INTL JOURNAL OF ELECTRONICS AND TELECOMMUNICATIONS, 2012, VOL. 58, NO. 4, PP. 323–326
Manuscript received July 24, 2012; revised October, 2012. DOI: 10.2478/v10177-012-0044-0

Probabilistic Sequence Mining – Evaluation and Extension of ProMFS Algorithm for Real-Time Problems

Krzysztof Hryniów and Andrzej Dzieliński

***Abstract*—Sequential pattern mining is an extensively studied method for data mining. One of new and less documented approaches is estimation of statistical characteristics of sequence for creating model sequences, that can be used to speed up the process of sequence mining. This paper proposes extensive modifications to one of such algorithms, ProMFS (probabilistic algorithm for mining frequent sequences), which notably increases algorithm's processing speed by a significant reduction of its computational complexity. A new version of algorithm is evaluated for real-life and artificial data sets and proven to be useful in real-time applications and problems.**

***Keywords*—ProMFS, sequential pattern mining, probabilistic mining, real-time.**

I. Introduction

SEQUENTIAL PATTERN MINING was first defined by Agrawal and Srikant in [1]

> Given a set of sequences, where each sequence consists of a list of itemsets, and given a user-specified minimum support threshold (min_support), sequential pattern mining is to find all frequent subsequences whose frequency is no less than min_support.

One of the algorithms used for sequential pattern mining is ProMFS algorithm (probabilistic algorithm for mining frequent sequences) proposed by Tumasonis and Dzemyda in [2]. This two-step procedure is based on the estimation of statistical characteristics of the main sequence, such as a probability of element occurrence and an average distance between different elements. Based on such characteristics the algorithm creates a shorter, model sequence which is analysed with GSP (Generated Sequence Pattern [3]) or similar algorithms such as SPARSE (Sequential PAttern mining with Restricted Search [4]), SPAM (Sequential PAttern Mining [5]), LAPIN (LAst Position INduction [6]) or in some specific cases even with BDFS(b)-diff-sets [7]. The frequency estimation of the subsequences in the sequence is based on the results of applying GSP algorithm to a model sequence. The ProMFS algorithm is meant to produce slightly less accurate results in shorter time than classical approach algorithms for very large data sets [2].

Extensive testing on both real-life and artificial data sets has shown that the originally proposed ProMFS algorithm can generate inadequate results in much longer time then the basic GSP algorithm. The experiments on many types of commonly available data, have shown that the original ProMFS algorithm's working speed is one to two orders of magnitude slower than that of GSP or even basic Apriori algorithm [8]. Moreover, the accuracy of generated results was low for many types of tested data. The longest sequences, which were reflecting the hardest-to-find, and the most interesting patterns, were rarely identified by ProMFS algorithm. Results of tests were shown in [9].

In [9] several modifications to ProMFS algorithm were proposed. They eliminated the accuracy problems of ProMFS caused by creating improper sequences. Two of three criteria were modified greatly improving accuracy – for tested data, ProMFSmod, the modified version of ProMFS algorithm, was able to find all the sequences that were found by basic GSP algorithm without ProMFSmod preprocessing. The working time has improved as well, due to the optimisation of ProMFS algorithm to work with FUP (Fast Update [10]) algorithm – it allowed frequently updated data sets to be computed in only slightly longer time than that of basic GSP algorithm (and for large data sets – even slightly shorter time).
Despite those modifications ProMFS algorithm was still incapable of achieving working times needed for real-time operations, especially for first runs and data sets that were not frequently modified (for King James Bible [11], one of real-life data sets used, working time for version without FUP was circa 37h – as illustrated in Tab. I).

In this article a modification of ProMFS algorithm for real-time applications is proposed, that has superior working time to ProMFS algorithm and due to that allows to use it as a preprocessing tool not only for other algorithms (as ProMFS was intended), but also for real-time rule-based systems.
It is worth noting that ProMFS algorithm does not need any information about a data set to work properly (it can be useful for determining proper parameters g and l) and used as a preprocessing tool it can give additional information about a data set to a second-step algorithm, as it gathers statistical data about used sequences.

