Incremental Maintenance of Discovered Mobile User Maximal Moving Sequential Patterns*

Shuai Ma, Shiwei Tang, Dongqing Yang, Tengjiao Wang, and Chanjun Yang

Department of Computer Science, Peking University, Beijing 100871, China
{mashuai,tjwang,cjyang}@db.pku.edu.cn
{tsw,dqyang}@pku.edu.cn

Abstract. In the context of mobile computing, a special sequential pattern, moving sequential pattern that reflects the moving behavior of mobile users attracted researchers' interests recently. While there have been a number of efficient moving sequential pattern mining algorithms reported, this paper concentrates on the maintenance of mined maximal moving sequential patterns. In particular, we developed an incremental approach, where maximal moving sequential patterns are stored in prefix trees, and new moving sequences can be easily combined with the existing patterns. A performance study indicated that the proposed approach performs significantly faster than straightforward approaches that mine from the whole updated database.

Keywords. Moving sequential pattern, Incremental maintenance, Data mining

1 Introduction

Mining moving sequential patterns has great significance for effective and efficient location management in wireless communication systems. We systematically describe the problem of mining moving sequential patterns as a special case of mining sequential patterns with the extension of support [1]. There are mainly four differences between mining conventional sequential patterns and moving sequential patterns. Firstly, if two items are consecutive in a moving sequence α, and α is a subsequence of β, those two items must be consecutive in β. This is because we care about what the next move is for a mobile user in mining moving sequential patterns. Secondly, in mining moving sequential patterns the support considers the number of occurrences in a moving sequence, so the support of a moving sequence is the sum of the number of occurrence in all the moving sequences of the whole moving sequence database. Thirdly, the Apriori property plays an important role for efficient candidate pruning in mining sequential patterns. For example, suppose <ABC> is a frequent length-3

* Supported by the National High Technology Development 863 Program of China under Grant No. 2002AA4Z3440; the National Grand Fundamental Research 973 Program of China under Grant No. G1999032705; the Foundation of the Innovation Research Institute of PKU-IBM.

sequence, and then all the length-2 subsequences {<AB>, <AC>, <BC>} must be frequent in mining sequential patterns. In mining moving sequential patterns <AC> may not be frequent. This is because a mobile user can only move into a neighboring cell in a wireless system and items must be consecutive in mining moving sequential patterns. In addition, <AC> is not a subsequence of <ABC> any more in mining moving sequential patterns and that any subsequence of a frequent moving sequence must be frequent is still fulfilled from that meaning, which is called Pseudo-Apriori property. The last difference is that a moving sequence is an order list of items, but not an order list of itemsets, where each item is a cell id.

Wen-Chih Peng et al. presented a data-mining algorithm, which involves mining for user moving patterns in a mobile computing environment in [2]. Moving pattern mining is based on a roundtrip model [3], and their LM algorithm selects an initial location S, which is either VLR or HLR whose geography area contains the homes of the mobiles users. Suppose a mobile user goes to a strange place for one month or longer, the method in [2] cannot find the proper moving pattern to characterize the mobile user. A more general method should not give any assumption of the start point of a moving pattern. Basically, algorithm LM is a variant one from GSP [4]. The Apriori-based methods can efficiently prune candidate sequence patterns based on Aprior property, but in moving sequential pattern mining we cannot prune candidate sequences efficiently because the moving sequential pattern only preserves Pseudo-Apriori property. In the meanwhile Apriori-based algorithms still encounter problem when a sequence database is large and/or when sequential patterns to be mined are numerous and/or long [5]. Based on the idea of projection and Pseudo-Apriori property, we propose a novel and efficient moving sequential pattern mining algorithm PrefixTree based on a key tree structure prefix trees, which can effectively represent candidate frequent moving sequences in [1]. The performance study shows that PrefixTree outperforms LM of our version.

It is a nontrivial work to maintain the discovered mobile user maximal moving sequential patterns because the moving sequence of a mobile user will frequently update every day and such updates may not only invalidate some existing frequent moving sequence but also turn some infrequent moving sequences into frequent ones. Wen-Chih Peng et al. also slightly revised LM in [6], but we should point out that the LM algorithm revised is not a real algorithm that incrementally mines the moving sequences. Firstly it relies on another algorithm MM for the calculation of Length-2 moving sequence; secondly, algorithm LM is executed to obtain new moving sequential patterns for every ω moving sequences as a solution for incremental update problem, thus LM is not designed for finding the moving sequential patterns of the updated database.

