Journal of Advanced Computer Science & Technology, 5 (2) (2017) 20-32

Journal of Advanced Computer Science & Technology

Website: www.sciencepubco.com/index.php/JACST
doi: 10.14419/jacst.v6i2.7011
Research paper

A heuristic to predict the optimal pattern-growth direction for the pattern growth-based sequential pattern mining approach

Kenmogne Edith Belise[1*], Nkambou Roger[2], Tadmon Calvin[3] and Engelbert MEPHU NGUIFO[4]

[1,3]*Faculty of Science, Department of Mathematics and Computer Science, LIFA, Po. Box. 67 Dschang, Cameroon,*
[2]*Computer Science Department, University of Québec at Montréal, 201 avenue du président-Kennedy Montréal (Québec) H2X 3Y7 Canada, Knowledge Management laboratory,*
[4]*Computer Science Institute - LIMOS - UMR CNRS 6158 Complexe scientifique des cézeaux, 1 rue de la chebarde, 63178 Aubière cedex 49, Bd François-Mitterrand - CS 60032 63001 Clermont-Ferrand Cedex 1*
[*]*Corresponding author E-mail:ebkenmogne@gmail.com*

Abstract

Sequential pattern mining is an efficient technique for discovering recurring structures or patterns from very large datasets, with a very large field of applications. It aims at extracting a set of attributes, shared across time among a large number of objects in a given database. Previous studies have developed two major classes of sequential pattern mining methods, namely, the candidate generation-and-test approach based on either vertical or horizontal data formats represented respectively by GSP and SPADE, and the pattern-growth approach represented by FreeSpan, PrefixSpan and their further extensions. The performances of these algorithms depend on how patterns grow. Because of this, we introduce a heuristic to predict the optimal pattern-growth direction, i.e. the pattern-growth direction leading to the best performance in terms of runtime and memory usage. Then, we perform a number of experimentations on both real-life and synthetic datasets to test the heuristic. The performance analysis of these experimentations show that the heuristic prediction is reliable in general.

Keywords: *sequence mining; sequential pattern; frequent pattern; pattern-growth direction; heuristic; prefixspan; suffixspan*

1. Introduction

Many real world applications have to deal with sequential data. Discovery of sequential patterns [13, 5] from large dataset was first introduced by Agrawal and Srikant [3] in 1995 based on their study of customer purchase sequences, as follows: *Given a set of sequences, where each sequence consists of a list of events (or elements) and each event consists of a set of items, and given a user-specified minimum support threshold of min_sup, sequential pattern mining finds all frequent subsequences, that is, the subsequences whose occurrence frequency in the set of sequences is no less than min_sup*. Sequential pattern mining is one of the important fields in data mining because of its variety of applications in web access pattern analysis, market basket analysis, fault detection in network, DNA sequences etc. It plays a vital role in different areas. It is essentially an enumeration problem over the sub-sequence partial order looking for those sequences that are frequent. The search can be performed in a breadth-first or depth-first manner, starting with more general (shorter) sequences and extending them towards more specific (longer) ones. Many algorithms were proposed for sequential pattern mining. The existing algorithms essentially differ in the data structures used to *index* the database to facilitate fast enumeration. These mining algorithms are broadly classified into two approaches known as the Apriori-based candidate generation approach [3, 4, 8, 9, 10, 14, 18, 3, 20, 22, 23] and the pattern growth approach [11, 17, 15, 16, 12, 19].

Since the first proposal of this data mining task and its associated efficient mining algorithms, there has been a growing number of researchers in the field and tremendous progress [13] has been made, evidenced by hundreds of follow-up research publications, on various kinds of extensions and applications, ranging from scalable data mining methodologies, to handling a wide diversity of data types, various extended mining tasks, and a variety of new applications.

Improvements in sequential pattern mining algorithms have followed similar trend in the related area of association rule mining and have been motivated by the need to process more data at a faster speed with lower cost. Previous studies have developed two major classes of sequential pattern mining methods : The Apriori-based approach [3, 4, 8, 9, 10, 14, 18, 3, 20, 22, 23] and the pattern growth approach [11, 17, 15, 16, 12, 19].

The Apriori-based approach form the vast majority of algorithms proposed in the literature for sequential pattern mining. Apriori-like algorithms depend mainly on the Apriori anti-monotony property, which states the fact that any super-pattern of an infrequent pattern cannot be frequent, and are based on a candidate generation-and-test paradigm proposed in association rule mining [1, 2]. This candidate generation-and-test paradigm is carried out by GSP [3], SPADE [23], and SPAM [4]. Mining algorithms derived from this approach are based on either vertical or horizontal data formats. Algorithms based on the vertical data format involve AprioriAll, AprioriSome and DynamicSome [3], GSP [3], PSP [14] and SPIRIT [8], while those based on the horizontal data format involve SPADE [23], cSPADE

Copyright © 2016 Author. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

[22], SPAM [4], LAPIN-SPAM [20], IBM [18] and PRISM [9, 10]. The generation-and-test paradigm has the disadvantage of repeatedly generating an explosive number of candidate sequences and scanning the database to maintain the support count information for these sequences during each iteration of the algorithm, which makes them computationally expensive. To increase the performance of these algorithms constraint driven discovery can be carried out. With constraint driven approach system should concentrate only on user specific or user interested patterns or user specified constraints such as minimum support, minimum gap or time interval etc. With regular expression these constraints are studied in SPIRIT [8].

To alleviate these problems, the pattern-growth approach, represented by FreeSpan [11], PrefixSpan [15, 16] and their further extensions, namely FS-Miner [6], LAPIN [12, 21], SLPMiner [19] and WAP-mine [17], for efficient sequential pattern mining adopts a divide-and-conquer pattern growth paradigm as follows. Sequence databases are recursively projected into a set of smaller projected databases based on the current sequential patterns, and sequential patterns are grown in each projected database by exploring only locally frequent fragments [11, 16]. The frequent pattern growth paradigm removes the need for the candidate generation and prune steps that occur in the Apriori-based algorithms and repeatedly narrows the search space by dividing a sequence database into a set of smaller projected databases, which are mined separately. The major advantage of projection-based sequential pattern-growth algorithms is that they avoid the candidate generation and prune steps that occur in the Apriori-based algorithms. They grow longer sequential patterns from the shorter frequent ones. The major cost of these algorithms is the cost of forming projected databases recursively. To alleviate this problem, a pseudo-projection method is exploited to reduce this cost. Instead of performing physical projection, one can register the index (or identifier) of the corresponding sequence and the starting position of the projected suffix in the sequence. Then, a physical projection of a sequence is replaced by registering a sequence identifier and the projected position index point. Pseudo-projection reduces the cost of projection substantially when the projected database can fit in main memory.

PrefixSpan [15, 16] and FreeSpan [11] differ at the criteria of partitionning projected databases and at the criteria of growing patterns. FreeSpan creates projected databases based on the current set of frequent patterns without a particular ordering (i.e., pattern-growth direction), whereas PrefixSpan projects databases by growing frequent prefixes. Thus, PrefixSpan follows the unidirectional growth whereas FreeSpan follows the bidirectional growth. Another difference between FreeSpan and PrefixSpan is that the pseudo-projection works efficiently for PrefixSpan but not so for FreeSpan. This is because for PrefixSpan, an offset position clearly identifies the suffix and thus the projected subsequence. However, for FreeSpan, since the next step pattern-growth can be in both forward and backward directions from any position, one needs to register more information on the possible extension positions in order to identify the remainder of the projected subsequences. Note that the mining can proceed from the suffix, which is essentially SuffixSpan [15, 16], an algorithm symmetric to PrefixSpan by growing suffixes from the end of the sequence forward. The main difference between PrefixSpan and SuffixSpan is that PrefixSpan follows the left-to-right pattern-growth direction while SuffixSpan follows the right-to-left pattern-growth direction. PrefixSpan [15, 16] performs much better than GSP[3], SPADE [23] and FreeSpan[11]. The major cost of PrefixSpan is the construction of projected databases.

