Contents lists available at ScienceDirect

Expert Systems with Applications

journal homepage: www.elsevier.com/locate/eswa

Using positional sequence patterns to estimate the selectivity of SQL LIKE queries

Mehmet Aytimur[a], Ali Cakmak[b,*]

[a] *Department of Computer Science, Istanbul Sehir University, Turkey*
[b] *Department of Computer Engineering, Istanbul Technical University, Istanbul, Turkey*

A R T I C L E   I N F O

Article history:
Received 3 March 2019
Revised 12 July 2020
Accepted 13 July 2020
Available online 19 July 2020

Keywords:
Selectivity estimation
Histograms
Data management
Sequence pattern mining
Information content

A B S T R A C T

Sequence patterns are frequently employed in many expert system applications in a wide range of domains from bioinformatics to smart homes and stock market analysis. Regular sequence patterns fail to express whether two consecutive items in a pattern are occurring right after each other in all pattern occurrences in an item database or not. Such a differentiation may be important for many intelligent system applications, for instance, to better address business questions like "should two frequently-bought-together items be located right next to each other on a retail store shelf, or is it ok to place them at some distance as long as they are in the same aisle?". In this paper, we propose a novel type of sequence pattern, called "positional sequence patterns", and illustrate its application on a special expert system, i.e., the query planner/optimizer of a database management system. Positional sequence patterns allow to accommodate extra information regarding whether a frequent ordered item pair always occurs next to each other without any gap in between in all pattern occurrences. Since positional sequence patterns are not considered by the existing sequence pattern mining algorithms, we also propose an algorithm to mine them. Next, we integrate the positional sequence patterns into the selectivity estimation component of the query optimizer as an expert system application. More specifically, in the knowledgebase of the query optimizer, a histogram-like structure of positional sequence patterns are created and stored. Then, during query optimization time, these histograms are utilized to infer the selectivity of flexible text queries that are enabled by the SQL LIKE operator. In particular, the proposed selectivity estimation method employs redundant pattern elimination based on pattern information content during histogram construction, and a partitioning-based matching scheme. The experimental results on a real dataset from DBLP show that the proposed approach outperforms the state of the art by around 20% improvement in error rates.

1. Introduction

Frequently occurring orders of items and events, called *sequence patterns*, are of interest for many expert and intelligent system applications (Fournier-Viger et al., 2017; Pokou et al., 2016; Schweizer et al., 2015). There exists a number of sequence mining algorithms (Le et al., 2017; Pei et al., 2004; Pasquier, Bastide, Taouil, & Lakhal, 1999; Wang, Han, & Pei, 2003; Wang & Han, 2004; Yan, Han, & Afshar, 2003; Zaki and Hsiao, 2000). Some of these works generate all possible patterns, while others eliminate "redundant patterns", and generate only so-called "closed sequence patterns". However, the generated regular sequence patterns do not differentiate between whether two consecutive items in a pattern are always occurring right after each other with no other items in between or not. As an example, a given character sequence pattern "A B C" may mean that in the corresponding database, "A" and "B" may always occur together as "AB" with no other characters in between, or there may be a long and varying sequence of other characters between "A" and "B" in different occurrences of the pattern. Such a differentiation may be important for some expert and intelligent system applications. As an example expert system application, in this paper, we consider the query optimizer of a database management system. In a database management system, the query optimizer is responsible for generating the most efficient query execution plan by considering

* Corresponding author at: İTÜ Ayazağa Kampüsü Bilgisayar ve Bilişim Fakültesi, 34467 Sarıyer, İstanbul, Turkey.

E-mail addresses: mehmetaytimur@std.sehir.edu.tr (M. Aytimur), ali.cakmak@gmail.com (A. Cakmak).

https://doi.org/10.1016/j.eswa.2020.113762
0957-4174/© 2020 Elsevier Ltd. All rights reserved.

Table 1
Non-matching query statistics for each minsup value.

Minsup	1%		1.5%		2%	
Query Group	1st group	2nd group	1st group	2nd group	1st group	2nd group
Number of non-matching queries	42	25	49	25	52	30
Average row count for non-matching queries	4288	9978	5657	8448	5645	10,801

different access paths, join orders, join methods, etc. In this context, the query optimizer may itself be considered as an expert system with a *knowledgebase* that stores different query transformation rules as well as various forms of database statistics. Using this knowledgebase, the query optimizer employs its built-in intelligence capabilities to automatically construct the execution plan with the lowest computing requirements. A critical task of the optimizer while deciding on the most optimal query plan is to estimate the selectivity of each predicate in a query. The selectivity of a predicate *p* indicates the fraction of database rows that would be retained after applying *p* as a filter on the rows. The optimizer estimates a predicate's selectivity usually based on statistics gathered from the database and stored in its knowledgebase (i.e., the database catalog) before query time (i.e., offline). With number-typed data, the query optimizers usually come up with sufficiently accurate predictions owing to vast past research. On the other hand, for predicates on text-based data, it is still a challenge for query optimizers to come up with reliably accurate selectivity estimates. The main reason for this disparity is that predicates on text-based data often contain flexible patterns that may match a wide array of rows. In contrast, predicates on numeric data are strict and well-formed. The particular need for flexible predicates on text data results from the fact that textual data is frequently not clean with many misspellings and typographical errors. Especially with the explosion of internet resources, the "dirtiness" of textual data is even more evident. Furthermore, due to the nature of the text data, shared character sequences (e.g., prefixes, suffixes, etc.) often determine a particular class of data rows that database users may often be interested in. SQL provides the LIKE operator to allow formulating wildcard predicates on text data. For instance, the predicate, *name LIKE 'Luck%'*, returns all the names that start with 'Luck'. Accurately predicting the selectivity of such flexible predicates on text data is still an open research problem.

There are several works (Jagadish, Ng, & Srivastava, 1999; Jagadish, Kapitskaia, Ng, & Srivastava, 2000; Krishnan, Vitter, & Iyer, 1996; Lee, Ng, & Shim, 2009, Aytimur & Çakmak, 2018) in the literature which study the selectivity estimation of the wildcard predicates. The majority of past studies often assume that the predicates are in the form of $\%w\%$ where w represents several characters, and they perform poorly (Lee et al., 2009) for more generic queries that are in the form of $\%s_1\%s_2\%...\%s_n\%$ where s_i represents one or more characters. Besides, the main characteristic of these techniques is that they process all or a large set of row ids in the database during selectivity prediction. Processing such large data for each query during optimization time (i.e., online) suffers from memory and execution time overhead.

We recently proposed an algorithm, called SPH, to predict the selectivity of LIKE query predicates (Aytimur & Çakmak, 2018) using regular sequence patterns. In SPH approach, first, frequent sequence patterns are computed from the database before query time. Then, a histogram structure is built out of the discovered patterns. In order to estimate the selectivity of a given LIKE predicate, the precomputed histogram is exploited during query optimization time. While SPH works reasonably well in many settings, it often overestimates the selectivity of LIKE predicates that are in the form of $\%s_1s_2\%\ s_3\%s_4s_5\%...\%s_n\%$ where each s_i represents a single character. We give an example.

Example:. Let "%A%B%C%D%A%" be a sequence pattern stored in a histogram with frequency 10. If LIKE predicate p is "%AB%C%DA%", then SPH estimates the number of rows in the result set of this query as 10 based on the above histogram endpoint pattern. However, the true frequency of such a predicate may often be significantly less than 10.

We observe three factors that contribute to the overestimation in SPH as follows:

(i) The employed patterns are too generic, which makes SPH fail to differentiate the consecutively placed characters from those that are not strictly consecutive and may have some other characters between them.
(ii) Histogram endpoints are often occupied by patterns that have high frequency and are almost completely subsuming one another with little difference in the information that they provide. This greatly narrows down the coverage and diversity of SPH histograms, and leads to the elimination of many other potentially useful patterns from consideration as endpoints during histogram construction.
(iii) SPH attempts to fully match a query predicate pattern to histogram endpoints, and ignores partial matches. This prevents SPH from taking advantage of the constructed histogram, and contributes to its overestimation.

In order to address the above issues, in this paper, we propose and employ *positional sequence patterns* in the query optimizer of a database management system for selectivity estimation. The proposed positional sequence patterns discriminate uninterruptedly appearing character pairs from those that may have other characters appearing between them. Next, since positional sequence patterns are not supported by the existing sequence pattern mining algorithms (Le et al., 2017; Pei et al., 2004; Pasquier, Bastide, Taouil, & Lakhal, 1999; Wang, Han, & Pei, 2003; Wang & Han, 2004; Yan, Han, & Afshar, 2003; Zaki and Hsiao, 2000), we also propose an algorithm to mine positional sequence patterns of the form, $\%a_1\%a_2\%...\%a_n\%$ where each a_i represents one or more characters. Note that in a regular sequence pattern form, each a_i represents a single character. Similar to SPH, we build histograms out of the computed patterns, and these histograms are later used to estimate the selectivity of flexible LIKE string predicates. However, we eliminate redundant patterns that decrease the diversity and coverage of the constructed histogram. Besides, we integrate a partial matching framework during selectivity estimation. We call the proposed approach in this paper *P-SPH* (*P* for positional sequence patterns).

We use the DBLP dataset in order to extensively evaluate our methods. Our results show that P-SPH decreases the error rate of selectivity estimations up to 20% in comparison to the state of the art.

Contributions: Our primary contributions in this paper are as follows:

- We propose a novel type of sequence pattern (i.e., *positional sequence pattern*) that carries more information than the standard sequence patterns which do not specify whether there is any character between two consecutive items in the sequence or not. That is, if two characters always appear at consecutive

positions in all appearances of a pattern, such information is also marked in the new type of patterns as well. Such a differentiation may be important for many expert and intelligent system applications, for instance, to better address business questions like "should two frequently-bought-together items be located right next to each other on a retail store shelf, or is it ok to place them at some distance as long as they are in the same aisle?".
- We propose an algorithm to compute this new pattern type by extending a standard sequence pattern mining algorithm.
- The proposed positional sequence patterns may be employed by expert and intelligent systems in various domains. As an example expert system application, we demonstrate the utility of the positional sequence patterns in the query optimizer of a database management system for the task of estimating the selectivity of flexible string predicates.
- For efficiency and scalability purposes, the query optimizer's knowledgebase stores only a small subset of mined positional sequence patterns. In order to improve the coverage of selected positional sequence pattern subset, we introduce *information content-based* elimination of some patterns that highly overlap with already selected histogram endpoint patterns.
- During selectivity estimation based on the histograms stored in the optimizer's knowledgebase, we propose a *slider-based partial pattern matching scheme* to improve the accuracy of optimizer's selectivity predictions.
- We assess the proposed approach comparatively in different aspects on real datasets, and show that it outperforms the state of the art approaches, and provides better accuracy in terms of the prediction accuracy and online memory usage.

The rest of this paper is organized as follows. In Section 2, we discuss the related work. Section 3 describes the computation of positional sequence patterns. In Section 4, we present histogram construction out of positional sequence patterns. Section 5 describes the selectivity estimation algorithm. In Section 6, we present a detailed experimental evaluation of the proposed methods, and Section 7 concludes with pointers for future research.

2. Related work

2.1. Sequence pattern mining

Sequence pattern mining was first proposed in by Agrawal & Srikant (1995). Since then, a number of algorithms have been proposed to compute sequential patterns in a dataset. SPADE (Zaki, 2001), PrefixSpan (Pei et al., 2004), and SPAM (Ayres, Gehrke, Yiu, & Flannick, 2002) are some of the well-known sequence mining approaches. SPADE is based on vertical id-list database format, and uses a lattice-theoretic approach to decompose the original search space into smaller spaces. PrefixSpan employs a horizontal format, and uses the pattern-growth method. SPAM adopts a vertical bitmap representation, and mines the longest sequential patterns. Recently, VMSP (Viger, Wu, Gomariz, and Tseng, 2014) is proposed. VMSP is based on vertical id-list format, and mines not all sequential patterns, but only the "maximal" sequential patterns. Wang and Han (2004) propose BIDE, and similarly, rather than mining all frequent sequences, it mines only "closed" patterns.

