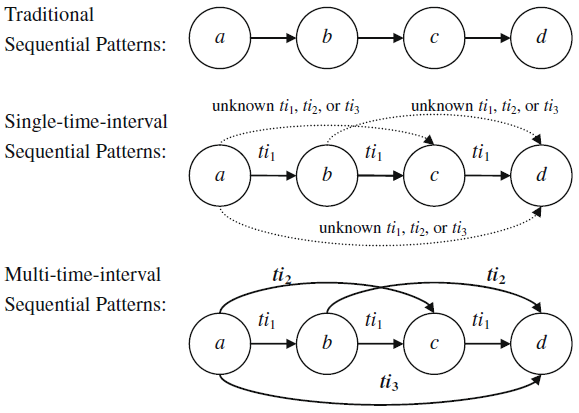
Multi-Time-Interval Sequential Pattern Mining

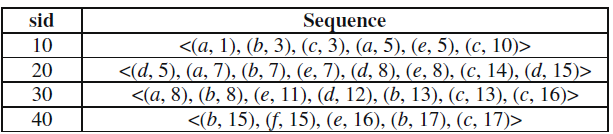
| J Sudarsanan  Sangeeth S V | * 181CO222 * 181CO246 |
| --- | --- |
|

# Overview

Multi-Time-Interval Sequential (MTIS) Patterns are patterns that contain information about the time differences between sequentially occurring events. For example, a person who purchases the first volume of a book series, is likely to purchase the second one a week after, and the third one another 2 weeks later, and so on. Just the time-interval sequential pattern mining algorithms would only be able to get the time interval information between consecutive events. By using multi-time-interval sequential patterns, we seek to establish such time-relations between every pair of events in a sequence.

  
***Fig. 1. Different Kinds of Sequential Patterns***

This example shows four events a, b, c and d, occurring sequentially. The diagram shows how a multi-time-interval pattern needs to establish the time intervals between one event and each successive event so that all the time intervals between any two events are well-defined.

  
***Fig. 2. A Sequence Database***

# Properties of Multi Time Interval Sequential Patterns

* Support: Support of a multi-time interval sequence pattern is defined as the ratio of number of item sequences in which it is contained to the total number of item sequences in our database.
* Descending Property: Each successive time interval in each &i of a multi-time interval sequence, is at most equal to the previous time interval. For example, is valid, whereas, is not valid.
* Containing Property: A multi-time interval sequence is said to be contained in a data sequence if for every pair of items, the difference in the timestamps is same as that indicated in the MTIS pattern. For example, is contained in data sequence where
* Anti-Monotonicity Property: If a multi-time-interval sequence is frequent, so are all of its subsequences. Accordingly, if a multi-time-interval sequence 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.

# Dataset

For the purposes of understanding and evaluating the two algorithms, they were run on the example dataset from the paper which is also shown in (***Fig 2****)*. For consequent testing and comparisons, we have made use of randomly generated item sequences using the same criteria of minimum support and time intervals. The dataset for the MTIS pattern mining task is to be in the form of a list of lists of total size of which each is a list of items together with their timestamps. Each list is associated with a Transaction ID which identifies the entire list as a single row in the dataset. For a pattern to satisfy the minimum support , the pattern has to be contained in a minimum of sequences in

The chosen dataset is the Jewelry Dataset which can be found [here](https://drive.google.com/file/d/1EfkvA3KH2fQObFF1jWUv7NNUqeQhj9Me/view?usp=sharing). The dataset is in a completely different format, so the following steps are taken to prepare the dataset.

* The date is in the format *“yyyy-mm-dd hh:mm:ss UTC”*. This string is to be converted to the relevant timestamp data that is required for every item in our dataset. In this instance, we are obtaining the year, month and day of every transaction.
* A single outlier exists in the quantity column (*quantity is two where every other transaction has one*)*.* This outlier is removed.
* All rows with any missing values are also removed as they do not convey much information to the algorithm.
* The *item\_name* is obtained as the ‘\_’-separated string concatenation of the following columns: *category\_id, category\_code, metal* and *color.*
* The Dataset contains the jewelry transaction data from November 2018 to November 2020. Each month is divided into 3 and each of these classes is allocated a separate *Transaction ID* for use by the algorithm.
* The *timestamp* is obtained as the integer day of the purchase in question.
* Now, using *LabelEncoder* from *sklearn.preprocessing* module, we obtain the integer labels of both the *Transaction ID* and the *item\_name*.

# 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. 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, experiments conducted using our implementations of the algorithms shows that PrefixSpan does not scale well with larger item sequences. On the dataset selected, Apriori performs significantly better due to the limited capacity of the host system.

We also perform a few optimizations in the runtime and space complexity on the MI-PrefixSpan algorithm. This optimization in the PrefixSpan algorithm is done by replacing the expensive table generation step with a more efficient spanning algorithm.

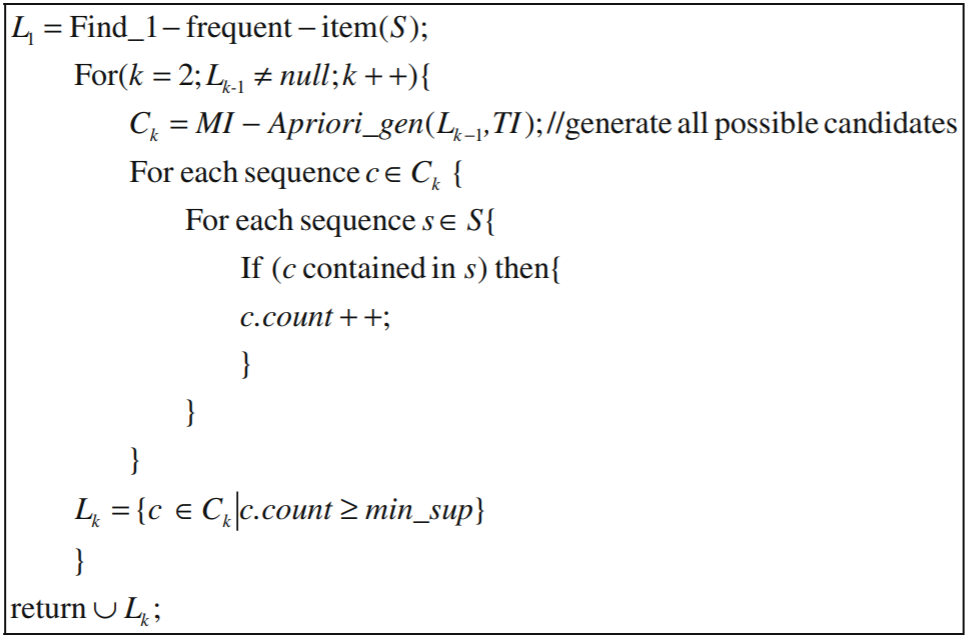
# Algorithms

There are two algorithms mentioned in the paper:

