*IOSR Journal of Computer Engineering (IOSR-JCE)*
*e-ISSN: 2278-0661,p-ISSN: 2278-8727, Volume 17, Issue 3, Ver. II (May – Jun. 2015), PP 110-114*
*www.iosrjournals.org*

# Frequent Pattern Mining with Serialization and De-Serialization

Paresh Tanna[1], Dr. Yogesh Ghodasara[2]
[1](PhD Scholar, School of Computer Science, RK University, India)
[2](Associate Professor, College of Information Tech., Anand Agriculture University, India)

***Abstract :*** *Competent frequent pattern mining techniques be critical for finding relationship rules. Here, examination of subject of finding association rules for objects into an enormous DB of customer purchase entries is discussed. Ruling huge patterns following DB entry set had guided lots of techniques. Like Apriori, DHP, ECLAT, FP Growth etc. At this point, we projected new technique known as Frequent Pattern Mining with Serialization and De-Serialization (FPMSD), that's proficient for finding frequent patterns. FPMSD utilizes DLI(down-level-index) i.e. patterns which co-found with delegate item can be recognized rapidly and straightly using effortless and quickest method. This would happen to advantageous compare to other frequent pattern mining techniques. Additionally serialization will save produced frequent patterns into a file and de-serialization will pull through saved patterns from file. This Serialization and De-serialization Technique(SDT) takes lesser time for patterns entry set gathering than getting this from scratch.*
***Keywords -*** *Association rule, Frequent pattern mining, DLI, SDT*

## I. Introduction

The objective of the techniques illustrated here is to perceive links or relatives among detailed items in huge DB entry sets. This is a universal assignment in lots of data mining problems. These influential probing methods have an extensive variety of functions in real usage areas [7]. These methods enable market analyst and to reveal concealed patterns in huge entry sets, like "consumers who purchases item X recurrently also purchase items Y and Z". This happens through utilization of association rule finding [1]. Association rules are generated with evaluating facts for recurrent patterns and utilizing criterion support and confidence to classify the mainly imperative relations [7].

## II. Projected Method: Fpmsd

Here projected innovative frequent pattern mining technique with indexing and concept of serialization and de-serialization. This technique formulates effectual exercise of indexing with serialization and de-serialization. FPMSD could be utilized for proficient bulky recurrent pattern creation. FPMSD utilizes both vertical as well as horizontal facts layout for creating bulky recurrent pattern from DB entry sets. Numerous disputes originated for Apriori[2], DHP[3], ECLAT[5] as well as FP Growth[4] techniques. There is a petite scope for improvement in FPM to make execution much faster than discussed techniques i.e. use of serialization and de-serialization technique with indexing for FPM. With some analytical survey, people could uncover several enhancements which preserve be recommended for these techniques like condense passes of entry sets in DB, minimize numeral of eligible sets of items, make easy support counting of candidates lacking DB check up[6]. By taking into consideration testing and evaluation work on these features, experimentation by down-level-index(DLI) is followed [9]. Object found support count similar to min_sup, the object itself and probable patterns from this object and their DLI having equal support count as similar to min_sup [9]. This demonstrates that DLI would takes less execution duration for plenty of task of support count and pattern creation. Also serialization will save created frequent patterns into a file and de-serialization will recover saved patterns from file done by preceding step. This Serialization and De-serialization Technique(SDT) takes lesser time for patterns entry set gathering than getting this from scratch.

**FPMSD Algorithm [10]:**
**Steps**:
1. Check if SDT File exists in the given location as per specified criteria else goto step no. 3.
2. De-serialize this SDT file into local object of dataset SDT_Dataset
   SDT_Dataset = Deserialization_Itemsets(SDT_FileName)

   If no. of transactions in SDT_Dataset is greater than DB_Count i.e. in actual count of actual DB
   SDT_Dataset = TransToMatrix_After_Deserialization(SDT_FileName)
   (i.e. adding additional transactions details into SDT_Dataset
   Else

SDT_Dataset = TransToMatrix( ) (i.e. Creating dataset as usual into vertical layout.)