This paper studies the problem of determining the optimal pattern-growth direction between left-to-right and right-to-left pattern-growth directions, i.e the pattern-growth direction leading to the best performance in terms of runtime and memory usage. To this end, a heuristic is introduced to predict the optimal pattern-growth direction. Then, a number of experimentations on both real-life and synthetic datasets are performed to test the heuristic. The performance analysis of these experimentations show that the heuristic prediction is reliable in general.
The rest of the paper is organized as follows. Section 2 states the problem of mining sequential patterns. Section 3 proposes a heuristic to predict the optimal pattern-growth direction. Section 4 presents experimental results and analyses them. Sub-section 4.1 presents real-life and synthetic datasets used in experimentations. Sub-section 4.2 presents experimental results. Sub-section 4.3 is devoted to performance analysis. Concluding remarks are given in section 5.

2. Statement of the problem of mining sequential patterns

The problem of mining sequential patterns, and its associated notation, can be given as follows:
Let $I = \{i_1,\ i_2,\ \ldots,\ i_n\}$ be a set of literals, termed **items**, which comprise the alphabet. An **itemset** is a subset of items. A **sequence** is an ordered list of itemsets. A sequence s is denoted by $\prec s_1,\ s_2,\ \ldots s_n \succ$, where s_j is an itemset. s_j is also called an **element** of the sequence, and denoted as $(x_1, x_2,\ \ldots, x_m)$, where x_k is an item. For brevity, the brackets are omitted if an element has only one item, i.e. element (x) is written as x. An item can occur at most once in an element of a sequence, but can occur multiple times in different elements of a sequence. The number of instances of items in a sequence is called the length of the sequence. A sequence with length l is called an **l-sequence**. The length of a sequence α is denoted $|\alpha|$. A sequence $\alpha = \prec a_1a_2\ldots a_n \succ$ is called **subsequence** of another sequence $\beta = \prec b_1b_2\ldots b_m \succ$ and β a **supersequence** of α, denoted as $\alpha \subseteq \beta$, if there exist integers $1 \leq j_1 < j_2 < \ \ldots\ < j_n \leq j_m$ such that $a_1 \subseteq b_{j1},\ a_2 \subseteq b_{j2},\ \ldots,\ a_n \subseteq b_{jn}$. Symbol ε denotes the **empty sequence**.
We are given a database S of input-sequences. A **sequence database** is a set of tuples of the form $\prec sid, s \succ$ where sid is a **sequence_id** and s a sequence. A tuple $\prec sid, s \succ$ is said to contain a sequence α, if α is a subsequence of s. The **support** of a sequence α in a sequence database S is the number of tuples in the database containing α, i.e.

$$support(S,\alpha) = |\{\prec sid, s \succ \mid \prec sid, s \succ \in S\ \wedge\ \alpha \subseteq s\}|.$$

It can be denoted as $support(\alpha)$ if the sequence database is clear from the context. Given a user-specified positive integer denoted $min\_support$, termed the **minimum support** or the **support threshold**, a sequence α is called a **sequential pattern** in the sequence database S if $support(S,\alpha) \geq min\_support$. A sequential pattern with length l is called an **l-pattern**. Given a sequence database and the $min\_support$ threshold, **sequential pattern mining** is to find the complete set of sequential patterns in the database.

3. Right-to-left direction versus left-to-right direction

In this section, we introduce a heuristic which recommends a pattern-growth direction between left-to-right direction (also called PrefixSpan) and right-to-left direction (also called SuffixSpan).
Let consider a dataset S, an item x_p, an itemset $s_j = (x_1, x_2,\ \ldots, x_m)$, a sequence $s = \prec s_1,\ s_2,\ \ldots s_n \succ$ and a support threshold st. The weight $Weight(x_p, S, st)$ of item x_p in dataset S according to threshold st is defined as.

$$Weight(x_p,S,st) = \begin{cases} Support(S,x_p) \text{ if} & Support(S,x_p) \geq st \\ 0 \text{ if} & Support(S,x_p) < st \end{cases}$$

The weight $Weight(s_j, S, st)$ of itemset s_j in dataset S according to threshold st is defined as follows.

$$Weight(s_j,S,st) = \sum_{p=1}^{m} Weight(x_p,S,st),$$

The left weight $LeftWeight(S,s)$ of sequence s in dataset S according to threshold st is defined as

$$LeftWeight(s,S,st) = \sum_{j=1}^{size}(size - j + 1)Weight(s_j, S, st),$$

where $size = n/2$. Similarly, the right weight $RightWeight(S,s)$ of sequence s in dataset S according to threshold st is defined as

$$RightWeight(s,S,st) = \sum_{j=leftIndex}^{n}(j - leftIndex + 1)Weight(s_j, S, st),$$

where $leftIndex = n - size + 1$. The cumulative left weight $LeftWeight(S,st)$ of all the sequences in dataset S according to threshold st is defined as

$$LeftWeight(S,st) = \sum_{s\in S} LeftWeight(s,S,st).$$

Similarly, the cumulative right weight $RightWeight(S)$ of all the sequences in dataset S according to threshold st is defined by

$$RightWeight(S,st) = \sum_{s\in S} RightWeight(s,S,st).$$

Recommendations are performed as follows. The left-to-right direction (also called PrefixSpan) is recommended if $LeftWeight(S) < RightWeight(S)$. The right-to-left direction (also called SuffixSpan) is recommended if $RightWeight(S) < LeftWeight(S)$.

4. Experimental results and performance analysis

4.1. Presentation of datasets

The datasets used here are collected from the webpage (http://www.philippe-fournier-viger.com/spmf/index.php) of SPMF software [7]. This webpage provides large datasets in SPMF format that are often used in the data mining litterature for evaluating and comparing algorithm performance.

4.1.1. Real-life datasets

BMSWebView1 (Gazelle) (KDD CUP 2000) : This dataset contains 59,601 sequences of clickstream data from an e-commerce. It contains 497 distinct items. It is called here BMS1_spmf. The average length of sequences is 2.42 items with a standard deviation of 3.22. In this dataset, there are some long sequences. For example, 318 sequences contains more than 20 items.

BMSWebView2 (Gazelle) (KDD CUP 2000) : This is a second dataset (called here BMS2) used in the KDD-CUP 2000 competition. It contains 77,512 sequences of clickstream data. It contains 3340 distinct items. The average length of sequences is 4.62 items with a standard deviation of 6.07 items.

Kosarak : This is a very large dataset containing 990 000 sequences of clickstream data from an hungarian news portal. The dataset in its original format can be found at http://fimi.ua.ac.be/data/. The SPMF format is called Kosarak_converted. However, this dataset is very large. Therefore, a subset of only 10 000 sequences (kosarak10k) and a subset of 25 000 sequences (kosarak25k) are provided in the SPMF format.

SIGN : This is a dataset of sign language utterance containing approximately 800 sequences. The original dataset file in another format can be obtained here with more details on this dataset.

BIBLE : This dataset is a conversion of the Bible into a sequence database (each word is an item). It contains 36 369 sequences and 13905 distinct items. The average length of a sequence is 21.6 items. The average number of distinct items per sequence is 17.84.

LEVIATHAN : This dataset is a conversion of the novel Leviathan by Thomas Hobbes (1651) as a sequence database (each word is an item). It contains 5834 sequences and 9025 distinct items. The average number of items per sequence is : 33.8. The average number of distinct items per sequence is 26.34.

MSNBC : This is a dataset of clickstream data. The original dataset contains 989,818 sequences obtained from the UCI repository. Here the shortest sequences have been removed to keep only 31,790 sequences. The number of distinct items in this dataset is 17 (an item is a webpage category). The average number of itemsets per sequence is13.33. The average number of distinct items per sequence is 5.33.

FIFA : A dataset of 20,450 sequences of clickstream data from the website of FIFA World Cup 98. It has 2,990 distinct items. The average sequence length is 34.74 items with a standard deviation of 24.08 items. This dataset was created by processing a part of the web logs from the World Cup 98.