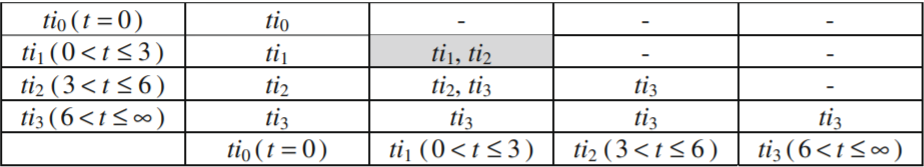
* MI-Apriori Algorithm: This algorithm tries to generate higher order sequences (Ck) from lower order sequences (Ck-1). The basic principle behind this algorithm is that any sequence of length k, must have two subsequences of length (k-1). ⇒ Anti-Monotonicity Property.
* MI-PrefixSpan Algorithm: The 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.

# 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.

  
***Fig. 3. The MI-Apriori Algorithm.***

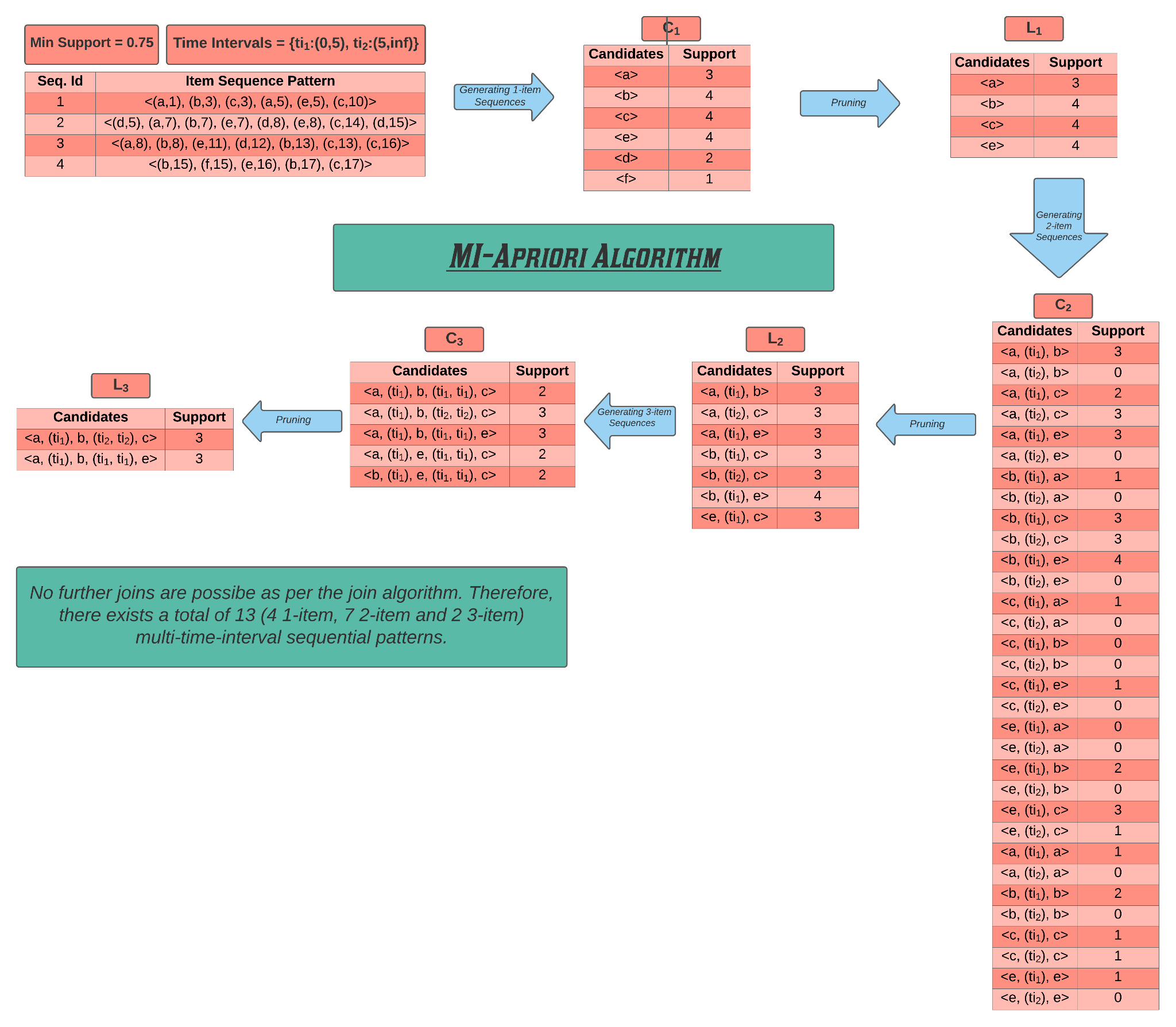
* The basic join operation seeks to take two k - MTIS patterns as input and combine them to form (k+1) - MTIS patterns. For example, consider, and
* The idea is to check the equality between the latter (k-1) elements of one pattern and the first (k-1) elements of the other pattern. Here, we can see that {b, (t1), e} forms the last part of P1 and the first part of P2.
* Now, the new pattern
* The question mark (?) is computed using the time-interval information matrix generated (***Fig 3***) using the set of time intervals
* Here,
* If there had been multiple entries in the time interval information matrix, all possible combinations would have been considered.

