Multi-Time-Interval Sequential Pattern Mining

J Sudarsanan	-	181CO222
Sangeeth S V	-	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.

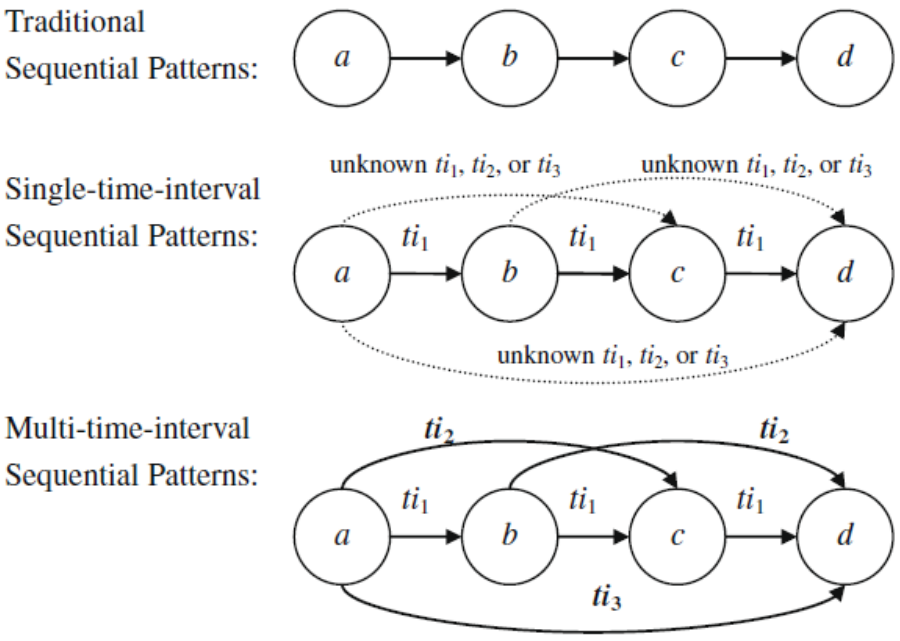

Fig. 1. Different Kinds of Sequential Patterns

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.

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

Properties of Multi Time Interval Sequential Patterns

I. *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.

II. *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, (ti_3, ti_2, ti_2, ti_1) is valid, whereas, (ti_2, ti_3, ti_2, ti_1) is not valid.

III. *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, $\{a, (ti_1), b, (ti_2, ti_1), c, (ti_2, ti_1, ti_1), d\}$ is contained in data sequence $A = \{(a, 1), (b, 3), (c, 5), (d, 6), (e, 8)\}$ where $ti_0: [0], ti_1: [0, 3], ti_2: [3, 6], ti_3: [6, \infty)$.

IV. *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.

Work Done: 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, $P_1 = \{a, (ti_0), b, (ti_1, ti_1), e\}$ and $P_2 = \{b, (ti_1), e, (ti_3, ti_2), c\}$.
- 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, (t1), e} forms the last part of P1 and the first part of P2.
- Now, the new pattern $P' = \{a, (ti_0), b, (ti_1, ti_1), e, (?, ti_3, ti_2), c\}$.
- The question mark (?) is computed using the time-interval information matrix generated (***Fig 3***) using the set of time intervals (ti_0, ti_1, ti_2, ti_3).
- Here, $ti_3 + ti_0 = ti_3 \Rightarrow P' = \{a, (ti_0), b, (ti_1, ti_1), e, (ti_3, ti_3, ti_2), c\}$.
- If there had been multiple entries in the time interval information matrix, all possible combinations would have been considered.

