Hindawi
Mathematical Problems in Engineering
Volume 2022, Article ID 5379086, 23 pages
https://doi.org/10.1155/2022/5379086

Research Article

Discovering Significant Sequential Patterns in Data Stream by an Efficient Two-Phase Procedure

Huijun Tang,[1,2] **Le Wang**,[2] **Yangguang Liu**,[2] **and Jiangbo Qian**[1]

[1]*Faculty of Electrical Engineering and Computer Science, Ningbo University, Ningbo 315211, China*
[2]*Faculty of Finance and Information, Ningbo University of Finance & Economics, Ningbo 315175, China*

Correspondence should be addressed to Huijun Tang; totti_2018@sina.com

Received 14 July 2022; Revised 8 November 2022; Accepted 18 November 2022; Published 13 December 2022

Academic Editor: Ghous Ali

One essential topic of mining sequential patterns in the data stream is to optimize the time-space computations. However, more importantly, it should pay more attention to the significance of mining results as a large portion of them just response to the user-defined constraints purely by accident and they may have no statistical significance. In this paper, we propose FSSPDS, an efficient two-phase algorithm to discover the significant sequential patterns (SSPs) in the data stream with typical sliding windows, which has never been considered in existing problems. First, for generating SSPs candidates with high-quality, FSSPDS takes testable support and pattern length constraints into account and insignificant patterns were removed timely by a pattern-growth method. In the second phase, appropriate permutation testing is used to test the significance of the SSPs candidates. Exact permutation p values are obtained in a novel combination way based on unconditional Barnard's test statistic which better reflects the process of data generations and collections. Experimental evaluations show that FSSPDS allows the discovery of SSPs in the data stream and rivals the state-of-the-art approaches efficiently under the control of family-wise error rate (FWER), especially for time efficiency, which was approximately an order of magnitude higher.

1. Introduction

Mining sequential patterns [1] in the transaction data stream is to output patterns that satisfy the user-defined constraints, such as support (the number of patterns that appear in the transactions), utility, and length. Lots of efficient algorithms are proposed to deal with this kind of problems, including CM-Spam [2], HUSP-ULL [3], and NegPSpan [4], but when there is transactions with the feature of labels, many of the results by traditional mining algorithms may lack statistical significance, i.e., they are appeared just by chance, and we are going to draw patterns which are statistically significant to one of the labels.

Statistically significant pattern mining (SSPM [5]) algorithms are used to solve such problems. It can be used in many applications; for example, in medical treatment, doctors are interested in some sequence of treatments that are statistically significant in the process of adverse drug reaction (ADR) signal detection (responsive vs. unresponsive). Often, SSPM is a two-phase method: producing significant pattern candidates first and then test their significance. It tests the significance of the patterns based on hypothesis testing. If the p value of the testing pattern is less than the test level threshold, it will be flagged as a significant pattern. There may exist a large number of patterns waiting for testing, so it turns into a multiple hypothesis testing problem under FWER or false discovery rate (FDR [6]).

Sliding window [2] is a typical data generation mode of the data stream. In this paper, we focus on mining SSPs in the data stream with sliding windows under the control of FWER. The challenge for achieving this goal is twofold.

One is producing candidates for SSP. Most of the current SSPM methods consider frequent sequential patterns (FSPs) as SSP candidates and many efficient FSPs mining algorithms are used to be candidates producing methods [7]. However, most of the algorithms did not consider the significant factors in the process of candidates generation. In fact, some low-support patterns (although they may be FSPs)

do not meet the minimal testable support requirements [8, 9], and they should be removed in the process of candidates generation so as to reduce the testing number in the second phase, and we are going to consider such requirements in candidates mining phase by introducing pattern lengths and testable support constraints. The insignificant pattern would be removed timely by a pattern-growth method.

The other is testing the candidates. The data size in a sliding window of the data stream is usually small and such data may not have sample representativeness. Permutation p values would be an effective strategy for calculating approximate or exact p values to test the significance of small datasets [10, 11]. In [10–12], the permutation testing algorithm can be acted as a useful method to get better results compared with traditional methods. Nevertheless, such an approach has two disadvantages when they are used to find SSPs in candidates produced in the first phase. One is existing literature produce p values based on the random swapping strategy [11] and it will take a high computational cost to get the final results. The other is that the p values based on the random sampling strategy [10] may be equivalent to 0, and the approximate p values may result in a bad estimation.

Additionally, in the second phase of pattern testing, Fisher's test statics [7, 13] is a frequently used data evaluation statistics, and it can be understood that the data generating process is similar to the observed data sample, and the supports are fixed. However, actually data generations and collections do not show such rules, especially, since the support of the pattern may be changing with time passing. For the example of click patterns drawing in an e-commerce website of members from two districts (regarding two classes), to test the significance of a certain click behavior occur more for one class (district), Fisher's test means that the behaviors are collected to a certain amount of members (overall), and the repeated experiment maintains the test pattern's support. However, there may be another way to collect the data in a fixed period of time; by such a method, the frequency of a click behavior occurrence is not fixed and would be always changed in the experiments. In such a scenario, the latter method can better reflect the process of data generation and collection. Unconditional test statistic such as Barnard's test statistic [9] calculates test statistic value which does not fix the support of testing pattern, and it can be a more appropriate than the calculation rule of traditional Fisher's test statistic.

In this study, we serve to propose FSSPDS for mining SSPs in a novel way. Our contributions are listed as follows.

We produce SSP candidates with a length control by a pattern-growth method under the testable support requirement of Barnard's unconditional test statistic and insignificant candidates are removed timely so as to increase the test level and find more SSPs.

We introduce the usage of Barnard's unconditional test statistic. For reducing the computational time, an approximate upper bound is proposed to reduce calculation time.

We discover SSPs in the data stream based on exact permutation p values by a new combinatorial calculation approach. By producing p values in a short time, it shows superiorities compared with the state-of-the-art exact permutation p values algorithms.

We run experiments on real-world datasets to prove the effectiveness of PSSPDS. Our method can be considered with higher efficiency compared with its counterparts.

The rest of this article is as follows. Section 2 reviews the related works. Section 3 describes the problem and defines related terms. Section 4 gives our corresponding algorithm PSSPDS. Theoretical analysis is given in Section 5. Section 6 shows experimental results on real datasets, and Section 7 gives conclusions.

2. Related Works

Many efficient algorithms have been introduced to deal with the problem of sequential pattern mining with frequency constraints, including Spade [14], PrefixSpan [15], CM-Spam [2], and Lapin [16]. In recent years, specific constraints-based sequential pattern mining has been paid much attention. Sequential association rule mining [17, 18] looks up association rules in transactional data. It does not consider the sequence of items but focuses on the fact that there is an intersection between the front and back itemsets. Episode sequential pattern mining [19] is used to look for patterns in a single sequence, rather than a group of sequences. Periodic sequential pattern mining [20, 21] is used to find patterns that occur frequently and periodically in long sequences. Subgraph mining [22, 23] is another field of sequential pattern mining, which aims to discover all frequent subgraphs in graph databases, the corresponding algorithms based on different data structures (such as list-structure [24], pattern-tree [25], and optimization algorithm [19]) are proposed to solve the related pattern mining problem from sequences database, and all these pattern mining approaches are based on the sequential database and the constraints threshold (selected by the user). Recently, Wang gives Miner-K [25] algorithm to mine the patterns with length constraints, and Nader proposes NEclat-Closed [24] to mine the closed pattern based on a vertical structure. They can obtain the related pattern results in a short time.

When the transaction with label feature, the results returned by the above algorithms may lack significance and some patterns are not statistically meaningful. SSPM algorithms look for significant patterns and have been widely used in e-commerce searching [26], essential protein recognition [27], and community detection [28, 29]. Hämäläinen [30] proposes the SSPM model first and regards significant pattern mining as a multiple-hypothesis testing problem. Webb [31] controls the error rate by introducing FWER and FDR in significant pattern discovery. Bonferroni's control [32] is a traditional correction method under the control of FWER.

The test statistic is an important part of SSPM. Fisher's conditional test statistic is a popular approach to measure the significance [7, 10, 31], and LAMP [8] strategy is used to reduce the calculation time as it puts forward testable support requirements to the testing patterns based on Fisher's test statistic. Barnard's test statistic [9, 14] is known

to be another effective method for calculating p value. Leonardo et al. [9] propose a novel structure UT for evaluating the significance of a pattern and it gives the testable support requirements based on Barnard's test statistic. Jiang et al. [33] put forward an unconditional test to get the p values of two different distributions, and it gives the conclusion that the p value produced by Barnard's test statistic has less risk but usually the computation is expensive when the test data sample is large.

The permutation-based method could be acted as an excellent technology to mine significant patterns on small data samples. He et al. [10] use a permutation test by returning all exact p values of the patterns. Llinares-L_opez and Sugiyama [34] propose a permutation test for the process of mining significant sequential patterns, and Pellegrina and Vandin [35] apply a permutation test to mine the top-k significant sequential patterns in the database. Recently, Tonon and Vandin [11] propose the algorithm PROMISE with two strategies: itemsets swapping and random permutations, and it can be known as a state-of-the-art method that draws significant sequential patterns in the transactional database under the control of FWER.

There are also some other methods for studying significant patterns. Riondato et al. [36] propose a significant pattern mining algorithm based on progressive sampling pattern testing. Thien et al. [37] apply the mining results of the significant pattern test to utility dataset analysis. Zihayat et al. [38] extract significant patterns in gene sequences. Fournier-Viger et al. [39] output significant subgraphs in large graphs. Cheng et al. [40] propose the algorithm LTC to look for significant patterns in the data stream which are not only frequent but also persistent.

To the best of our knowledge, no SSPM studies have hitherto considered the significant factors in candidates mining and focused on producing permutation p values based on unconditional test statistics for mining statistically significant sequential patterns in the data stream. Our present paper demonstrates the feasibility and the advantages of our efficient two-phase algorithm.

3. Related Works

3.1. Significant Sequential Patterns. A transaction data stream DS (data stream) can be known as $\mathrm{DS} = \{t_1, t_2, \ldots, t_n\}$, where t_i is the ith transaction. $I = \{x_1, x_2, \ldots, x_m\}$ be a set of literals. Each transaction is assigned to the label G^0 or G^1. A sliding window W is defined as drawing transactions from ith to jth arrival of transactions with a pregiven sliding length. We are going to mine SSPs based on the following definitions.

Definition 1. Pattern X is flagged as a candidate of SSP if its support is higher than or equal to $MinS$.

X could be a sequence of items in I, the number of X occurrences in W is known as the support $S(X)$, and given a support threshold λ $(0 < \lambda < 1)$, $MinS = \lambda|W|$, and if $S(X) \geq MinS$, it is said to be a candidate of SSP.

Definition 2. FSP X is flagged as SSP if its p value is less than a test level threshold.

Hypothesis testing is used to highlight the significance of the pattern in a sliding window. $\pi(X, G^i)$ is the probability that X with label $G^i (i \in \{0, 1\})$. H_0: $\pi(X, G^0) = \pi(X, G^1)$ is considered as the null hypothesis; our goal is to assess the significance of X based on the observed contingency table [15] for evaluating whether it supports H_0. If the p value of X is known, H_0 will be rejected iff $P_X < \alpha$, where α is the significant level, and then X is considered as a significant pattern. P_X is the p value of X based on the observed data sample.

3.2. Permutation Testing and Test Statistic Selection. According to excellent performance on small data samples, we are going to produce p values in Definition 2 by permutation testing. Permutation testing judges whether the observed patterns are significant through the distribution of the patterns. The general process of the testing significance of pattern X can be shown in Figure 1.

Fisher's test statistic is a frequently used statistic value in the traditional permutation testing [15–17]. Its calculation process is based on the 2×2 contingency table which is known in Table 1.

$S_1(X)$ and $S_0(X)$ are the supports of X belonging to G^1 and G^0. n_1 and n_0 are the total numbers of rows with each label, and n is the whole transaction number. Given the support of X, $P_F(S_1(X))$ is calculated as follows:

$$P_F(S_1(X)) = \frac{\binom{n_1}{S_1(X)}\binom{n_0}{S_0(X)}}{\binom{n}{S(X)}}. \tag{1}$$

The final test statistic value for X was established as follows:

$$P^F(X) = \sum_{P_F(x) <= P_F(S_1(X))} P_F(x). \tag{2}$$

Barnard's test statistic is another test statistic as previously mentioned and different from Fisher's test statistic, it does not fix the row or column value of the contingency table which will better reflect the data generations and collections, and for X, the nuisance parameter π is the assumed value based on the hypothesis H_0. The following is defined:

$$P(S(X), S_1(X), \pi) = \binom{n_0}{S_0(X)}\binom{n_1}{S_1(X)}\pi^{S(X)}(1-\pi)^{(n-S(X))}, \tag{3}$$

where n_0 and n_1 are fixed and $S(X)$ acted as a random variable according to the support of the pattern. Given the value of π, define the function as follows:

$$P_S(S(X), \pi) = \sum_{P(x,y,\pi) <= P(S(X), S_1(X), \pi)} P(x, y, \pi), \tag{4}$$

$P_s(S(X), \pi)$ is the sum of the test statistic of observing a contingency table for X that is as or more extreme than the

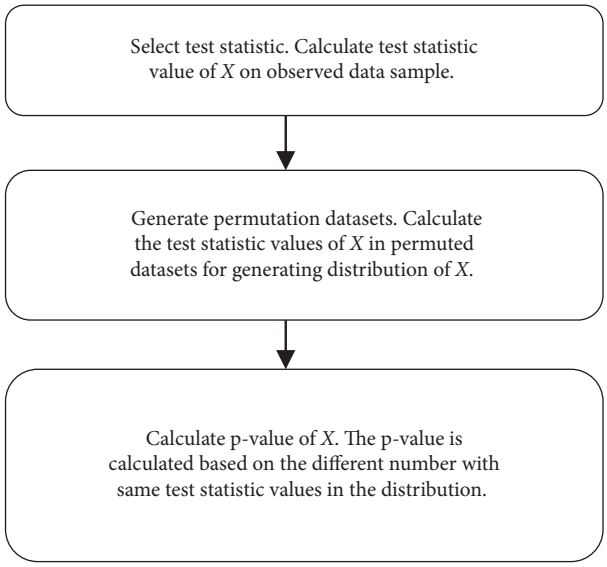

Figure 1: Permutation testing in data stream.