II. ProMFS Algorithm Overview

ProMFS algorithm is based on three statistical characteristics of the sequence: probability of an element in the sequence, probability of an element's appearance after another one and average distance between different elements

K. Hryniów and A. Dzieliński are with Institute of Control and Industrial Electronics, Warsaw University of Technology, Koszykowa 75, 00-662, Warsaw, Poland (hryniowk@ee.pw.edu.pl; adziel@ee.pw.edu.pl).

in the sequence. Each element is defined as single, different ASCII character. ProMFS needs two parameters defined:
– l – length of model sequence S;
– minimum support g threshold, for second-step algorithm.

In the first step the algorithm creates matrices with characteristics of a sequence. Probabilities of an element in the sequence are stored in a matrix P:

$$P(i_j) = \frac{V(i_j)}{VS} \tag{1}$$

where:
$V(i_j)$ is the number of occurrences of element i_j in the sequence S;
VS is the length of main sequence S.

Probability of an element occurring after another element is marked as $P(i_j|i_v)$ and $D(i_j|i_v)$ is the distance between elements i_j and i_v in the sequence. Matrix A stores average distances between elements, which are used for the complementary function $\rho(C_r, A_{r,j})$ (rho). After each step, function $\rho(C_r, A_{r,j})$ modifies matrix Q:

$$\rho(C_r, a_{r,j}) \rightarrow Q[i_j, r + a_{r,j}] = Q[i_j, r + a_{r,j}] + 1 \tag{2}$$

At the start of algorithm's work, all elements in matrix Q are null. Matrix M stores minimal distances between elements i_j and i_v that are used to improve algorithm's accuracy as used with combination with average distances they limit the occurrence of improbable subsequences in created model sequence C.
After creating all matrices and filling them with statistical characteristics of sequence, algorithm starts by putting element with maximum $P(i)$ into the first place of model sequence C and changing matrix Q with $\rho(C_r, A_{r,j})$. Next elements of model sequence are chosen according to algorithm presented below until model sequence C is of length l. Elements with maximal value of $Q(i_j, C_r)$ are chosen and inserted into model sequence C:

$$C(r) = max(Q(i_j, C_r)). \tag{3}$$

If $Q(i_j, C_r)$ is equal for two or more elements we choose next element for model sequence based on the second criterion

$$C(r) = max(P(C_{r-1}|i_j), P(C_{r-1}|i_v)). \tag{4}$$

If we have again equal values we use the third criterion

$$C(r) = max(P(i_j|i_v)). \tag{5}$$

In case of equal values, the element with less occurrences in the model sequence is chosen. After model sequence C is of length l the first step of algorithm stops and another algorithm (such as GSP) is used with model sequence C as input instead of longer main sequence S.
Presented algorithm is using modified criteria from [9] that are more accurate then basic criteria presented in [2]. For simplicity presented formulas (4) and (5) are for case with two elements with equal values.

III. FastProMFS Algorithm Overview

FastProMFS algorithm uses different method of loading data sets and generating probability matrices than classical ProMFS. To speed up the process additional two elements are used – character set template T (vector containing the next position in sequence S of each ASCII character) and vector L, containing the last positions of each ASCII character in the sequence.
Both vectors consist of 255 elements (number of characters in ASCII). Each element in vector T represents the difference between the current position of the algorithm and the position where the nearest character with that ASCII code is located. For example:

$$T[67] = 2; T[68] = 10; T[69] = 0;$$

means that next 'A' letter is 2 positions ahead from current position, nearest 'B' letter is 10 positions ahead and 'C' letter is non-existent after current position. The last case improves algorithm's speed, as it reduces the need to search for non-existing characters. Also, introduction of vector T allows to search for only one character, rather than all of them, as positions of all characters (with the exception of current) for the next element checked are the same, only one position closer (all numeric values in vector T are decreased by one).
Vector L is not modified in the course of algorithm's run, as it stores the last position of each of ASCII characters that is unchangeable. After the algorithm moves to that position, corresponding element in vector T is zeroed permanently.