In this paper we propose an efficient algorithm called PrefixTree+, which takes the advantage of PrefixTree algorithm and avoids rerunning the mining algorithm from scratch each time.

The rest of the paper is organized as follows. In section 2, we briefly describe the algorithm of PrefixTree. Section 3 describes our incremental mining algorithm PrefixTree+. The experimental results are given in section 4. Discussion and future work are made in section 5.

2 PrefixTree Algorithm

In this section, we will briefly describe PrefixTree algorithm, which forms the basis for our incremental algorithm.

For each item C_i, we call the items that may appear just after it candidate consecutive items. It is easy to know that only the items after C_i in a moving sequence may be the consecutive items of C_i, denoted by CCI(C_i). And for any item $C_j \in$ CCI(C_i), length-2 moving sequence $<C_iC_j>$ is frequent. Mobile user can only move into a neighboring cell in a wireless system, so the unbound of the number of candidate consecutive items for an item is the number of neighbor cells. Though there maybe be a huge number of cells, the number of neighboring cells is often small. For example it is two in one-dimension model, and six in two-dimension hexagonal model, and eight in two-dimension mesh model, and is relative small even in graph model [7]. And the other concepts of Prefix, Projection and Postfix are similar to the ones in PrefixSpan [5], and are changed according to the characteristics of moving sequences.

PrefixTree algorithm only need scan the database three times, and the key idea of PrefixTree is the use of prefix trees. Prefix tree is a compact representation of candidate moving sequential patterns. The root is the frequent item, and is defined at depth one. Each node contains three attributes: one is the item, one is the count attribute which means the support of the item, and the last one is the flag indicating whether the node is traversed. The items of a node's children are all contained in its candidate consecutive items. In the first two scans PrefixTree generates the frequent itmes, frequent length-2 moving sequential patterns and CCIs of each frequent item, and the prefix trees are constructed in the third scan. It is easy for us to generate the moving sequential patterns based on the prefix trees. Every moving sequence from the root node to the leaf node is a candidate frequent moving sequences. We can get all the moving sequential patterns by scanning all the prefix trees once. The support of each node decreases with the depth increase, so a new frequent moving sequence is generated when we traverse the prefix trees from the root to the leaves when encountering a node whose count is less than the support threshold.

3 PrefixTree+ Algorithm

In this section, we firstly describe the basic theory of incremental mining moving sequences, and then present PrefixTree+ algorithm for computing the maximal moving sequential patterns in the updated database.

A. Sarasere et al. presented an efficient algorithm called Partition for mining association rules in [8]. The key to correctness of the Partition algorithm is that any potential large itemset appears as a large itemset in at least one of the partitions, and similarly we have the following lemma for mining moving sequential patterns, which is also similar to the Partition algorithm, and it is easy to know the following lemma is true. Interesting readers could see [8] for more information.

Lemma 1: Any moving sequence that is potential frequent with respect to the updated database D′ must occur as a frequent moving sequence in at least one of the partitions: the original database D and the incremental part Δ^+.

Based on the above lemma, we modify the PrefixTree algorithm to PrefixTree+ for computing the maximal moving sequential patterns in the updated database, which is shown as below,

```
Algorithm PrefixTree+
Input: D, $\Delta^+$, Materialized prefix trees of D, Minimum support threshold min_sup
Output: The complete set of maximal moving sequential patterns of D′
1: Reconstructing the prefix trees of D by scanning the materialized sequential patterns of the prefix trees once.
2: Adding the count of prefix trees of D by scanning $\Delta^+$ once.
3: Constructing the prefix trees of $\Delta^+$ by scanning $\Delta^+$ three times.
4: Adding the count of prefix trees of $\Delta^+$ by scanning D once.
5: Generating moving sequential patterns based on the prefix trees of D and $\Delta^+$.
6: Generating maximal moving sequential patterns.
7: Generating the materialized prefix trees of D′ based on the prefix trees of D and $\Delta^+$.
```

We keep the prefix trees of the original database by materializing them, and we can construct the prefix trees again based on the materialized moving sequential patterns. Thus the prefix trees of the original moving sequence database are not need to mine any more, in the meanwhile we can get the their supports of the updated database by scanning the incremental part once. After constructing the prefix trees of the incremental part, we can get their supports of the updated database by scanning the original moving sequence database once. Based on the prefix trees of the original database and the incremental part separately, we could get the maximal moving sequential patterns. The materialized sequential patterns of the prefix trees for the updated database, which will be used for the next time mining, is the union of the materialized prefix trees of D and Δ^+, so we can easily get the materialized prefix trees of D′ even if we do not know the prefix trees of D′ at all.