Table 1: 2×2 contingency table for X.

Transactions	$S_t(X) = 1$	$S_t(X) = 0$	Rows
$t \in G^1$	$S_1(X)$	$\eta_1 - S_1(X)$	η_1
$t \in G^0$	$S_0(X)$	$\eta_0 - S_0(X)$	η_0
Columns	$S(X)$	$\eta - S(X)$	η

observed one if H_0 is true. The nuisance parameter π is in the range of (0, 1), and the final test statistic value of X is known as follows:

$$P_B(X) = \max_{\pi \in [0,1]} \left(P_S(S(X), \pi)\right). \tag{5}$$

This maximum is calculated over all possible values of the nuisance parameter. Its test statistic value is usually less than which is produced by Fisher's test statistic.

For a given test level α, a testable support requirement by Barnard's test statistic [24] is given by the following:

$$P_S\left(\frac{x_s, x_s}{n}\right) = \alpha. \tag{6}$$

Based on formula (6), for a testing pattern X, if it is to be a significant pattern, then its support must meet $S(X) > x_s$.

In step 2 of Figure 1, generating permuted datasets will take high computational cost; in most practical cases, the permutation number is constrained as a fixed value for reducing the running time, but p value produced by such strategy is an approximation of the exact distribution which may lead to a bad estimation.

4. The Method

We are going to mine SSPs in a novel way under the framework of FWER. In the mining process of SSPs candidates, we introduce pattern lengths and the testable support constraints and insignificant patterns are removed timely. In the testing process, a close upper bound of Barnard's test statistic is proposed to reduce calculation time, and permutation testing with a combination strategy is introduced to get exact p values, and the proposed algorithm is verified to have a significant improvement compared with state-of-the-art methods.

4.1. Mining SSP Candidates. Longer patterns tend to have low support and are more likely to be insignificant by testable support requirement. In the process of SSP candidates mining, we introduce a user-specified length to reduce the search computations; according to the excellent mining performance by tree structure with length constraint in [37, 38], we establish a pattern tree and mine all SSP candidates from the tree under pattern length and testable support control; the mining process mainly consists of two phases, as shown in Algorithm 1.

To draw SSPs candidates more efficiently, two pruning strategies are proposed to optimize the mining process, and they are used in the process of CreateTree and SSPs_Candidates in Algorithm 1, respectively.

Theorem 1. *If $S(X) \leq MinS$, supersets of X are not SSP candidates.*

Proof. Let Xe be a super set of X, then $S(Xe) \leq S(X)$; if $S(X) \leq MinS$, then X will not be a candidate based on Definition 1, so Xe must not be a SSP candidate.

Theorem 2. *If $S(X) < x_s$ in formula (6), supersets of X are not SSP candidates.*

Proof. Let Xe be a super set of X. The same as Theorem 1, $S(Xe) \leq S(X)$. When $S(X) < x_s$, then $S(Xe) < x_s$;based on formula (6), Xe will not be an SSP candidate.

Based on the efficient two pruning strategies, we use the efficient tree structure by a pattern-growth method to produce the candidates. We take the data in Figure 2(a) as an example and set $\lambda = 0.5$ and $k = 3$ as follows:

(1) Calculate the testable support value $x_s = 3$ based on formulas (4) and (6). Remove the unpromising items whose support is less than 3. Therefore, delete "G."

(2) The header table consists of two parts which are the support and the link pointer. By one scan, create the header table H, add T_1 to the tree, and there are two types of nodes as shown in Figure 2(b); one is an ordinary node, such as node "A" and "C." Another node is a leaf node such as "F.1," which means "F" is the leaf node and the support of the sequential path is 1.

(3) Add sequential transactions to the tree, Figure 2(c) shows the result after the second transaction T_2 is added. Since there are two "D" items in this transaction, record the position of the previous "D," which is represented by $S = \{2\}$, and set the link pointer to the previous item.

(4) Figure 2(d) shows the result of adding the third transaction. In the process of adding, if the sequential

```
Input: DS: dataset
       λ: support threshold
       k: user-specified length
       x_s: testable support
Output: SSPs candidates
//create a Tree T and a header table H
(1) T, H = CreateTree (DS, λ, x_s)
//find SSPs Candidates
(2) Candidates = SSPs_Candidates (x_s, T, H, base-item, k, and λ)
(3) Output Candidates
```

Algorithm 1: SSPsCandidates.

transaction items already exist in the tree, you only need to add the corresponding support. The result after adding all transactions to the tree is shown in Figure 2(e). In order to describe the sequence more clearly, the pointer link is hidden in Figure 2(e).

The creation algorithm is shown in Algorithm 2. We first calculate the testable support value, remove these items whose support values or support are not satisfied from the header table (lines 1–7), insert sequential transactions after removing the insignificant items, accumulate the support values of the item in each path into the header table, and add the link pointer of the new node to the header table (lines 8–16); line 17 returns the tree T and the header table H.

The above process effectively constructs the global tree and maintains the data into a tree and a header table, and algorithm SSPs_Candidates uses pattern-growth method to mine all SSPs candidates. The algorithm processes the items in the header table from a bottom-to-top sequence. Based on the final tree and head table in Figure 2(e), here we demonstrate the example by mining SSPs candidates with item "F" and "E" as the tail nodes. The process could be known in Figure 3.

The support of item "F" is bigger than 3, and we can create a subtree and subheader table for base-item ("F"). According to the node pointer, analyze the paths with "F." The path <"A" and "C"> is obtained from <"A," "C," and "F"> with leaf node "F," <"A," "C," "D," "B," and "E"> and <"C" and "E"> are obtained from the leaf node "C" and "B." The subtree and subheader table of item ("F") are shown in Figure 3(a). Delete the items whose support does not meet the support requirements; thus, only item "C" is left. Then, the subheader table and subtree with base <"C" and "F"> are established, as shown in Figure 3(b). However, since the support of the remaining item "A" is less than 3, the program is interrupted. Continue to search the candidates of item "E" by the same operation steps, the subtree and subheader table with base "E" are shown in Figure 3(c), and the tree with base <"B" and "E"> is shown in Figure 3(d). It can be seen that patterns whose last item is "E" are also removed as the supports that are not satisfied and continue to look for candidates with the next item until all items in header table H are processed.

The specific process of SSPs_Candidates is shown in Algorithm 3. For the current processing item, if the length and support constraints are satisfied, it will be added to the base item (Lines 1–5). If the length is not satisfied but the support is reached, then a subheader table and a subtree will be established and continue to pursuit the candidates by a recursion (Lines 6–9). When the current item is completed and processed, remove it and go to look for candidates of the next item in H (Line 10). Line 12 returns the final candidates. □

4.2. Testing SSP Candidates

4.2.1. Close Upper Bound of Barnard's Test Statistic. To formula (5), one has to overcome the high computation to calculate the sum of the value which has equal or lower probability than being observed, and we look for a close upper bound of Barnard's test statistic.

Lemma 1. $\operatorname{argmax}_{\pi}\{P(S(X), S_1(X), \pi)\} = S(X)/n$.

Proof. To formula (3), when nuisance parameter π is unique variable and other variables are fixed, based on [24], $P(S(X), S_1(X), \pi)$ is a function of π $(0 < \pi < 1)$ and the derivative of π could be calculated as follows:

$$\begin{aligned}\frac{\partial P(S(X), S_1(X), \pi)}{\partial \pi} &= \binom{n_0}{S(X) - S_1(X)} \binom{n_1}{S_1(X)} \left(S(X)\pi^{(S(X)-1)}(1-\pi^{(n-S(X))} + (S(X)-n)\pi^{S(X)}(1-\pi^{(n-S(X)-1)}\right) \\ &= \binom{n_0}{S(X) - S_1(X)} \binom{n_1}{S_1(X)} \pi^{(S(X)-1)}(1-\pi^{(n-S(X)-1)}(S(X) - n\pi).\end{aligned} \tag{7}$$

```
Input: DS: dataset
       λ: support threshold
       x_s: testable support
Output: a tree T and a header table H
 (1) Initiate a header table H containing the fields of the item, support, and links
 (2) For each transaction T_d of DS, do
 (3)    For each item X in T_d do
 (4)       Calculate H.X.support
 (5)    End For
 (6) End For
 (7)    Delete unpromising items from H with support and x_s constraints
 (8) Initialize a Tree T with an empty root node
 (9) For each transaction T_d of DS, do
(10)    Delete unpromising items from T_d
(11)    Insert the sequential itemset S of T_d
(12)    For each item X in S
(13)       Update H.X.support
(14)       Add the links
(15)    End For
(16) End For
(17) Return T and H
```

Algorithm 2: GreateTree.

```
Input: T: a tree, H: a header table, and base-item
       λ: support threshold
       k: user-specified length
       x_s: testable support
Output: candidates
 (1) Candidates = ()
 (2) For each item Q in H (with a bottom-up sequence) do
 (3)    If H.Q.support > x_s then
 (4)       base-item = Q ∪ base-item
 (5)       If |base-item| ≤ k and base-item.support ≥ MinS then
 (6)          Copy the base-item to Candidates
 (7)          Create a subtree subT and a subheader table subH
 (8)          SSPs_Candidates (subT, subH, base-item, k, x_s, and λ)
 (9)       End If
(10)    End If
(11) Remove Q from H
(12) End For
(13) Return Candidates
```

Algorithm 3:SSPs_Candidates.

It could be known that when $\pi = S(X)/n$, the value of the first derivative is 0. In this circumstance, the second derivative could be calculated as follows:

$$\frac{\partial^2 P(S(X), S_1(X), \pi)}{\partial^2 \pi} = -n \binom{n_0}{S(X) - S_1(X)} \binom{n_1}{S_1(X)} \pi^{(S(X)-1)} (1-\pi)^{(n-S(X)-1)}, \tag{8}$$

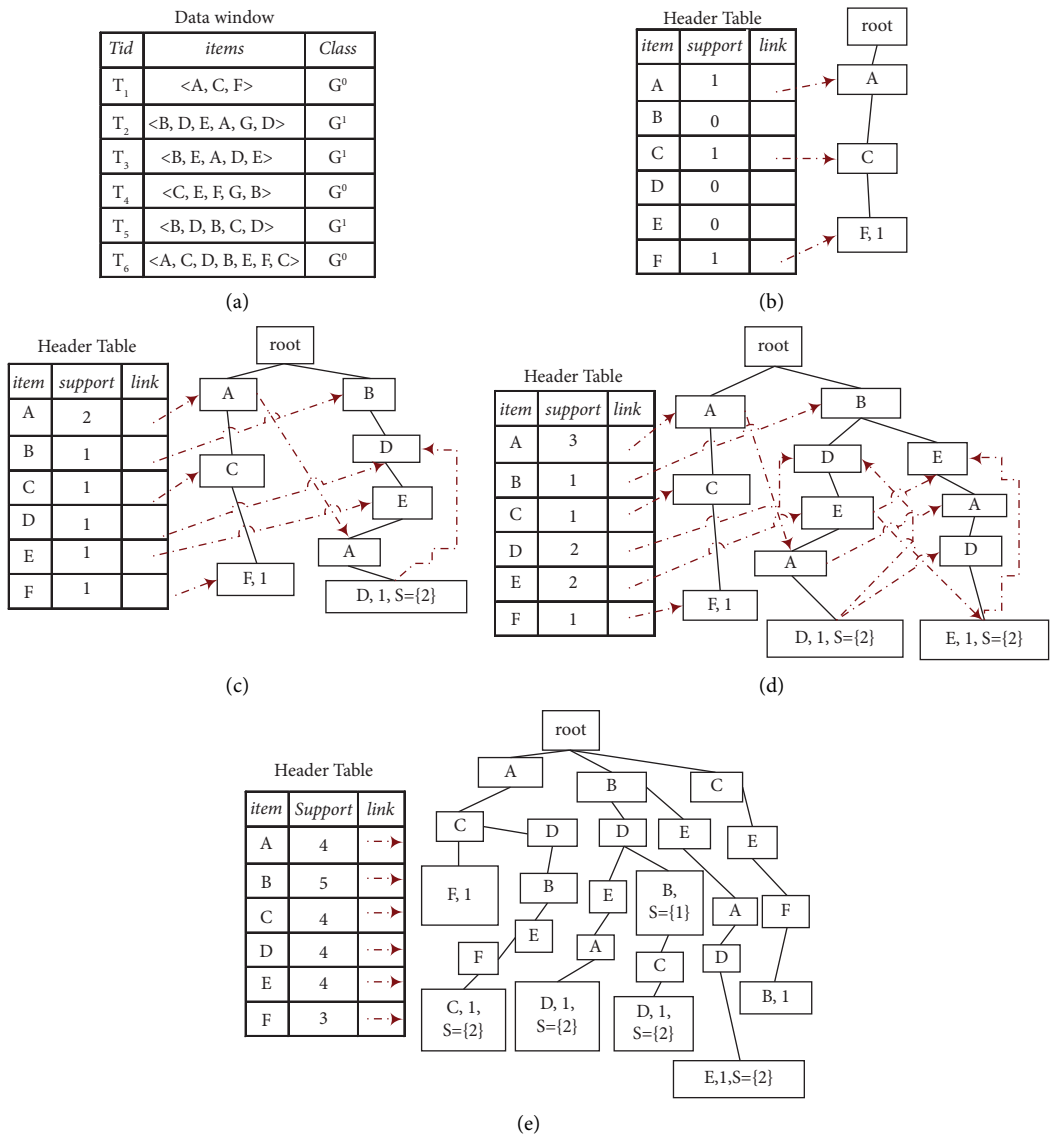

FIGURE 2: A case of creating a tree and a header table. (a) The header table, (b) the tree after adding T_1, (c) the tree after adding T_2, (d) the tree after adding T_3, and (e) the tree after adding all transactions (pointer hidden).