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

The MI-Apriori Algorithm is an intuitive approach to obtaining multi time-interval-sequential patterns from a list of item sequences. This approach was developed from the principles of the existing Apriori algorithm for Sequential Pattern Mining. Therefore, it has the same advantages and disadvantages.

* Advantages:
  + An easy to understand approach to this new mining task that builds upon pre-existing approaches.
  + An iterative incremental algorithm decreases the load on the CPU, GPU, etc.
* Disadvantages:
  + The algorithm is very time intensive. Each joinCk method generates a large number of potential patterns in the candidate set, of which each one has to be separately taken and evaluated based on the minimum support criterion. Moreover, the compatibility condition checking is itself an expensive operation.
* Space Complexity:
  + The memory used by the MI-Apriori algorithm is of the order of at each level Here, is the candidate set of MTIS-patterns that are of length and is the pruned set of MTIS-patterns of length that satisfies the minimum support requirement.
* Time Complexity:
  + The most expensive step of the MI-Apriori algorithm is the joinCk method which takes in two patterns of length and outputs a length pattern.
  + For this task, the algorithm considers every pair of patterns in and checks their compatibility. This operation is
  + Then support for every such generated pattern is computed. Therefore, the time complexity comes out to be where is the dataset, is the size of the maximum item sequence among the transactions and is the set passed into the joinCk method.

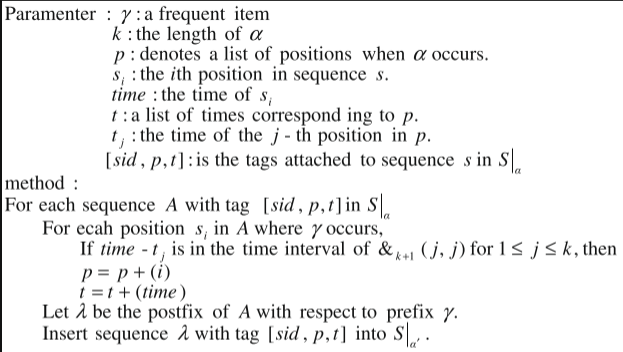
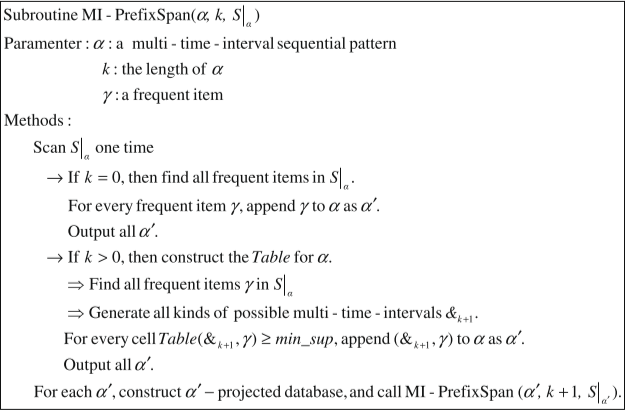
# Numerical Example - Apriori

  
***Fig. 5. MI-Apriori Algorithm explained with a numerical example***

# 

# 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.

  
***Fig. 6. The MI-PrefixSpan Algorithm.***

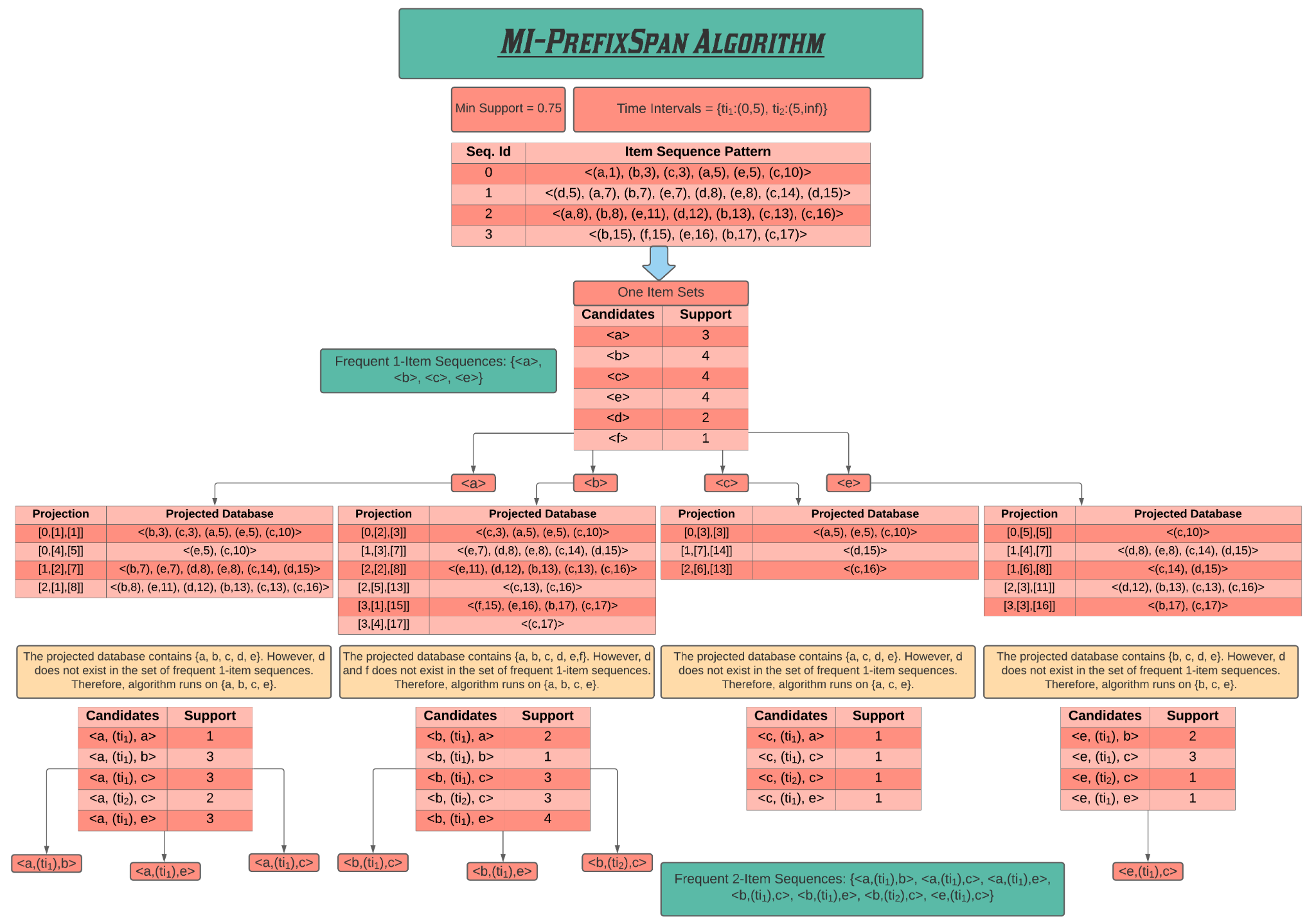
The projected database for a prefix is denoted by a 3-tuple notation for each row in the projected database.