4.1.2. Synthetic datasets

The synthetic datasets used here are collected from the webpage of SPMF software [7]. To generate synthetic sequence databases, you can use the sequence database generator provided in SPMF, which is flexible and easy to use. It is also possible to generate sequence databases by using the IBM Generator. Files generated by the IBM Generator can be converted in SPMF format by using the conversion tool provided in SPMF. Another alternative for generating synthetic sequences databases is to use a Matlab program provided by Ashwin Balani. Here are some synthetic sequence databases generated with the IBM Quest Dataset Generator for Sequential Pattern Mining, converted to the SPMF format:

1: data.slen_10.tlen_1.seq.patlen_2.lit.patlen_8.nitems_5000_spmf
2: data.slen_10.tlen_1.seq.patlen_3.lit.patlen_8.nitems_5000_spmf
3: data.slen_8.tlen_1.seq.patlen_5.lit.patlen_8.nitems_5000_spmf
4: data.slen_8.tlen_1.seq.patlen_6.lit.patlen_8.nitems_5000_spmf

4.2. Experimental results

We exhaustively experimented on both real world and synthetic datasets presented in section 4.1 in order to assess the performance of our approach.

All experiments are done on a 4-cores of 2.16GHz Intel(R) Pentium(R) CPU N3530 with 4 gigabytes main memory, running Ubuntu 14.04 LTS. All the algorithms are implemented in Java and grounded on SPMF software [7].

The experiments consisted of running the pattern-growth algorithms related to the left-to-right and right-to-left directions on each dataset while decreasing the support threshold until algorithms became too long to execute or ran out of memory. For each dataset, we recorded the number of frequent patterns discorved, the execution time, the memory usage and the pattern-growth direction recommeded by the heuristic. The experimental results are presented in tables. The execution times of the heuristic (without the time of loading data) are not presented as they are negligible compared to the execution times of the two mining algorithms, i.e PrefixSpan and SuffixSpan.

Table 1: Execution times (in milliseconds) and memory sizes (in mb) performances following the two pattern-growth directions and heuristic prediction on the real-life dataset BIBLE.

Dataset BIBLE		Left-to-right		Right-to-left		Heuristic
Threshold (in %)	Pattern count	Runtime (in ms)	Memory (in mb)	Runtime (in ms)	Memory (in mb)	Prediction
0.02	5293	553637	930.267	455317	876.033	Right-to-left
0.03	2216	226704	721.167	181669	673.497	Right-to-left
0.04	1185	117672	595.961	92805	572.001	Right-to-left
0.05	774	73706	544.005	57235	517.643	Right-to-left
0.06	507	51357	517.617	39998	486.966	Right-to-left
0.07	371	36581	501.661	29194	454.309	Right-to-left

Table 2: Execution times (in milliseconds) and memory sizes (in mb) performances following the two pattern-growth directions and heuristic prediction on the real-life dataset LEVIATHAN.

Dataset LEVIATHAN		Left-to-right		Right-to-left		Heuristic
Threshold (in %)	Pattern count	Runtime (in ms)	Memory (in mb)	Runtime (in ms)	Memory (in mb)	Prediction
0.02	33387	378601	682.674	484164	767.991	Left-to-right
0.03	12534	199897	531.461	271939	568.760	Left-to-right
0.04	6286	128508	490.904	178264	516.615	Left-to-right
0.05	3698	91745	477.662	126534	472.160	Left-to-right
0.06	2300	63960	441.430	87292	435.925	Left-to-right
0.07	1577	49631	426.322	69908	409.915	Left-to-right

Table 3: Execution times (in milliseconds) and memory sizes (in mb) performances following the two pattern-growth directions and heuristic prediction on the real-life dataset MSNBC.

Dataset MSNBC		Left-to-right		Right-to-left		Heuristic
Threshold (in %)	Pattern count	Runtime (in ms)	Memory (in mb)	Runtime (in ms)	Memory (in mb)	Prediction
0.38	13	10677	535.886	10538	467.362	Right-to-left
0.39	13	10699	500.293	10531	499.377	Right-to-left
0.40	13	10712	518.660	10363	496.498	Right-to-left
0.41	12	10493	509.333	10136	498.207	Right-to-left
0.42	11	10067	511.344	10004	464.402	Right-to-left
0.43	11	10099	475.897	9897	508.419	Right-to-left

Table 4: Execution times (in milliseconds) and memory sizes (in mb) performances following the two pattern-growth directions and heuristic prediction on the real-life dataset FIFA.

Dataset FIFA		Left-to-right		Right-to-left		Heuristic
Threshold (in %)	Pattern count	Runtime (in ms)	Memory (in mb)	Runtime (in ms)	Memory (in mb)	Prediction
0.20	934	41786	557.274	35459	539.499	Right-to-left
0.21	737	35441	511.141	29795	518.640	Right-to-left
0.22	557	27387	469.111	25036	486.738	Right-to-left
0.23	403	21638	443.496	20000	440.658	Right-to-left
0.24	318	18019	430.906	16885	424.836	Right-to-left
0.25	277	16290	419.911	14934	396.822	Right-to-left

Table 5: Execution times (in milliseconds) and memory sizes (in mb) performances following the two pattern-growth directions and heuristic prediction on the real-life dataset BMS1_spmf.

Dataset BMS1_spmf		Left-to-right		Right-to-left		Heuristic
Threshold (in %)	Pattern count	Runtime (in ms)	Memory (in mb)	Runtime (in ms)	Memory (in mb)	Prediction
0.01	77	544	101.077	528	101.188	Right-to-left
0.02	22	232	53.360	230	54.084	Right-to-left
0.03	11	169	54.025	169	53.438	Right-to-left
0.04	5	128	45.337	128	45.347	Left-to-right
0.05	4	122	44.098	121	44.522	Left-to-right
0.06	3	112	43.452	116	43.443	Left-to-right

Table 6: Execution times (in milliseconds) and memory sizes (in mb) performances following the two pattern-growth directions and heuristic prediction on the real-life dataset BMS1_spmf.

Dataset BMS1_spmf		Left-to-right		Right-to-left		Heuristic
Threshold (in %)	Pattern count	Runtime (in ms)	Memory (in mb)	Runtime (in ms)	Memory (in mb)	Prediction
0.0010	3991	6803	217.695	6506	222.413	Right-to-left
0.0011	3010	5942	206.596	5536	196.607	Right-to-left
0.0012	2382	5320	194.733	4974	183.731	Right-to-left
0.0013	1961	4935	183.512	4611	184.140	Right-to-left
0.0014	1686	4596	178.610	4350	173.401	Right-to-left
0.0015	1428	4312	171.280	4056	170.134	Right-to-left

Table 7: Execution times (in milliseconds) and memory sizes (in mb) performances following the two pattern-growth directions and heuristic prediction on the real-life dataset BMS2.

Dataset BMS2		Left-to-right		Right-to-left		Heuristic
Threshold (in %)	Pattern count	Runtime (in ms)	Memory (in mb)	Runtime (in ms)	Memory (in mb)	Prediction
0.0010	23294	31424	460.304	15792	353.775	Right-to-left
0.0011	17820	27545	439.496	14414	359.994	Right-to-left
0.0012	14029	24160	407.116	13166	342.035	Right-to-left
0.0013	11594	21890	376.972	12195	337.262	Right-to-left
0.0014	9401	19168	359.311	11235	339.543	Right-to-left
0.0015	7840	17361	355.599	10471	330.676	Right-to-left

Table 8: Execution times (in milliseconds) and memory sizes (in mb) performances following the two pattern-growth directions and heuristic prediction on the real-life dataset SIGN.