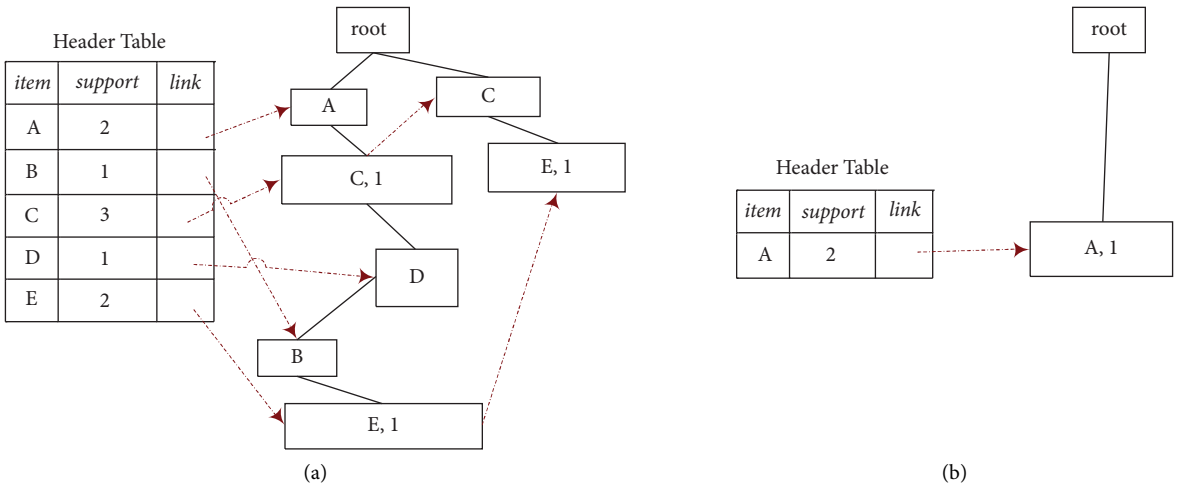

FIGURE 3: Continued.

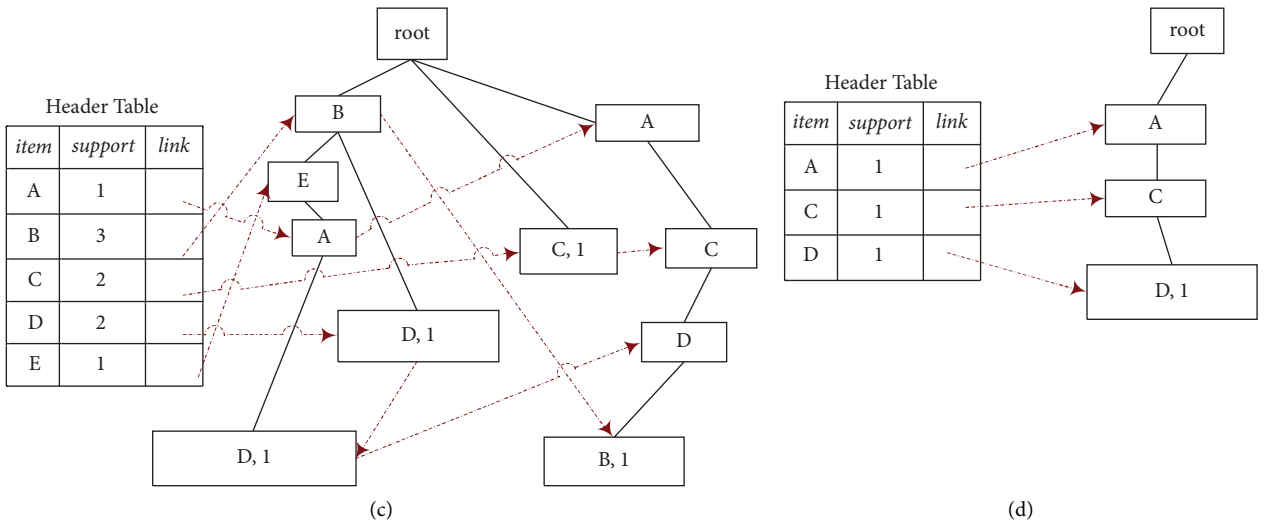

FIGURE 3: A case of creating a subtree and a header table. (a) Subtree of item "F," (b) the subtree of itemset <C and F>, (c) subtree of item "E," and (d) subtree of itemset <B and E>.

and the second order derivative is always less than 0, and Lemma 1 holds.

For any nuisance parameter π, to formula (3), $P(S(X), S_1(X), \pi) = B(n_1, S_1(X), \pi) * B(n_0, S_0(X), \pi)$ where $B(z, h, \pi) = \binom{z}{h}\pi^h(1-\pi)^{z-h}$, the maximal value of $B(n_1, S_1(X), \pi)$ is at $q_1 = \lfloor (n_1 + 1)\pi \rfloor = Q_1\pi$, and to $B(n_0, S_0(X), \pi)$ is at $q_0 = \lfloor (n_0 + 1)\pi \rfloor = Q_0\pi$, assuming $S_1(X) \le q_1$ and $S_0(X) \le q_0$ (the results for other situations are analogous), we can know that the p value of data points in the range of $(S_1(X), q_1)$ and $(S_0(X), q_0)$ is not less than the observed. Defining $D(q_1, q_0)$ as a minimal product of the data point number of each range, we can calculate the value of $D(q_1, q_0)$ by Lemma 2. □

Lemma 2. $D(q_0, q_1) = \min(|Q_0 S_1(X)/Q_1 - S_0(X)|, |Q_1 S_0(X)/Q_0 - S_1(X)|)$.

Proof. Assuming $S_1(X)/Q_1 >= S_0(X)/Q_0$,

$$
\begin{aligned}
D(q_0, q_1) &= \operatorname*{argmin}_{\pi} \left| (|Q_1\pi - S_1(X)| + 1)(Q_0\pi - S_0(X))| + 1) \right|, \\
(1)\pi &>= \frac{S_1(X)}{Q_1}, \\
D(q_0, q_1) &= Q_1Q_0\pi^2 - ((Q_0(S_1(X) - 1) + Q_1(S_0(X) - 1))\pi + (S_1(X) - 1)(S_0(X) - 1) - 1, \\
\operatorname*{argmin}_{\pi} D(q_0, q_1) &= Q_0\frac{S_1(X)}{Q_1} - S_0(X), \\
(2)\frac{S_0(X)}{Q_0} &<= \pi <= \frac{S_1(X)}{Q_1}, \\
D(q_0, q_1) &= -Q_1Q_0\pi^2 + (Q_0(S_1(X) + 1) + Q_1(S_0(X) - 1))\pi - (S_1(X) + 1)(S_0(X) - 1) - 1, \\
\operatorname*{argmin}_{\pi} D(q_0, q_1) &= \min\left(Q_0\frac{S_1(X)}{Q_1} - S_0(X), Q_1\frac{S_0(X)}{Q_0} - S_1(X) \right), \\
(3)\pi &<= \frac{S_0(X)}{Q_0}, \\
D(q_0, q_1) &= Q_1Q_0\pi^2 - (Q_0(S_1(X) + 1) + Q_1(S_0(X) + 1))\pi + (S_1(X) + 1)(S_0(X) + 1) - 1, \\
\operatorname*{argmin}_{\pi} D(q_0, q_1) &= Q_0\frac{S_1(X)}{Q_1} - S_0(X),
\end{aligned}
\tag{9}
$$

thus proving Lemma 2; the proposition is also tenable when $S_1(X)/Q_1 \le S_0(X)/Q_0$.

We propose an upper bound of the p value based on Lemmas 1 and 2, the upper bound of p value could be calculated in O(1) and it is proved in Lemma 3. This dramatically speeds up the p value calculation of the unconditional test. □

Lemma 3. $P_B(X) \le P(X(S), X_1(S), X/n)((n_0+1)(n_1+1) - D(q_1, q_0))$.

$$\begin{aligned} P_B(X) &= \max_\pi \left\{ \sum_{P(x,a,\pi) \le P(S(X), S_1(X), \pi)} P(x,a,\pi) \right\} \\ &\le \max_\pi \{ P(S(X), S_1(X), \pi) | P(x,a,\pi) | \} \le P(S(X), S_1(X), f(S))((n_0+1)(n_1+1) - D(q_1,q_2)). \\ &= P\left(\frac{S(X), S_1(X), X}{n} \right)((n_0+1)(n_1+1) - D(q_1,q_2)) \end{aligned} \tag{10}$$

4.2.2. Permutation-Based p Values. Similar to [10], suppose there are N patterns (including X) in the observed data window waiting for test, the p value of X is calculated as follows:

$$P - \text{value}(X) = \frac{\#(P_i < P_B) i \in \{1 \ldots ..., ND_p\}}{ND_p}, \tag{11}$$

where D_p is the number of permutation datasets, P_B is the test statistic value according to the tested pattern X on observed dataset, and P_i are test statistic values according to N patterns with permutated datasets; they are all calculated by Lemma 3.

Each permutation dataset matches a contingency table. So, looking for p values is transformed to look for contingency tables, the contingency table is decided by n_1 and $S_1(X)$, and the same contingency table produces the same p values; if permutation datasets are produced randomly, it will be very time-consuming. In fact, there are many permutation datasets with the same contingency tables, and we are going to produce permutation p values based on a combination strategy for improving the calculation efficiency by following two steps.

Step 1. (permutation datasets generation). Unlike [10], our permutation strategy produces a permuted dataset that does not change the pattern support and the length of each transaction. This strategy ensures that besides the label number, everything else is fixed. Suppose a sliding window W consists of 6 transactions in Figure 2(a), random two permutation datasets using this approach are shown in Figures 4(b) and 4(c). The transactions have the same order as them in W, and the support of the pattern is not changed; instead, the sequential patterns with the certain label are changed: S_0(<A and C>) is 2 in Figure 2(a) but it is 1 in the Figure 4(a). So, we will take n_1 in Table 1 as a variable quantity based on the calculation of Barnard's test statistic which is more in line with the process of data generations and collections.

Step 2. (datasets combination with the same test statistic). Literature [10] proposed the algorithm by producing permutation p values based on a combination strategy. We follow the strategy based on Barnard's test statistic. Different from [10], in our setting, n_1 is variable and it could be changed in (0, n). For a testing pattern X, set $L = \min(S(X), n_1)$ and $U = \max(0, (n_1 - (n - S(X))))$. $S_1(X)$ is in the range of (L, U); selecting x transactions from $S(X)$ transactions (contain X) and $S(X) - x$ transactions from $n - S(X)$ transactions (not contain X) could be known as follows:

$$r(x, n_1) = \binom{S(X)}{x} \binom{n - S(X)}{S(X) - x}. \tag{12}$$

Based on the contingency table, x takes the value in (L, U). Thus, according to the different value of n_1, the total number of the contingency tables is calculated as follows:

$$\sum_{n_1=0}^{n} \sum_{x=L}^{U} r(x, n_1). \tag{13}$$

Based on formulas (12) and (13), the p values of SSP candidates could be known in Algorithm 4. For each testing pattern (Line 2), the number of permutation datasets could be calculated, and the combination process could be known in Lines 3–13. The original test statistic value for each pattern in the observed data window could be calculated by Line 15, and to each test statistic value in list p by combination, it finds the number that is less than the observed one (Lines 16–20), and Lines 21-22 calculate each p value of testing pattern in the candidates and add it to the final results. Algorithm 4 can be considered an efficient algorithm to optimize the calculation of p values.