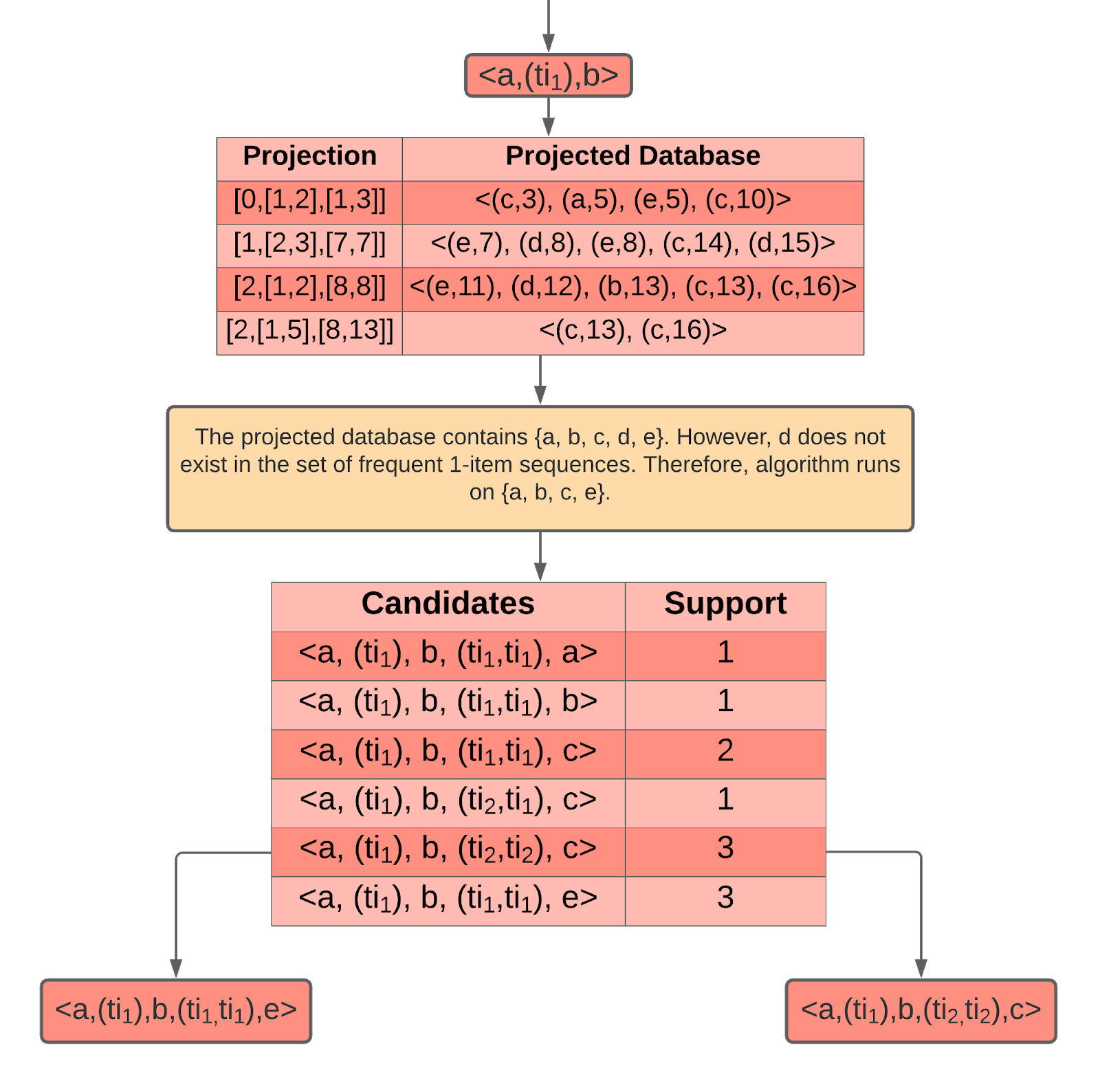
* The first part denotes the *Transaction ID* in the dataset.
* The second part is a list which corresponds to the indices of the items in the prefix.
* The last part denotes a list of the timestamps of each of the items in the prefix.

Consider the following:

* Item sequence: and
* Prefix: where
* The projection notation and the corresponding projected database would look like:

# Numerical Example - PrefixSpan

  
***Fig. 7(a). MI-PrefixSpan Algorithm explained with a numerical example.***

  
***Fig. 7(b). MI-PrefixSpan - contd.***

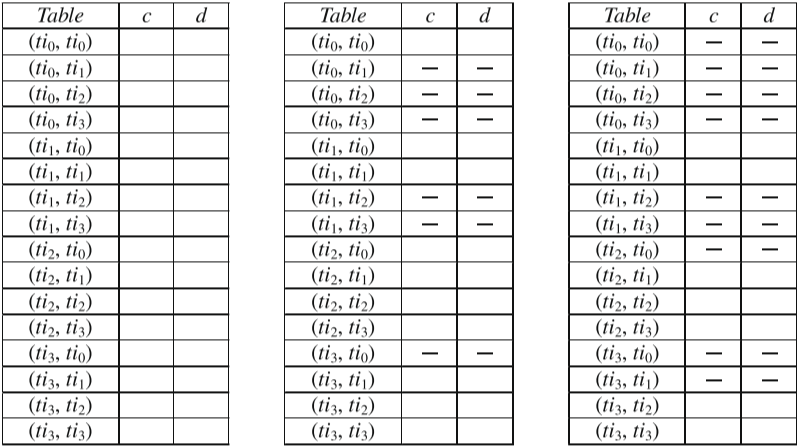
The MI-PrefixSpan algorithm extends the PrefixSpan algorithm from sequential pattern mining to accomodate the need for the time interval relationships between all items in the pattern.