Dataset SIGN		Left-to-right		Right-to-left		Heuristic
Threshold (in %)	Pattern count	Runtime (in ms)	Memory (in mb)	Runtime (in ms)	Memory (in mb)	Prediction
0.10	105544	54587	809.646	52698	791.261	Right-to-left
0.11	78002	44124	660.625	43204	677.467	Right-to-left
0.12	59065	35821	577.677	36039	606.811	Right-to-left
0.13	45460	30486	523.951	30928	519.777	Right-to-left
0.14	34365	24706	438.836	25558	464.903	Right-to-left
0.15	27213	20686	435.898	21978	330.676	Right-to-left

Table 9: Execution times (in milliseconds) and memory sizes (in mb) performances following the two pattern-growth directions and heuristic prediction on the real-life dataset kosarak10k.

Dataset kosarak10k		Left-to-right		Right-to-left		Heuristic
Threshold (in %)	Pattern count	Runtime (in ms)	Memory (in mb)	Runtime (in ms)	Memory (in mb)	Prediction
0.0015	88111	28667	407.745	9526	354.966	Right-to-left
0.0017	31060	11560	352.353	5444	341.654	Right-to-left
0.0019	19477	8006	346.310	4012	335.946	Right-to-left
0.0020	16655	7285	346.767	3634	294.289	Right-to-left
0.0022	14670	6669	345.432	3403	291.439	Right-to-left
0.0023	11265	5504	340.230	2942	285.448	Right-to-left

Table 10: Execution times (in milliseconds) and memory sizes (in mb) performances following the two pattern-growth directions and heuristic prediction on the real-life dataset kosarak25k.

Dataset kosarak25k		Left-to-right		Right-to-left		Heuristic
Threshold (in %)	Pattern count	Runtime (in ms)	Memory (in mb)	Runtime (in ms)	Memory (in mb)	Prediction
0.0015	72778	65839	553.194	14700	406.531	Right-to-left
0.0020	14705	16660	331.601	7399	278.002	Right-to-left
0.0025	8396	11077	298.574	5131	238.042	Right-to-left
0.0030	4957	7711	242.761	4010	210.039	Right-to-left
0.0035	6049	6049	227.242	3297	184.912	Right-to-left
0.0040	2645	4924	195.322	2868	171.507	Right-to-left

Table 11: Execution times (in milliseconds) and memory sizes (in mb) performances following the two pattern-growth directions and heuristic prediction on the real-life dataset Kosarak_converted.

Dataset Kosarak		Left-to-right		Right-to-left		Heuristic
Threshold (in %)	Pattern count	Runtime (in ms)	Memory (in mb)	Runtime (in ms)	Memory (in mb)	Prediction
0.00160	92129	275767	933.843			Right-to-left
0.00163	69537	208065	932.711	36847	766.786	Right-to-left
0.00165	62940	186873	931.178	35762	749.994	Right-to-left
0.00170	47761	143341	837.263	32545	695.050	Right-to-left
0.00180	27250	86661	683.175	27271	544.967	Right-to-left
0.00190	18255	61624	610.035	23422	510.269	Right-to-left

Table 12: Execution times (in milliseconds) and memory sizes (in mb) performances following the two pattern-growth directions and heuristic prediction on the synthetic dataset data.slen_10.tlen_1.seq.patlen_3.lit.patlen_8.nitems_5000_spmf.

Synthetic dataset		Left-to-right		Right-to-left		Heuristic
Threshold (in %)	Pattern count	Runtime (in ms)	Memory (in mb)	Runtime (in ms)	Memory (in mb)	Prediction
0.0010	172897	16127	778.533	18038	780.393	Right-to-left
0.0011	119236	11498	609.879	11444	608.268	Right-to-left
0.0012	93881	9300	527.715	9292	527.741	Right-to-left
0.0013	62149	7698	413.225	7733	411.667	Right-to-left
0.0014	26596	3678	290.741	3648	280.505	Right-to-left
0.0015	23706	3229	261.430	3249	257.898	Right-to-left

Table 13: Execution times (in milliseconds) and memory sizes (in mb) performances following the two pattern-growth directions and heuristic prediction on the synthetic dataset data.slen_10.tlen_1.seq.patlen_2.lit.patlen_8.nitems_5000_spmf.

Synthetic dataset		Left-to-right		Right-to-left		Heuristic
Threshold (in %)	Pattern count	Runtime (in ms)	Memory (in mb)	Runtime (in ms)	Memory (in mb)	Prediction
0.00167	97187	11884	614.571	8002	606.597	Right-to-left
0.00170	35174	5931	326.843	4102	322.446	Right-to-left
0.00172	9710	2263	215.494	2070	191.259	Right-to-left
0.00200	2214	1211	126.248	1401	126.246	Right-to-left
0.00220	2146	1197	126.164	1373	126.161	Left-to-right
0.00250	2123	1168	125.961	1471	125.962	Left-to-right

Table 14: Execution times (in milliseconds) and memory sizes (in mb) performances following the two pattern-growth directions and heuristic prediction on the synthetic dataset data.slen_8.tlen_1.seq.patlen_5.lit.patlen_8.nitems_5000_spmf.

Synthetic dataset		Left-to-right		Right-to-left		Heuristic
Threshold (in %)	Pattern count	Runtime (in ms)	Memory (in mb)	Runtime (in ms)	Memory (in mb)	Prediction
0.0010	181353	18257	809.431	17556	795.277	Right-to-left
0.0011	153240	15058	754.734	13744	743.977	Right-to-left
0.0012	88461	9623	534.108	9012	539.216	Right-to-left
0.0013	71833	7619	475.363	7299	486.217	Right-to-left
0.0014	58953	6447	423.811	6206	434.475	Right-to-left
0.0015	48724	5666	383.937	5225	393.443	Right-to-left

Table 15: Execution times (in milliseconds) and memory sizes (in mb) performances following the two pattern-growth directions and heuristic prediction on the synthetic dataset data.slen_10.tlen_1.seq.patlen_6.lit.patlen_8.nitems_5000_spmf.

Synthetic dataset		Left-to-right		Right-to-left		Heuristic
Threshold (in %)	Pattern count	Runtime (in ms)	Memory (in mb)	Runtime (in ms)	Memory (in mb)	Prediction
0.0014	140339	10903	768.477	9851	757.935	Right-to-left
0.0015	21536	3512	293.686	3520	294.380	Right-to-left
0.0016	9285	2497	226.348	2641	224.654	Right-to-left
0.0017	4490	2049	193.117	2222	189.378	Right-to-left
0.0018	3284	1865	178.694	2061	182.001	Right-to-left
0.0019	2121	1704	171.187	1788	171.177	Left-to-right

4.3. Performance analysis

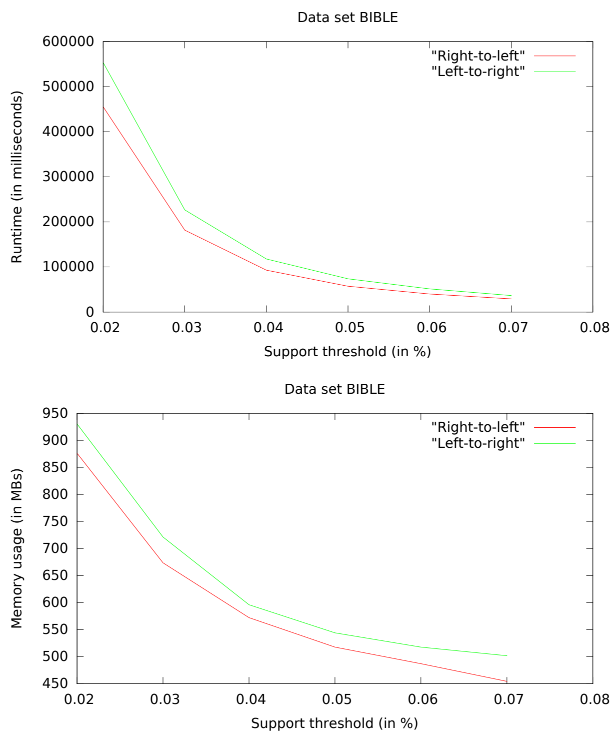

Figure 1: The heuristic recommends the right-to-left pattern-growth direction. It is $1.21 - 1.25$ times faster and requires almost $1.04 - 1.10$ times less memory than the other direction.