End If

3. Transform listing of entries in DB to Vertical Layout
4. Generate Entry Set from DB.
5. Discover Frequent 1-pattern from the given Vertical Entry Set, is merely the count of entry for transactions in DB only.
6. Arrange given patterns by chronological sequence in Entry Set by its min_sup.
7. Collect unique listing of items as KeyItemGroup from Entry Set i.e. Listing of patterns only in chronological sequence with its min_sup
8. Create BooleanTable for every entry available in KeyItemGroup set
9. Create DLI for each entry in KeyItemsGroup
10. For every Entry in KeyItemsGroup

```
If Entry.DLI < > "Nill"
    If Entry.Count == minimum_sup Then
        SearchPatternsSimmilarToMinimumSup(Entry, Entry.Count)
    Else
        SearchPatternsLargerThanMinimumSup(Entry, Entry.Count)
    End If
Else
    If Entry.Count > minimum_sup
        AND  Entry_Order < Entry.Length Then
      SearchPatternsDLINone(Entry)
    End If
End If
```

**Example 1: Taking an illustration of a 10 entries in DB and minimum_sup is 2.**

**Table 1 : DB Entries**

| TID | Items |
|---|---|
| ED1 | W1, W2,W3,W5,W6,W15 |
| ED2 | W1,W3,W7 |
| ED3 | W5,W9 |
| ED4 | W1,W3,W4,W5,W7 |
| ED5 | W1,W3,W5,W7,W12 |
| ED6 | W5,W10 |
| ED7 | W1,W2,W3,W5,W6,W16 |
| ED8 | W1,W3,W4 |
| ED9 | W1,W3,W5,W7,W13 |
| ED10 | W1,W3,W5,W7,W14 |

**Table 2 : 1- Frequent Patterns**

| Itemset | Sup Count |
|---|---|
| W2 | 2 |
| W4 | 2 |
| W6 | 2 |
| W7 | 5 |
| W1 | 8 |
| W3 | 8 |
| W5 | 8 |

**Table 3 : Chronological Ordered Entries**

| TID | Items | Ordered Items |
|---|---|---|
| ED1 | W1, W2,W3,W5,W6,W15 | W2,W6,W1,W3,W5 |
| ED2 | W1,W3,W7 | W7,W1,W3 |
| ED3 | W5,W9 | W5 |
| ED4 | W1,W3,W4,W5,W7 | W4,W7,W1,W3,W5 |
| ED5 | W1,W3,W5,W7,W12 | W7,W1,W3,W5 |
| ED6 | W5,W10 | W5 |
| ED7 | W1,W2,W3,W5,W6,W16 | W2,W6,W1,W3,W5 |
| ED8 | W1,W3,W4 | W4,W1,W3 |
| ED9 | W1,W3,W5,W7,W13 | W7,W1,W3,W5 |
| ED10 | W1,W3,W5,W7,W14 | W7,W1,W3,W5 |

**Table 4 : BooleanTable Characterization of DB**

| TID | W2 | W4 | W6 | W7 | W1 | W3 | W5 |
|---|---|---|---|---|---|---|---|
| ED1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 |
| ED2 | 0 | 0 | 0 | 1 | 1 | 1 | 0 |
| ED3 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| ED4 | 0 | 1 | 0 | 1 | 1 | 1 | 1 |
| ED5 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| ED6 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| ED7 | 1 | 0 | 1 | 0 | 1 | 1 | 1 |
| ED8 | 0 | 1 | 0 | 0 | 1 | 1 | 0 |
| ED9 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| ED10 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |

= ED1 ∩ ED7
= 1010111 ∩ 1010111
= 1010111

So, ED2's DLI is W6 W1 W3 W5 i.e. (W2, W6 W1 W3 W5)

Progressing with this procedure consequently for given items and lastly DLI array will be (W2, W6 W1 W3 W5), (W4, W1 W3), (W6, W1 W3 W5), (W7, W1 W3), (W1, W3), (W3, Ø), (W5, Ø).

| Table 5 : Frequent Pattern Ceneration | | | | | | |
|---|---|---|---|---|---|---|
| ITEMS | | | | | | |
| **W2** | **W4** | **W6** | **W7** | **W1** | **W3** | **W5** |
| W2 : 2<br>W2,W6 : 2<br>W2,W1 : 2<br>W2,W3 : 2<br>W2,W5 : 2<br>W2,W6,W1 : 2<br>W2,W6,W3 : 2<br>W2,W6,W5 : 2<br>W2,W1,W3 : 2<br>W2,W1,W5 : 2<br>W2,W3,W5 : 2<br>W2,W6,W1,W3 : 2<br>W2,W6,W1,W5 : 2<br>W2,W6,W3,W5 : 2<br>W2,W1,W3,W5 : 2<br>W2,W6,W1,W3,W5 : 2 | W4 : 2<br>W4,W1 : 2<br>W4,W3 : 2<br>W4,W1,W3 : 2 | W6 : 2<br>W6,W1 : 2<br>W6,W3 : 2<br>W6,W5 : 2<br>W6,W1,W3 : 2<br>W6,W1,W5 : 2<br>W6,W3,W5 : 2<br>W6,W1,W3,W5 : 2 | W7 : 5<br>W7,W1 : 5<br>W7,W3 : 5<br>W7,W1,W3 : 5<br>W7,W5 : 4<br>W7,W1,W5 : 4<br>W7,W3,W5 : 4<br>W7,W1,W3,W5 : 4 | W1 : 8<br>W1,W3 : 8<br>W1,W5 : 6<br>W1,W3,W5 : 6 | W3 : 8<br>W3,W5 : 6 | W5 : 8 |
| TOTAL = 43 | | | | | | |
| 16 | 4 | 8 | 8 | 4 | 2 | 1 |