$ti_0\,(t=0)$	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 - Apriori

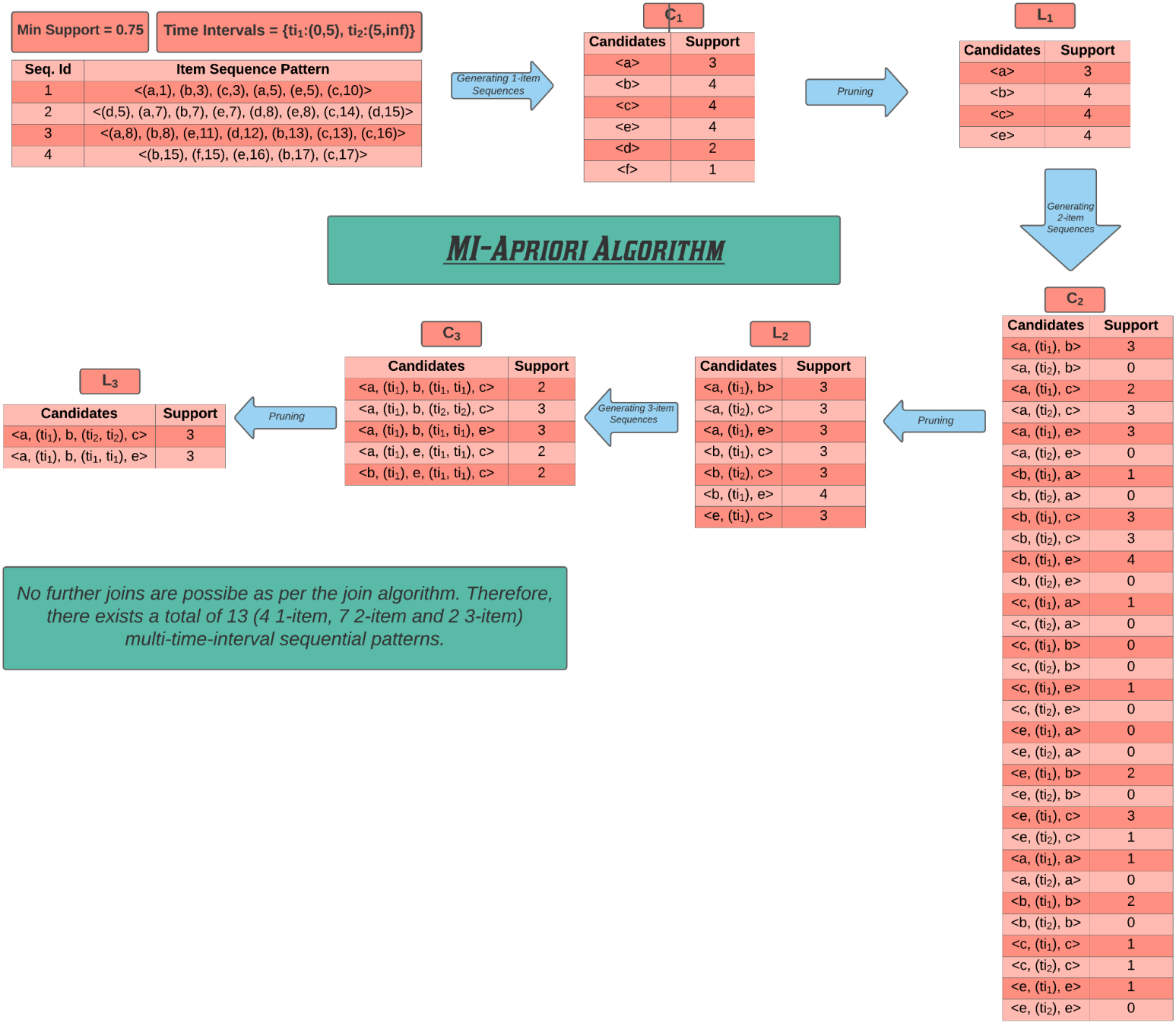

Fig. 5. MI-Apriori Algorithm explained with a numerical example

Numerical Example - PrefixSpan *(Implementation-Work In-Progress)*

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.