2.2. Selectivity estimation for wildcard predicates

The selectivity estimation of wildcard predicates has been studied extensively in previous studies. Jin and Li (2005) propose an algorithm, called SEPIA. They employ a kind of frequency table in order to keep some summary data. To et al. (2013) propose information entropy-based histograms in order to estimate the selectivity of a LIKE predicate. They develop three algorithms (ME, MSE, and MB) that estimate the selectivity by using information entropy-based histograms. Jagadish et al. (1998) exploit optimal histograms to estimate the selectivity. They propose an algorithm which finds the optimal bucket boundaries. Poosala et al. (1996) offer equi-width and equi-height histogram structures. Muralikrishna and DeWitt (1988) propose an algorithm to build multi-dimensional histograms to estimate the selectivity of multi-dimensional queries. Lin et al. (2017) claim that previous histogram-based selectivity estimation approaches may negatively affect the performance of the queries because of the periodical data scans, and they get stuck with the curse of dimension. Hence, to overcome the above issues, they propose a new method that employs Ward's minimum variance method. The Ward method finds k-nearest "query feedback records" (QFR) for a new predicate, and a self-tuning histogram is constructed based on the k-nearest QFRs. Raju and Murthy (2017) also use a histogram structure to estimate the selectivity of a query. Instead of the classical histogram construction, they propose an adaptive way of histogram construction. In particular, histogram bucket contents are customized according to the predicate values that appear in executed queries.

Krishnan et al. (1996) employ suffix trees in order to structure the textual data, and estimate the selectivity of the substrings in a wildcard predicate. They propose an algorithm called KVI, and it assumes the independence of substrings. The suffix tree approach is used in many other studies as well. Jagadish et al. (1999) propose MO algorithm in order to overcome the accuracy problem due to KVI's independence assumption. The MO algorithm is based on the Markov assumption. Chaudhuri et al. (2004) observe that MO often underestimates selectivities, and they introduce a new algorithm called CRT which is based on the Short Identifying Substring (SIS) assumption. Li et al. (2015) claim that KVI and MO algorithms experience underestimation and overestimation problems. They analyze the main issues which lead to both underestimation and over estimation cases, and then propose two new algorithms, EKVI and EMO, which are extended versions of the KVI and MO algorithms respectively. Lee et al. (2009) exploit both suffix tree structure and MO as tools in their approach, and propose two new algorithms, MOF and LBS. They use the edit distance in order to find all base substrings of a given query predicate pattern. They estimate the selectivity by using minimal base substrings and information stored in an N-gram table. MOF is essentially an extended version of the authors' an earlier work (Lee et al., 2007). MOF is further extended into LBS, which assigns a signature to each minimal base substring, and keeps them in an N-gram table with their database frequencies. As for the minimal base substrings which are not stored in the N-gram table, LBS employs MO and suffix trees to estimate the frequency. Moreover, Kim et al. (2010) exploit a similar approach for selectivity estimation. Their method is based on inverted-gram indices. They employ signatures that are generated by random permutation for each substring. Both techniques proposed by Lee et al. (2009) and Kim et al. (2010) rely on similar techniques, and their reported results are almost the same. Besides, Mazeika et al. (2007) propose VSol to estimate the selectivity of the approximate string queries. They use edit distance and q-grams for estimation. In order to access the q-grams, and estimate the selectivity of a query, they use a hash index. VSol's approach is very similar to LBS (Lee et al., 2009).

Recently, Aytimur and Çakmak (2018) propose a novel approach, called SPH. SPH first mines all frequent closed sequence patterns by using an existing sequence mining algorithm (Wang and Han, 2004). Then, it builds a histogram structure from mined patterns. Histograms store the sequence patterns as their endpoint values, and their corresponding frequencies as endpoint counts.

During the selectivity estimation, all buckets of the histogram are visited, and based on the nature of matching between the histogram bucket endpoints and the predicate, it returns an estimated selectivity value. SPH is currently the state-of-the-art approach to estimate the selectivities of LIKE query predicates.

This study improves over SPH in the following aspects: (i) instead of regular sequence patterns, we propose and employ a new type of sequence pattern (i.e., positional sequence pattern) which accommodates strictly consecutive character pairs as part of the pattern, (ii) during histogram construction, rather than using all computed patterns blindly, we carefully filter patterns that sufficiently differ from others based on information-theoretic measures, (iii) during selectivity estimation phase, SPH considers only full pattern matching, while our approach in this work also features partial pattern matching in its selectivity estimation step with a slider-based search.

2.3. Selectivity estimation for other types of query predicates

Gupta and Garg (2016) study the problem of the selectivity estimation for range queries. To this end, they employ micro-clusters and cosine coefficients. More specifically, the proposed technique keeps data summary and its density distributions on micro-clusters. The technique performs well for balanced, moderately skewed, and highly skewed data distributions. Shin (2018) proposes a new approach that computes the exact selectivity during the optimization phase. In particular, the approach runs an extra aggregate query to compute the selectivity of a predicate in the query optimization period. The proposed method deals well with complex predicates. Yang et al. (2019) develop sampling-based techniques to solve the problem of selectivity estimation on set containment. The main aim of their study is to estimate the cardinality of the containment search. A random sampling-based and a list-based approach are proposed to solve the problem.

2.4. Approximate string matching

Finding mismatched parts between an input pattern and a target string is another issue in the string matching problem. Chen and Wu (2018) propose an indexing mechanism that finds all matches of a pattern in a string with up-to k mismatches. Kociumaka et al. (2019) also utilize a similar k-mismatches approach to find the longest common substring between given two strings. Hasan et al. (2019) employ two deep learning-based approaches to estimate the cardinality of point and range queries. In the first approach, they model the selectivity estimation as a density estimation problem. In the second technique, they estimate the selectivity of unknown queries from a training set where selectivities of queries known. Also, they provide a perspective on the challenges of adapting deep learning to relational data.

2.5. Adaptations to special-purpose databases

Approximate string matching is also popular in spatial databases. As an example, Yao et al. (2010) employ MHR tree to efficiently answer approximate string-matching queries in large spatial databases. Their technique is based on the min-wise signature and the linear hashing technique, which are also exploited in LBS approach.

Moreover, with the rapid progress in high-throughput sequencing technologies, genomics repositories stand out as another form of massive text-based databases (Chaitanya, 2019). Due to their increasing size and the need for complicated queries, genomics databases may greatly benefit from highly-specialized query processing methods and index structures (Jalili et al., 2018, Layer et al., 2018, Papadopoulos et al., 2016). Such methods usually adopt non-relational (i.e., key-value store) model. The proposed method, in this paper, on the other hand, is intended to support query processing in a relational setting. Nevertheless, genome researchers may still take advantage of the proposed positional sequence patterns to discover motifs (Bailey et al., 2015) that frequently appear in particular genomic regions of interest. To this end, one first needs to compile a set of possibly interesting genomic regions. Then, positional sequence pattern mining may be run on this dataset to discover the interesting motifs. As an example, a genomics researcher may create a dataset that includes 1000 basepair upstream region of each known gene on a genome of an organism. Then, the extracted positional patterns may point out candidate transcriptions factor binding sites, as well those genes that share common transcription factors. Positional sequence patterns may provide a finer grained view of frequent motifs than regular sequence patterns. That is, between two highly conserved regions in a motif, there may be a region with a high rate of mutations. Positional sequence patterns would allow differentiating between such highly conserved and less conserved regions in motifs, while regular sequence patterns may not directly accommodate them.

2.6. Process mining

Besides, the proposed method in this study may be adapted into the process mining field (van der Aalst et al., 2007). The primary aim of process mining is to discover, monitor, and improve the real processes by extracting knowledge from event logs. Conformance checking takes an event log and Petri net, and diagnoses the differences between the observed and modeled behavior (Carmona et al., 2018). More specifically, conformance checking techniques take a process model and an event log as inputs and return a set of differences between the behaviors captured in the process model and event log. Rozinat and Van der Aalst (2008) employ an incremental method to conform a process model to an event log. It is the first comprehensive analysis on conformance checking. The authors employ fitness and appropriateness in conformance analysis. (Gómez López, Borrego Núñez, Carmona, & Martínez Gasca, 2016) define the alignment problem between a process model and event log as a constraint satisfaction problem. They demonstrate that encoding the alignment problem as a constraint problem has various advantages over other conformance analysis approaches in the literature. Solvers can comfortably handle a large number of instances in order to obtain a valid output. Moreover, Carmona (2019) provides an overview of how process discovery and conformance checking problems can be solved in a distributed manner. The author focuses on different ways to partition event logs and models.

The volume of event data and the event log data generation frequency have rapidly increased. Event logs may contain billions of events. Moreover, processes mining is a continuous task, as processes in a system frequently change. Finding the best alignment may require solving many optimization problems or repeated state-space explorations. Hence, conformance checking can be time-consuming and inefficient under the above conditions. Our proposed method may be combined with the conformance checking techniques. More specifically, instead of using all event logs in every step of the conformance checking, a summary structure can be built offline, and then, a conformance checking task can be run on that summary structure to validate the process model with the event log of the same process. P-SPH employs a custom histogram as a summary structure which is built on the top of the frequent positional sequence patterns. Similarly, all frequent patterns in an event log may be mined, and then, a histogram structure may be built on the mined frequent patterns. Then, the conformance checking task may compare the process with the frequent event log in the histogram. A combination of P-SPH and conformance checking techniques may reduce the running time requirements considerably without compromising on the accuracy.

3. Positional sequence patterns

Sequence mining, in general, aims to find the statistically relevant sequence patterns in a given database. Since its introduction in the 90 s, it has been used in a wide range of expert system applications such as analyzing the DNA and RNA sequences to figure out coding regions, discovering customer shopping behaviors, analyzing telephone calling patterns in call centers, and finding user click patterns in web clickstreams. Standard (regular) sequence patterns, and the problem of mining such patterns may be formally defined as follows.

Definition 1 (*(Proper Sequence Containment):*). Given two sequences S = $s_1s_2\ldots s_n$ and Q = $q_1q_2\ldots q_k$ where each s_i ($1 \le i \le n$) and q_j ($1 \le j \le k$) are characters from an alphabet, and i, j indicate the position of characters in S and Q, respectively, let Q[i] denote the character at position i in Q. Then, S is properly contained in Q, if there exists a set of n positions, $p_1 < p_2 < \ldots < p_n$, such that;

$$\forall(s_i,\ s_j) \in S \textit{ and } i < j \rightarrow Q[p_i] = s_i \textit{ and } Q\left[p_j\right] = s_j \textit{ and } j - i \leqslant p_j - p_i$$

Example:. Consider S = ABD. S is properly contained in Q_1 = ACBCD. However, S is not properly contained in Q_2 = ACDCB.

Definition 2 (*(Regular Sequence Pattern):*). Given a sequence database D and a frequency threshold minsup, a sequence P is called a regular sequence pattern, if the number of rows that properly contain P in D is equal to or greater than minsup.

Definition 3 (*(Regular Sequence Pattern Mining Problem):*). Given a sequence database D and a frequency threshold minsup, the Regular Sequence Pattern Mining Problem is to compute the set of all regular sequence patterns.