4.3. FSSPDS. We are going to draw significant sequential patterns in the data stream under the framework of FWER based on the p values by Algorithm 4. FSSPDS could be designed for m windows. W_1 is the first window that can reach W_m after $m - 1$ sliding. To data stream with m windows, the process of FSSPDS for returning SSPs in the data

TID	Transaction itemsets	Class
1	<A, C, F>	G^0
2	<B, D, E, A, G, D>	G^1
3	<B, E, A, D, E>	G^1
4	<C, E, F, G, B>	G^1
5	<B, D, B, C, D>	G^0
6	<A, C, D, B, E, F, C>	G^1

(a)

TID	Transaction itemsets	Class
1	<A, C, F>	G^1
2	<B, D, E, A, G, D>	G^0
3	<B, E, A, D, E>	G^1
4	<C, E, F, G, B>	G^0
5	<B, D, B, C, D>	G^0
6	<A, C, D, B, E, F, C>	G^1

(b)

TID	Transaction itemsets	Class
1	<A, C, F>	G^1
2	<B, D, E, A, G, D>	G^0
3	<B, E, A, D, E>	G^0
4	<C, E, F, G, B>	G^1
5	<B, D, B, C, D>	G^1
6	<A, C, D, B, E, F, C>	G^1

(c)

Figure 4: The permuted datasets.

Input: support threshold λ, test level α, and transaction data window W.
Output: p values of testing patterns.
(1) $n = |W|$, $P = R = ()$, total = 0, and number = 0
(2) For each FSSP candidate X do
(3) For $n_1 = 0$ to n do
(4) $L = \min(S(X), n_1)$
(5) $U = \max(0, (n_1 - (n - S(X))))$
(6) For $s = L$ to U do
(7) num = $r(s, n_1)$
(8) $p = P_B(X)$//by Lemma 3
(9) total+ = num
(10) P.add (<X, p, and num>)
(11) End for
(12) End for
(13) End for
(14) For each FSSP candidate X do
(15) $px = P_B(X)$
(16) For each item *Pitem* in P do
(17) If $Pitem.p < px$ then
(18) number+ = Pitem.num
(19) End If
(20) End for
(21) p value = number/total
(22) R.add (<X, p value>)
(23) End For
(24) Return R

Algorithm 4: p values.

stream under the control of FWER could be known in Algorithm 5. For each sliding window, SP_i in Line 3 returns SSP candidates by algorithm 1, and R_i in Line 4 outputs the p values by Algorithm 4. If its p value does not exceed the corrected test level, the pattern is added to the result set (Lines 5–12). Line 14 returns the final results.

5. Complexity Analysis

First, we study the computational complexity of producing SSP candidates. By existing algorithms in [1, 2], SSP candidates are obtained in time $O(|DS|^2 \cdot L \cdot M^2)$. DS is the dataset, L is the distinguish items in DS, and M is the maximum sequential length in the dataset. By introducing the length constraint k, the complexity is performed in time $O(|D|^2 \cdot L \cdot M \cdot k)$. It is known that k is not bigger than M, and often it is far smaller than H. The complexity of FSSPDS achieves a better result than traditional candidates mining methods. Of course, it is important to note that some effective SSP candidates with long lengths may be removed, but actually, we can know that there are few candidates which are lost when we choose a relatively bigger length constraint value in the experiments. Instead, in most cases, the mining algorithm with length constraint accelerates the completion of candidates discovering tasks and finding more SSP patterns.

Second, we study the complexity of test statistics with permutation p values. According to Lemma 3, the test statistic could be obtained in O(1); such an upper bound is

```
Input: data stream W, test threshold, minimum support threshold λ, and length k
Output: SSPs with FWER ≤ α.
(1) SSP = (), calculate testable support x_s by α
(2) For each sliding window W_i do
(3)    SP_i = SSPsCandidates (W_i, λ, x_s, and k)
(4)    R_i = p values (λ, α, W_i)
(5)    For each pattern X in SP_i do
(6)       Add X to SSP
(7)    End For
(8)    For each pattern X in SSP do
(9)       If R_i.X. p value > α/|SSP| then
(10)         Remove X from SSP
(11)      End If
(12)   End For
(13) End For
(14) Return SSP
```

Algorithm 5: FSSPDS.

efficient-to-compute. In fact, in [24], a close upper bound of Barnard's test statistic is established as $P_B(X) \le P(X(S), X_1(S), S(X)/n)$ $((n_1+1)\ (n_0+1))$ which can be used as an effective value to assess the test static value of the significant patterns. Our new upper bound which is proved in Lemma 3 is less than this effective value. It is more close to the exact test statistic value. Algorithm 4 produced permutation p values based on combination strategy; if permutation p values in a sliding window W are produced randomly and there are J FSSP candidates waiting for testing, there should be $2^{|W|}$ permutation datasets, and the complexity of such calculation should be acted as $O(J{\cdot}2^{|W|})$. However, by our efficient combination strategy, the complexity in Algorithm 4 is $O(J{\cdot}|W|{\cdot}(U-L) \le J{\cdot}|W|^2)$. It can be proved that $2^{|W|} > |W|^2$ by simple mathematical induction when $|W| \ge 4$. Thus, we can know that the running time by our permutation strategy could be greatly reduced and no significant pattern is lost.

6. Experiments

6.1. Environment and Dataset. The code used for the evaluation has been developed in Python, and the platform of the experiment is configured as follows: Windows 10 system, 2G Memory, and intel (R) Core(TM) i3-2310 CPU @2.10 GHz. We evaluate the performance of FSSPDS on six datasets, mushrooms, a2a are from libSVM [41]. T10I4D100K, bms-web2, retail, and bms-pos are obtained from SPMF [42], and they are labeled by [32]. They all have two classes. Table 2 shows the details of datasets, where |I| is the size of the alphabet, avgLength is the average length of the transactions, and |D| is the number of sequences. The window size is initialized with 100 transactions and the length $k = l \times MaxLength$, and $MaxLength$ is the maximal transaction length and l is initialized as 0.85.

6.2. Efficiency Evaluation

6.2.1. Candidates Evaluation. Firstly, we evaluate the performance for producing SPP candidates of the proposed algorithm FSSPDS compared with Miner-K [25] based on index tree and TSPIN [21] of vertical structure and CM-Spam in [2], and Figures 5(a)–5(f) show the running time, and Figures 6(a)–6(f) show the maximal memory consumption of the four algorithms under different minimum support threshold, respectively. From Figure 5, FSSPDS achieves the best time performance of the four algorithms. Miner-K takes less time than TSPIN and CM-Spam. FSSPDS spends significantly less time than the other three methods by benefiting from removing the insignificant patterns earlier. By the two requirements of testable support and length constraints, FSSPDS can always achieve better efficiency under different support thresholds. From Figure 6, we also come to the conclusion that FSSPDS can complete all tasks with less memory consumption.

Table 3 shows the number of candidates produced by the four algorithms. TSPIN and CM-Spam return exact number without length constraint and K-miner returns number with length constraints which is less than TSPIN and CM-Spam. The number returned by FSSPDS is the least of the four algorithms. For example, on Mushrooms when the threshold is 0.6, FSSP produces 616 candidates (the number of real SSPs is 535), but K-miner returns 786 candidates and the number by TSPIN and CM-Spam is 1332. The insignificant patterns are removed timely in the mining process and the testing number could be reduced. Overall, FSSPDS has obvious advantages in producing candidates with less number and running time, and FSSPDS can return SSPs smoothly.

6.2.2. Significance Evaluation. For evaluating the significance of FSSPDS, it is compared with four algorithms. The first is the recent advanced permutation p values producing method in [11] with Fisher's test statistic which is denoted as FSSPPROM; the second version that we denote FSSPEPAR is the one based on Fisher's test statistic with combined permutation p values strategy in [10]; the third, we call it PROMBD, is the one that uses upper bound of Barnard's test statistic by SPuManTE [9] with permutation p values producing method in [11]; and the last one FSSPDS*

Table 2: Data characteristics.

Dataset	$\|I\|$	avgLength	$\|D\|$	Class
Mushrooms	112	21.0	8124	2
a2a	108	13.87	2265	2
T10I4D100K	870	10.1	100000	2
Bms-pos	1656	6.5	515597	2
Retail	16470	10.3	88162	2
Bms-web2	330285	4.59	77158	2

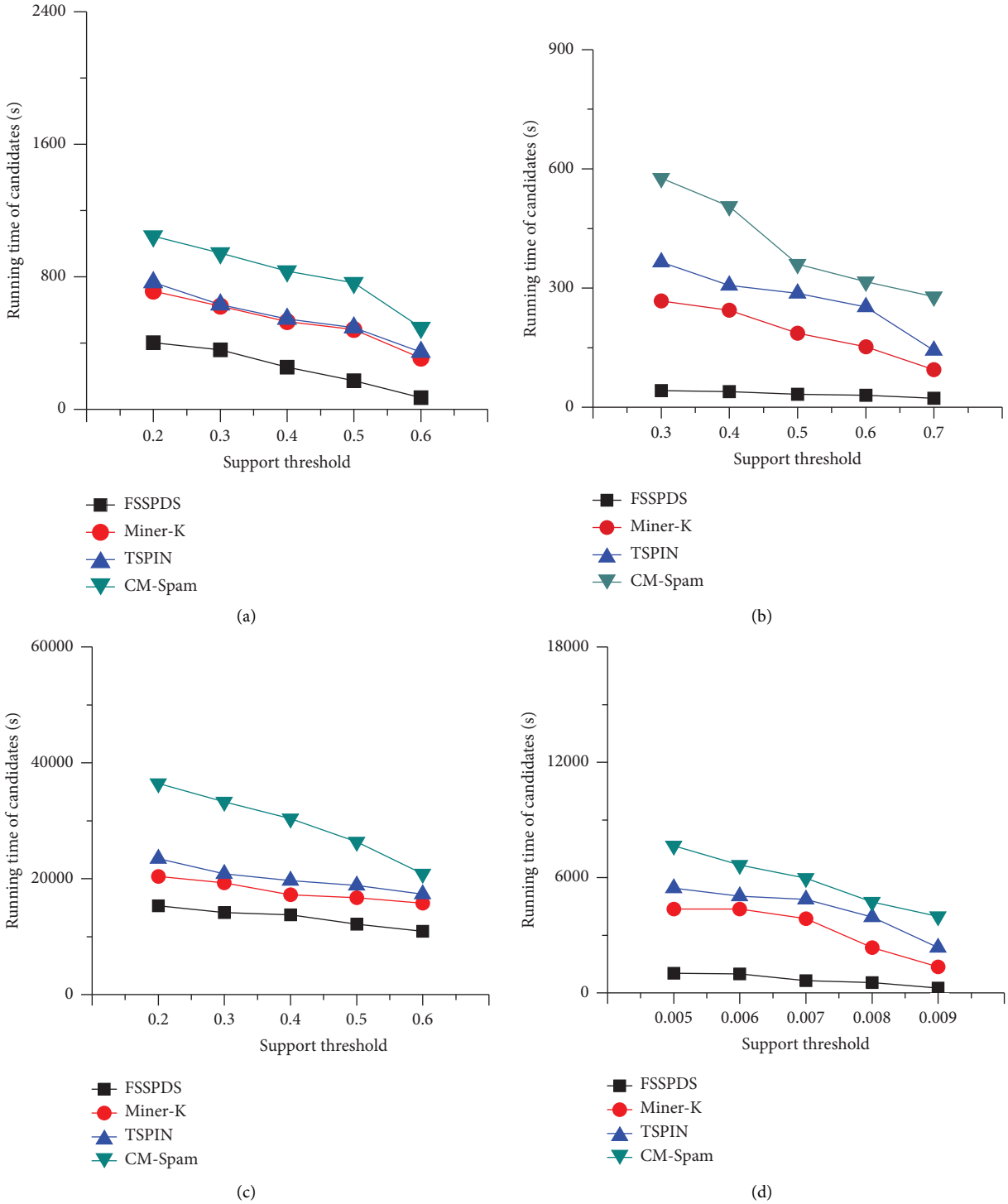

Figure 5: Continued.