Algorithm

- Input data – sequence S, length of model sequence l, minimum support g threshold (for second-step algorithm);
- Step 1 – sequence S is loaded from a data set; each loaded character (and combination) is counted for generation of matrices $P(i)$ and $P(i|j)$; vector L is created;
- Step 2 – number of different ASCII characters in sequence S is set as m;
- Step 3 – for the first element in sequence S vector T is created; minimal distances between elements in sequence are updated;
- Step 4 – in loop, for each next element $S[n]$, where $n < VS$:
  - a) $T[i]$ is zeroed, where i is ASCII code of n;
  - b) $T[j] = T[j] - 1$, $\forall(j)! = i$;
  - c) minimal distances are updated on base of T;
  - d) if $n \geq L[i]$ algorithm moves to next element; else if $T[i]$ was zero before step 4a it stays so, otherwise $T[i]$ is updated (and also $M[i][i]$);
- Step 5 – matrix A is created;
- Step 6 – matrices $P(i)$ and $P(i|j)$ are created.

After the creation of matrices A, M, $P(i)$ and $P(i|j)$, the rest of algorithm proceeds in the same way as in modified ProMFS algorithm presented in [9].

A. Example

Let us take a sample sequence $'ABCAB'$ to illustrate how modified algorithm works in each step. For simplicity ASCII signs 'A', 'B' and 'C' are mapped to positions 1, 2 and 3 respectively (instead of 67, 68 and 69) in vectors L and T. Length of model sequence is set to 3.

- Step 1 – $L = \begin{bmatrix} 4 & 5 & 3 \end{bmatrix}$
- Step 2 – there are only three different symbols in sequence, so $m = 3$.
- Step 3 – for first element ('A') we create vector $T = \begin{bmatrix} 3 & 2 & 1 \end{bmatrix}$;
  Minimal distances in matrix M are updated and now $M = \begin{bmatrix} 3 & 2 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$
- Step 4 – for second element in sequence ('B'):
  - a) field in T corresponding to 'B' is zeroed; $T = \begin{bmatrix} 3 & 0 & 2 \end{bmatrix}$;
  - b) distances in vector T are lowered by one; $T = \begin{bmatrix} 2 & 0 & 1 \end{bmatrix}$;
  - c) minimal distances are updated for element 'B'; $M = \begin{bmatrix} 3 & 2 & 1 \\ 2 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$;
  - d) as $n = 2 < L[2]$ and $T[2]$ was greater then zero at the beginning of step 4a, we update $T[2]$ and $M[2][2]$; $T = \begin{bmatrix} 2 & 3 & 1 \end{bmatrix}$, $M = \begin{bmatrix} 3 & 2 & 1 \\ 2 & 3 & 1 \\ 0 & 0 & 0 \end{bmatrix}$
- Step 4 – for third element in sequence ('C'):
  - a) field in T corresponding to 'C' is zeroed; $T = \begin{bmatrix} 2 & 3 & 0 \end{bmatrix}$;
  - b) distances in vector T are lowered by one; $T = \begin{bmatrix} 1 & 2 & 0 \end{bmatrix}$;
  - c) minimal distances are updated for element 'C'; $M = \begin{bmatrix} 3 & 2 & 1 \\ 2 & 3 & 1 \\ 1 & 2 & 0 \end{bmatrix}$;
  - d) as $n = 3 = L[3]$ this step is skipped – and from now on algorithm is not searching for 'C' signs.
- Step 4 – for fourth element in sequence ('A'):
  - a) field in T corresponding to 'A' is zeroed; $T = \begin{bmatrix} 0 & 2 & 0 \end{bmatrix}$;
  - b) distances in vector T are lowered by one; $T = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix}$;
  - c) minimal distances are not updated as $M[1][2]$ had the same value and zeroes are ignored as there cannot be zero distance between elements. Such value means that given combination is not present in the sequence;
  - d) as $n = 4 = L[1]$ this step is skipped – and from now on algorithm is not searching for 'A' signs.
- Step 4 – for fifth element in sequence ('B'):
  - a) field in T corresponding to 'B' is zeroed; $T = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}$;
  - b) as all the elements of vector T are zeros this step is skipped;
  - c) minimal distances are not updated as all elements in T are zeros;
  - d) as $n = 5 = L[2]$ this step is also skipped.
- Step 5 – matrix of average distances $A = \begin{bmatrix} 3 & 2 & 1 \\ 2 & 3 & 1 \\ 1 & 2 & 0 \end{bmatrix}$.
  In this case this matrix has the same values as as matrix M due to short sequence size.
- Step 6 – matrices $P(i) = \begin{bmatrix} \frac{2}{5} & \frac{2}{5} & \frac{1}{5} \end{bmatrix}$ and $P(i|j) = \begin{bmatrix} 0 & \frac{2}{5} & 0 \\ 0 & 0 & \frac{1}{5} \\ \frac{1}{5} & 0 & 0 \end{bmatrix}$ are created.