4 Experimental Results and Performance Study

All experiments are performed on a 1.7GHz Pentium 4 PC machine with 512M main memory and 60G hard disk, running Microsoft Windows 2000 Professional. All the methods are implemented using JBuilder 6.0.

The synthetic datasets used for our experiments come from SUMATRA (Stanford University Mobile Activity TRAces) [9]. BALI-2: Bay Area Location Information (real-time) dataset records the mobile users' moving and calling activities in a day. The mobile user averagely moves 7.2 times in a day in 90 zones, so the average length of moving sequence is 8.2. We extract about 42,000 moving sequences from BALI-2 used for our experiments.

Let reading a moving sequence in a data file costs 1 unit of I/O. Let ρ_s be the support threshold (percentage), L be the length of the longest moving sequential pattern, and N be the number of the materialized moving sequential patterns of the prefix trees. The I/O cost of LM is equal to $L(|\Delta^+|+|D|)$; the I/O cost of PrefixTree is equal to $3(|\Delta^+|+|D|)$; and the I/O cost of PrefixTree+ is approximately equal to $4|\Delta^+|+|D|+2N$. N is usually is a small number, and N is about 500~2,500 in all our experiments. If L is bigger than 3, the I/O cost of LM is bigger than the one of PrefixTree; otherwise, a reverse conclusion. If the result of subtracting $(4|\Delta^+|+|D|+2N)$ from $3(|\Delta^+|+|D|)$ is bigger than zero, i.e. $(2|D|-|\Delta^+|-2N)>0$, the I/O cost of PrefixTree is bigger than the one of PrefixTree+; otherwise, a reverse conclusion. From the I/O costs analysis we could get a coarse conclusion that PrefixTree+ is more efficient than PrefixTree only if $|\Delta^+|$ is about less two times of $|D|$, which give a condition when to use PrefixTree+ algorithm. In fact, in most cases $|\Delta^+|$ is much smaller than $|D|$, so PrefixTree+ is good approach from this view. And even when $|\Delta^+|$ is around two times of $|D|$, PrefixTree+ is still nearly as efficient as PrefixTree in our experiments.

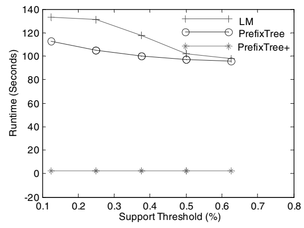

Fig. 1. CPU Costs

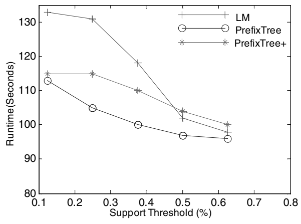

Fig. 2. CPU Costs

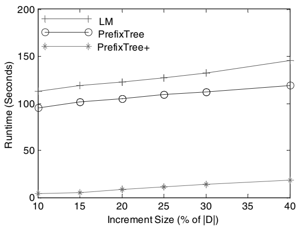

Fig. 3. CPU Costs

Fig. 1 shows the run time of LM, PrefixTree, and PrefixTree+ according to the support threshold, where $|D|$ is 40,000 and $|\Delta^+|$ is 2,000. When the support is high, there is only a limited number of moving sequential patterns, and the length of patterns is short, PrefixTree and LM are close to each other according to their runtime. However as the support threshold decreases, PrefixTree is more efficient than LM. Compared with LM and PrefixTree, PrefixTree+ results in the run time improvement of up to about two orders of magnitude.

