Efficient Frequent Pattern Mining over Data Streams

Syed Khairuzzaman Tanbeer, Chowdhury Farhan Ahmed, Byeong-Soo Jeong, Young-Koo Lee
Department of Computer Engineering, Kyung Hee University
1 Seochun-dong, Kihung-gu, Youngin-si, Kyunggi-do, 446-701 Republic of Korea
{tanbeer, farhan, jeong, yklee}@khu.ac.kr

ABSTRACT

This paper proposes a prefix-tree structure, called CPS-tree (Compact Pattern Stream tree) that efficiently discovers the exact set of recent frequent patterns from high-speed data stream. The CPS-tree introduces the concept of dynamic tree restructuring technique in handling stream data that allows it to achieve highly compact frequency-descending tree structure at runtime and facilitates an efficient FP-growth-based [1] mining technique.

Categories and Subject Descriptors: H.2.8

[**Database Management**]: Database Applications – *data mining*.

General Terms: Algorithms, Performance.

Keywords: Data mining, frequent patterns, data stream.

1. INTRODUCTION

Recently, finding frequent patterns from data streams has become one of the important and challenging problems, since capturing the stream content memory efficiently with a single-pass and efficient mining have been major issues. The FP-growth mining technique [1] is one of the efficient algorithms where the achieved performance gain is mainly based on the highly compact frequency-descending FP-tree structure that ensures the tree to maintain as much prefix sharing as possible. However, the two database scans and prior threshold knowledge requirements of the FP-tree restrict its use in data stream. DSTree [3] uses the FP-growth mining technique to mine exact set of recent frequent patterns from stream data with a single-pass. However, it provides poor compactness in tree structure and inefficient mining phase, since it uses frequency-independent canonical order tree structure. Therefore, in this paper, we propose a novel tree structure, called CPS-tree (Compact Pattern Stream tree), that constructs an FP-tree like compact prefix-tree structure with a single-pass over stream data and provide the same mining performance as the FP-growth technique through the efficient tree restructuring process.

2. CONSTRUCTION and MINING of the CPS-TREE

To capture the recent stream contents CPS-tree uses the sliding window mechanism. To facilitate the window sliding and tree updating, each window *W* is decomposed into a number of equal-sized non-overlapping batches of transaction, called pane *P*. Let the window slides pane-by-pane.

The CPS-tree follows the FP-tree construction mechanism to insert transactions into tree. At first, the transactions are inserted

CIKM'08, October 26–30, 2008, Napa Valley, California, USA.
ACM 978-1-59593-991-3/08/10.

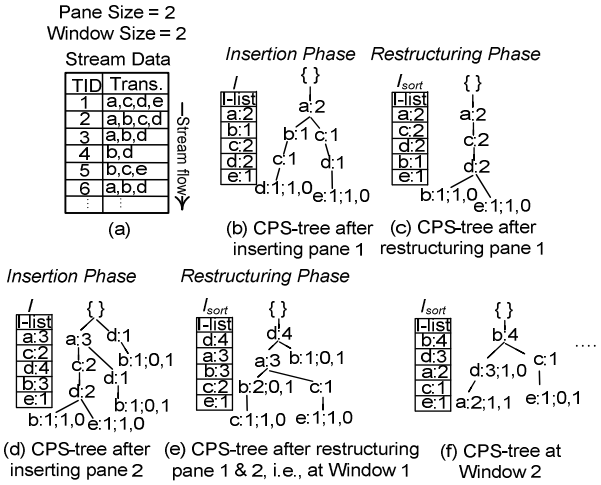

Figure 1. The CPS-tree construction

(*Insertion phase*) according to a predefined item order (e.g., lexicographic item order). The item order of CPS-tree is maintained by a list, called *I*, with respective frequency count of each item. After inserting a part of transactions, if the item order of *I* deviates from the current frequency-descending item order to a degree, CPS-tree is dynamically restructured (*Restructuring phase*) by the current frequency-descending item order I_{sort}. During the next *Insertion phase* transactions are inserted in I_{sort} order. The pane-wise information is separately maintained into the tree in a list, called *pane-counter* in the last node, called *tail-node* of each transaction. The step-by-step CPS-tree construction and restructuring phases are shown in Figures 1(b) to 1(c) for the database of Figure 1(a) with $W = 2$ panes and $P = 2$ transactions. Consider the tree restructuring is performed after each pane.

We refresh the CPS-tree at each window slide in order to provide a ready-to-mine platform with the exact content of the current window. Upon sliding of window the first value in the *pane-counter* in each *tail-node* and same value from the count value of each node up to the *root* in the path are removed, and the other remaining values in the list are shifted left by one slot to reflect the expiration of oldest pane. The CPS-tree after deleting the expired pane and inserting a new pane (i.e., at window 2) is shown in Figure 1(f). The DSTree for windows 1 and 2 requires

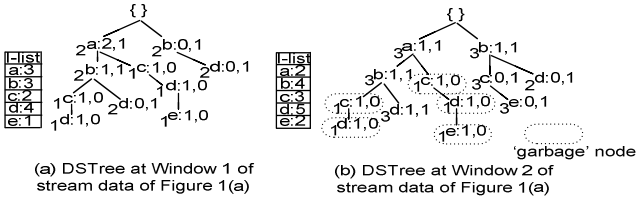

Figure 2. DSTrees on the stream data of Figure 1(a)

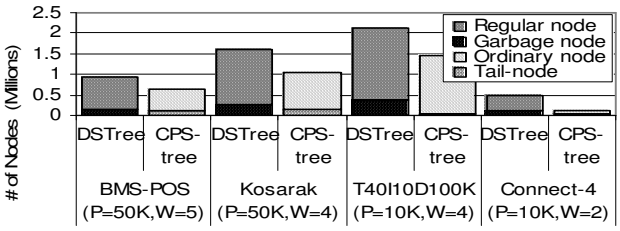

Figure 3. Node count of DSTree and CPS-tree

comparatively higher nodes (as in Figures 2). Moreover, the tree at window 2 is overburdened with some '*garbage*' nodes that are generated since DSTree does not update the tree at each window.