Next steps of algorithm are identical to those in [9] and for given example look following:

- Step 7 – creation of vector $C = \begin{bmatrix} \end{bmatrix}$ and matrix $Q = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$.
- Step 8 – first element is put into model sequence. As two elements in $P(i)$ have the same maximum value ('A' and 'B') and both are not represented in model sequence C criteria (1) – (3) are skipped and element with lower index value is chosen (exactly as in [2]). $C = \begin{bmatrix} A \end{bmatrix}$; $Q = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$.
- Step 9 – for $r = 2$
  - a) second element is chosen according to criterion (1) and $max(Q(i_j, C_2)) = 1$ is for element 'B'; $C = \begin{bmatrix} A & B \end{bmatrix}$
  - b) $Q = \begin{bmatrix} 0 & 0 & 2 \\ 0 & 1 & 1 \\ 0 & 0 & 2 \end{bmatrix}$.
- Step 9 – for $r = 3$
  - a) third element is chosen according to criterion (1) and $max(Q(i_j, C_3)) = 2$ for both elements 'A' and 'C'. Using criterion (2) we get $max(P(A|B), P(B|C) = 0.2$ for C' ; $C = \begin{bmatrix} A & B & C \end{bmatrix}$;
  - b) $Q = \begin{bmatrix} 0 & 0 & 3 \\ 0 & 1 & 2 \\ 0 & 0 & 2 \end{bmatrix}$.
- Step 10 – as model sequence C is full, second-step algorithm is used with model sequence $C = \begin{bmatrix} A & B & C \end{bmatrix}$ as an input.

IV. Algorithm Evaluation

FastProMFS algorithm was evaluated both theoretically (with computational complexity analysis) and experimentally – by running it on a large real-life and artificial data sets.

A. Theoretical Evaluation

Algorithm's computational complexity for each step can be calculated as follows:

- Step 1 – computational complexity of this step is $O(n)$;
- Step 2 – computational complexity of this step is $O(1)$;
- Step 3 – step is finished in VS operations in extreme case, computational complexity of this step is $O(n)$;
- Step 4 – step is finished in $m^2 * (\frac{VS}{m} - 1)$, which can be approximated as $m * VS$ for most cases; as $VS >> m$ algorithm is convergent to $O(n)$;
  In one extreme case step is finished in $\frac{VS^2}{2}$, but it is only in case when $m = \frac{VS}{2}$ (and elements are sorted in specific way) and as $VS >> m$ it can be ignored for practical solutions.
- Step 5 – step if finished in m^2, as $VS >> m$ computational complexity of this step is $O(n)$
- Step 6 – step if finished in m^2, as $VS >> m$ computational complexity of this step is $O(n)$

As the latter part of ProMFS algorithm has computational complexity of $O(n)$, overall computational complexity of FastProMFS algorithm is $O(n)$ making it fast and feasible for real-time applications.

B. Experimental Evaluation

For experimental evaluation, the algorithm was compiled using Microsoft Visual Studio 2010 Ultimate and run on Intel i7 960@3.20 GHz CPU, with ASUS P6T7WS motherboard, 24 GB DDR3 RAM, Windows Server 2008 R2 64-bit operating system with data on Seagate ST31000524AS hard drive. Working speeds of loading data sets and generating matrices (steps that differ in ProMFS and FastProMFS) for different data sets are presented below.

TABLE I
COMPARISON OF WORKING TIMES FOR THE MOST TIME-CONSUMING PART OF PROMFS AND FASTPROMFS ALGORITHMS (STEPS 1-6)

Data set description	Data set size [in thousands of characters]	Number of different ASCII characters	ProMFS working time [s]	FastProMFS working time [s]
King James Bible	4834,8	81	130810	13
Pan Tadeusz	461,4	78	1209	1
randomly generated	500	68	1092	1

Above results are for real-life (first and second) and artificial (third) data sets.