Example:. Consider a sequence database shown in Fig. 1 that has four rows of sequences with letters from the following alphabet, $\Sigma = \{A, B, C, D, E\}$. Assume that the minimum support threshold is 3. Then, the complete set of regular sequence patterns and their corresponding frequencies are as follows {A:4, AB:4, AC:4, ACB:4, ACBE:3, ACE:4, B:4, BA:3, BAB:3, BAE:3, BC:3, BCB:3, BCE:3, BE:4, C:4, CC:3, CCB:3, CCE:3}

In this paper, we introduce a new type of sequence patterns, called *positional sequence patterns*. The difference between regular sequence patterns and positional sequence patterns is that the latter distinguishes item pairs that always appear next to each other in the same order in all the occurrences of the pattern from those that may have other items between them in some or all occurrences of the pattern. Therefore, positional sequence patterns are more specific and carry more information than regular sequence patterns. We give an example.

Sequence id	Sequence
1	ABCABE
2	BCACDBE
3	BACDCEDB
4	ACECBE

Fig. 1. An example sequence database.

Example:. Consider a regular sequence pattern R = ACCB. In SQL LIKE syntax, this pattern may be expressed as A%C%C%B. That is, between any characters in R, there may be zero or more other characters in the actual occurrences of the pattern. Now, consider a positional sequence pattern in SQL LIKE syntax, P = AC%CB. P carries the extra information that A is always followed by C with no other characters in between in all occurrences of P. In addition, in at least one occurrence of P, there is another character between the double C characters at the middle. Similarly, the last part of the pattern suggests that there is no character between C and B in all occurrences of P.

We next formally define positional sequence patterns in a similar manner that the regular sequence patterns are defined above.

Definition 4 (*(Positional Sequence):*). Given an alphabet Σ, a sequence S = $s_1s_2\ldots s_n$ is called a positional sequence, if $\forall s_i \in S \rightarrow s_i \in \Sigma \cup \{'\%'\}$ where i in s_i indicates the position of a particular character in the sequence ($1 \le i \le n$).

Definition 5 (*(Positional Sequence Containment):*). Given a positional sequence S = $s_1s_2\ldots s_n$ and regular sequence Q = $q_1q_2\ldots q_k$, let Q[i], where $1 \le i \le n$, denote the character at position i in Q, and the slicing operator S[i:j] specify the subsequence s = $s_is_{i+1}\ldots s_{j-1}s_j$ of S. Then, S is positionally contained in Q if there exists a set of n positions, $p_1 < p_2 < \ldots < p_n$, such that;

$$\forall(s_i, s_j) \in S,\ s_i \neq '\%',\ s_j \neq '\%' \textit{ and } i < j$$
$$\rightarrow \begin{cases} Q[p_i] = s_i \textit{ and } Q\left[p_j\right] = s_j \textit{ and } j - i \leqslant p_j - p_i \textit{ if } '\%' \in S[i+1 : m-1] \\ Q[p_i] = s_i \textit{ and } Q\left[p_j\right] = s_j \textit{ and } j - i = p_j - p_i \textit{ if } '\%' \notin S[i+1 : m-1] \end{cases}$$

Example:. Consider S = A%BD. S is positionally contained in Q_1 = ACBD. However, S is not positionally contained in Q_2 = ACBAD.

Definition 6 (*(Positional Sequence Pattern):*). Given a sequence database D, a frequency threshold minsup, a positional sequence P is called a positional sequence pattern, if the number of rows that positionally contain P in D is equal to or greater than minsup.

Definition 7 (*(Positional Sequence Pattern Mining Problem):*). Given a sequence database D, and a frequency threshold minsup, Positional Sequence Pattern Mining Problem is to compute the set of all positional sequence patterns.

Example:. Consider the sequence database shown in Fig. 1. Assume that the minimum support threshold is 3. Then, the complete set of positional sequence patterns (in SQL LIKE predicate syntax) and their corresponding frequencies are as follows {**AC:3**, **AC%B:3**, **AC%E:3**, A:4, A%B:4, A%C:4, **A%C%BE:3**, A%C%B:4, A%C%E:4, B:4, B%A:3, B%A%B:3, B%A%E:3, B%C:3, B%C%B:3, B%C%E:3, B%E:4, C:4, C%C:3, C%C%B:3, C%C%E:3} where the patterns in bold are new additions to the list of patterns that are obtained with regular sequence pattern mining as shown in the previous example.

We next discuss the computation of positional sequence patterns.

3.1. Mining positional sequence patterns

There is already a number of available techniques (Pasquier et al., 1999; Wang, Han, & Pei, 2003; Yan, Han, & Afshar, 2003; Zaki and Hsiao, 2000) proposed in the literature to compute regular sequence patterns. Since positional sequence patterns are a superset of regular sequence patterns, rather than designing a new algorithm, we choose to extend one of the existing regular sequence mining methods to compute positional sequence patterns. Among many others, in this work, we choose to extend BIDE (Wang & Han, 2004) for three reasons: (i) it eliminates the candidate maintenance step of standard pattern mining methods; hence, its memory and running time requirements are lower, (ii) rather than mining all possible patterns, it skips 'redundant' patterns, and mines only a subset of all patterns (i.e., 'closed' patterns) which are not contained in other patterns, and (iii) its source code is publicly available to the researchers; thus, it can be readily modified for extensions.

```
**Algorithm 1:** BIDE
**Input:**  An input sequence database SDB, a minimum support threshold min_sup.
**Output:** The complete set of frequent closed sequences, FCS
1. FCS= Ø;
2. F1=frequent 1-sequences(SDB , min_sup);
3. for (each 1-sequence f1 in F1) do
4.     $SDB^{f1}$ =pseudo projected database(SDB)
5. for (each f1 in F1) do
6.     if (!BackScan(f1, $SDB^{f1}$))
7.         BEI =backward extension check(f1, $SDB^{f1}$);
8.         call bide($SDB^{f1}$, f1, min_sup, BEI, FCS);
9. return FCS;
```

```
**Algorithm 2:** bide
**Input:**  a projected sequence database Sp_SDB, a prefix sequence Sp,
 a minimum support threshold min_sup, and the number of backward extension items BEI
**Output:** The current set of frequent closed sequences, FCS
10. LFI = locally frequent items ($S_p$_SDB;
11. FEI = |{z in LFI |z.sup = $sup^{SDB}(S_p)$ }|
12. if ((BEI not equal FEI)==0)
13.     FCS=FCS U {Sp};
14. for (each I in LFI) do
15.     $S_p^{\,i}$ <$S_p$,i>;
16.     $SDB^{S}_{pi}$ pseudo projected database ($S_p$_SDB, $S_p^{\,i}$);
17. for (each I in LFI) do
18.     if (!BackScan($S_p^{\,i}$, $SDB^{S}_{pi}$))
19.         BEI =backward extension check($S_p^{\,i}$, $SDB^{S}_{pi}$);
20.         call bide($SDB^{S}_{pi}$, $S_p^{\,i}$, min_sup, BEI, FCS);
```

We next summarize how BIDE works. Then, we present our extensions on it in order to mine positional sequence patterns. BIDE focuses on a special class of patterns, called 'closed' patterns. Algorithm 1 and 2 summarize the working principles of BIDE. We first provide a definition of 'closed' pattern.

Definition 8 (*(Regular Closed Sequence Pattern):*). Assume that S_a and S_b are two regular sequence patterns. If S_b properly contains S_a, then S_b is called a supersequence of S_a. If a sequence pattern has no supersequence with the same frequency, then it is called a regular closed sequence pattern.

Example. Consider the following sequence patterns and their frequencies: S_a = ABC (freq: 5), S_b = ABBC (freq: 5), S_c = ABB (freq: 6). Here, S_a is not a regular closed pattern, since S_b is a supersequence of S_a, and it has the same frequency. On the other hand, although S_b is also a supersequence of S_c, S_c is still a regular closed pattern, as its frequency is higher than S_b.

Definition 9 (*(First instance of a prefix sequence):*). Given an input sequence t and a prefix sequence s, if t contains s, the subsequence from the beginning of t to the end of the first appearance of s is called the first instance of prefix sequence in t. As an example, the first instance of prefix sequence CD in sequence ABCDAB is ABCD.

Definition 10 (*(Projected sequence of a prefix sequence):*). The projected sequence of a prefix sequence is the remaining part of the input sequence t, after removing the first instance of the prefix sequence p from t. As an example, the projected sequence of prefix sequence CD in sequence ABCDAD is AD.

Definition 11 (*(Projected database of a prefix sequence):*). The projected database of a prefix sequence t in a database D is the complete set of the projected sequences of t in D.

Given a sequence database, BIDE first scans the entire database to find all frequent sequences that have length 1 in line 1. It then builds the projected database for each frequent length-1 sequence in lines 3 to 4. Next, it calls BackScan to check whether frequent 1-sequence can be pruned or not in line 6, and if not, it computes the

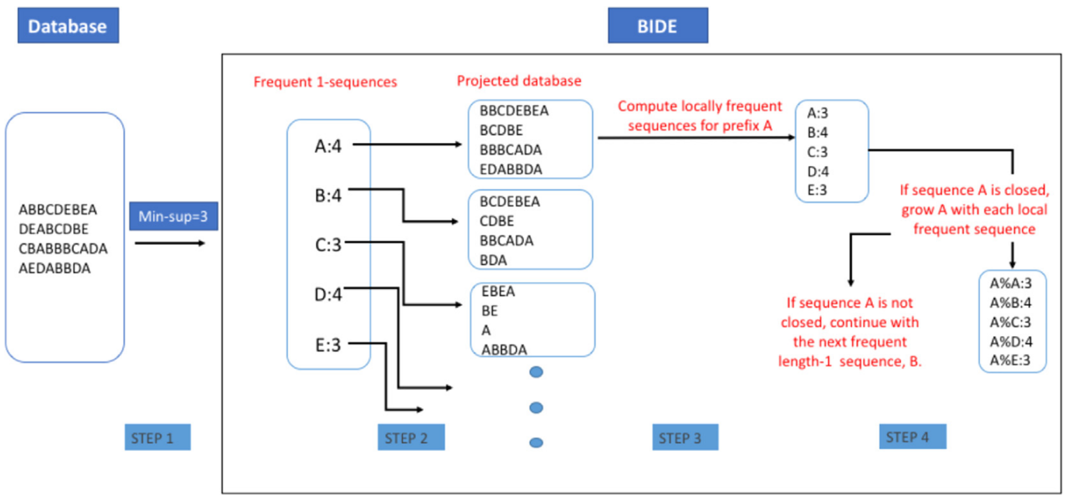

Fig. 2. A high-level overview of BIDE steps for length-1 patterns.

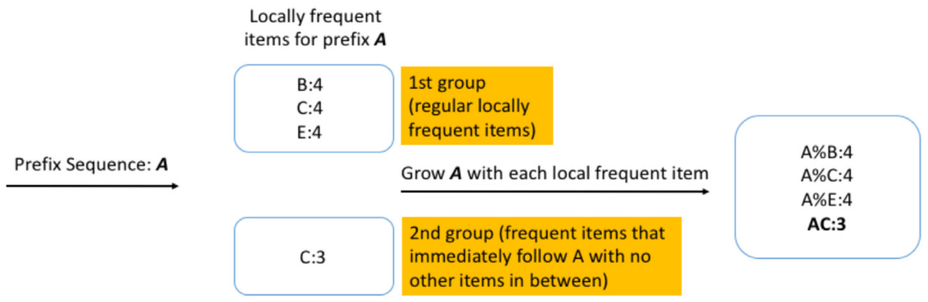
Locally frequent items for prefix A
Prefix Sequence: A
B:4
C:4
E:4
1st group (regular locally frequent items)
Grow A with each local frequent item
C:3
2nd group (frequent items that immediately follow A with no other items in between)
A%B:4
A%C:4
A%E:4
AC:3

Fig. 3. The new workflow in step 3 of the extended algorithm.

number of backward-extension items in line 7, and calls the subroutine bide in line 8. In line 10, it finds all locally frequent items for a prefix and computes the number of forward-extension items in line 11. If no forward-extension and backward extension are possible, it marks the prefix as a frequent closed sequence in lines 12 and 13. It then extends the prefix to get a new prefix, and finds pseudo projected database for the new prefix in lines 15 and 16. Next, it applies BackScan check from line 17 to line 19 and calls subroutine bide in line 20. Fig. 2 summarizes the workflow of BIDE for the frequent length-1 sequences (the produced patterns are presented in SQL LIKE syntax).