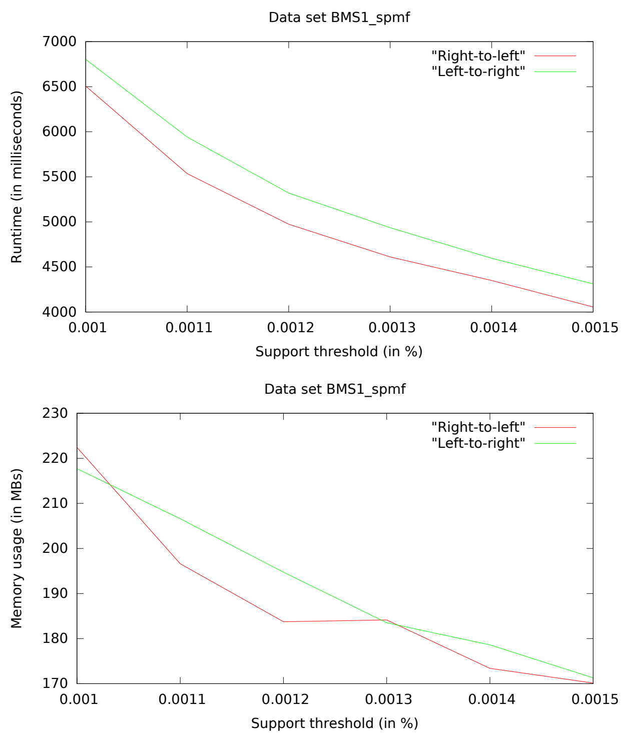

Figure 3: The heuristic recommends the right-to-left pattern-growth direction. It is $1.04 - 1.05$ times faster. It requires almost $1 - 1.05$ times less memory than the other direction if the support threshold is greater than 0.001 and a little more memory otherwise.

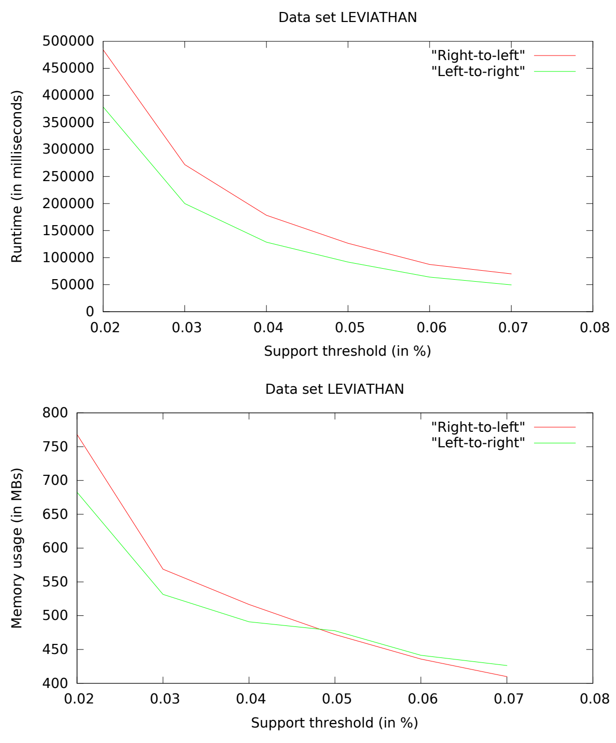

Figure 2: The heuristic recommends the left-to-right pattern-growth direction. It is $1.27 - 1.4$ times faster, and requires less memory if the support threshold is less than 0.05 and a little more memory otherwise.

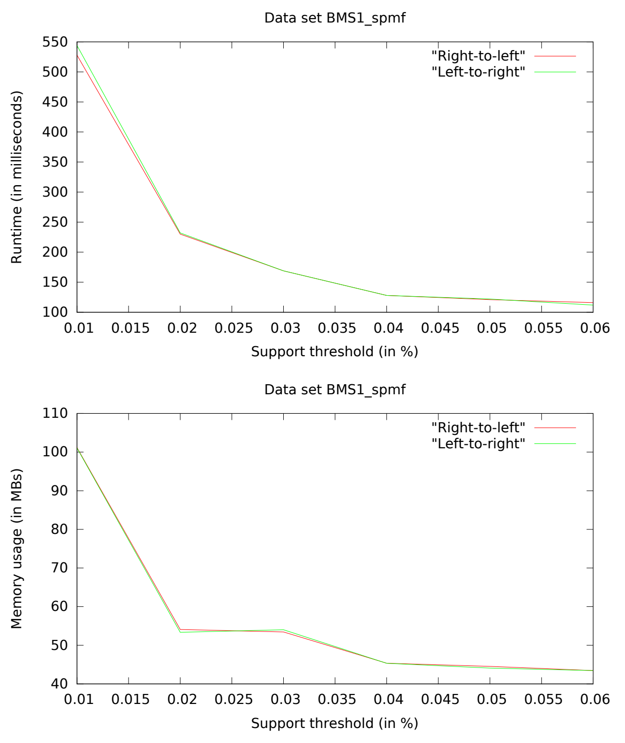

Figure 4: The heuristic recommends the right-to-left pattern-growth direction if the support threshold is within the range $0.01 - 0.03$ and the left-to-right pattern-growth direction if it is within the range $0.04 - 0.06$. The runtime (resp. memory usage) performances of the two directions are very close.

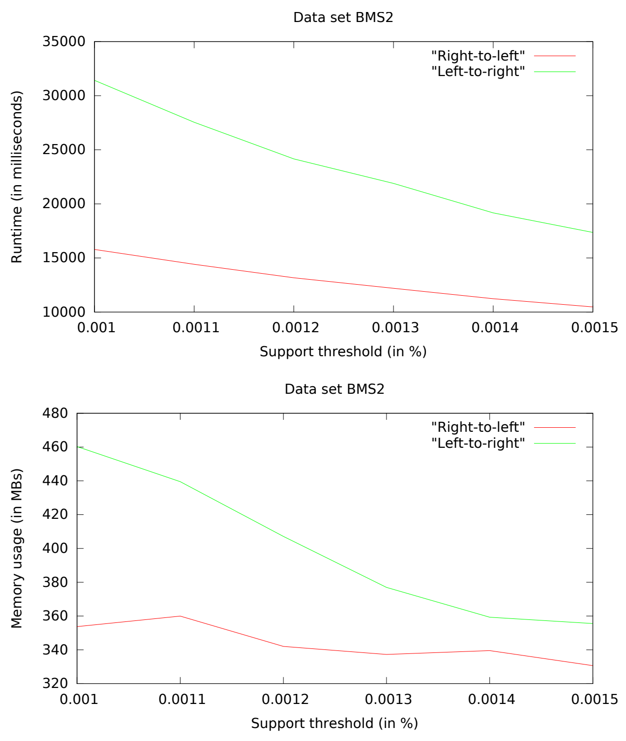

Figure 5: The heuristic recommends the right-to-left pattern-growth direction. It is $1.5 - 2$ times faster and requires almost $1.07 - 1.3$ times less memory than the other direction.

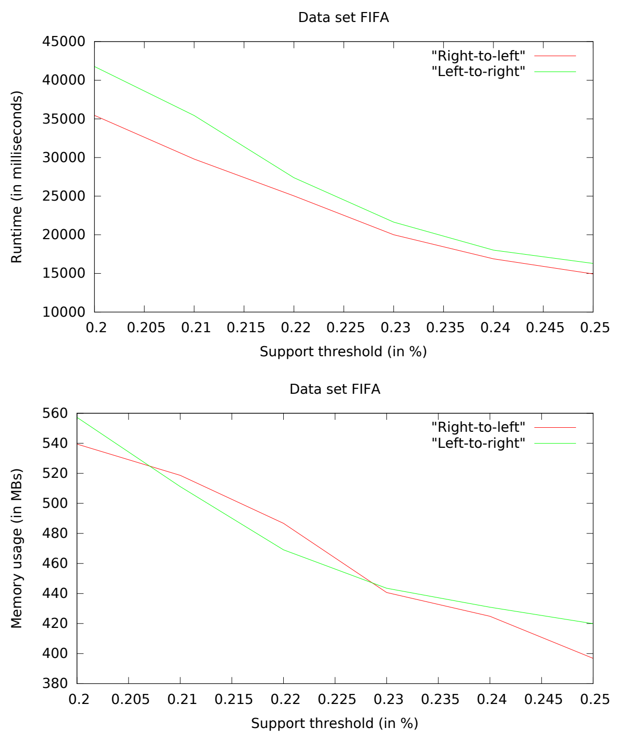

Figure 7: The heuristic recommends the right-to-left pattern-growth direction. It is $1.096 - 1.178$ times faster. It requires less memory if the support is between $0.206 - 0.23$ and a little more memory otherwise.