Fig. 2 shows the run time of LM, PrefixTree, and PrefixTree+ according to the support threshold, where $|D|$ is 14,000 and $|\Delta^+|$ is 28,000. Now the ratio of $|\Delta^+|/|D|$ is 2. The efficiency of PrefixTree+ is between LM and PrefixTree at first, and is a little worse than the ones of LM and PrefixTree as the support threshold increases. This experiment proves our analysis that PrefixTree+ is still nearly as efficient as PrefixTree in our experiments when $|\Delta^+|$ is around two times of $|D|$, and also give the condition that when PrefixTree+ should be used for the problem of maintaining maximal moving sequential patterns.

The above experiments also show that the CPU cost of PrefixTree+ is insensitive to the change of support threshold, and the reason is that the change of support threshold affects little to the I/O cost for algorithm PrefixTree+.

We use a database D of 30,000 moving sequences and support threshold ρ_s of 0.125%. We vary the size of Δ^+ to show the algorithms' scalability. Fig. 3 shows the run time of LM, PrefixTree, and PrefixTree+ according to the size of Δ^+. When the size of Δ^+ increases, the run time of all the three algorithms increases. PrefixTree is more efficient than LM, and PrefixTree+ results in the run time improvement of up to about two orders of magnitude compared with LM and PrefixTree.

In summary, our performance study shows that PrefixTree+ is more efficient and scalable than LM and PrefixTree in most and reasonable cases. Compared with LM and PrefixTree, PrefixTree+ results in the run time improvement much. In addition, we give the condition that when PrefixTree+ should be used for the problem of maintaining maximal moving sequential patterns.

5 Discussion and Future Work

This paper studies an efficient, fast, and incremental updating technique for maintenance of maximal moving sequential patterns. We propose an incremental mining algorithm PrefixTree+, which strives to use the mining results of last time and improve the mining efficiency. Its novelty is materializing prefix trees, and using the lemma that any moving sequence that is potential frequent with respect to database D must occur as a frequent moving sequence in at least one of the partitions. Our performance study shows that PrefixTree+ is more efficient and scalable than LM and PrefixTree.

Another interesting and important study is when to update the mining results again, such as the problem in association rules [10]. Our next step will focus on the problem of when to update the discovered maximal moving sequential patterns.

Acknowledgement. We are very thankful to Professor Lu Hong Jun for giving us many good suggestions and advices, which help improve the quality of this paper.

References

1. Shuai Ma, Tengjiao Wang, Shiwei Tang, Dongqing Yang, and Jun Gao. Mining Mobile User Maximal Moving Sequential Patterns. Technical report PKU_CS_DB_TR20030601, Department of Computer Science, Peking University, Beijing, China, 2003.
2. Wen-Chih Peng, Ming-Syan Chen. Mining User Moving Patterns for Personal Data Allocation in a Mobile Computing System. Proc. of the ICPP Conference, pp. 573-580, 2000.
3. N. Shivakumar, J. Jannink, and J. Widom. Per-user Profile Replication in Mobile Environments: Algorithms Analysis and Simulation Result. ACM/Baltzer Journal of Mobile Networks and Applications, v.2 n.2, p.129-140, 1997.
4. Ramakrishnan Srikant, Rakesh Agrawal. Mining Sequential Patterns: Generalizations and Performance Improvements. Proc. of the 5th EDBT Conference, pp. 3-17, 1996.

5. J. Pei, J. Han, B. Mortazavi-Asl et al. PrefixSpan: Mining Sequential Patterns Efficiently by PrefixProjected Pattern Growth. Proc. of the 17th ICDE Conference, pp. 215-224, 2001.
6. Wen-Chih Peng, Ming-Syan Chen. Developing Data Allocation Schemes by Incremental Mining of User Moving Patterns in a Mobile Computing System. IEEE Transactions on Knowledge and Data Engineering, 15(1): 70-85 (2003).
7. Vincent W. S. Wong and Victor C. M. Leung. Location Management for Next Generation Personal Communication Networks. IEEE network, special issue on next generation wireless broadband networks, vol. 14, no. 5, pp. 8-14 2000.
8. A. Sarasere, E. Omiecinski, and S. Navathe. An Efficient Algorithm for Mining Association Rules in Larges Databases. Proc. of the 21st VLDB Conference, pp. 432-444, 1995.
9. SUMATRA: Stanford University Mobile Activity TRAces. http://www.db.stanford.edu /sumatra/.
10. S. Lee and D. Cheung. Maintenance of Discovered Association Rules: When to Update? Proc. of SIGMOD DMKD Workshop, 1997.