* Advantages:
  + Significantly faster than the MI-Apriori Algorithm and experimental results show the same.
  + A recursive approach cuts down the number of scans of the dataset to get the “supports” of the candidate items.
  + The PrefixSpan algorithm is much more versatile for actual use. If needed, we could specify the prefix our pattern has to start with and the algorithm can be applied with little to no adjustment. However, the only way to do the same task with Apriori algorithm is to generate all possible patterns and thn cross check the prefix of each obtained pattern.
* Disadvantages:
  + The algorithm is very memory intensive. The recursive algorithm generates a lot of data at each level and this occupied memory can be deallocated only after that particular branch of the recursion tree has been completely exhausted. This causes a huge load on the RAM which can crash the whole thing.
* Space Complexity:
  + The memory used by the MI-PrefixSpan algorithm is significantly higher than its counterpart. It is of the order of for each candidate pattern as the projections have to be stored for each candidate pattern. This combined with the recursion stack’s memory requirements, a huge memory requirement persists. Here, is the transactions in the dataset used.
* Time Complexity:
  + The time complexity of the PrefixSpan algorithm is relatively better than the apriori algorithm.
  + For each candidate pattern , the projected database has to be scanned through once to get the possible extensions of the candidate prefix.
  + This operation eventually amounts to time complexity of where is the total number of patterns generated, is the transactions in the dataset and is the size of the maximum item sequence among the transactions.

# Optimization

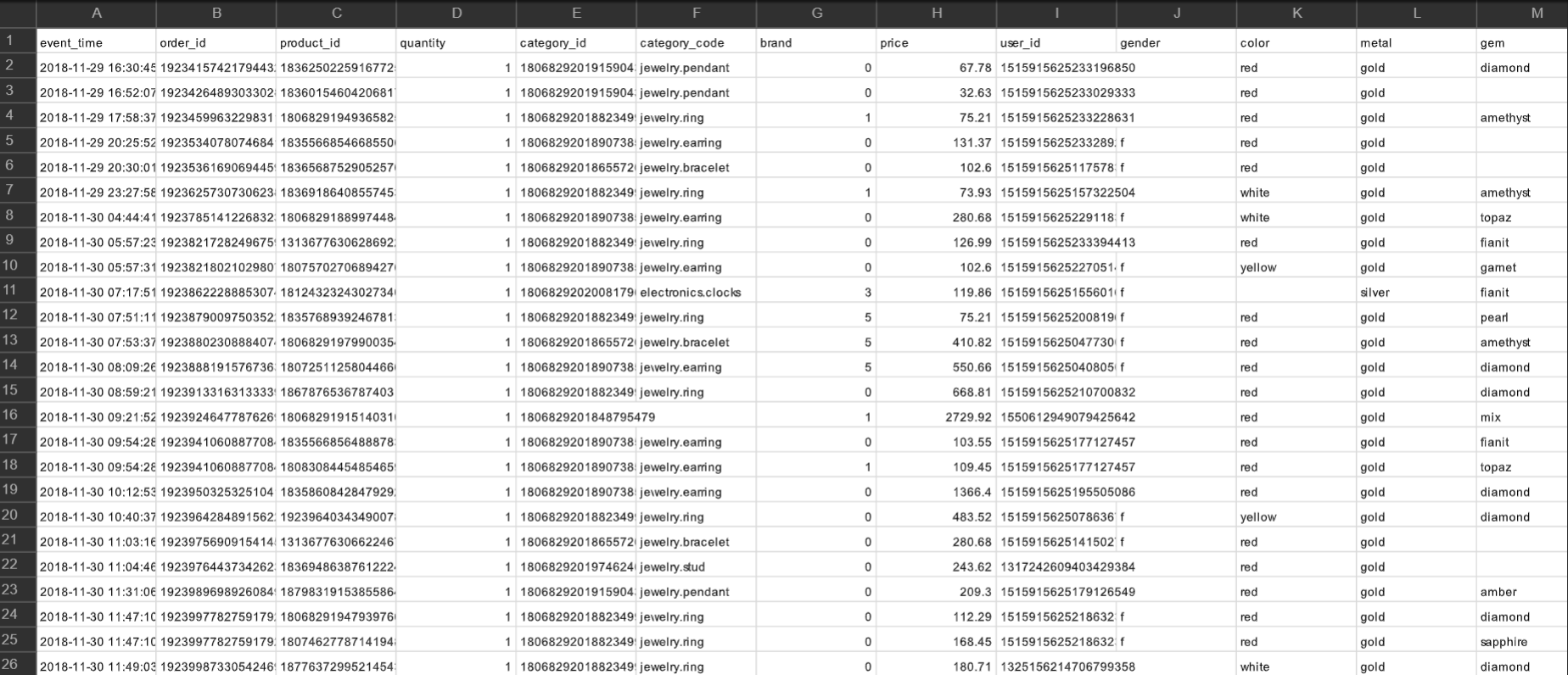
The reference paper taken, speaks of a table generation step in the PrefixSpan algorithm for the purpose of extending the prefix of length currently at hand. The purpose of this is to generate all possible time intervals of length and then compute the support for each valid time interval sequence. This operation requires one scan of the projected database which in the worst case is the entire dataset itself. Therefore the time complexity would be , where is the set of frequent one-items, is the time intervals used for generating the patterns, is the length of the pattern to be generated, is the transactions in the dataset and is the size of the maximum item sequence among the transactions. This also creates an additional space requirement in the order of for the time interval matrix.