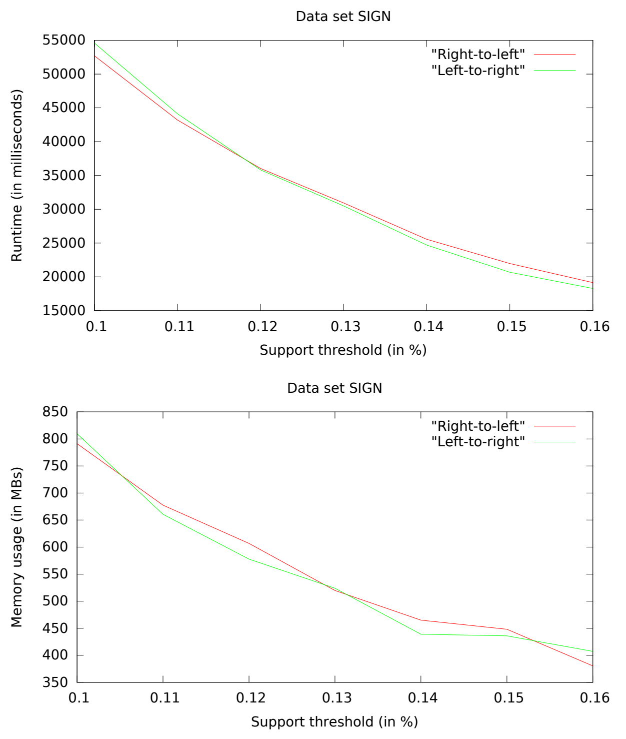

Figure 6: The heuristic recommends the right-to-left pattern-growth direction. It is slightly faster if the support threshold is less than 0.12 and slightly slower otherwise. The memory requirements are close.

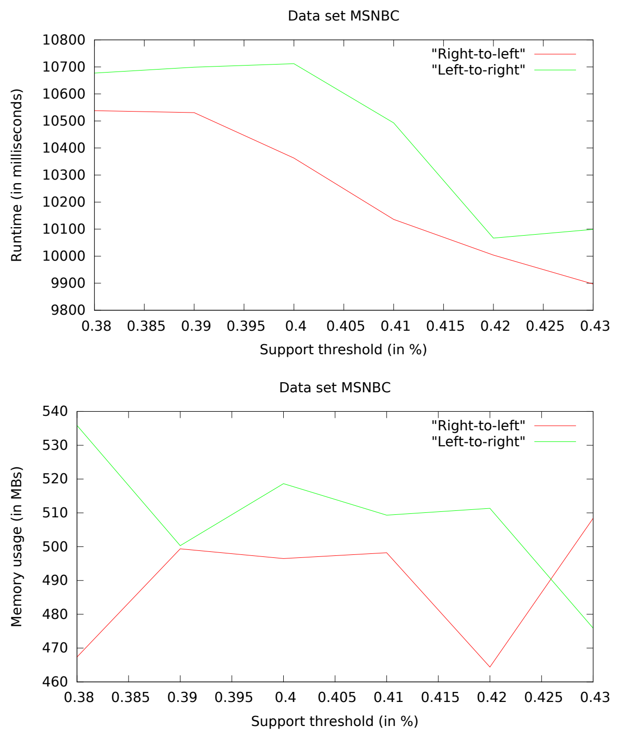

Figure 8: The heuristic recommends the right-to-left pattern-growth direction. It is $1.006 - 1.033$ times faster and requires almost $1.06 - 1.14$ times less memory than the other direction. It is slightly slower if the support threshold is more than 0.425.

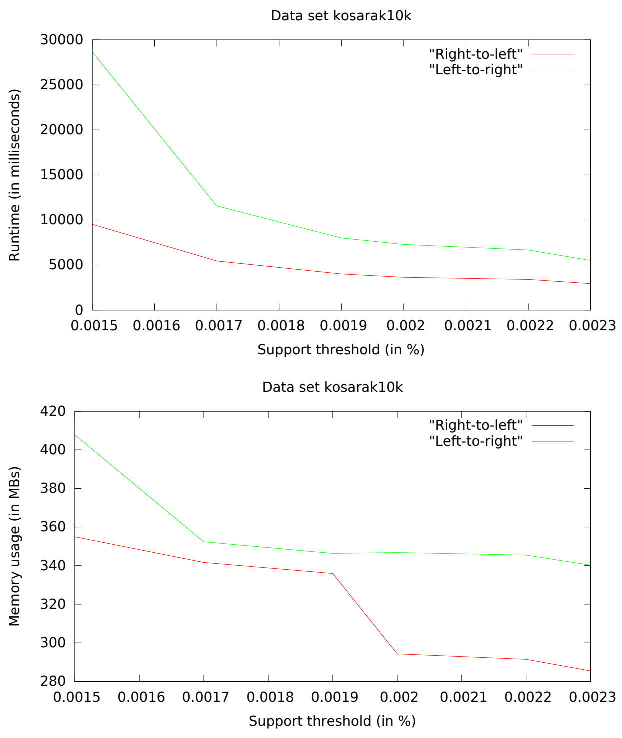

Figure 9: The heuristic recommends the right-to-left pattern-growth direction. It is $1.87 - 3$ times faster and requires almost $1.03 - 1.19$ times less memory than the other direction.

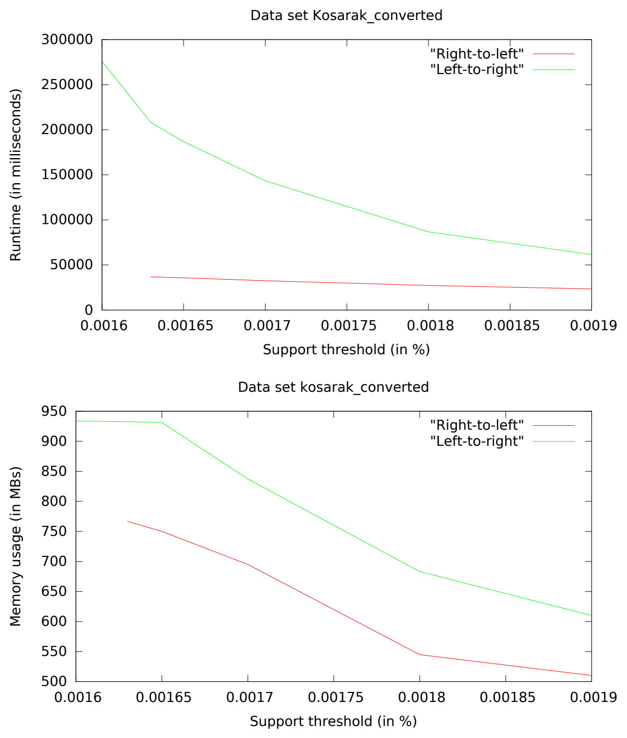

Figure 11: The heuristic recommends the right-to-left direction. It is $2.6 - 5.6$ times faster and requires almost 1.2 times less memory than the other direction.