It is clearly visible that proposed modifications greatly reduce working time of the most time-consuming steps of the algorithm, thus making FastProMFS is much faster then basic ProMFS algorithm. For large data sets it can be used in real-time conditions with proper second-step algorithm. In Tab. II are shown results for pattern mining with combination of ProMFS and GSP variants. As can be seen FastProMFS preprocessing is faster then GSP without preprocessing, but best results are obtained for the real-time modifications of both algorithms used – FastProMFS presented in this paper and GPU GSP presented in [12].

It is clearly visible that FastProMFS is capable of working on large data sets in real-time conditions. FastProMFS' accuracy is slightly lower then that of GSP algorithm as longest sequences found by it were one character shorter for all tested data sets. It is considered acceptable trade-off, especially when compared to losses of ProMFS algorithm presented in [9].

TABLE II
WORKING TIMES OF PROMFS AND GSP VARIANTS

Data set description	Working time of algorithm [s]				
	GSP	ProMFS with		FastProMFS with	
		GSP	GPU GSP	GSP	GPU GSP
King James Bible	6250	$> 24h$	$> 24h$	5217	123
Pan Tadeusz	451	1592	1231	384	29
randomly generated	399	1432	1128	342	24

Above results are for very low minimum support threshold (0,03% of data set size).

V. CONCLUSIONS

In the article, a new version of ProMFS algorithm was proposed. It eliminates it's biggest drawback – the algorithm's high computational complexity and large running times. It was evaluated for both real-life and artificial data sets and proved that algorithm is capable of real-time applications as preprocessing step for other algorithms. It has been shown that FastProMFS has low computational complexity of $O(n)$ for real-life data sets and is competitive with other frequent pattern mining algorithms under real-time constraints.
Further improvement of FastProMFS algorithm remains an open problem as adaptation for GPGPU (general purpose computation on graphics hardware) problems would be impossible without extensive modifications.

REFERENCES

[1] R. Agrawal and R. Srikant, "Mining sequential patterns," in *Proceedings of the Eleventh International Conference on Data Engineering*, 1995, pp. 3–14.
[2] R. Tumasonis and G. Dzemyda, "A probabilistic algorithm for mining frequent sequences," in *ABDIS*, 2004.
[3] R. Agrawal and R. Srikant, "Mining sequential patterns: Generalizations and performance improvements," in *International Conference on Extending Database Technology*, 1996, pp. 3–17.
[4] C. Antunes and A. Oliveira, "Sequential Pattern Mining Algorithms: Tradeoffs between Speed and Memory," in *2nd Workshop on Mining Graphs, Trees and Seq*, 2004.
[5] J. Ayres, J. Gehrke, T. Yiu, and J. Flannick, "Sequential PAttern Mining using a Bitmap Representation," in *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2002, pp. 429–435.
[6] Z. Yang, Y. Wang, and M. Kitsuregawa, "LAPIN-SPAM: An improved algorithm for mining sequential pattern," in *Proceedings of the 21st International Conference on Data Engineering Workshops*, 2005, p. 1222.
[7] R. Dass, "An Efficient Algorithm for Frequent Pattern Mining for Real-Time Business Intelligence Analytics in Dense Datasets," in *HICSS 06 Proceedings of the 39th Annual Hawaii International Conference on System Sciences*, 2006, p. 170b.
[8] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules in large databases," in *Proceedings of the 20th International Conference on Very Large Data Bases, VLDB*, 1994, pp. 487–499.
[9] K. Hryniów, "Probabilistic sequence mining – evaluation and extension of promfs algorithm," in *IIPhDW2009*, Szklarska Poreba, Poland, 2009.
[10] D. W. Cheung, J. Han, V. Ng, and C. Wong, "Maintenance of Discovered Association Rules in Large Databases: An Incremental Updating Technique," in *Proceedings of the Twelfth International Conference on Data Engineering*, 1996, pp. 106–114.
[11] "King James bible," (1611 Authorized Version, 1769 Revised Edition), http://printkjv.ifbweb.com/.
[12] K. Hryniów, "Parallel pattern mining – application of GSP algorithm for Graphics Processing Units," in *13th International Carpathian Control Conference*, Slovakia, 2012, pp. 233–236.