The suggested optimization is to skip the table generation step and simply scan through the projected database keeping count of each possible pattern extension and finally pruning the list based on corresponding support. This will remove the table generation matrix from the algorithm and instead replace it with a list of candidate single-item sequences and their timestamps. Moreover only patterns existing in the dataset will end up being stored, thus removing much of the memory overhead. The time complexity would be where is the set of frequent one-items, is the set of extensions possible for current prefix of length , is the transactions in the dataset and is the size of the maximum item sequence among the transactions. The space complexity will also be reduced to just .

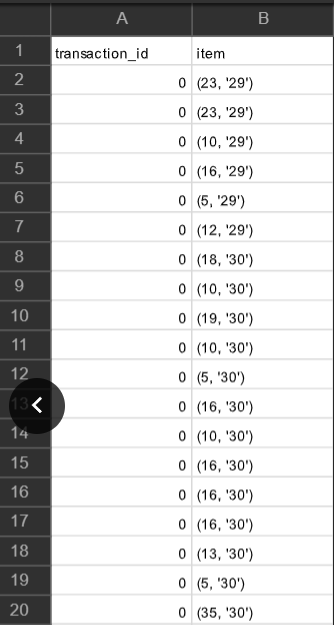
  
***Fig 8. Table Generation Step in MI-PrefixSpan Algorithm as described in the paper. This is for length 2 prefixes and the table is already in the order of 16 which is .***

# Screenshots

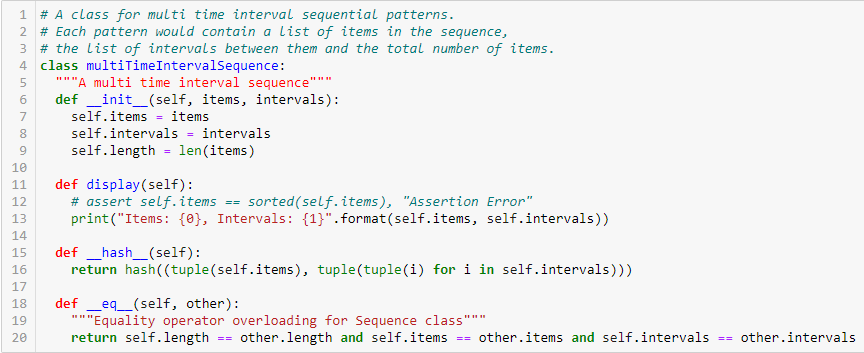
* Dataset is initially in a very cluttered raw format and is unfit for direct use in our algorithm.

  
***Fig 9. Unprepared Dataset***

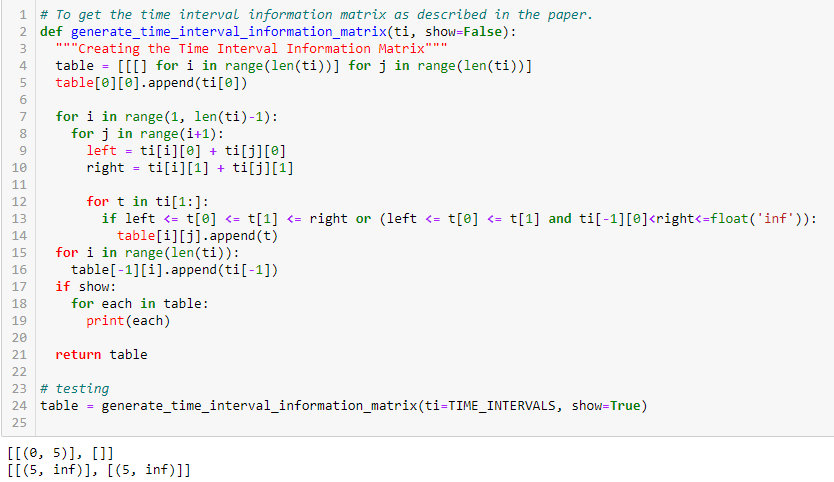
* The dataset is prepared using the procedure described in the Dataset section of this report.

  
***Fig 10. Prepared Dataset with each label pointing to specific items in the dataset.***

* The *MultiTimeIntervalSequence* class with the equality and hashing procedure defined.

  
***Fig 11. The Multi-Time-Interval-Sequence class.***

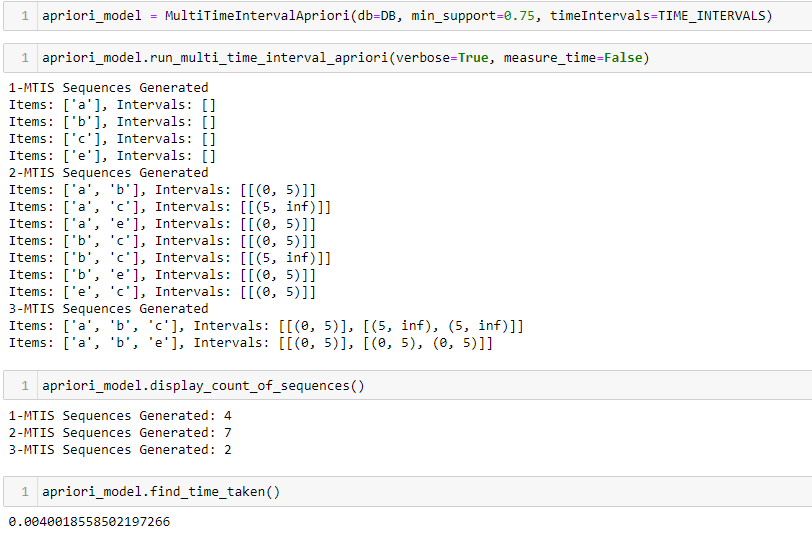
* The time interval information matrix table generation step which is integral to the apriori algorithm.