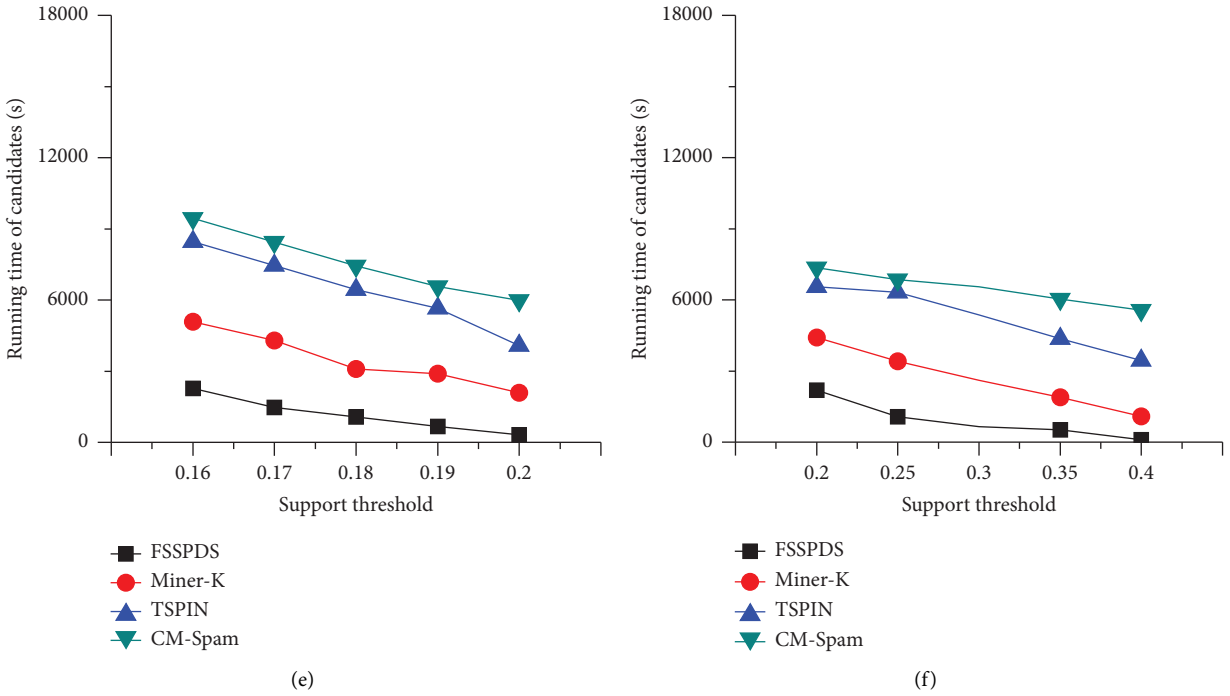

Figure 5: Running time of producing SSP candidates. (a) Mushrooms, (b) a2a, (c) bms-pos, (d) bms-web2, (e) retail, and (f) T10I4D100K.

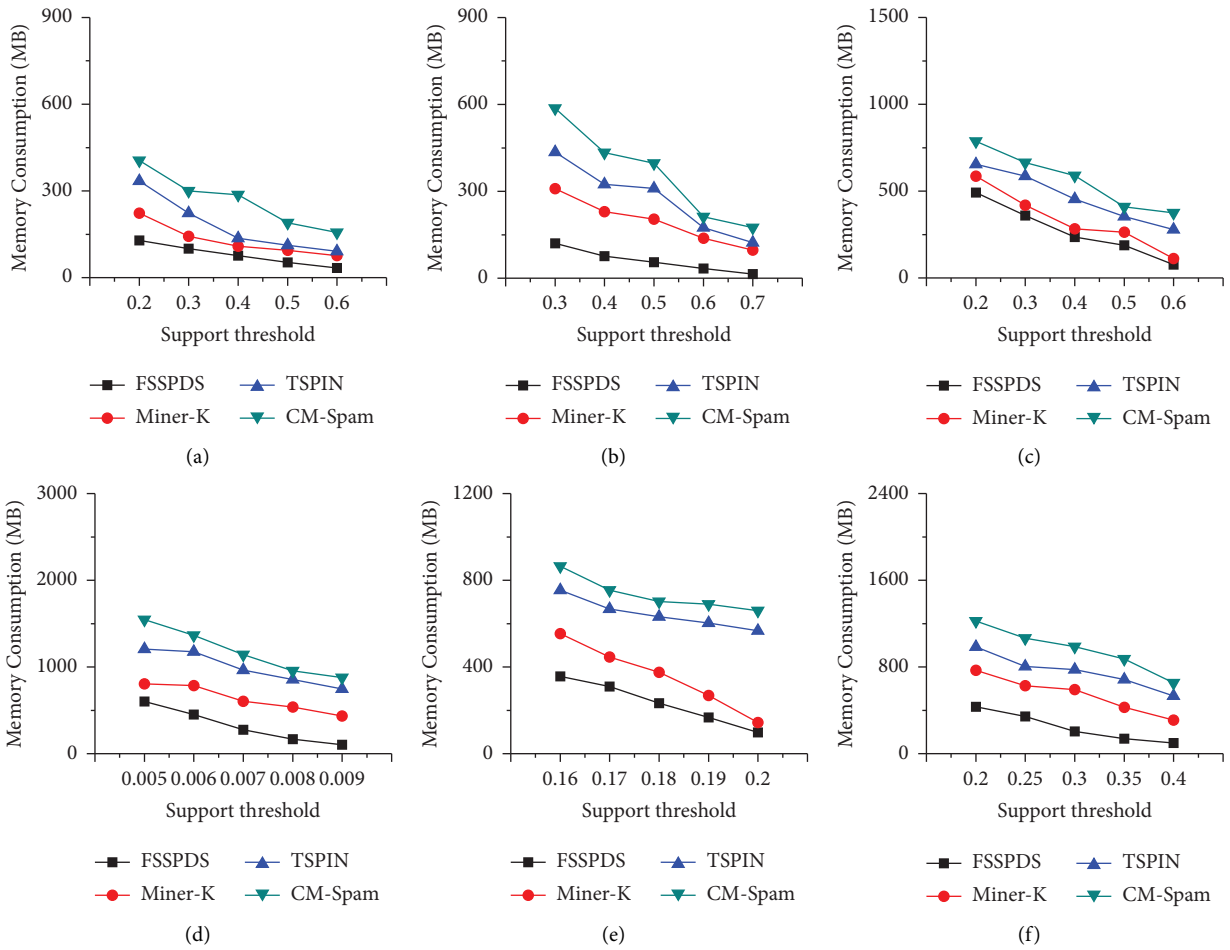

Figure 6: Memory consumption of producing SSP candidates. (a) Mushrooms, (b) a2a, (c) bms-pos, (d) bms-web2, (e) retail, and (f) T10I4D100K.

Table 3: Number of SSP candidates.

Datasets	Support threshold	SSP candidates by FSSPDS	SSPs by FSSPDS	SSP candidates by K-miner	SSP candidates by TSPIN/CM-spam
Mushrooms	0.2	1065	695	1234	1986
	0.3	954	643	1098	1765
	0.4	832	616	986	1564
	0.5	721	603	897	1456
	0.6	616	535	786	1332
a2a	0.3	154	98	187	415
	0.4	76	51	143	324
	0.5	43	29	109	254
	0.6	23	17	54	109
	0.7	19	13	43	87
bms-pos	0.2	235	175	313	56
	0.3	154	76	213	456
	0.4	46	15	109	315
	0.5	23	7	87	265
	0.6	7	1	13	67
bms-web2	0.005	109	65	187	324
	0.006	87	36	165	265
	0.007	76	22	154	187
	0.008	54	16	134	167
	0.009	21	16	109	154
Retail	0.16	209	165	365	546
	0.17	198	154	265	453
	0.18	187	113	208	412
	0.19	165	98	187	398
	0.2	109	56	145	276
T10I4D100K	0.2	1654	1225	1986	3245
	0.25	1567	999	1876	2986
	0.3	1432	885	1765	2098
	0.35	1398	576	1564	1986
	0.4	1098	304	1256	`1786

focusses on SSPs mining the same as FSSPDS but without the testable support and length constraints.

Figures 7(a)–7(f) show the running time on six datasets with different support thresholds. It shows that FSSPDS spent the shortest time on each dataset, benefiting from reducing search space by length and testable support constraints, and mining SSP candidates by FSSPDS can save a lot of time. Also, by using our effective combination strategy for permutation testing, the testing time is greatly reduced. For example, on a2a with a support threshold 0.3, FSSPDS spent 52.03125 seconds, while FSSPPROM, FSSPDS*, FSSPBD, and PSSPEPAR spent 667.53125 s, 465.4 s, 476.5 s, and 500 s, respectively. With threshold increases, the number of the SSPs decreases, so the running time became less with a bigger support threshold. However, from Figure 7, the running time of FSSPDS always have computational advantages. It can also show that the running time of FSSPDS* is less than other algorithms except for FSSPDS; this could confirm that by our effective candidates mining and testing process, the running time can be reduced even without considering length constraints. Additionally, from Figure 7, the time consumption of FSSPDS is relatively stable compared with other algorithms according to the increase of support thresholds.

Figures 8(a)–8(f) show the memory consumption of the algorithms on the six datasets. Algorithm FSSPDS reduces the number of items by using Theorems 1 and 2, and also, FSSPDS combines the same p values for calculation and thus the storage space could be reduced greatly. Benefiting from the pruning and combination strategies, it is known that the memory consumption of FSSPDS* is less than other algorithms except for FSSPDS. From Figure 8, FSSPDS is relatively stable compared with other algorithms and has a certain advantage in memory utilization.

As previously mentioned, the p values produced by these algorithms may be 0 which may result in a bad estimation. The percent of SSPs whose p values are zeros by the five algorithms could be seen from Table 4, it could be known that the proportion by FSSPDS is the smallest, and FSSPDS* is performed better than the other algorithms except for FSSPDS. We can get all p values based on formula (7) and FSSPDS achieves a better result than others. On mushrooms, Figure 9(a) shows p values distribution, and Figure 9(b) is the corresponding p values, the variance of p values could be known from Figure 9(c), the distribution by FSSPDS is more concentrated, and there are more patterns worth testing.

Figures 10(a)–10(f) show the pattern number comparison of the five algorithms on the six datasets. We can observe that the number by FSSPDS is significantly bigger

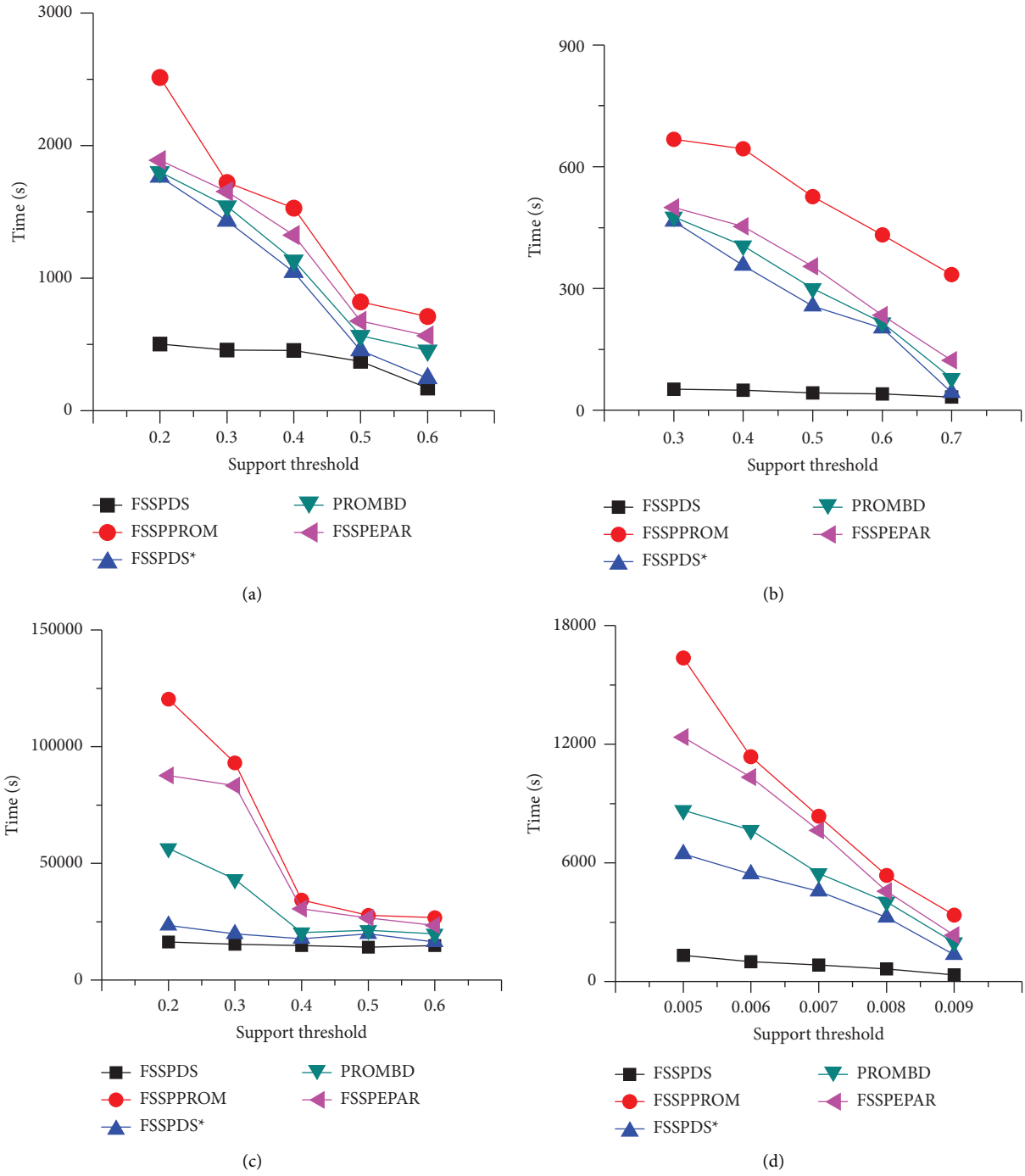

Figure 7: Continued.