3.1.1. Extensions for positional sequence pattern computation

In order to mine positional sequence patterns, we extend step 3 in Fig. 2 and line 10 in algorithm 1. In step 3, all locally frequent items are computed for a prefix sequence in the corresponding projected database. That is, BIDE goes over all sequences in a projected database, and counts whether an item of interest exists in the projected sequence or not. Our extension divides this step into two parts. In the first part, locally frequent items are computed in the same way as in BIDE. In the second part, we compute the locally frequent sequences where there is no item between the last item of the prefix sequence and the local item. Fig. 3 illustrates the

Endpoint Number	Endpoint Value	Endpoint Frequency
12	**AC%E**	3
22	**A%C%E**	4
32	B%C%B	3
42	B%E	4
48	C%C%E	3

Fig. 4. Positional pattern-based histogram construction with bucket count 5.

new workflow in step 3 of the extended algorithm for the database provided in Fig. 1 (the produced patterns are presented in SQL LIKE syntax).

In Fig. 3, there are two groups of locally frequent items for prefix *A*. The first group includes all regular locally frequent items as before. The second group includes only locally frequent items which do not have any characters between the last character of the prefix (frequent closed sequence) and the first character of the projected sequences for that prefix. When prefix A is extended with these locally frequent items, the new prefix sequences are {*A% B*:4, *A%C*:4, *A%E*:4, ***AC*:3**} where the last sequence is obtained from the second group of locally frequent items.

Note that a locally frequent item may appear in both groups 1 and 2. In such cases, after verifying that their frequencies are over the minimum support threshold, we grow the prefix sequence as follows:

- If both frequencies are equal, extend the prefix sequence with only the local item from the second group (closure check).
- Otherwise, extend the prefix sequence with both local items.

The above first bullet-point enforces the "closed pattern" property. That is, since the patterns that are extended with locally frequent items from the second group are more specific than their counterparts in group 1, the former ones are preferred over the latter to make sure that the computed patterns are closed.

3.2. Histogram construction

Out of the mined positional sequence patterns, a histogram is built in a similar way as in SPH (Aytimur & Çakmak, 2018) with some extensions. We first briefly explain the histogram building steps, and then present our extensions to eliminate redundant endpoints.

In order to build a histogram, first, all patterns are lexicographically sorted. The capacity of each bucket is determined by dividing the total frequencies of the patterns by the number of buckets. In the next step, bucket endpoints are determined. To this end, the sorted pattern list is traversed starting from the first one while keeping a running sum S of pattern frequencies. Whenever $S \geq n * C$, where n is the currently built bucket's number with initial value 1, and C is the bucket capacity, that particular pattern is set as the current bucket's endpoint, and n is updated as $n = floor(S/C) + 1$. Then, this process is repeated until all the histogram endpoints are determined. We give an example below.

Example. Consider the set of positional sequence patterns that are mined with minimum support threshold 3 from the database in Fig. 1, i.e., {**A%C%BE**:3, **AC%B**:3, **AC%E**:3, A%C%B:4, A%C%E:4, B%A%B:3, B%A%E:3, B%C%B:3, B%C%E:3, B%E:4, C%C%B:3, C%C%E:3}. Assume that the specified bucket count is 5. Since the total pattern frequency is 39, the capacity for each bucket is 7. Fig. 4 shows the histogram constructed by employing the above-summarized approach.

3.2.1. Eliminating redundant endpoints

We observe that many endpoint patterns in the constructed histograms are highly similar with a small difference in their frequencies. Such patterns occupy space in scarcely available histogram endpoints, while not providing much extra coverage in terms of matching a diverse set of LIKE query predicates. We give an example.

Example. Consider two positional frequent sequences, A%C%BE and AC%B, with the same frequency. Almost all query predicate patterns that match the second pattern also match to the first pattern. Therefore, it would be redundant to keep both patterns in the histogram. Instead, the first pattern may be kept, as it differentiates between a wider spectrum of possible query predicate patterns, and the second pattern may be discarded from the histogram.

As an extension to SPH (Aytimur & Çakmak, 2018), we eliminate such patterns based on pattern containment and information-theoretic filtering as formally defined next.

Definition 12 (*(Pattern Containment):*). Given a positional sequence pattern P, let Striped(P) denote a sequence which contains all non-wildcard characters of P in the same relative order as in P. Then, a positional sequence pattern Q is contained in another positional sequence pattern S, if Striped(Q) is properly contained in Striped(S).

In order to eliminate the adverse effects of the redundant pattern issue, one may consider eliminating a pattern if it is contained in at least one other pattern. However, such an approach may lead to major estimation errors, as the "information" contained in these patterns may differ significantly. Hence, we propose to compute the *information content* (Cover & Thomas, 2012) of patterns, and eliminate a pattern *p*, if the information content difference between *p* and another pattern that *contains p* is "ignorably" small.

Definition 13 (*(Information Content of a Pattern):*). Given a positional sequence pattern R and a database D, let freq(R) denote the number of rows that contain R in D, and |D| denote the number of rows in D. Then, the information content of R, IC(R), is computed as follows:

$$IC(R) = -loglogP(R)$$

where $P(R)$ denotes the probability of R, and computed as $P(R) = \frac{freq(R)}{|D|}$.

Definition 14 (*(Redundant Pattern):*). Given a positional sequence pattern set S computed over a database, a pattern P $\in$ S is considered as redundant, if there exists another pattern R $\in$ S and R $\neq$ S such that (i) R contains S, and (ii) IC(R) $-$ IC(S) $< \delta$ where δ is a small threshold.

In the above definition, the information content difference threshold, δ, is determined experimentally as explained in the empirical evaluation section. Before histogram computation, we eliminate *redundant patterns*. Then, the remaining patterns are considered during histogram construction. Once a pattern-based histogram is built offline during database statistics gathering time, it is stored in the query optimizer's knowledgebase (i.e., database dictionary/catalog) to be later used during query optimization time for selectivity estimation.

4. LIKE predicate selectivity estimation with Pattern-based histograms

We estimate the selectivity of LIKE query predicates based on the constructed histogram in a similar way to SPH (Aytimur & Çakmak, 2018) with some extensions. At a high level, the selectivity of a LIKE predicate pattern p is estimated according to the type of the match between predicate p and histogram endpoints as summarized below. Note that the order is important, i.e., an exact match is preferred over an encapsulated match, if both are applicable.

- o p exactly matches a histogram endpoint b. [Exact match case]
  - o $selectivity = [the\ endpoint\ frequency\ of\ b]/[the\ database\ size]$
- o p is contained in a set B of histogram bucket endpoints. [Encapsulated match case]
  - o $selectivity = [the\ minimum\ endpoint\ frequency\ in\ B]/[the\ database\ size]$

where set B includes bucket endpoint values and their corresponding frequencies for buckets in the histogram that encapsulate p, and minimum endpoint frequency is the minimum endpoint frequency value in B.

- o otherwise: [*No match case*]
  - o $selectivity = [t\ \%\ of\ the\ minimum\ support\ threshold]/[the\ database\ size]$

where SPH determines t experimentally as 10%, and we use the same setting.

In the above approach, due to the limited number of histogram endpoints, many queries fall into "no match case" in which an average selectivity is assigned independent of the query predicate. On the other hand, it is often the case that some parts of the query predicate may match the histogram endpoints, which may provide

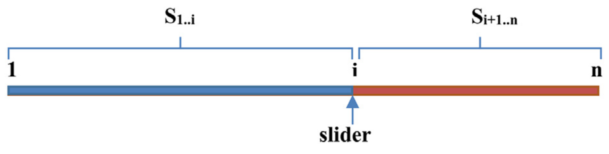

Fig. 5. Slider-based partitioning of query predicate string S.

better estimation boundaries. Hence, in addition to the above matching approaches of SPH, we introduce a *partitioning-based matching* strategy that is employed when exact or encapsulated matching attempts fail as summarized next.

4.1. Partitioning-based matching

The idea behind partition-based matching is stated by the following Lemma.

Lemma (***(Substring Selectivity):***). Given a LIKE query predicate string S of length n, assume that S is divided into two pieces, $S_{1..i}$ and $S_{i+1...n}$, at position i. Then, $\text{Selectivity}(S) \leq \text{Min}[\text{Selectivity}(S_{1..i}), \text{Selectivity}(S_{i+1..n})]$.

The proof of the above lemma is straightforward and intuitive. Therefore, we omit formal proof here for brevity. We perform the partition-based matching by introducing a slider, initially positioned at the first character in a given query predicate string S. Then, we advance the slider position one-by-one until it reaches to the last character in S (Fig. 5). At each slider position, we attempt to compute the selectivity estimates of $S_{1..i}$ and $S_{i+1...n}$ with exact or encapsulated match options. Next, the computed partition selectivities are compared to the minimum of the previously computed selectivities of partitions for the earlier positions of the slider. If any of the selectivities computed for the current position of the slider is smaller than the currently known minimum selectivity among all previous partitions, then we update the minimum selectivity accordingly. Once the slider reaches position n, the minimum selectivity estimate computed over all slider positions is assigned as the selectivity estimate for S.

***Partition length-based filtering*:** We observe that bluntly employing the above approach may sometimes lead to highly over-estimated selectivity predictions. This happens in the following cases: a very short partition (e.g., one character) of a query string may have an exact or encapsulated match in the histogram with many high-frequency endpoints, and the remaining partitions may not have any match. Hence, the selectivity of that very short partition is used as the selectivity estimate (as it is the "minimum" among all partitions). However, since there is a large length difference between the original query predicate string and its short partition(s), their selectivities usually differ quite a bit as well. In order to alleviate this side-effect of partitioning, we eliminate partitions of length smaller than $n - \varepsilon$ from consideration, where n is the length of the query predicate string, and ε is an experimentally determined threshold value. In the experimental evaluation section, we study the effect of ε, and accordingly, determine its optimal value.

The partitioning-based matching is placed after the *encapsulated match case* in the above list of attempted matches. That is, if the exact and encapsulated match is not possible, then the partitioning-based match is explored. If slider-based partitioning does not yield any selectivity estimate either (i.e., none of the partitions qualifies for an exact or encapsulated match), then 'no match case' is employed as before. We next give an example that comprehensively illustrates the effect of new histograms and partition-based matching on the selectivity estimation process.

Sequence id	Sequence
1	BCAB
2	BCAC
3	ACXCB
4	ACYCB
5	ACZCB
6	AYCCB
7	AZCCB
8	ACXCB

Fig. 6. An example sequence database D.

Endpoint Number	Endpoint Value	Endpoint Frequency
15	A%B	7
28	A%C%C%B	6
42	B	8
59	C%B	7

Fig. 7. Histogram constructed from regular sequence patterns.