  
***Fig 12. Time Interval Information Matrix Generation***

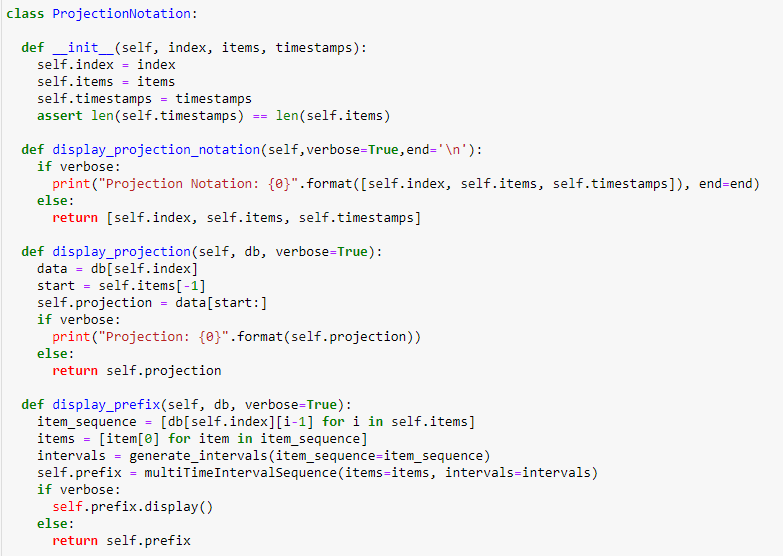
* The *joinCk* method which generates MTIS patterns incrementally.

  
***Fig 13. joinCk method which is the critical step in MI-Apriori Algorithm***

* Working of the Apriori Algorithm on the same example shown in (***Fig 5.***).

  
***Fig 14. Working of the MI-Apriori Algorithm.***

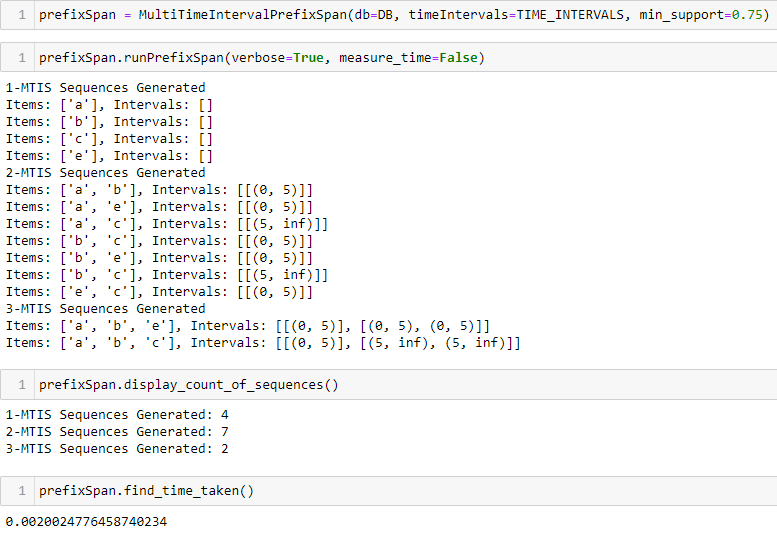
* *ProjectionNotation* class for storing the projections efficiently

  
***Fig 15. ProjectionNotation class***

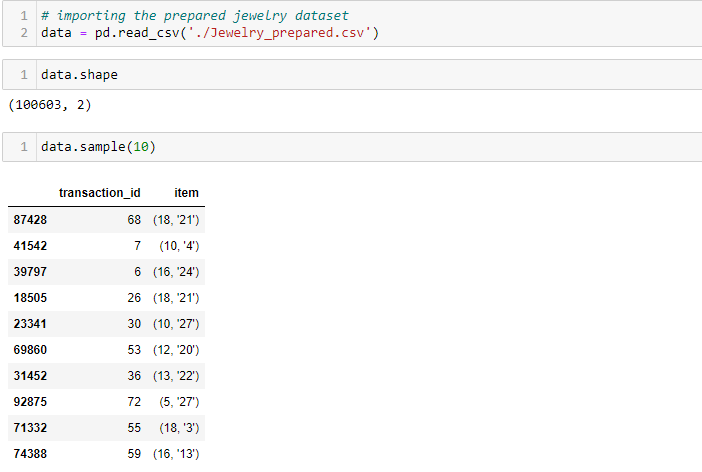
* The recursive pattern generation step which forms the crux of the PrefixSpan algorithm

  
***Fig 16. Recursive step in PrefixSpan algorithm***

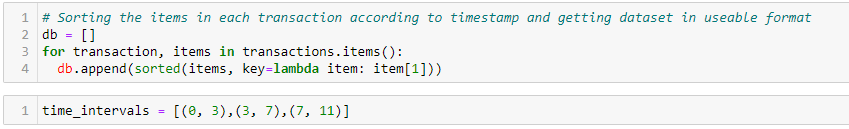
* Working of the PrefixSpan algorithm on the same example as shown in (***Fig 7***).

  
***Fig 17. Working of the MI-PrefixSpan Algorithm***

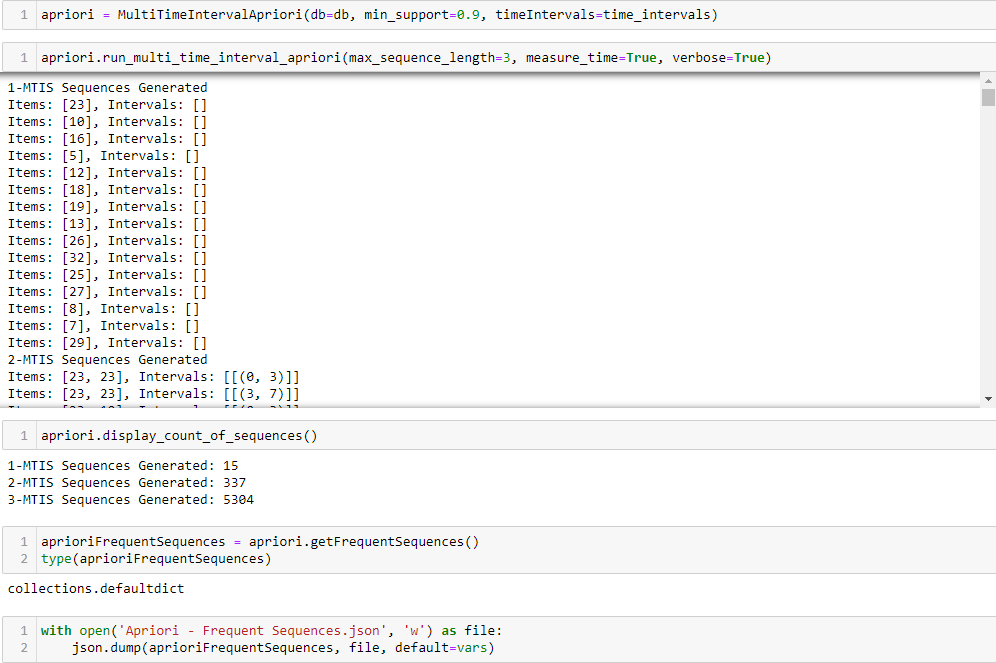
* Importing dataset from the .csv file.