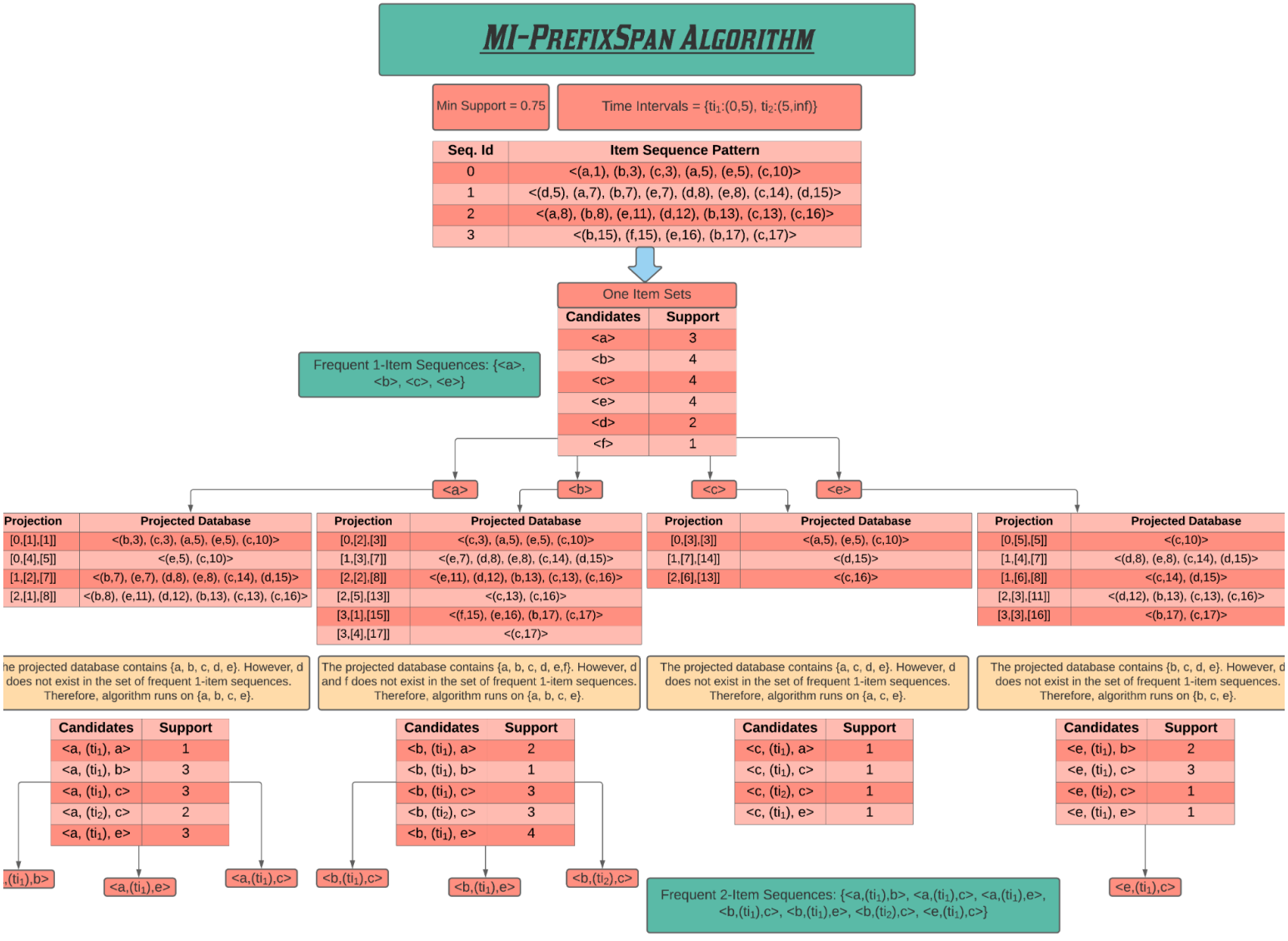

Fig. 6(a). MI-PrefixSpan Algorithm explained with a numerical example.

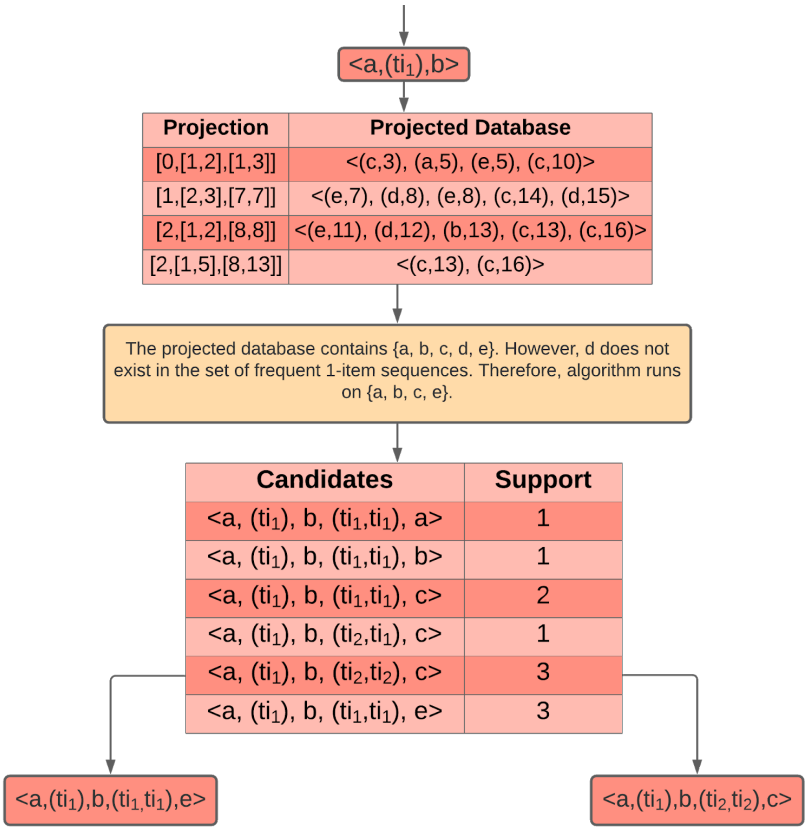

Fig. 6(b). MI-PrefixSpan - contd.