In order to restructure the CPS-tree, we use our proposed efficient tree restructuring mechanism, called Branch sorting method (BSM) [4], and the Path adjusting method proposed in [2]. We refer interested readers to [4] and [2] for getting detail about the BSM and Path adjusting method respectively. Once the CPS-tree is constructed on current window, we use the bottom-up FP-growth mining technique to generate the exact set of recent frequent patterns. The mining operation is highly efficient due to the frequency-descending tree structure.

3. EXPERIMENTAL ANALYSIS

We compare different performance issues of our CPS-tree with those of the DSTree, since it has been reported in [3] that DSTree outperforms other related algorithms to find recent frequent patterns from data stream. All programs are written in Microsoft Visual C++ 6.0 and run with Windows XP on a 2.66 GHz CPU and 1 GB memory. Runtime includes tree construction, tree restructuring (for CPS-tree only) and mining time. Several real and synthetic datasets are used. In the experiments, the size of window is indicated by the two parameters *W* and *P*.

The results on memory consumption on different datasets are shown in the form of the number of nodes in Figure 3. It is clear from the figure that the total number of nodes the CPS-tree requires is significantly less compared to that the DSTree does in each dataset. The reason is that CPS-tree's dynamic tree *Restructuring phase* enables it to obtain as much prefix sharing as possible that remarkably reduces the number of nodes compared to any frequency-independent tree. In addition, the CPS-tree is free from the 'curse' of '*garbage*' nodes. Moreover, it further reduces the size by maintaining only a few *tail-node*s compared to ordinary nodes in the tree structure.

The runtime comparison between the CPS-tree and DSTree over datasets of different types has been performed by varying the threshold (*min_sup*, ∂) values and widow parameters. In Table 1 we report the runtime of both trees for two (one high and one low) *min_sup* values over different datasets. The runtime distribution for tree construction, tree restructuring (only for CPS-tree), tree update (expired pane deletion time for CPS-tree and '*garbage*' node deletion time for DSTree), mining time (for two *min_sup* values) and total time are shown explicitly. The data in the table clearly demonstrate that CPS-tree outperforms DSTree in overall execution time by multiple orders of magnitudes on both high and low *min_sup* values over all types of datasets used in the experiment, which is due to the remarkable improvement CPS-tree achieves in mining time on the dynamically-obtained frequency-descending tree structure.

Therefore, from the above experiments we summarize that despite additional tree restructuring cost, our CPS-tree consistently outperforms state-of-the-art algorithms on both runtime and memory consumption in mining exact set of recent frequent patterns from data stream.

4. CONCLUSIONS

We propose the prefix-tree structure CPS-tree that introduces dynamic tree restructuring mechanism in data stream and efficiently finds recent frequent patterns from high-speed data stream with a single-pass.

5. ACKNOWLEDGEMENT

This study was supported by a grant of the Korea Health 21 R&D Project, Ministry For Health, Welfare and Family Affairs, Republic of Korea (A020602).

6. REFERENCES

[1] Han, J., Pei, J., and Yin Y. 2000. Mining frequent patterns without candidate generation. In Proc. of Int. Conf. on Management of Data. 1-12.

[2] Koh, J.-L., and Shieh, S.-F. 2004. An efficient approach for maintaining association rules based on adjusting FP-tree structures. In Lee Y-J, Li J, Whang K-Y, Lee D (eds) Proc. of DASFAA 2004. Springer-Verlag, Berlin Heidelberg New York, 417–424.

[3] Leung, C. K.-S., and Khan, Q. I. 2006. DSTree: A tree structure for the mining of frequent sets from data streams. In Proc. of the 6th Int. Conf. on Data Mining (ICDM). 928-932.

[4] Tanbeer, S. K., Ahmed, C. F., Jeong, B.-S., and Lee, Y.-K. 2008. CP-tree: a tree structure for single-pass frequent pattern mining. In Proc. of PAKDD, Lect Notes Artif Int, 1022-1027.

Table 1. Runtime distribution (Sec.)

Dataset with Window Parameters and *min_sup*s	Tree Structure	Tree Construction	Tree Restructuring	Tree Update	Mining Time		Total Time	
					∂_1	∂_2	∂_1	∂_2
BMS-POS (P=50K W=3)	DSTree	25.27	-	3.41	108.43	528.24	137.11	556.92
∂_1 = 6%, ∂_2 = 3%	CPS-tree	26.29	4.46	1.80	5.09	56.32	37.64	88.87
Connect-4 (P=10K, W=2)	DSTree	16.96	-	1.82	195.93	1141.90	214.71	1160.68
∂_1 = 99%, ∂_2 = 90%	CPS-tree	12.50	10.06	1.46	0.03	132.22	24.04	156.24
Kosarak (P=50K, W=4)	DSTree	301.19	-	9.03	52.00	1619.98	362.22	1930.20
∂_1 = 8%, ∂_2 = 1%	CPS-tree	300.46	16.63	4.69	0.06	88.56	321.84	410.34
T40I10D100K (P=10K, W=4)	DSTree	20.72	-	9.32	0.01	1834.55	30.05	1864.59
∂_1 = 25%, ∂_2 = 10%	CPS-tree	16.85	6.90	3.34	0.01	630.44	27.10	657.53