  
***Fig 18. Importing dataset***

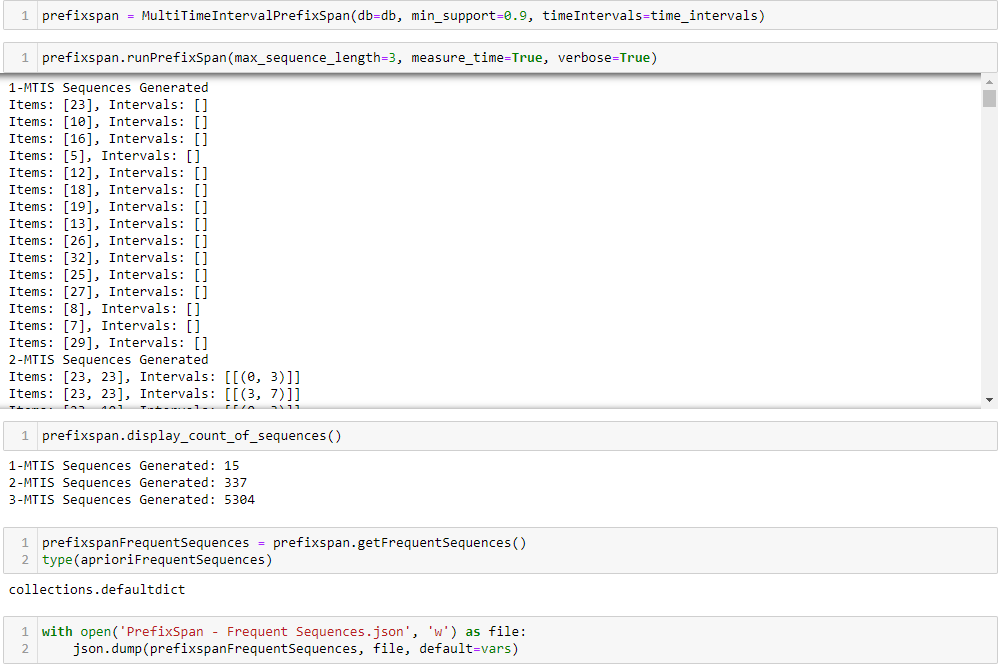
* Getting dataset and time intervals in the required format for applying the algorithms.

  
***Fig 19. Getting the dataset as a list of lists and defining the time intervals to be used in the algorithm.***

* MI-Apriori Algorithm on the Jewelry Dataset. The generated patterns are stored in a json file found here. (Doesn’t run on colab as it overloads the available RAM. Use a better local processor.)

  
***Fig 20. Running the MI-Apriori Algorithm on the Jewelry Dataset***

* MI-PrefixSpan Algorithm on the Jewelry Dataset. The generated patterns are stored in a json file found here. (Doesn’t run on colab as it overloads the available RAM. Use a better local processor.)

  
***Fig 21. Running the MI-PrefixSpan Algorithm on the Jewelry Dataset.***

# Links

* [Unprepared Dataset](https://drive.google.com/file/d/1EfkvA3KH2fQObFF1jWUv7NNUqeQhj9Me/view?usp=sharing)
* [Prepared Dataset](https://drive.google.com/file/d/1i9y2CvFgZZazt549Uxz2eNQ4pvmKZoho/view?usp=sharing)
* [Item Classes](https://drive.google.com/file/d/1-1Kc5lZnpUHFHAKtymqmL7kkPaQOkVxy/view?usp=sharing)
* [Transaction Classes](https://drive.google.com/file/d/1-0NOaxYMGXyXyGcqRv365XS-AL3ylPZ5/view?usp=sharing)
* [Reference Paper](https://drive.google.com/file/d/1pKB4I3eQGA2A7RjBcvpUlO63pjxmU4pW/view?usp=sharing)
* [Dataset Preparation Colab Notebook](https://colab.research.google.com/drive/1_e-NHTZr6-WIhGN_LS2py25sbVzUL69h?usp=sharing)
* [MTIS Colab Notebook](https://colab.research.google.com/drive/1lD_wU3R3fiUApudN9uBYJgkxSrIm0t1a?usp=sharing)
* [Apriori Frequent Patterns](https://drive.google.com/file/d/1hY82L5KbDJMK35Y4ahRiJV9lFFB9Z8tl/view?usp=sharing)
* [PrefixSpan Frequent Patterns](https://drive.google.com/file/d/1rzi_YkIxEWE4xjY9y4VlcAQrbMOERW0P/view?usp=sharing)

# Conclusion

We have discussed the two well-known algorithms for mining Multi-Time-Interval Sequential patterns from a database of sorted time-stamped item sequences. Both algorithms have been implemented and executed on a dataset with successful results. Experimental results are in accordance with the results in the paper. In small datasets, the PrefixSpan algorithm is vastly superior to the Apriori algorithm. However, on large datasets, Apriori fares significantly better in terms of memory utilization.

# References

[1] Ya-Han Hu, et. al, *“On mining multi-time-interval sequential patterns”*, Data & Knowledge Engineering, Volume 68, Issue 10, 2009.