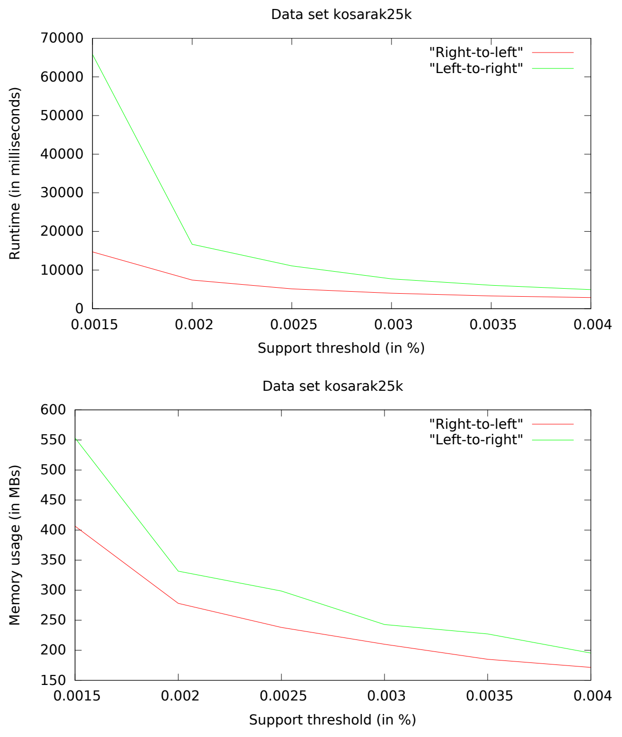

Figure 10: The heuristic recommends the right-to-left direction. It is $1.71 - 4.47$ times faster and requires almost $1.13 - 1.36$ times less memory than the other direction.

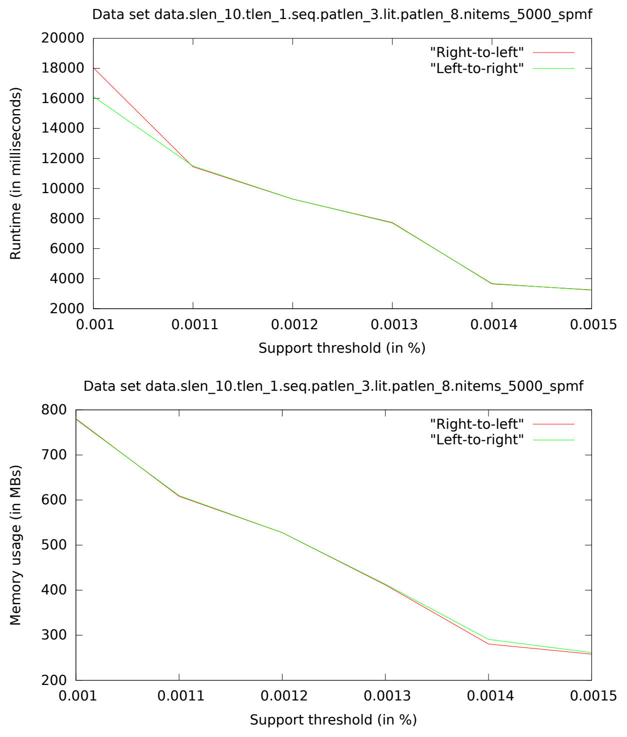

Figure 12: The heuristic recommends the right-to-left direction. It is slightly slower if the support threshold is less than 0.0011. Otherwise the runtimes are very close. The memory requirements are very close.

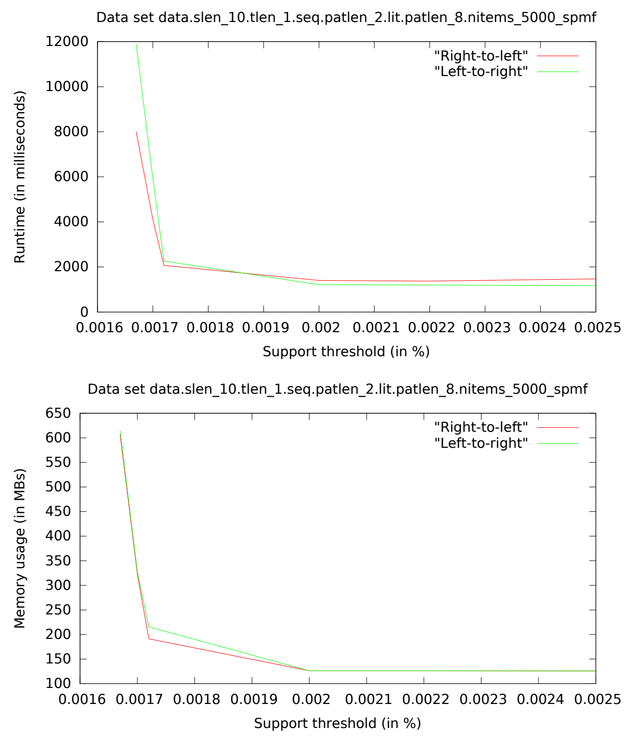

Figure 13: The heuristic recommends the right-to-left direction if the support threshold is within the range $0.00167-0.002$ and the left-to-right direction if it is within the range $0.0022-0.0025$. This prediction is good for three reasons: (1) it requires a little less memory, (2) it is slightly faster if the support threshold is within $0.00167-0.0017$ or $0.0022-0.0025$, and (3) the runtime (resp. memory usage) performances of the two directions are very close.

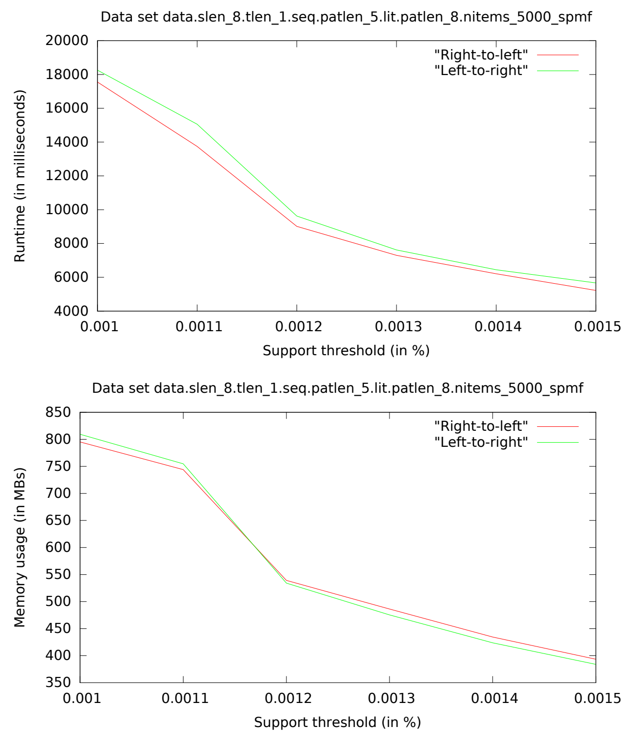

Figure 14: The heuristic recommends the right-to-left pattern-growth direction. It is slightly faster. The memory requirements are very close.

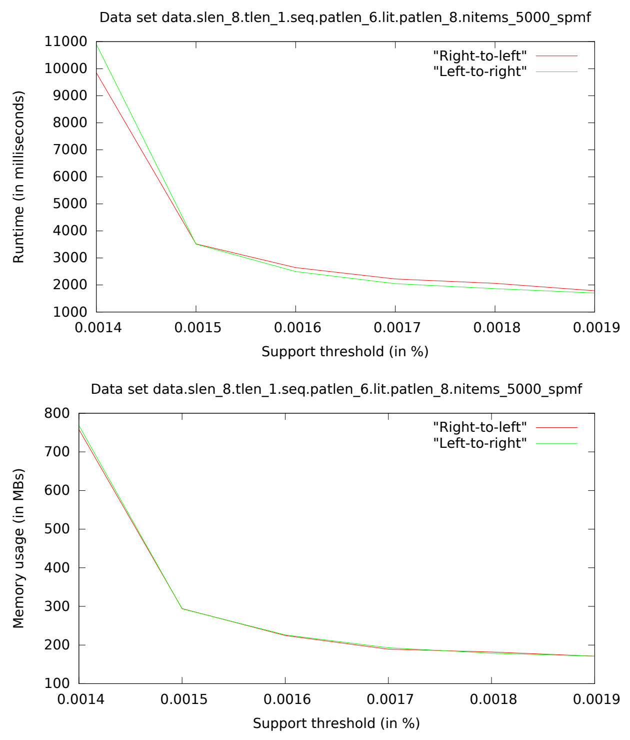

Figure 15: The heuristic recommends the right-to-left direction if the support threshold is within the range $0.0014-0.0018$ and the left-to-right direction if it is within the range $0.0019-0.002$. This prediction is good: (1) it requires almost the same memory, (2) it is slightly faster if the support threshold is less than 0.0015 and (3) the runtime (resp. memory usage) performances of the two directions are very close.