In given illustration, minimum_sup is 2, and items W2, W4 and W6 encompass sup_count 2. Therefore sup_count and frequent patterns creation for those given items is extremely simple. Like, to discover frequent patterns, simply require to produce potential groupings for item and DLI of item, sup_count would get similar just like minimum_sup. Following the method we can discover 16 + 4 + 8 = 28 entirety from 43 for W2, W4 and W6 straightly. Next for W7, that has sup_count larger than minimum_sup, frequent patterns would get probable groupings of item W7 and its DLI (W1 W3) and sup_count would be similar like W7. Item W5 that doesn't exist in DLI of W7, compute sup_count for these that is, for W7, W5. Except for remaining similar to (W7,W1,W5), (W7,W3,W5), (W7,W1,W3,W5) gets similar sup_count as (W7,W5). Discussed procedure is iteratively followed for W1, W3 as well as W5. As a consequence, this method utilize lesser duration for frequent patterns creation contrast to discussed techniques. Addition to this DLI, performance is also improved with Serialization and De-serialization techniques.

**Using Serialization and De-Serialization**

Serialization will save created frequent patterns into a file and de-serialization will recover saved patterns from file done by preceding step. This Serialization and De-serialization Technique(SDT) takes lesser time for patterns entry set gathering than getting this from scratch. Little java code is shared here to get insight into the concept followed by performance evaluation.

```
Serialize_Itemsets()
{
        FileOutputStream fos =  new FileOutputStream(SDTFleName);
        ObjectOutputStream oos = new ObjectOutputStream(fos);
        oos.writeObject(mapItemTIDS);
        oos.writeObject(ll);
        oos.writeObject(trans_count);
        oos.close();
        fos.close();
}
DeSerialize_Itemsets(String SDTFileName)
{
        Map<String, BitSet> e = null;
        FileInputStream fileIn = new FileInputStream(sdfilename);
        ObjectInputStream in = new ObjectInputStream(fileIn);
        e = (Map<String, BitSet>) in.readObject();
        ll = (LinkedList<Integer>)in.readObject();
        tCount = (Integer)in.readObject();
        in.close();
        fileIn.close();
}
```

Here, SDTFileName is used as a transitional tool to carry out this task.

## III. Investigational Output

We contrast the gig of the projected technique by different techniques like Apriori, DHP, ECLAT and FP Growth. This technique is developed in Java. Here preferring entry set from [8] for generating contradiction report of projected technique. Each entry set is referred from FIMI repository page http://fimi.cs.helsinki.fi. Table 6 illustrates distinctiveness of discussed entry sets. Performance time (in secs) requisites by Apriori, DHP, ECLAT, FP Growth and projected technique i.e. FPMSD are revealed in beneath fig 1st , 2nd , 3rd and 4th.

**Table 6. Dataset Layout**

[1st – Apriori, 2nd – DHP, 3rd – ECLAT, 4th – FP Growth 5th – FPMSD]

| Entry Set Layout | No. of Trnas | Listing of Techniques | Notes |
|---|---|---|---|
| T10I4D100 | 100 | 1st ,2nd ,3rd ,4th ,5th | Top hundred entries from T10I4D100K |
| T10I4D1000 | 1000 | 1st ,2nd ,3rd ,4th ,5th | Top thousand entries from T10I4D100K |
| T10I4D50000 | 50000 | 3rd ,5th | Top fifty thousand entries from T10I4D100K |
| T10I4D100K | 100000 | 3rd ,5th | - |