Endpoint Number	Endpoint Value	Endpoint Frequency
20	A%C%CB	6
36	AC%CB	3
54	C	8
74	C%CB	6

Fig. 8. Histogram constructed from positional sequence patterns.

Example. Consider the database D in Fig. 6 that contains 8 rows. Assume that the minimum support threshold is 2, and the allowed number of buckets in a histogram is 4. 12 regular sequence patterns and 18 positional sequence patterns are computed from D. Fig. 7 shows the resulting histogram constructed from regular sequence patterns, and Fig. 8 shows the histogram constructed from positional sequence patterns. Note that the histogram built from positional sequence patterns (Fig. 8) is more specific than the one constructed from regular patterns (Fig. 7). As an example, the endpoint pattern for bucket 2 in Fig. 7 has two forms in the other histogram in Fig. 8. We next demonstrate the selectivity computation for the above-discussed four match cases with both histograms.

Exact match case: Assume that a LIKE query predicate p = *AC%CB* is given. SPH (Aytimur & Çakmak, 2018) would declare an *exact match* to the second bucket of the histogram in Fig. 7, and compute the selectivity as 0.75 (i.e., 6/8 where 6 is the corresponding bucket endpoint frequency, and 8 is the number of rows in the database). Similarly, P-SPH would declare an *exact match* to the second bucket of the histogram in Fig. 8, and compute the selectivity as 0.375 (i.e., 3/8 where 3 is the corresponding bucket endpoint frequency, and 8 is the number of rows in the database). The true selectivity for p is 0.375.

Encapsulated match case: Assume that a LIKE query predicate p = *%C%C%* is given. No *exact match* is possible in either of the histograms. However, there is an *encapsulated match* with the second bucket of the histogram in Fig. 7. Since it only matches with a single bucket, SPH's selectivity estimate is 0.75 (i.e., 6/8). As for the histogram in Fig. 8, there are encapsulated matches with buckets 1, 3, and 4, where the maximum endpoint frequency is 6. Thus, the selectivity estimate of P-SPH is 0.75 (i.e., 6/8) as well. The true selectivity for p is 0.875 (i.e., 7/8).

Partitioned encapsulated match case: Assume that a LIKE query predicate p = *Z%CB* is given. No exact or encapsulated match is possible in any of the histograms. SPH will employ the no match case, and estimate the selectivity as 0.025 (i.e., 0.2/8) (0.2 is *t% of* min *support threshold* where SPH uses an experimentally determined value 10 for t). P-SPH takes advantage of a partitioning-based match in this case. According to Fig. 8, a partitioned match is possible between p and the first, second, and fourth endpoint values. Out of these, since the second bucket endpoint provides the smallest estimate, the selectivity is 0.375 (3/8). The true selectivity for p is 0.25 (2/8).

No match case: Assume that a LIKE query predicate p = *D%A%D%E* is given. In this case, none of the above match types (including partition-based match) is possible in either histogram. Therefore, *no match case* route is taken, and the selectivity is estimated as 0.025 (i.e., 0.2/8). The true selectivity for p is 0.

5. Experimental results

This section presents our experimental results to evaluate the proposed P-SPH algorithm. All of our experiments were performed on a DELL R720 machine with 2 × XEON E-5-2620v2 2.10 GHz CPU and 80 GB of RAM.

5.1. Dataset

We perform various experiments using a real dataset from DBLP. The dataset is the same as the one used in SPH (Aytimur & Çakmak, 2018) and contains 800,000 full author names. The lengths of full author names vary between 18 and 60 with an average of 22.5.

5.2. Test query set

We evaluate the performance of P-SPH using the same query workload as described in SPH (Aytimur & Çakmak, 2018). More specifically, the query workload includes three different groups of queries, and each group has 100 queries, except that the negative query set contains 24 queries. The way that these query sets are generated is described in SPH paper (Aytimur & Çakmak, 2018), but we also include a brief summary here as well.

- Queries in the first group are in the form of *%w% and %w$_1$%w$_2$%* where w$_i$ is a word that has a length between 5 and 12. In order to generate this group of queries, one or two words with a length between 5 and 12 are chosen. Then, a random number (from 0 to 2) of underscore characters (i.e., "_") are inserted at random positions in a word. In SQL LIKE syntax, the underscore character represents a wildcard that matches any single character. This group of query predicates has minimum, average, and maximum lengths of 5, 6.7, and 17, respectively. The average selectivity is 4.77%.
- To construct the second group of the queries, a random row, R, from the database is chosen, and a random number, k, between 3 and the length of this selected row is drawn. Then, k characters are removed from R, and a random number (from 2 to 8) of "%" signs are inserted at random positions in R. That is, the generated queries are in the form of *%s$_1$%s$_2$%*. . .. . . . *%s$_n$%* where s_i represents one or more characters. The average, minimum, and maximum lengths for the query predicates in this set are 8.4, 3, and 16, respectively. The average selectivity is 2.67%.
- The negative query set includes queries which do not match any rows in the database. The generation of the negative queries is almost the same as the second group of queries. The only difference is that a smaller number (from 1 to 3) of "%" symbols are randomly inserted in R to decrease the likelihood that the generated query predicate matches any rows in the database. Out of 100 generated queries, 24 of them are truly negative queries with 0 matching rows. Hence, this set, in its final form, contains 24 queries.

5.3. Evaluation metrics

We employ two metrics to test the accuracy of selectivity estimation as proposed in KVI study (Krishnan et al., 1996). The first metric is the relative error which is employed for the positive query sets. The relative error is defined as $|f_{true} - f_{est}|/f_{true}$, where f_{true} is the actual true selectivity of the query, and f_{est} is the estimated selectivity. Since the third group queries (negative query set) has the actual selectivity of 0, the relative error metric is not applicable here (i.e., due to the division by 0 error). Instead, the *absolute error* metric is used in this group. Absolute error is defined as $|f_{est} - f_{true}|$. We use the same metrics in order to evaluate our technique and compare it with the state of the art SPH (Aytimur & Çakmak, 2018). Moreover, in SPH, the authors exclude those queries that have the actual frequency of 10 or less. Similarly, we also exclude such queries in this work as well.

5.4. Results

In this section, we evaluate different aspects of our approach in terms of estimation accuracy, query time, space overhead, and compare it to the state of the art.

5.4.1. The effect of the minsup threshold

The minimum support threshold, during frequent sequence mining, directly affects the number of patterns in the result set. This section evaluates the effect of the minimum support threshold on the computed pattern count and accuracy. Fig. 9 shows the number of regular and positional sequence patterns for different minimum support values.

Observation 1: The total number of patterns (positional or regular) decreases dramatically, as the minimum support threshold increases.

The total number and the rate of increase for positional patterns are slightly higher than that of the regular patterns. The reason for this difference is that the positional sequence pattern set includes all regular sequence patterns as well as some additional patterns. For instance, consider two patterns and their frequencies: *A%B%A %D%A%B*: **5** and *A%B%A%D%AB*: **3**. The regular sequence pattern set includes only the first one, while the positional sequence pattern set includes both patterns.

As per the above observation, since the minimum support threshold greatly affects the total number of patterns, it is critical to increase the minimum support threshold to the extent that it

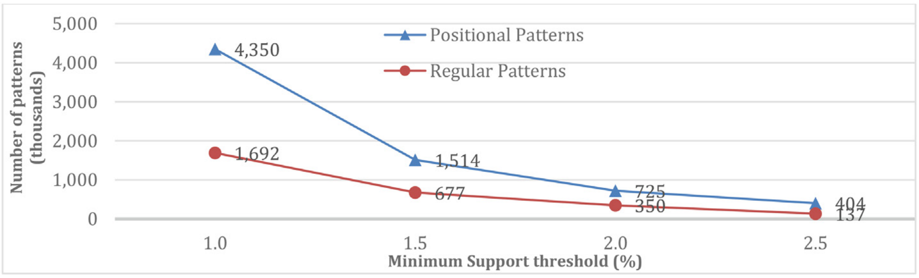

Fig. 9. The number of regular and positional sequence patterns for different minsup values.

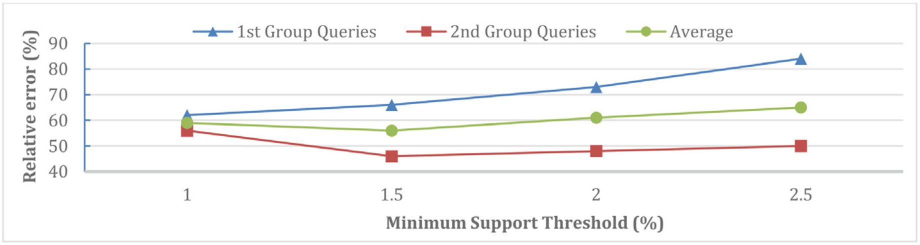

Fig. 10. The change of accuracy with different minsup values.

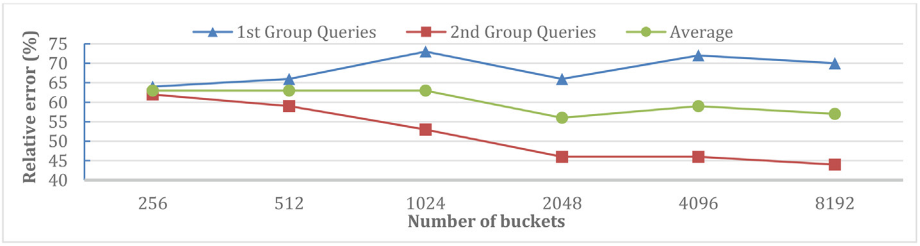

Fig. 11. Relative error for different numbers of buckets for P-SPH with minsup 1.5%

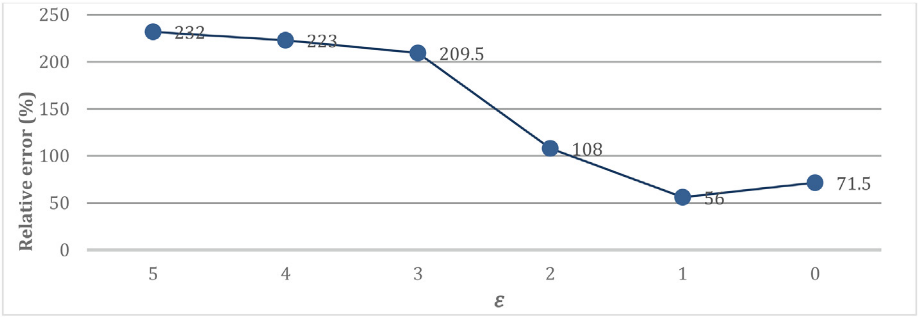

Fig. 12. Average relative error for different values of.ε

does not harm the selectivity estimation accuracy considerably. Next, we investigate how high the minimum support threshold could be while keeping the selectivity estimation accuracy decrease tolerable. Fig. 10 shows the change of accuracy with different minimum support threshold values (bucket count: 2048).

Observation 2: The average accuracy gets the highest value when the minimum threshold is 1.5%.

The accuracy of estimation decreases in general, as the minimum support threshold increases. The reason is that as the minsup threshold gets larger, the mined patterns get shorter. Hence, the ratio of queries that can be answered with exact or encapsulated match decreases, and P-SPH more often uses partitioning-based and no-match cases, which are more error-prone. There is one exception to this expected behavior. That is, the second group queries experiences a decrease in error rate when the minsup threshold increases from 1% to 1.5%. The reason is that for the first group of queries, the number of non-matching queries increases as the minsup value increases; hence, the error rate increases as well. On the other hand, for the second group of queries, the number of non-matching queries stays the same when minsup changes from 1% to 1.5%. What is more, the average actual row count for non-matching queries decreases. Therefore, the error rate decreases for the 2nd group queries when minsup increases from 1% to 1.5%. Detailed statistics for each minsup value are presented in Table 1. Based on the above observation, unless noted otherwise, we use 1.5% as the minimum support threshold for the remaining experiments.