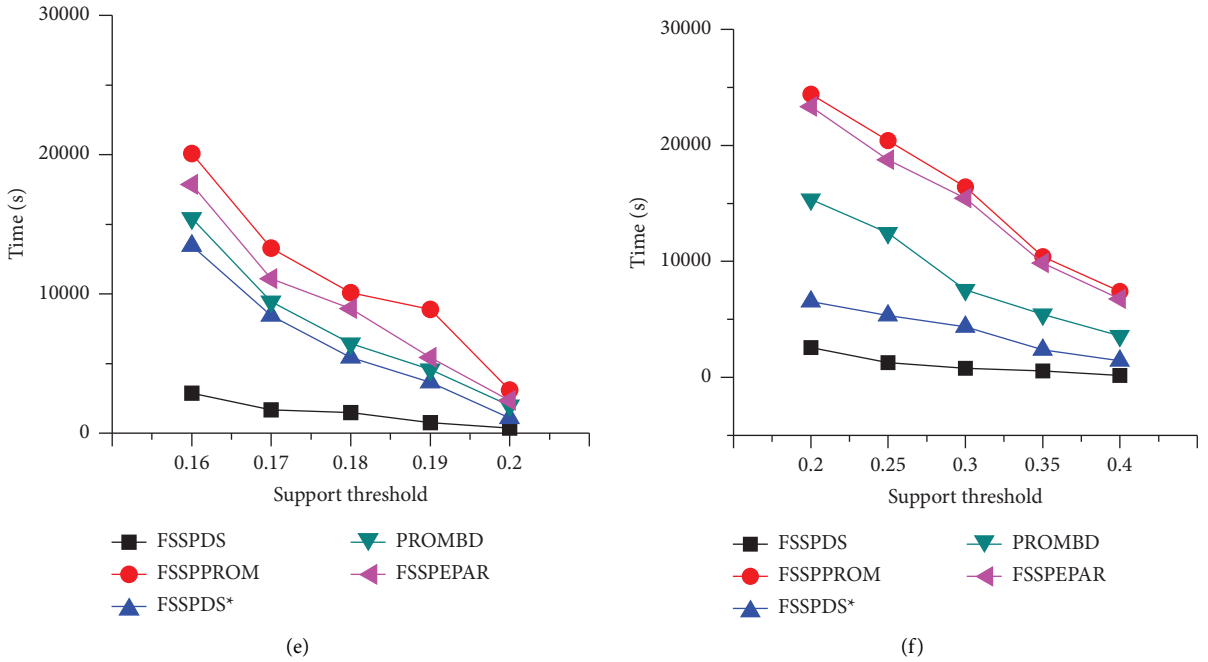

Figure 7: Running time of SSPs. (a) Mushrooms, (b) a2a, (c) bms-pos, (d) bms-web2, (e) retail, and (f) T10I4D100K.

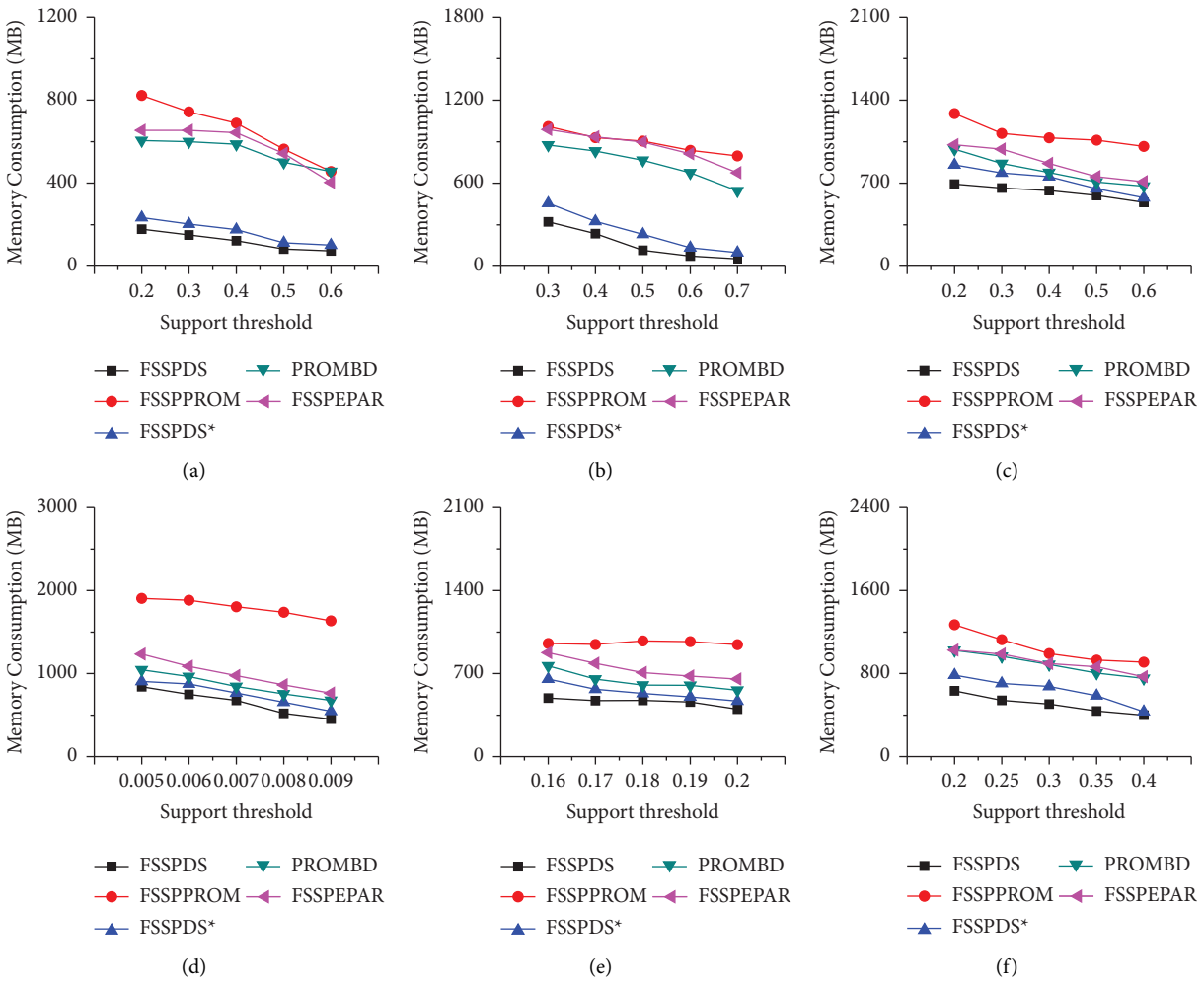

Figure 8: Memory consumption of SSPs. (a) Mushrooms, (b) a2a, (c) bms-pos, (d) bms-web2, (e) retail, and (f) T10I4D100K.

Table 4: Percent of SSPs whose p values are zeros.

Datasets	Support threshold	FSSPDS (%)	FSSPPROM (%)	FSSPDS* (%)	PROMBD (%)	FSSPEPAR (%)
Mushrooms	0.2	4.9	22.74	8.30	7.94	22.65
	0.3	4.98	18.45	7.40	7.62	21.49
	0.4	2.24	16.33	4.15	6.22	21.14
	0.5	2.32	14.29	3.60	6.14	18.84
	0.6	0.8	9.91	3.26	4.63	18.26
a2a	0.3	6.12	34.04	7.41	11.54	54.55
	0.4	6.41	26.19	11.63	12.50	66.67
	0.5	11.63	28.00	20.83	20.59	70.00
	0.6	18.52	29.41	35.71	53.85	71.43
	0.7	23.08	42.86	33.33	62.50	60.00
bms-pos	0.2	8.50	23.49	11.11	25.19	30.91
	0.3	10.81	63.27	22.45	25.58	41.07
	0.4	12.50	50.00	35.71	43.48	61.76
	0.5	33.33	40.00	25.00	50.00	70.00
	0.6	40.00	100.00	100.00	100.00	100.00
bms-web2	0.005	7.69	28.95	19.57	21.57	51.22
	0.006	8.89	30.00	26.09	40.74	54.84
	0.007	8.82	100.00	30.00	33.33	66.67
	0.008	15.38	100.00	23.65	33.33	66.67
	0.009	12.50	100.00	100.00	100.00	100.00
Retail	0.16	24.05	74.23	34.03	77.01	48.28
	0.17	22.84	65.41	42.04	85.53	50.39
	0.18	19.16	69.07	47.87	80.36	55.67
	0.19	28.04	64.29	38.46	69.77	44.12
	0.2	38.96	66.15	56.60	73.53	100.00
T10I4D100K	0.2	17.45	36.78	19.58	25.65	43.96
	0.25	18.87	34.91	21.16	28.14	39.12
	0.3	22.56	39.81	26.22	35.15	44.07
	0.35	25.08	77.59	30.04	47.25	64.44
	0.4	27.70	88.63	37.39	50.84	72.89

than the other algorithms. For example, on dataset retail when the support threshold is 0.16, FSSPDS produces 237 patterns, but the number by FSSPPROM, FSSPDS*, PROMBD, and FSSPEPAR is 163, 124, 87, and 145, respectively. FSSPDS can always mine more patterns of all algorithms. On most of the datasets, the number returned by FSSPDS* is close to FSSPDS and more than the other algorithms by benefiting from the calculation advantage of FSSPDS.

We also analyze the impact of length parameters l on the number of SSPs. The number of SSPs with different length parameters by FSSPDS could be known from Table 5. It could see that when l is more than 0.8, the number returned by FSSPDS is bigger than the number returned by FSSPDS* and Figure 11 shows the ratio of the two returned numbers with different length parameters. It is known that when l increases, the value of the ratio is bigger than 1 which means that we can obtain more SSPs by length constraint than without constraint. Such conclusion could be explained that when we make length constraint on the datasets, insignificant patterns are removed timely in the mining process; thus, more significant patterns could be drawn under the calculation rule of FWER.

6.2.3. Scalability Evaluation. In order to test the scalability of the FSSPDS, we evaluate its efficiency according to the different window sizes. We assess the performance under varied size. With $\lambda = 0.2$, Figure 12(a) shows the running time on mushrooms, Figure 12(b) shows the memory consumption on T10I4D100K, and Figure 12(c) shows the number of tested patterns whose p values are 0 on bms-pos. According to the results of our calculations on all these six datasets, it can be known that FSSPDS can achieve a better result for all five algorithms in terms of running time, memory consumption, and effective p values. It can mine SSPs smoothly under different data sizes.

Figure 13 shows the number of SSPs with different length parameters and window sizes, Figure 13(a) returns the number of SSPs with $\lambda = 0.16$ and $l = 0.7$ on retail, Figure 13(b) returns the number of SSPs with $\lambda = 0.005$ and $l = 0.8$ on bms-web2, and Figure 13(c) returns the number of

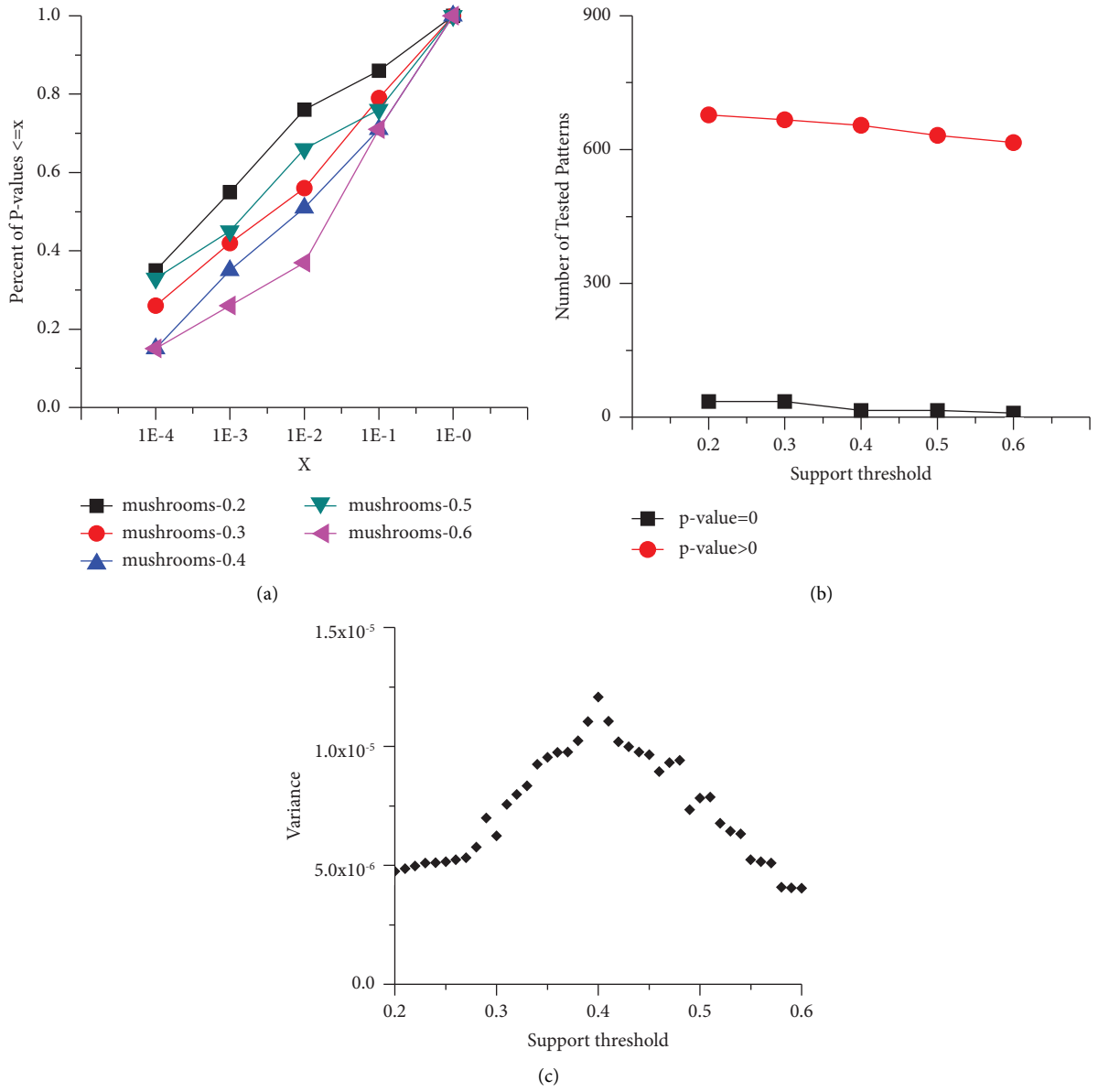

Figure 9: p values of the tested patterns by FSSPDS. (a) Distribution, (b) p value $= 0$ and p value > 0, and (c) variance.