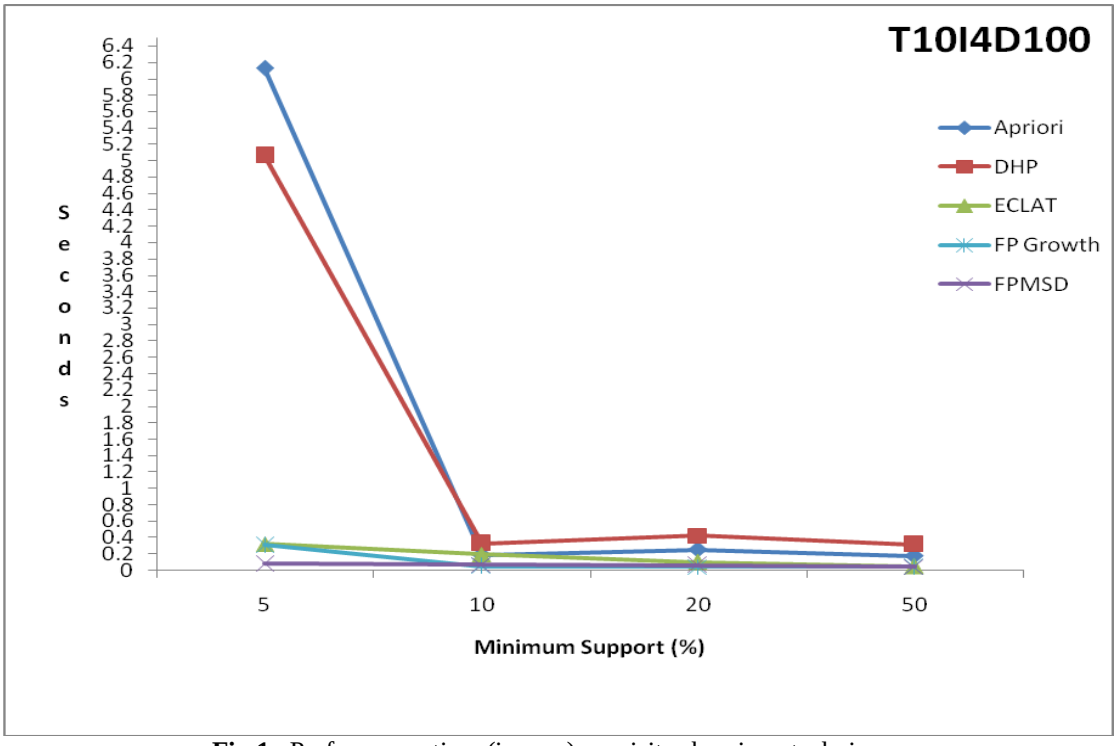


**Fig 1.** Performance time (in secs) requisites by given techniques

The entry set in Fig. 1 demonstrates that projected FPMSD executes fastest on lesser to larger minimum_sup with tiny amount of data.

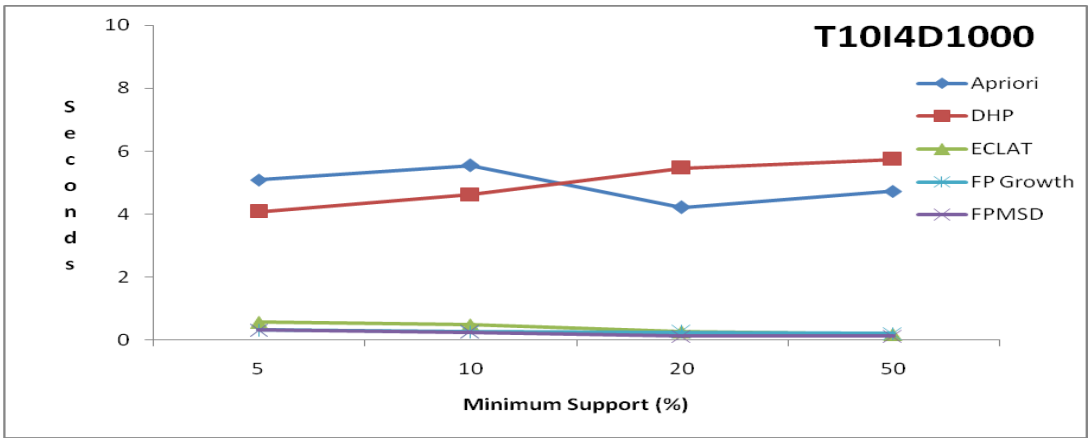


**Fig 2.** Performance time (in secs) requisites by given techniques

The entry set in Fig. 2 demonstrates that projected FPMSD executes fastest on lesser to larger minimum_sup with some higher to tiny amount of data.

