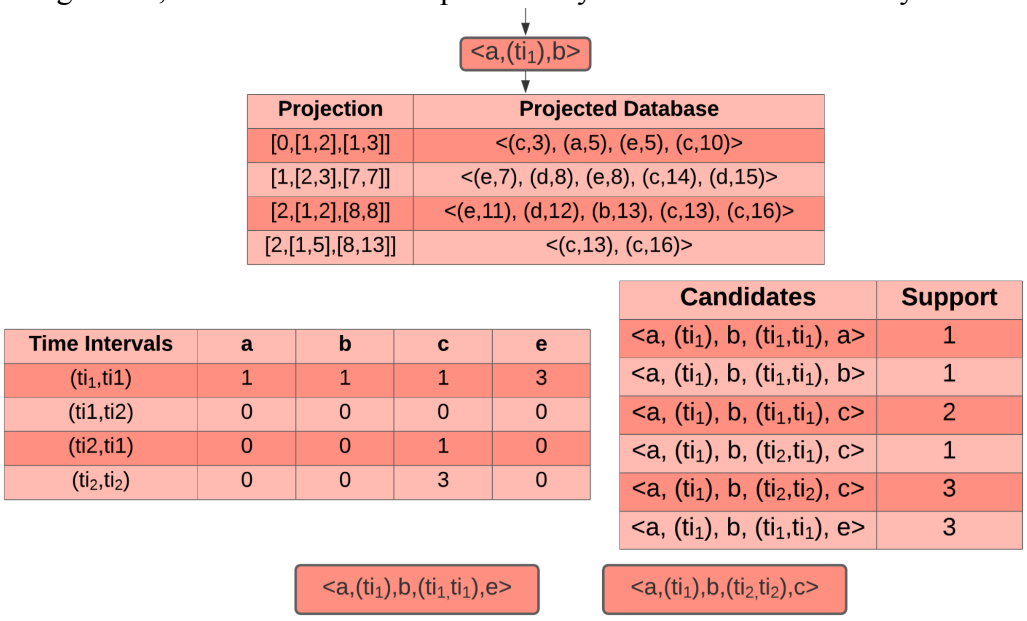
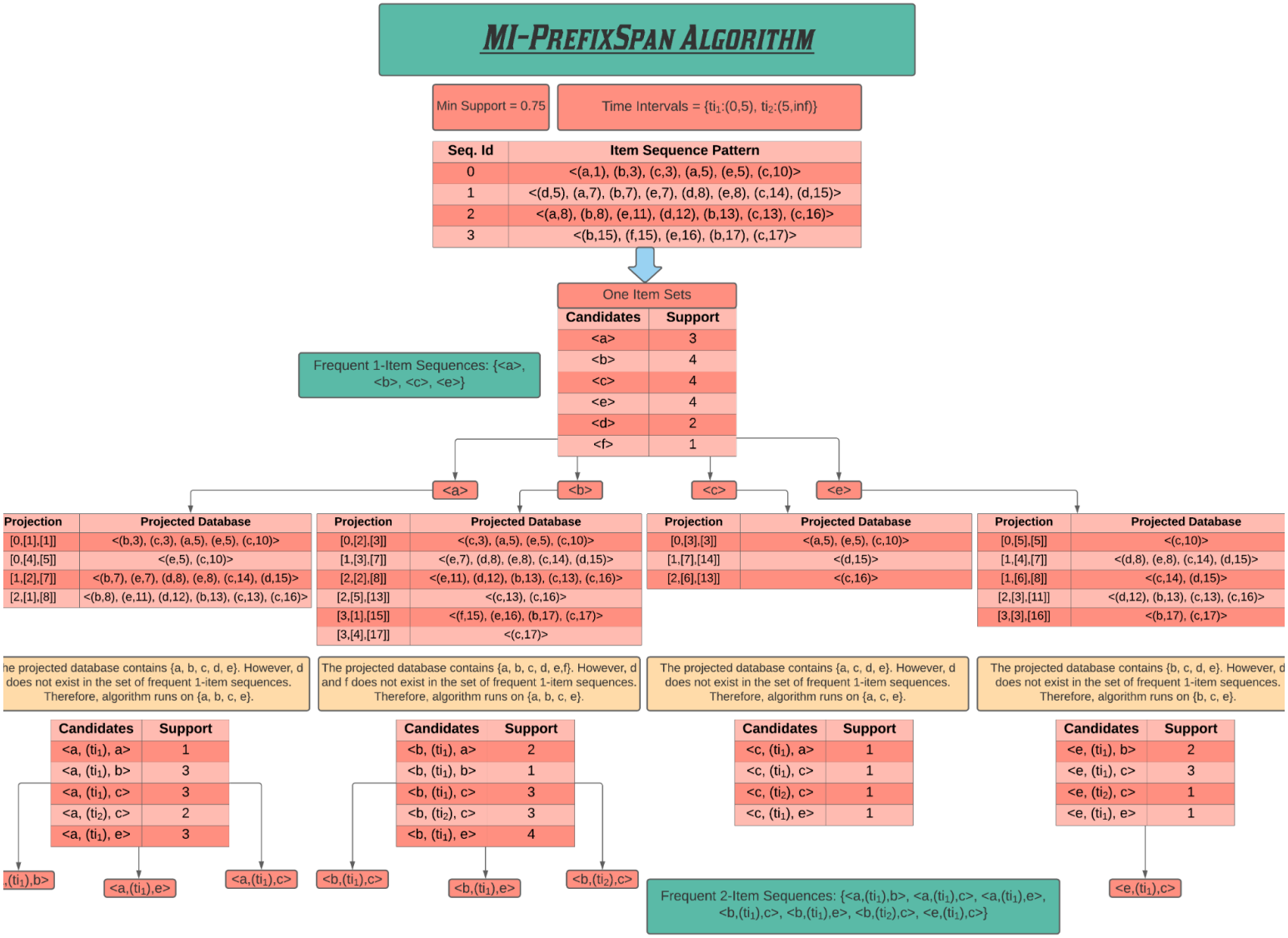