5.4.2. The effect of the number of buckets

The number of buckets in a histogram is an important setting, as more number of buckets means more endpoint values, and may increase the selectivity estimation accuracy. However, lower numbers of buckets require less search time and less memory space. Hence, there is a trade-off between estimation accuracy and higher resource consumption. In this experiment, we investigate the relationship between the selectivity estimation accuracy and the number of buckets. Fig. 11 plots the relative estimation error for different numbers of buckets.

Observation 3: *As the number of buckets increases, the average relative error decreases until the number of buckets reaches 2048. After that point, no meaningful accuracy improvement is observed.*

The above observation shows that there is no need to use more than 2048 buckets to increase selectivity estimation accuracy. Hence, unless noted otherwise, we use 2048 buckets in the remaining experiments.

5.4.3. The effect of partitioning

In this section, we evaluate the contribution of partitioning-based matching. First, we determine the optimal value for the threshold ε that provides the best accuracy. Fig. 12 shows the change of accuracy for different values of ε (bucket count: 2048, minsup: 1.5%).

Observation 4: *The lowest relative error is obtained when ε is 1,* i.e., *partitions which are shorter than $|lengthofquerypredicate| - 1$ are not considered during partitioning-based selectivity estimation.*

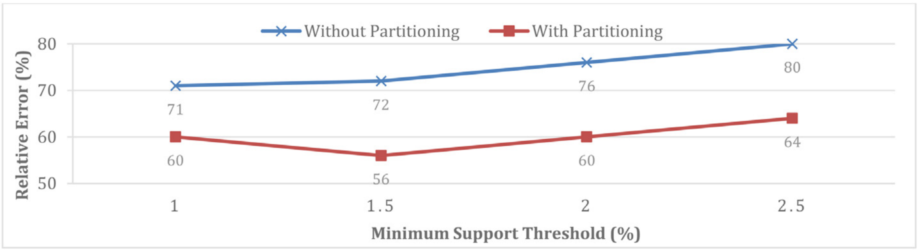

Fig. 13. Relative error for different minimum support thresholds values with and without partitioning.

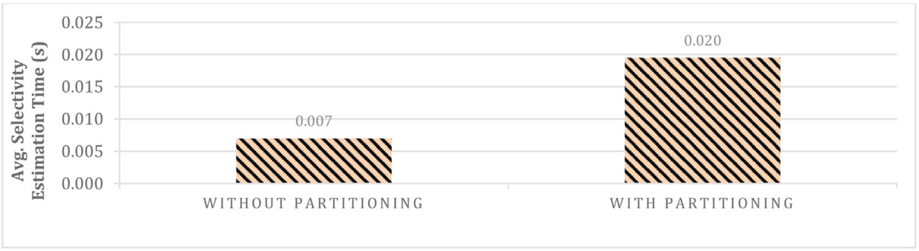

Fig. 14. Selectivity estimation time with and without partitioning.

The relative error increases when the allowed partition lengths get shorter. This is because short partitions are very generic, and their selectivity may be a lot higher than that of the original predicate. Hence, the relative error increases.

Next, we present selectivity estimation accuracies with and without partitioning-based matching. Fig. 13 shows the average relative error for different minimum support threshold values with and without partitioning (bucket count: 2048, $\varepsilon = 1$).

Observation 5: Partitioning-based matching decreases the relative selectivity estimation error up to 16%.

When partitioning-based match is not available, long query predicate strings do not match any endpoint of the histogram, although there are matching rows in the database. On the other hand, partitioned query predicate strings allow such queries to partially match histogram endpoint values, and provides better estimation accuracy.

Finally, we evaluate the impact of partitioning-based matching on query processing time. Fig. 14 shows the average selectivity estimation time with and without partitioning (bucket count: 2048, minsup: 1.5%).

Observation 6: Partitioning-based matching increases the average running time around 3 times.

The increase in query processing time due to partitioning-based matching is expected, as it adds an extra step of computation during the selectivity estimation. Although the selectivity estimation time relatively increases significantly, the total estimation time is still at the level of milliseconds. Thus, the improved accuracy as shown in Fig. 13 justifies this estimation time compromise.

5.4.4. The effect of redundant pattern elimination

In this section, we evaluate the effect of redundant pattern elimination. We first experimentally identify the best value for the information content difference threshold, δ, described in Section 3.3.1. Fig. 15 shows the change of relative estimation error with different information content difference threshold values where the horizontal axis values are multiplied by 10^5 to improve the readability (bucket count: 2048, minsup: 1.5%).

Observation 7: The lowest relative error is obtained when the information content difference threshold for redundant pattern elimination is -0.00216.

Based on the above observation, for all the experiments in this paper, we set the information content difference threshold for redundant pattern elimination as -0.00216.

Next, we present the impact of redundant pattern elimination on the selectivity estimation accuracy. Fig. 16 shows the average relative error for different minimum support threshold values with and without redundant pattern elimination (bucket count: 2048, $\varepsilon = 1$).

Observation 8: Redundant pattern elimination decreases the relative error up-to 7%.

Redundant pattern elimination does not harm the estimation accuracy, i.e., it either improves the accuracy or performs nearly the same as the case without redundant pattern elimination. This is expected, as redundant patterns occupy some of the very limited histogram bucket endpoints, and removing them opens up space for more distinctive patterns to be included in a histogram.

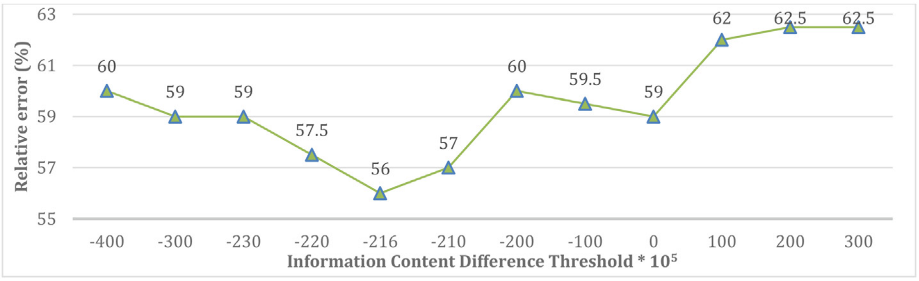

Fig. 15. The change of relative estimation error with different information content difference threshold values.

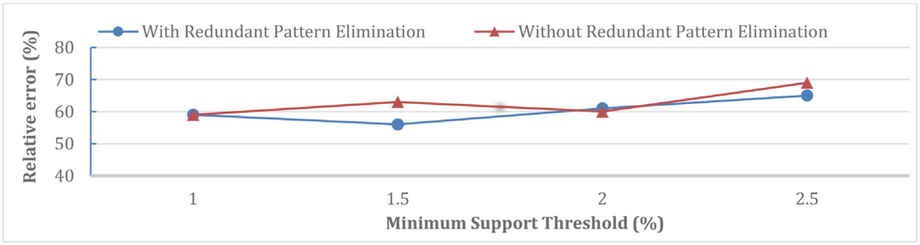

Fig. 16. Average relative error with and without redundant pattern elimination.

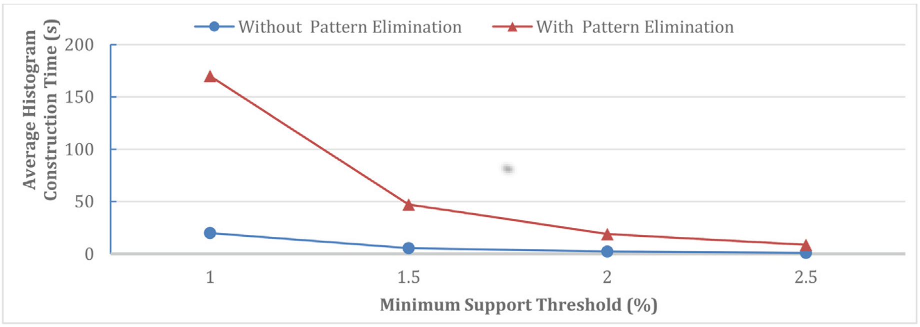

Fig. 17. Build time with and without redundant pattern elimination.

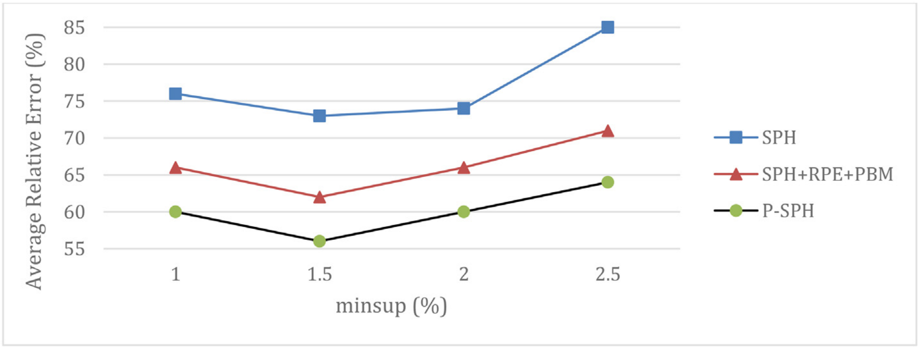

Fig. 18. Average relative error comparison for different minimum support thresholds values.

Finally, we evaluate the impact of redundant pattern elimination on histogram construction time. Fig. 17 shows the change of histogram construction time with and without redundant pattern elimination (bucket count: 2048).

Observation 9: Redundant pattern elimination significantly increases the histogram construction time.

Although the above observation points to some considerable overhead in histogram construction time, the bottleneck in the statistics gathering pipeline is the pattern mining stage, which is responsible for more than 99% of the total spent time as shown in the next section. Therefore, the above increase in histogram construction time is invisible in practice.

5.4.5. Comparison with the state of the art

In this section, we compare our proposed method (P-SPH) with the existing SPH, LBS, and KVI algorithms.

5.4.5.1. Accuracy-based comparison. We first conduct a comparative study based on selectivity estimation accuracy. SPH originally does not have the partition-based matching (PBM) and redundant pattern elimination (RPE) features. On the other hand, these techniques are generic enough to be transferred from P-SPH to SPH. Hence, for comparison purposes, we also create an enhanced version of SPH extended with PBM and RPE. Fig. 18 shows the average accuracy values for the original SPH, extended SPH, and P-SPH approaches with different minsup thresholds (number of buckets: 2048).

Observation 10: P-SPH provides up to 20% lower error rates in comparison to the original SPH.

Observation 11: P-SPH provides up-to 7% better selectivity estimation accuracies than the extended SPH for all minsup values. Since the only difference between P-SPH and the extended SPH is that the former employs positional sequence patterns while the latter employs regular sequence patterns, the success of P-SPH over the extended SPH is attributed to the positional sequence pattern usage.

We next perform a similar comparison by changing histogram bucket counts and keeping the minsup value fixed this time. Fig. 19 shows the average accuracy values for the above three approaches with different bucket counts (minsup: 1.5%).

Observation 12: In all bucket count configurations, P-SPH outperforms SPH (up-to 17%). Both approaches, in general, benefit from increased bucket counts in terms of lowering the estimation error rates.