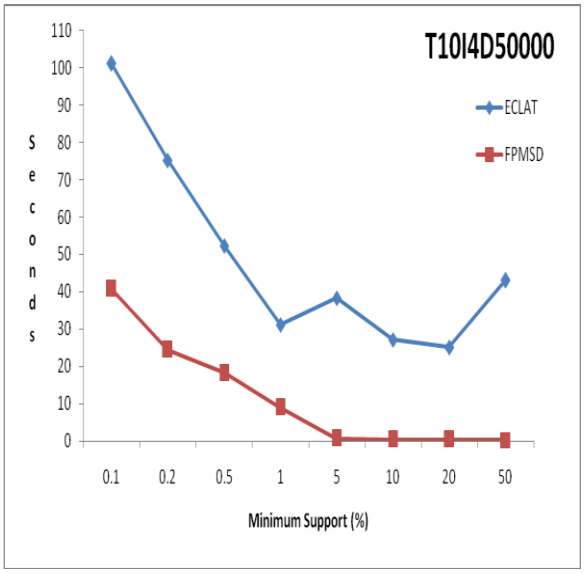


**Figure 3.**

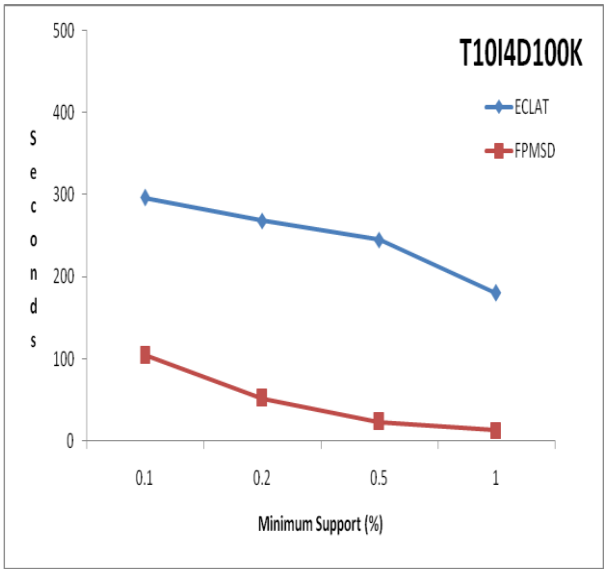


**Figure 4.**

**Fig 3 and 4** shows performance time (in secs) requisites by given techniques

The entry set in Fig. 3 and 4 demonstrates that projected FPMSD executes fastest on lesser to larger minimum_sup with higher amount of data.

## IV. Conclusion

Projected FPMSD executes fastest on lesser to larger minimum_sup with tiny to higher amount of data. Ccontrasting with Apriori, DHP, ECLAT and FP Growth, projected technique diminishes duration for greater frequent patterns creation and candidate frequent patterns that sup_count doesn't required calculation.

## References

[1] Data Mining: Concepts and Techniques, Jiawei Han and Micheline Kamber, MORGAN KAUFMANN PUBLISHER, An Imprint of Elsevier

[2] R. Agrawal and S. Srikant, "Fast Algorithms for Mining Association Rules in Large Databases", Proceedings of the 20th International Conference on Very Large Data Bases, September 1994.

[3] J. Park, M. Chen and Philip Yu, "An Effective Hash-Based Algorithm for Mining Association Rules", Proceedings of ACM Special Interest Group of Management of Data, ACM SIGMOD'95, 1995.

[4] Han, Pei & Yin, "Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach", Data Mining and Knowledge Discovery, Volume 8, Issue 1 , pp 53-87,2004

[5] M. Zaki, S. Parthasarathy, M. Ogihara, and W. Li, " New Algorithms for Fast Discovery of Association Rules", Proc. 3rd ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD'97, Newport Beach, CA), 283-296 AAAI Press, Menlo Park, CA, USA 1997

[6] Shruti Aggarwal, Ranveer Kaur, "Comparative Study of Various Improved Versions of Apriori Algorithm", International Journal of Engineering Trends and Technology (IJETT) - Volume4Issue4- April 2013

[7] Agrawal, R., T. Imielin´ ski, and A. Swami (1993). Mining association rules between sets of items in large databases. In Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, SIGMOD '93, New York, NY, USA, pp. 207–216. ACM.

[8] Synthetic Data for Associations and Sequential Patterns. http://fimi.cs.helsinki.fi

[9] W.Song, B.R.Yang, Z.Y.Xu, "Index-BitTableFI: An improved algorithm for mining frequent itemsets," Knowledge Based Systems 21 (4) , 507–513(2008).

[10] Paresh Tanna, Dr. Yogesh Ghodasara, "Improved Frequent Pattern Mining Algorithm with Indexing,", IOSR Journal of Computer Engineering (IOSR-JCE), e-ISSN: 2278-0661,p-ISSN: 2278-8727, Volume 16, Issue 6, Ver. VII (Nov – Dec. 2014), PP 73-78.