The performance analysis per dataset shows that our heuristic is reliable in general. Thus, it can be used to improve the performances of the pattern growth-based sequential pattern mining approach. The proposed heuristic is based on two tasks: (1) the mining of frequent items and (2) the computation of the cumulative left and right weights of all the sequences in the dataset. Task (1) is included in PrefixSpan [11, 15, 16] and SuffixSpan [11, 15, 16] algorithms and task (2) requires only one scan of the dataset. Because of this, the heuristic runtime is negligible compared to the runtime of the two sequential-pattern mining algorithms, i.e PrefixSpan and SuffixSpan.

5. Conclusion

In this paper, we have studied the impact of the pattern-growth direction on the performances of the pattern growth-based sequential pattern mining algorithm. Our study show that the pattern-growth direction affect both the runtime and memory usage performances. This has motivated us to introduce a heuristic which recommends a pattern-growth direction between left-to-right direction (also called PrefixSpan) and right-to-left-direction (also called SuffixSpan). Performance analysis related to experimental results obtained from eight real-life datasets and four synthetic datasets, commonly used for evaluating and comparing sequential pattern algorithms, shows that the heuristic prediction is reliable in general. Futhermore the heuristic runtime is negligible compared to the runtime of the two sequential-pattern mining algorithms, i.e PrefixSpan and SuffixSpan.

References

[1] R. Agrawal, T. Imielinski, and A. N. Swami. Mining association rules between sets of items in large databases. pages 207–216, 1993.

[2] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *VLDB'94, Proceedings of 20th International Conference on Very Large Data Bases, September 12-15, 1994, Santiago de Chile, Chile*, pages 487–499, 1994.

[3] R. Agrawal and R. Srikant. Mining sequential patterns. In *Proceedings of the Eleventh International Conference on Data Engineering, March 6-10, 1995, Taipei, Taiwan*, pages 3–14, 1995.

[4] J. Ayres, J. Flannick, J. Gehrke, and T. Yiu. Sequential pattern mining using a bitmap representation. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, July 23-26, 2002, Edmonton, Alberta, Canada*, pages 429–435, 2002.

[5] T. Dam, K. Li, P. Fournier-Viger, and Q. Duong. An efficient algorithm for mining top-rank-k frequent patterns. *Appl. Intell.*, 45(1):96–111, 2016.

[6] M. El-Sayed, C. Ruiz, and E. A. Rundensteiner. Fs-miner: efficient and incremental mining of frequent sequence patterns in web logs. In *Sixth ACM CIKM International Workshop on Web Information and Data Management (WIDM 2004), Washington, DC, USA, November 12-13, 2004*, pages 128–135, 2004.

[7] P. Fournier-Viger, A. Gomariz, T. Gueniche, A. Soltani, C. Wu, and V. S. Tseng. SPMF: a java open-source pattern mining library. *Journal of Machine Learning Research*, 15(1):3389–3393, 2014.

[8] M. N. Garofalakis, R. Rastogi, and K. Shim. SPIRIT: sequential pattern mining with regular expression constraints. In *VLDB'99, Proceedings of 25th International Conference on Very Large Data Bases, September 7-10, 1999, Edinburgh, Scotland, UK*, pages 223–234, 1999.

[9] K. Gouda, M. Hassaan, and M. J. Zaki. Prism: A primal-encoding approach for frequent sequence mining. In *Proceedings of the 7th IEEE International Conference on Data Mining (ICDM 2007), October 28-31, 2007, Omaha, Nebraska, USA*, pages 487–492, 2007.

[10] K. Gouda, M. Hassaan, and M. J. Zaki. Prism: An effective approach for frequent sequence mining via prime-block encoding. *J. Comput. Syst. Sci.*, 76(1):88–102, 2010.

[11] J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, and M. Hsu. Freespan: frequent pattern-projected sequential pattern mining. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining, Boston, MA, USA, August 20-23, 2000*, pages 355–359, 2000.

[12] C. Hsieh, D. Yang, and J. Wu. An efficient sequential pattern mining algorithm based on the 2-sequence matrix. In *Workshops Proceedings of the 8th IEEE International Conference on Data Mining (ICDM 2008), December 15-19, 2008, Pisa, Italy*, pages 583–591, 2008.

[13] N. R. Mabroukeh and C. I. Ezeife. A taxonomy of sequential pattern mining algorithms. *ACM Comput. Surv.*, 43(1):3, 2010.

[14] F. Masseglia, F. Cathala, and P. Poncelet. The PSP approach for mining sequential patterns. In *Principles of Data Mining and Knowledge Discovery, Second European Symposium, PKDD '98, Nantes, France, September 23-26, 1998, Proceedings*, pages 176–184, 1998.

[15] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M. Hsu. Prefixspan: Mining sequential patterns by prefix-projected growth. In *Proceedings of the 17th International Conference on Data Engineering, April 2-6, 2001, Heidelberg, Germany*, pages 215–224, 2001.

[16] J. Pei, J. Han, B. Mortazavi-Asl, J. Wang, H. Pinto, Q. Chen, U. Dayal, and M. Hsu. Mining sequential patterns by pattern-growth: The prefixspan approach. *IEEE Trans. Knowl. Data Eng.*, 16(11):1424–1440, 2004.

[17] J. Pei, J. Han, B. Mortazavi-Asl, and H. Zhu. Mining access patterns efficiently from web logs. In *Knowledge Discovery and Data Mining, Current Issues and New Applications, 4th Pacific-Asia Conference, PADKK 2000, Kyoto, Japan, April 18-20, 2000, Proceedings*, pages 396–407, 2000.

[18] L. Savary and K. Zeitouni. Indexed bit map (IBM) for mining frequent sequences. In *Knowledge Discovery in Databases: PKDD 2005, 9th European Conference on Principles and Practice of Knowledge Discovery in Databases, Porto, Portugal, October 3-7, 2005, Proceedings*, pages 659–666, 2005.

[19] M. Seno and G. Karypis. Slpminer: An algorithm for finding frequent sequential patterns using length-decreasing support constraint. In *Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM 2002), 9-12 December 2002, Maebashi City, Japan*, pages 418–425, 2002.

[20] Z. Yang and M. Kitsuregawa. LAPIN-SPAM: an improved algorithm for mining sequential pattern. In *Proceedings of the 21st International Conference on Data Engineering Workshops, ICDE 2005, 5-8 April 2005, Tokyo, Japan*, page 1222, 2005.

[21] Z. Yang, Y. Wang, and M. Kitsuregawa. LAPIN: effective sequential pattern mining algorithms by last position induction for dense databases. In *Advances in Databases: Concepts, Systems and Applications, 12th International Conference on Database Systems for Advanced Applications, DASFAA 2007, Bangkok, Thailand, April 9-12, 2007, Proceedings*, pages 1020–1023, 2007.

[22] M. J. Zaki. Sequence mining in categorical domains: Incorporating constraints. In *Proceedings of the 2000 ACM CIKM International Conference on Information and Knowledge Management, McLean, VA, USA, November 6-11, 2000*, pages 422–429, 2000.

[23] M. J. Zaki. SPADE: an efficient algorithm for mining frequent sequences. *Machine Learning*, 42(1/2):31–60, 2001.