Screenshots

➢ For now, the algorithm is being run on the example dataset provided in the numerical example along with the same time interval and support criteria. The Apriori algorithm is run on this dataset to get all the frequent patterns.

```
TIME_INTERVALS = [(0, 5), (5, float('inf'))]
# TIME_INTERVALS = [(0, 0), (0, 3), (3, 6), (6, float('inf'))]
MIN_SUP = 0.75
DB = [
    [('a', 1), ('b', 3), ('c', 3), ('a', 5), ('e', 5), ('c', 10)],
    [('d', 5), ('a', 7), ('b', 7), ('e', 7), ('d', 8), ('e', 8), ('c', 14), ('d', 15)],
    [('a', 8), ('b', 8), ('e', 11), ('d', 12), ('b', 13), ('c', 13), ('c', 16)],
    [('b', 15), ('f', 15), ('e', 16), ('b', 17), ('c', 17)]
    ]
for row in DB:
  print(row)
```

```
[('a', 1), ('b', 3), ('c', 3), ('a', 5), ('e', 5), ('c', 10)]
[('d', 5), ('a', 7), ('b', 7), ('e', 7), ('d', 8), ('e', 8), ('c', 14), ('d', 15)]
[('a', 8), ('b', 8), ('e', 11), ('d', 12), ('b', 13), ('c', 13), ('c', 16)]
[('b', 15), ('f', 15), ('e', 16), ('b', 17), ('c', 17)]
```

Fig. 7. The Sample Dataset being used to evaluate the Algorithm

➢ The *'multiTimeIntervalSequence'* class has the following attributes:

- *Items:* A list of items in the pattern.
- *Intervals:* The list of intervals between the events
- *Length:* The number of items in the pattern.

```
# A class for multi time interval sequential patterns.
# Each pattern would contain a list of items in the sequence,
# the list of intervals between them and the total number of items.
class multiTimeIntervalSequence:
  """A multi time interval sequence"""
  def __init__(self, items, intervals):
    self.items = items
    self.intervals = intervals
    self.length = len(items)

  def display(self):
    # assert self.items == sorted(self.items), "Assertion Error"
    print("Items: {0}, Intervals: {1}".format(self.items, self.intervals))

  def __hash__(self):
    return hash((tuple(self.items), tuple(tuple(i) for i in self.intervals)))

  def __eq__(self, other):
    """Equality operator overloading for Sequence class"""
    return self.length == other.length and self.items == other.items and self.intervals == other.intervals
```

Fig. 8. The Definition of the Multi Time Interval Sequential Pattern Class

➢ Creating an object of the '*MultiTimeIntervalApriori'* class with the same DB, Time interval and support information. Then, we can call methods to find each frequent pattern, the count of the sequences and the time taken by the algorithm to generate the frequent patterns.

```
[37] apriori_model = MultiTimeIntervalApriori(db=DB, min_support=0.75, timeIntervals=TIME_INTERVALS)

[38] apriori_model.run_multi_time_interval_apriori(verbose=True, measure_time=False)

     1-MTIS Sequences Generated
     Items: ['a'], Intervals: []
     Items: ['b'], Intervals: []
     Items: ['c'], Intervals: []
     Items: ['e'], Intervals: []
     2-MTIS Sequences Generated
     Items: ['a', 'b'], Intervals: [[(0, 5)]]
     Items: ['a', 'c'], Intervals: [[(5, inf)]]
     Items: ['a', 'e'], Intervals: [[(0, 5)]]
     Items: ['b', 'c'], Intervals: [[(0, 5)]]
     Items: ['b', 'c'], Intervals: [[(5, inf)]]
     Items: ['b', 'e'], Intervals: [[(0, 5)]]
     Items: ['e', 'c'], Intervals: [[(0, 5)]]
     3-MTIS Sequences Generated
     Items: ['a', 'b', 'c'], Intervals: [[(0, 5)], [(5, inf), (5, inf)]]
     Items: ['a', 'b', 'e'], Intervals: [[(0, 5)], [(0, 5), (0, 5)]]

[39] apriori_model.display_count_of_sequences()

     1-MTIS Sequences Generated: 4
     2-MTIS Sequences Generated: 7
     3-MTIS Sequences Generated: 2

[40] apriori_model.find_time_taken()

     0.010011434555053711
```

Fig. 9. The code and output generated from running the MI-Apriori Algorithm

Conclusion

The code for the algorithm and its execution on the sample dataset can be found in the following colab-notebook→*https://colab.research.google.com/drive/1OVFSADzx_KYDFl1B-J4y9v6ZfsHzcH8P?usp=sharing*. We have discussed the two well-known algorithms for mining Multi-Time-Interval Sequential patterns from a database of sorted time-stamped item sequences. The Apriori algorithm for this sequential pattern mining task has been implemented and the PrefixSpan algorithm is a work in progress.

References

[1] Ya-Han Hu, Tony Cheng-Kui Huang, Hui-Ru Yang, Yen-Liang Chen, *"On mining multi-time-interval sequential patterns"*, Data & Knowledge Engineering, Volume 68, Issue 10, 2009.