Fig. 11. Both Implementations Compared: Table Generation⇒O(16) and Without Table Generation⇒O(6)

Numerical Example:



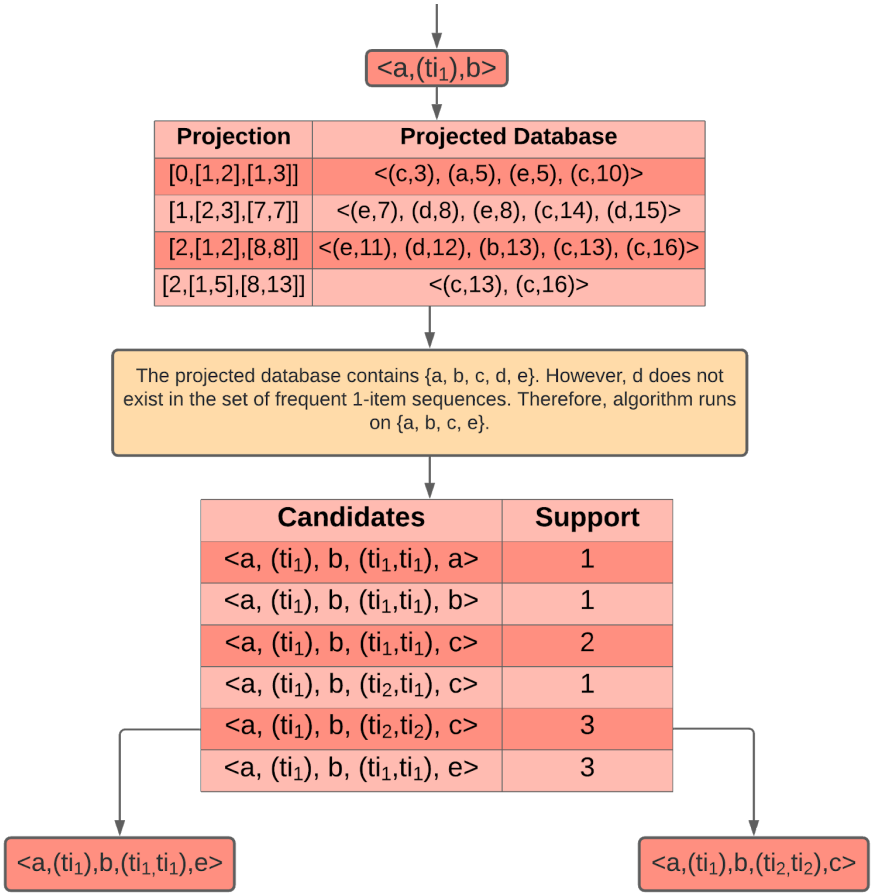


Fig. 12. The working of the MI-PrefixSpan algorithm illustrated with a simple example.

Future Work:

In this work, we have explored both MI-Apriori and MI-PrefixSpan algorithms for sequence data mining. Other commonly used algorithms for pattern mining are FP-Growth and Eclat algorithms. We have also only used randomly generated datasets for the testing, comparison and analysis of the algorithms. This may not reflect the performance on real-world datasets. So, more extensive testing and execution on such real-world datasets is essential for practical applications of these algorithms.

References:

[1] Hu, Y. H., Huang, T. C. K., Yang, H. R., & Chen, Y. L. (2009). On mining multi-time-interval sequential patterns. Data & Knowledge Engineering, 68(10), 1112-1127.

[2] Chen, Y. L., Chiang, M. C., & Ko, M. T. (2003). Discovering time-interval sequential patterns in sequence databases. Expert Systems with Applications, 25(3), 343-354.