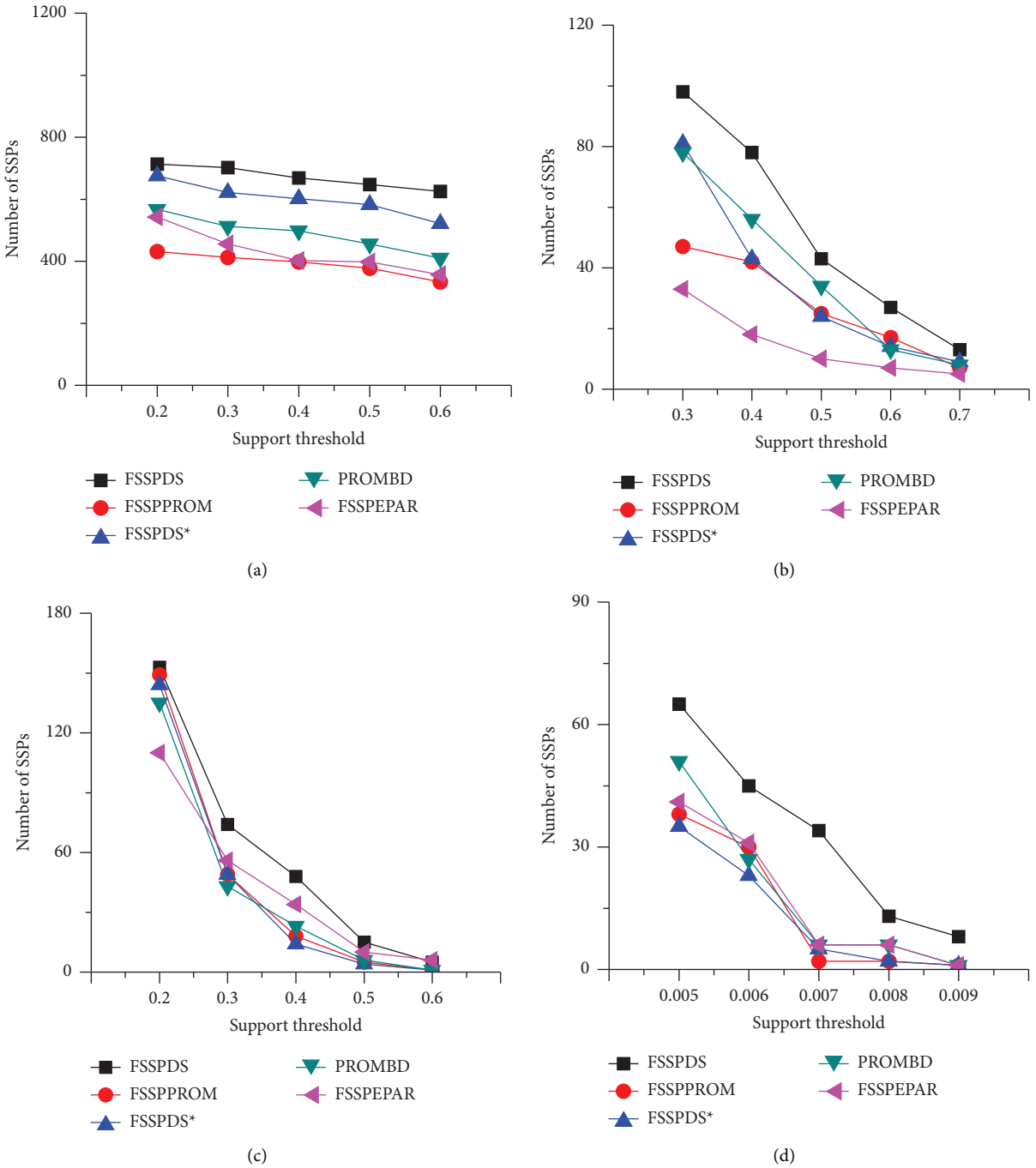

Figure 10: Continued.

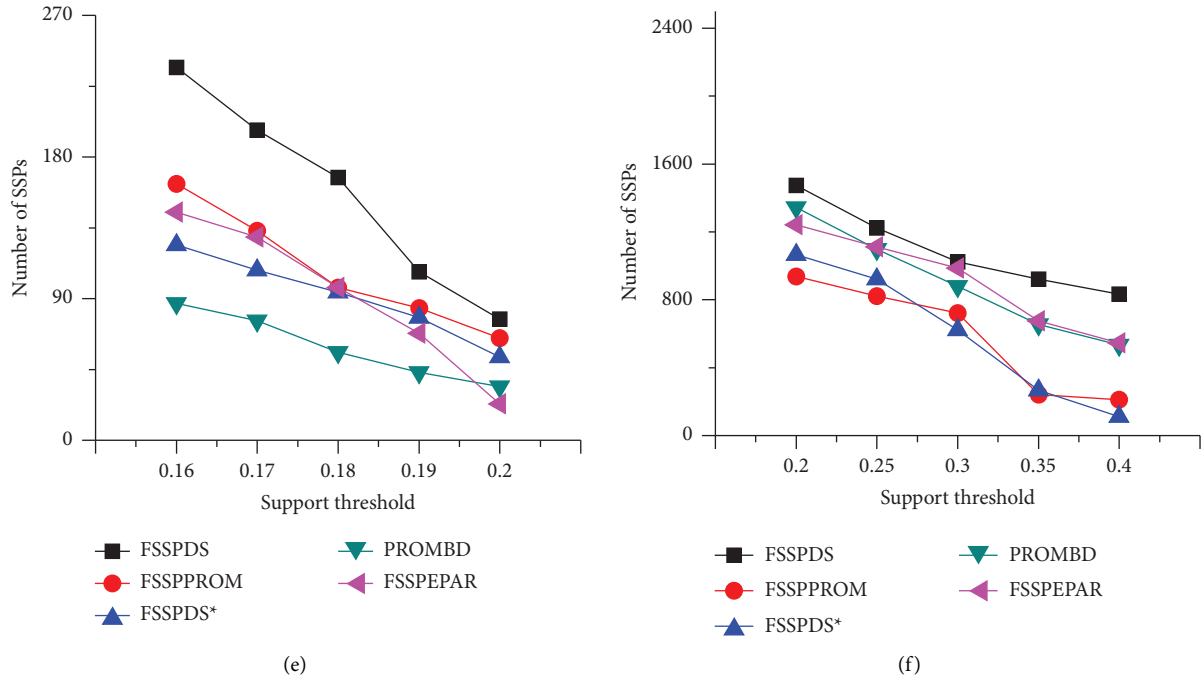

Figure 10: Number of SSPs. (a) Mushrooms, (b) a2a, (c) bms-pos, (d) bms-web2, (e) retail, (f) T10I4D100K.

Table 5: Number of SSPs with different length parameters.

Datasets	Support threshold	FSSPDS ($l = 0.7$)	FSSPDS ($l = 0.8$)	FSSPDS ($l = 0.9$)	FSSPDS*
Mushrooms	0.2	567	685	705	675
	0.3	532	632	654	622
	0.4	532	616	616	602
	0.5	512	598	609	583
	0.6	491	535	535	521
a2a	0.3	65	87	123	81
	0.4	32	45	56	43
	0.5	18	28	32	24
	0.6	12	16	21	14
	0.7	7	12	15	9
bms-pos	0.2	112	156	234	144
	0.3	43	56	78	49
	0.4	7	15	18	14
	0.5	6	7	7	4
	0.6	1	1	1	1
bms-web2	0.005	23	54	67	35
	0.006	12	25	39	23
	0.007	8	17	25	15
	0.008	8	12	18	12
	0.009	8	12	18	11
Retail	0.16	54	156	231	124
	0.17	34	112	212	108
	0.18	19	98	176	94
	0.19	19	98	98	78
	0.2	7	56	56	53
T10I4D100K	0.2	564	1125	1543	1065
	0.25	435	987	1098	921
	0.3	347	865	954	621
	0.35	267	568	764	267
	0.4	198	246	543	111

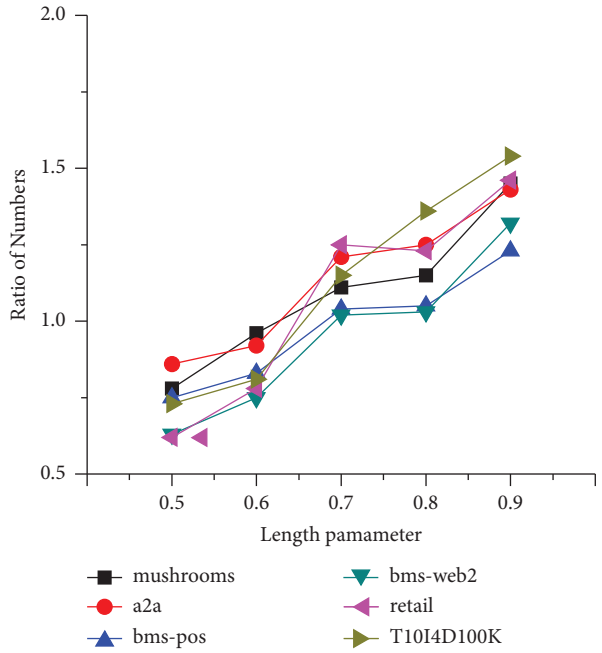

Figure 11: Ratio of numbers.

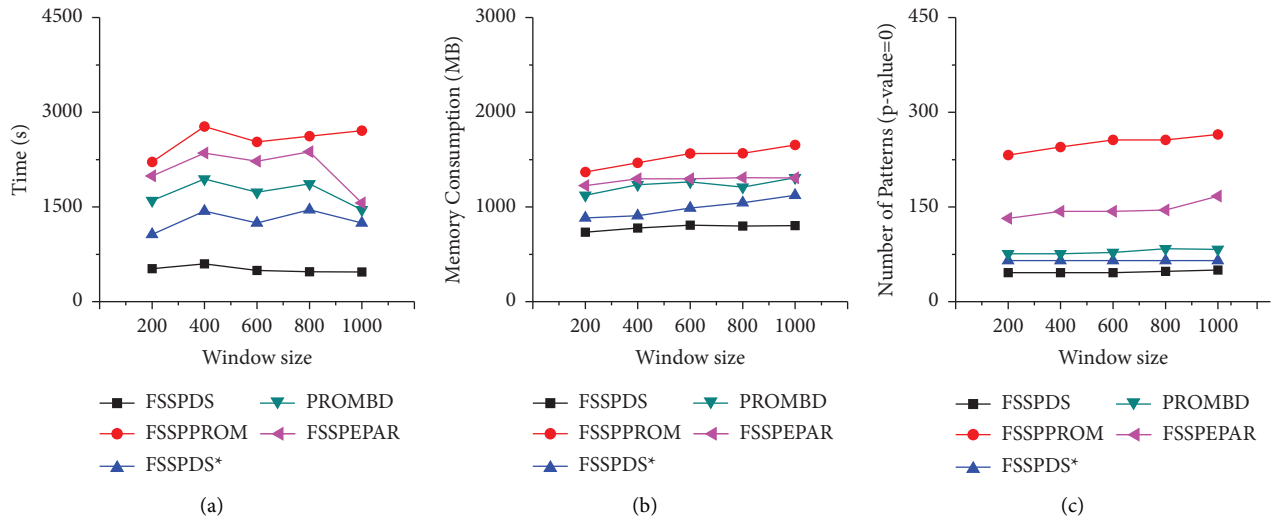

Figure 12: Efficiency with different window sizes. (a) Mushrooms, (b) T10I4D100K, and (c) bms-pos.