Observation 13: Positional sequence patterns lead to better selectivity estimates than the regular sequence patterns as P-SPH

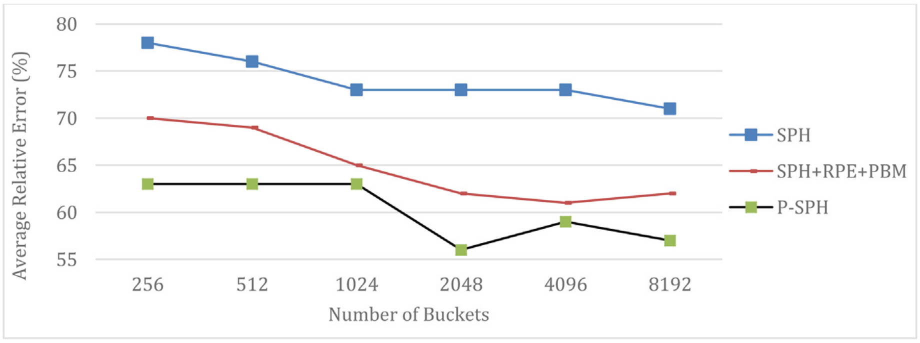

Fig. 19. Relative error comparison for different number of buckets.

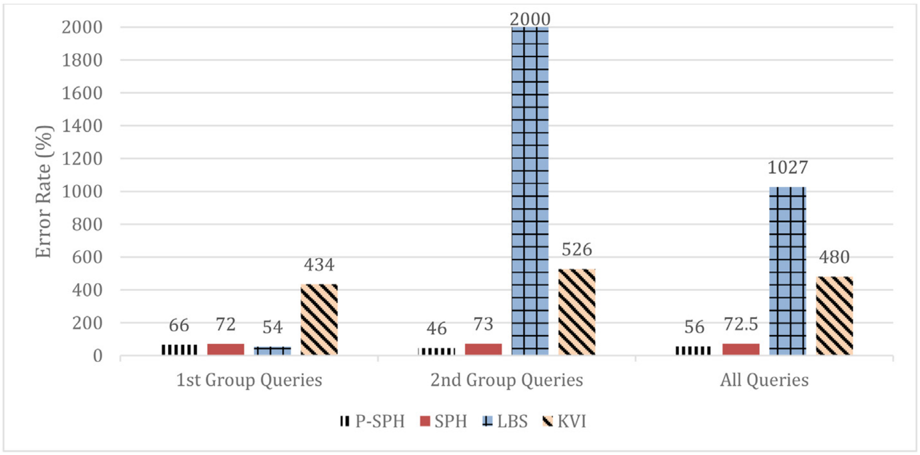

Fig. 20. Accuracy comparison of P-SPH, SPH, KVI, and LBS.

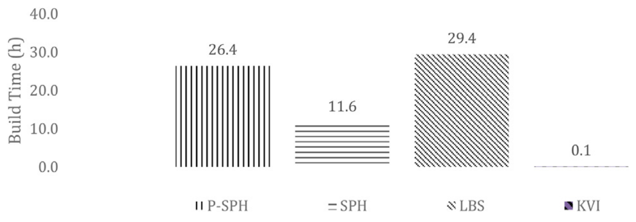

Fig. 21. Average build time for P-SPH, SPH, KVI, and LBS.

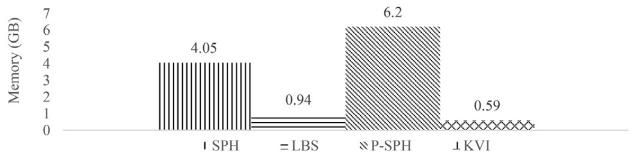

Fig. 22. Average build phase space overhead for all algorithms.

outperforms (up-to 7%) the extended SPH featuring partitioning-based matching and redundant pattern elimination for all bucket configurations.

The above two experiments show that P-SPH outperforms both SPH and its extended version for all configurations of minsup and bucket counts.

We next compare P-SPH with two earlier methods, KVI and LBS in terms of the selectivity estimation accuracy. We also include SPH to illustrate all methods in one picture. Fig. 20 shows the average relative error for P-SPH, SPH, KVI, and LBS approaches. In all following experiments, bucket count is set to 2048 and minimum support is set to 1.5%.

Observation 14: P-SPH provides the least estimation error rate by a significant margin when the whole query set is considered, even though LBS performs slightly better for the first group of queries.

Time and space overhead-based comparison:

In this section, we compare different approaches in terms of memory and running time requirements. Here, we perform our analysis in two parts: (i) offline build phase where patterns are computed and a histogram is built during database statistics gathering time, and (ii) online phase, which involves the estimation of query predicate selectivity during query optimization time.

First, we present time and memory requirements for the offline build phase. Figs. 21 and 22 show the build phase time and memory requirements, respectively, for all compared algorithms.

Observation 15: In terms of build time, P-SPH takes considerably more time than SPH, and it is comparable with LBS, while KVI takes the least amount of time owing to its simpler methodology.

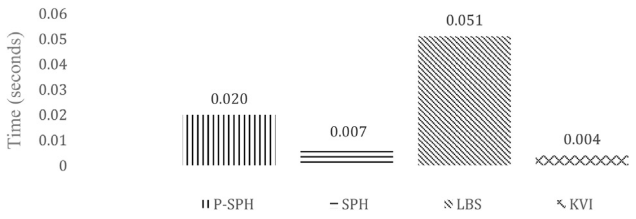

Fig. 23. The selectivity estimation (online phase) time comparison.

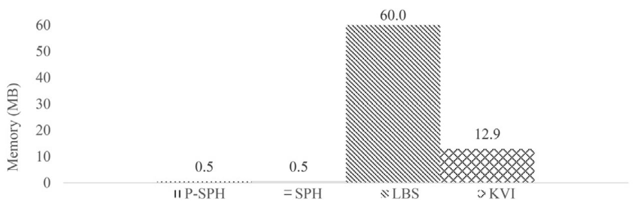

Fig. 24. Online phase memory requirement comparison.

The above observation is directly explained by Fig. 9, which shows that there is a similar relation between the numbers of positional and regular patterns. The number of positional patterns is significantly higher. Therefore, it takes more time to compute them. Besides, since this stage takes place offline before query time and possibly on a separate machine than the production servers, the time spent for this stage is considered a lot less critical than the online query optimization time, which is discussed in the next set of experiments.

Observation 16: In terms of build phase memory, P-SPH and SPH are comparable. KVI and LBS require significantly less build phase memory than P-SPH and SPH.

Even though P-SPH computes 2 times more patterns than SPH, the memory requirements do not reflect this difference. This is mostly because the proposed positional sequence mining approach follows the depth-first strategy of the original BIDE algorithm that it extends.

Since the histogram building phase takes place offline on a possibly different server, it does not directly affect the performance of the query optimizer during actual query execution time. Hence, the higher space requirement of P-SPH during this offline phase is not considered to be a major concern point.

Next, we present time and memory requirements for the online phase. Fig. 23 shows the selectivity estimation time, and Fig. 24 presents the online phase memory requirements for P-SPH, SPH, KVI, and LBS.

Observation 17: The selectivity estimation time of SPH is considerably less than that of P-SPH. LBS has the worst estimation time, while KVI has the least estimation time.

Two factors contribute to the above observation: (i) P-SPH uses partitioning-based matching, which adds extra computation time, whereas SPH lacks this feature, and (ii) the endpoint values in histograms build with positional sequence patterns are longer than those built with regular sequence patterns. Hence, P-SPH spends more time than SPH while computing encapsulated or exact matches.

5.5. Discussion

The above experimental results, in brief, imply the following insights:

- Positional sequence patterns are richer in variety and number than the regular sequence patterns, as the former allows to express more information about the items in a pattern, i.e., whether each item pair is strictly consecutive or not. Hence, more resources are needed to compute and store positional sequence patterns than that for regular sequence patterns.
- Minsup threshold value that is employed during pattern computation has a dramatic effect on the prediction accuracy and pattern mining cost. It is also tightly connected to the resulting pattern content and length. As the minimum support threshold values get larger, the cost of pattern mining step decreases. However, with high minsup values, the average length of the mined sequential patterns gets smaller and smaller. This decreases the ratio of query predicates that matches at least one histogram bucket endpoint. Accordingly, prediction accuracy may get lower.
- Increasing the size of the knowledgebase of the query optimizer does not improve the selectivity estimation accuracy forever. After a point (about 2048 buckets), the improvement is negligible. This may imply that once the coverage of histogram endpoints over the database reaches maturity, additional endpoints do not contribute much. Hence, there is no need to pay the additional cost of keeping a larger knowledgebase after this critical point.
- Very small partitions in partial matching case harm the prediction accuracy, as they match almost all histogram endpoints, and they are not specific enough to represent the original unpartitioned query predicate.
- Partitioning's benefit percentage is lower than the extra time requirement that it introduces. Hence, if optimization time is overly vital for a company's database management system, the partitioning may be turned off in favor of optimization time.
- All extensions that the P-SPH algorithm combines over SPH contribute to the accuracy improvement both individually when they are applied alone, as well as in a group when they are applied together. This is evident when each of these extensions is integrated with SPH, even though SPH is not using the positional sequence patterns. What is more, with all extensions added to SPH, P-SPH still performs better owing to the richer representation power of newly proposed positional sequence patters.

Despite the significant improvements over the state of the art, we also note the following limitations of our study.

- A production database of a company is often subject to updates, insertions, and deletions which change the content of the database over time. Some of these modifications may not lead to a significant change in terms of the mined frequent positional sequence patterns, while some others may cause considerable differentiation in the sequence pattern set. This study does not offer an adaptive solution to determine the best time point for re-computing the pattern set, and then re-construct the corresponding histograms accordingly.
- All pattern mining algorithms require minimum support threshold (i.e., minsup) to be provided as input for the pattern mining process. High threshold values may lead to missing some important patterns, while low threshold values increase the number of patterns and pattern computation time dramatically. Even though this study suggests the optimal minsup threshold that minimizes the prediction error rate (see Fig. 10), the suggested values are specific to this dataset, and it has to be re-determined for each dataset separately. This introduces another layer of tuning that an expert system has to take care of.
- Similarly, for "no match case", during selectivity estimation, the current study employs a fixed selectivity estimation independent of the query predicate. This implies two limitations: (i) since the employed constant estimation value is not adaptive for different query predicates, it may lead to large estimation discrepancies for some predicates, and (ii) the employed threshold is tuned for the current query workload and the employed dataset. As the set of queries changes and the database content evolves, this fixed value needs to be re-determined.
- The query workloads that we have employed are extensive in terms of their coverage of different selectivity levels. However, they are synthetic query workloads that are generated randomly. Hence, they may not fully reflect a real query workload's performance.

- In regular traditional histogram structures that are constructed for numeric values, histogram endpoints are sorted, and binary search is employed to efficiently locate the matching bucket endpoint(s). However, for the histograms that we employ in this study, there is no order that one can enforce on patterns. Hence, a similar binary search may not be applied during matching. Therefore, all histogram endpoints are attempted one by one for a possible match for each query predicate, which leads to spending extra time during histogram endpoint search.
- In order to build the proposed histogram structure, we first mine the positional sequential patterns. Even though this step is performed offline and takes place before the query optimization time possibly on a different machine than the production server, pattern mining is a resource-intensive step.
- Finally, this study mostly focuses on selectivity estimation accuracy. However, the effect of accuracy improvements on the actual query execution plans and database performance is not explored. Concentrating purely on prediction accuracy values may be misleading, as the suggested selectivity estimation improvements may not directly translate to database performance enhancements at similar levels.