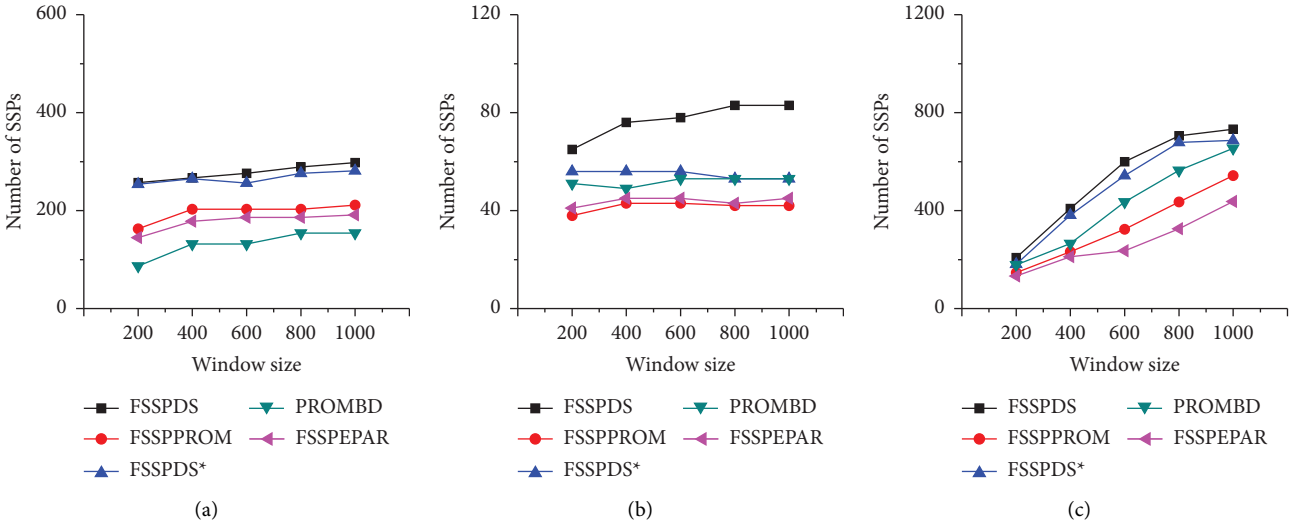

Figure 13: Number of SSPs with different sizes. (a) Retail ($l = 0.7$), (b) bms-web2 ($l = 0.8$), and (c) a2a($l = 0.9$).

SSPs with $\lambda = 0.3$ and $l = 0.9$ on a2a. It can get the same conclusions that pattern numbers by FSSPDS are higher than the other algorithms. FSSPDS is relatively stable and it has little effect on varied window size.

7. Conclusion and Feature Work

We introduce FSSPDS, an efficient algorithm to mine statistically significant sequential patterns in the data stream. Firstly, insignificant candidates can be removed timely by introducing testable supports and length constraints with a pattern-growth method based on the tree structure. According to better reflect the data generation process in the time sliding window, different from the traditional Fisher's test statistic, we mine SSPs based on unconditional test statistics with permutation p values under the framework of FWER. To overcome the computation drawbacks of p values production, we proposed a close upper bound of the unconditional test statistic and used a combination strategy to look for effective permutation p values. The experimental results on real datasets demonstrate the effectiveness of FSSPDS.

FSSPDS is still time-consuming on some datasets, especially on dense datasets where the length of each transaction is related long, and the permutation process spent lots of time. Some useful pruning strategies could be used for accelerating the calculation process. For example, the boundary pruning and static buffering technology in [10] could be used for reducing the computational cost. The continuous computation technology could be used in formula (8) as $r(S_1(X), n_1 + 1) = r(S_1(X), n_1)(n_1 + 1)^2/(n_1 - s + 1)(n - n_1)$. Additionally, the data size is fixed in our experiments, efficiency evaluation on datasets with unfixed size could be studied, and we will investigate these problems in our future work.

Data Availability

The data used to support the findings of this study are included within the article.

Conflicts of Interest

The authors declare that they have no conflicts of interest to report regarding the present study.

Acknowledgments

This work was supported by the National Social Science Foundation of China (grant no. 21BGL088) and Project of Zhejiang Provincial Public Welfare Technology Application and Research (grant no. LGF21H180010).

References

[1] P. Fournier-Viger, J. C.-W. Lin, and R. Uday Kiran, "A survey of sequential pattern mining," *Data Science and Pattern Recognition*, vol. 1, no. 1, pp. 54–67, 2017.

[2] P. Fournier-Viger, A. Gomariz, M. Campos, and R. Thomas, "Fast vertical mining of sequential patterns using Co-occurrence information," in *Proceedings of the PACKDDM*, pp. 40–52, Sydney, Australia, April 2014.

[3] W. Gan, J. C.-W. Lin, J. Zhang, H. C. Chao, and P. S. Yu, "Fast utility mining on sequence data," *IEEE Transactions on Cybernetics*, vol. 51, no. 2, pp. 487–500, 2021.

[4] T. Guyet and R. Quiniou, "NegPSpan: efficient extraction of negative sequential patterns with embedding constraints," *Data Mining and Knowledge Discovery*, vol. 34, no. 2, pp. 563–609, 2020.

[5] W. Hämäläinen and G. I. Webb, "A tutorial on statistically sound pattern discovery," *Data Mining and Knowledge Discovery*, vol. 33, no. 2, pp. 325–377, 2019.

[6] N. Das and S. K. Bhandari, "Bound on FWER for correlated normal," *Statistics & Probability Letters*, vol. 168, Article ID 108943, 2021.

[7] L. De Jong, W. Roseboom, and G. Kramer, "A composite filter for low FDR of protein-protein interactions detected by in vivo cross-linking," *Journal of Proteomics*, vol. 230, Article ID 103987, 2021.

[8] A. Terada, M. Okada-Hatakeyama, K. Tsuda, and J."Statistical significance of combinatorial regulations," *Statistical of the National Academy of Sciences*, vol. 110, no. 32, pp. 12996–13001, 2013.

[9] L. Pellegrina, M. Riondato, and F. Vandin, "Spumante: significant Pattern Mining with Unconditional Testing," in *S. I. G. K. D. D., Anchorage, A. K.*pp. 1528–1538, USA, 2019.

[10] J. Wu, Z. He, F. Gu, J. Zhou, and C. Yang, "Computing exact permutation p-values for association rules," *Information Sciences*, vol. 346-347, pp. 146–162, 2016.

[11] A. Tonon and F. Vandin, "Permutation Strategies for Mining Significant Sequential Patterns," in *Proceedings of the ICDM*, pp. 1330–1336, Beijing, China, May 2019.

[12] E. Chung and J. P. Romano, "Exact and asymptotically robust permutation tests," *Annals of Statistics*, vol. 41, no. 2, pp. 484–507, 2013.

[13] W. Hämäläinen, "New upper bounds for tight and fast approximation of Fisher's exact test in dependency rule mining," *Computational Statistics & Data Analysis*, vol. 93, pp. 469–482, 2016.

[14] B. Vo, T. P. Hong, and B. Le, "DBV-Miner: a Dynamic Bit-Vector approach for fast mining frequent closed itemsets," *Expert Systems with Applications*, vol. 39, no. 8, pp. 7196–7206, 2012.

[15] J. Pei, J. Han, and B. Mortazavi-Asl, "Mining sequential patterns by pattern-growth: the prefixspan approach," *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, no. 11, pp. 1424–1440, 2004.

[16] Z. Yang and M. Kitsuregawa, "LAPIN-SPAM: an improved algorithm for mining sequential pattern," *Proc ICDE Workshops*, p. 1222, 2005.

[17] I. Fister, "Information cartography in association rule mining," *IEEE transactions on emerging topics in computational intelligence*, vol. 6, no. 3, pp. 660–676, 2022.

[18] K. H. T. Dam, T. Given-Wilson, and A. Legay, "Packer classification based on association rule mining," *Applied Soft Computing*, vol. 127, Article ID 109373, 2022.

[19] J. C. W. Lin, L. Yang, P. Fournier-Viger, T. P. Hong, L. S. L. Wang, and J. Zhan, "Mining high-utility itemsets based on particle swarm optimization," *Engineering Applications of Artificial Intelligence*, vol. 55, pp. 320–330, 2016.

[20] W. Song, W. Ye, and P. Fournier-Viger, "Mining sequential patterns with flexible constraints from MOOC data," *Applied Intelligence*, vol. 52, no. 14, pp. 16458–16474, 2022.

[21] P. Fournier-Viger, Y. Wang, P. Yang, U. Yun, and R. U. Kiran, "TSPIN: mining top-k stable periodic patterns," *Applied Intelligence*, vol. 52, no. 6, pp. 6917–6938, 2021.
[22] J. N. Xing and X. B. Ma, "DP-gSpan: a pattern growth-based differentially private frequent sub graph mining," in *Proceedings of the TRUSTCOM*, pp. 397–404, Shenyang, China, 2021.
[23] B. Rao and S. Mishra, "An approach to detect patterns (Subgraphs) with edge weight in graph using graph mining techniques," *Computational Intelligence in Data Mining*, vol. 711, pp. 807–817, 2021.
[24] N. Aryabarzan and B. Minaei-Bidgoli, "NEclat-Closed: a vertical algorithm for mining frequent closed itemsets," *Expert Systems with Applications*, vol. 174, Article ID 114738, 2021.
[25] Le Wang, "An algorithm for mining fixed-length high utility itemsets," in *Lecture Notes in Computer Science*, pp. 1–16, Springer, Cham, 2022.
[26] D. Conklin, "Mining contour sequences for significant closed patterns," *Journal of Mathematics and Music*, vol. 15, no. 2, pp. 112–124, 2021.
[27] C. Zhao, D. Liu, B. Teng, and Z. He, "BagReg: protein inference through machine learning," *Computational Biology and Chemistry*, vol. 57, no. 12, pp. 12–20, 2015.
[28] M. Ryu, G. Lee, and K. Lee, "Online sequential extreme studentized deviate tests for anomaly detection in streaming data with varying patterns," *Cluster Computing*, vol. 24, no. 3, pp. 1975–1987, 2021.
[29] Z. Y. He, H. Liang, Z. Chen, C. Zhao, and Y. Liu, "Computing exact P-values for community detection," *Data Mining and Knowledge Discovery*, vol. 34, no. 3, pp. 833–869, 2020.
[30] H. Wilhelmiinaand, "Kingfisher: an efficient algorithm for searching for both positive and negative dependency rules with statistical significance measures," *Knowledge and Information Systems*, vol. 32, no. 2, pp. 383–414, 2002.
[31] G. I. Webb, "Layered critical values: a powerful direct-adjustment approach to discovering significant patterns," *Machine Learning*, vol. 71, no. 2-3, pp. 307–323, 2008.
[32] C. Low-Kam, C. Raïssi, M. Kaytoue, and J. Pei, "Mining statistically significant sequential patterns," in *Proceedings of the ICDM*, pp. 488–497, Texas, TX, USA, February 2013.
[33] T. Jiang, T. Chen, X. X. Meng, and M. Xiao, "A new method for calculating p-value under unconditional exact test," *Thermal Science*, vol. 25, no. 3, pp. 2385–2397, 2021.
[34] F. Llinares-L_opez and M. Sugiyama, "Fast and Memory-Efficient Significant Pattern Mining via Permutation Testing," in *Proceedings of the SIGKDD*, pp. 725–734, New York, USA, August 2015.
[35] L. Pellegrina and F. Vandin, "Efficient Mining of the Most Significant Patterns with Permutation Testing," in *Proceedings of the SIGKDD*, pp. 2070–2080, London, England, December 2018.
[36] M. Riondato and E. Upfal, "Mining frequent itemsets through progressive sampling with rademacher averages," in *Proceedings of the SIGKDD*, pp. 1005–1014, Sydney, Australia, June 2015.
[37] T. Q. Tran, K. Fukuchi, and Y. Akimoto, "Statistically significant pattern mining with ordinal utility," *Proc. SIGKDD, Virtual Event, CA*, pp. 1645–1655, USA, 2020.
[38] M. Zihayat, H. Davoudi, and A. An, "Mining significant high utility gene regulation sequential patterns," *BMC Systems Biology*, vol. 11, no. S6, p. 109, 2017.
[39] P. Fournier-Viger, C. Cheng, Z. Cheng, and J. C. W. Lin, "Mining significant trend sequences in dynamic attributed graphs," *Knowledge-Based Systems*, vol. 182, Article ID 104797, 2019.
[40] S. Cheng, D. Yang, Y. Tong, H. Zhang, and B. Cui, "LTC: a fast algorithm to accurately find significant items in data streams," *IEEE Transactions on Knowledge and Data Engineering*, vol. 99, pp. 1–15, 2020.
[41] C.-C. Chang and C.-J. Lin, "LIBSVM: a library for support vector machines," *ACM Transactions on Intelligent Systems and Technology*, vol. 2, no. 3, pp. 1–27, 2011.
[42] P. Fournier-Viger and A. Gomariz, "Spmf: a java open source pattern mining library," *Journal of Machine Learning Research*, vol. 15, pp. 3389–3393, 2014.