6. Conclusion & future work

In this paper, we propose a new approach to estimate the selectivity of SQL LIKE query predicates. To this end, we introduce a new type of sequence patterns called *positional sequence patterns*, and extend a regular sequence mining algorithm to compute positional sequence patterns. A histogram is built on top of the mined positional sequence patterns during database statistics gathering time, and then this histogram is later employed during query optimization time to compute the estimated selectivities. In order to increase the coverage of histograms, we introduce an information content-based redundant pattern elimination approach. Besides, to take advantage of partial matches between histogram endpoints and query predicate strings, we also propose a partitioning-based matching algorithm. We assess the proposed techniques on a real dataset from DBLP, and demonstrate that our methods significantly outperform the state of the art in terms of selectivity estimation accuracy.

As part of future work, we would like to investigate the following directions:

- One future research direction is to parallelize the computation of positional sequential patterns. Parallel processing may improve the efficiency and scalability of extracting positional sequence patterns from massive datasets so that expert systems that work on big data may more efficiently benefit from the proposed positional sequence patterns. To this end, Apache Spark- and Hadoop-based distributed mining algorithms will be investigated.
- Our proposed positional sequence patterns are mined based on only the concept of the frequency. On the other hand, expert systems that are employed in the retail business may be interested in customer's buying choices that are both frequent as well as bring high profits. In this setting, *profit* may be included in the mining process as an additional constraint in addition to frequency to mine frequent and high-profit sequence patterns in a customer transaction database.
- Besides, one may consider incorporating a "*confidence*" metric into the positional sequence pattern mining process, as targeting high-profit patterns may not provide a complete picture of whether items in those high-profit patterns are tightly associated or not. *Confidence* will provide a measure of the conditional probability that will supply expert systems with *positional buying rules* in the form of {item1, item2} → {item3} similar to traditional association rules with additional positional consideration for items.
- Query optimizers traditionally keep their knowledgebase small to decrease the memory space requirements and increase the scalability of database management systems to handle multiple queries simultaneously. Therefore, effectively selecting the limited histogram bucket endpoint values from the pool of all mined patterns is critical. One may consider analyzing the past query workloads to assign a "utility score" to each positional sequence pattern, and then prioritize the patterns with high utility score for inclusion in the limited histogram buckets in the knowledgebase of the query optimizer. This may improve the accuracy of the query optimizer's predictions.
- Another task that query optimizers perform while automatically building a query execution plan is to determine the order of joining different tables in a given query. A new expert system may be designed to analyze a company database's query workload and mine positional sequence patterns in join orders of past query execution plans. Next, these join order patterns are stored in the knowledgebase of the query optimizer, and they are used to quickly decide on the join order of a new future query. Such a pattern-based approach may reduce the compilation time and resources that the query optimizer consumes during query compilation.
- In this work, we focus on the accuracy of the selectivity estimations. Another future direction would be incorporating these estimations into the query optimizer of a well-known database management system (e.g., Postgres, Oracle, etc.). Then, the effect of improved prediction accuracy on overall database performance may be empirically assessed on a real query workload benchmark such as TPC-H, TPC-DS, etc.
- Incremental maintenance (Chakkappen et al., 2008) of histograms and mined sequential patterns may save a considerable amount of time by eliminating the unnecessary recomputation of positional sequence patterns over the whole database. Therefore, developing an incremental maintenance model for pattern-based histograms may be another promising research direction.
- Usually, query optimizers populate their knowledgebase based on a sample taken from the whole database. Determining the minimum sampling percentage without harming the representative power of the mined sequential patterns is another open research problem.
- Finally, further research is needed to adaptively set the estimation value for no match cases based on the underlying predicate, and the current histogram and database content for more robust selectivity estimation.

CRediT authorship contribution statement

Mehmet Aytimur: Data curation, Formal analysis, Investigation, Methodology, Software, Validation, Writing - original draft, Writing - review & editing. **Ali Cakmak:** Conceptualization, Formal analysis, Funding acquisition, Investigation, Methodology, Project administration, Resources, Supervision, Writing - original draft, Writing - review & editing.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This research is in part funded by TUBITAK (Scientific and Technological Research Council of Turkey) under grant number 117E086.

References

Agrawal, R., & Srikant, R. (1995, March). Mining sequential patterns. In ICDE (Vol. 95, pp. 3-14).

Ayres, J., Flannick, J., Gehrke, J., & Yiu, T. (2002). Sequential pattern mining using a bitmap representation. *In Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 429–435). ACM..

Aytimur, M., & Çakmak, A. (2018). Estimating the selectivity of LIKE queries using pattern-based histograms. *Turkish Journal of Electrical Engineering & Computer Sciences, 26*(6), 3319–3334.

Bailey, T. L., Johnson, J., Grant, C. E., & Noble, W. S. (2015). The MEME suite. *Nucleic Acids Research, 43*(W1), W39–W49.

Carmona, J., van Dongen, B., Solti, A., & Weidlich, M. (2018). *Conformance Checking*. Cham: Springer.

Carmona, J. (2019). Decomposed Process Discovery and Conformance Checking. *Encyclopedia of Big Data*. Technologies.

Chaitanya, K. V. (2019). *Important Databases Related to Genomes. In Genome and Genomics* (pp. 261–269). Singapore: Springer.

Chakkappen, S., Cruanes, T., Dageville, B., Jiang, L., Shaft, U., Su, H., & Zait, M. (2008). Efficient and scalable statistics gathering for large databases in Oracle 11g. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data* (pp. 1053–1064). ACM.

Chaudhuri, S., Ganti, V., & Gravano, L. (2004). Selectivity estimation for string predicates: Overcoming the underestimation problem. *In Proceedings. 20th International Conference on Data Engineering* (pp. 227–238). IEEE.

Chen, Y., & Wu, Y. (2018). On the string matching with k mismatches. *Theoretical Computer Science, 726*, 5–29.

Cover, T. M., & Thomas, J. A. (2012). *Elements of information theory*. John Wiley & Sons.

Fournier-Viger, P., Wu, C. W., Gomariz, A., & Tseng, V. S. (2014). In *May). VMSP: Efficient vertical mining of maximal sequential patterns* (pp. 83–94). Cham: Springer.

Fournier-Viger, P., Lin, J. C. W., Kiran, R. U., Koh, Y. S., & Thomas, R. (2017). A survey of sequential pattern mining. *Data Science and Pattern Recognition, 1*(1), 54–77.

Gómez López, M. T., Borrego Núñez, D., Carmona, J., & Martínez Gasca, R. (2016). Computing alignments with constraint programming: The acyclic case. Org. *In Proceedings of the International Workshop on Algorithms & Theories for the Analysis of Event Data 2016: Torun.* (pp. 96–110).

Gupta, S., & Garg, D. (2016). Selectivity estimation of range queries in data streams using micro-clustering. *International Arab Journal of Information Technology (IAJIT), 13*(4).

Hasan, S., Thirumuruganathan, S., Augustine, J., Koudas, N., & Das, G. (2019). Multi-attribute selectivity estimation using deep learning. https://arxiv.org/abs/1903.09999.

Jagadish, H. V., Koudas, N., Muthukrishnan, S., Poosala, V., Sevcik, K. C., & Suel, T. (1998, August). Optimal histograms with quality guarantees. In VLDB (Vol. 98, pp. 24-27).

Jagadish, H. V., Ng, R. T., & Srivastava, D. (1999, May). Substring selectivity estimation. In PODS (Vol. 99, pp. 249-260).

Jagadish, H. V., Kapitskaia, O., Ng, R. T., & Srivastava, D. (2000). One-dimensional and multi-dimensional substring selectivity estimation. *The VLDB Journal—The International Journal on Very Large Data. Bases, 9*(3), 214–230.

Jalili, V., Matteucci, M., Goecks, J., Deldjoo, Y., & Ceri, S. (2018). Next generation indexing for genomic intervals. *IEEE Transactions on Knowledge and Data Engineering, 31*(10), 2008–2021.

Jin, L., & Li, C. (2005). In *Selectivity estimation for fuzzy string predicates in large data sets* (pp. 397–408). VLDB Endowment.

Kim, Y., Woo, K. G., Park, H., & Shim, K. (2010, March). Efficient processing of substring match queries with inverted q-gram indexes. In 2010 IEEE 26th International Conference on Data Engineering (ICDE 2010) (pp. 721-732). IEEE.

Kociumaka, T., Radoszewski, J., & Starikovskaya, T. (2019). Longest common substring with approximately k mismatches. *Algorithmica, 81*(6), 2633–2652.

Krishnan, P., Vitter, J. S., & Iyer, B. (1996). June). Estimating alphanumeric selectivity in the presence of wildcards. *ACM SIGMOD Record, 25*(2), pp. 282–293). ACM.

Layer, R. M., Pedersen, B. S., DiSera, T., Marth, G. T., Gertz, J., & Quinlan, A. R. (2018). GIGGLE: A search engine for large-scale integrated genome analysis. *Nature Methods, 15*(2), 123.

Le, B., Duong, H., Truong, T., & Fournier-Viger, P. (2017). FCloSM, FGenSM: Two efficient algorithms for mining frequent closed and generator sequences using the local pruning strategy. *Knowledge and Information Systems, 53*(1), 71–107.

Lee, H., Ng, R. T., & Shim, K. (2007). In *September). Extending q-grams to estimate selectivity of string matching with low edit distance* (pp. 195–206). VLDB Endowment.

Lee, H., Ng, R. T., & Shim, K. (2009). March). Approximate substring selectivity estimation. *In Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology* (pp. 827–838). ACM.

Li, D., Zhang, Q., Liang, X., Guan, J., & Xu, Y. (2015). Selectivity estimation for string predicates based on modified pruned count-suffix tree. *Chinese Journal of Electronics, 24*(1), 76–82.

Lin, X., Zeng, X., Liu, J., & Chen, W. (2017). Cardinality estimation applying micro self-tuning histogram. *International Journal of Innovative Computing, Information and Control, 13*(4), 1077–1094.

Mazeika, A., Böhlen, M. H., Koudas, N., & Srivastava, D. (2007). Estimating the selectivity of approximate string queries. *ACM Transactions on Database Systems (TODS), 32*(2), 12.

Muralikrishna, M., & DeWitt, D. J. (1988). *Equi-depth histograms for estimating selectivity factors for multi-dimensional*. Management of Data: In Proc. Int'l Conf.

Papadopoulos, S., Datta, K., Madden, S., & Mattson, T. (2016). The TileDB array data storage manager. *Proceedings of the VLDB Endowment, 10*(4), 349–360.

Pasquier, N., Bastide, Y., Taouil, R., & Lakhal, L. (1999). In *Discovering frequent closed itemsets for association rules* (pp. 398–416). Berlin, Heidelberg: Springer.

Pei, J., Han, J., Mortazavi-Asl, B., Wang, J., Pinto, H., Chen, Q., ... Hsu, M. C. (2004). Mining sequential patterns by pattern-growth: The prefixspan approach. *IEEE Transactions on Knowledge and Data Engineering, 16*(11), 1424–1440.

Pokou, Y. J. M., Fournier-Viger, P., & Moghrabi, C. (2016). Authorship attribution using small sets of frequent part-of-speech skip-grams. *In The Twenty-Ninth International Flairs Conference*.

Poosala, V., Haas, P. J., Ioannidis, Y. E., & Shekita, E. J. (1996). June). Improved histograms for selectivity estimation of range predicates. In *ACM Sigmod. Record (Vol., 25*(2), pp. 294–305). ACM.

Raju, G. T., & Murthy, V. (2017). March). Selectivity estimation in web query optimization. *In Proceedings of the Second International Conference on Internet of things, Data and Cloud Computing* (p. 200). ACM..

Rozinat, A., & Van der Aalst, W. M. (2008). Conformance checking of processes based on monitoring real behavior. *Information Systems, 33*(1), 64–95.

Schweizer, D., Zehnder, M., Wache, H., Witschel, H. F., Zanatta, D., & Rodriguez, M. . In *December). Using consumer behavior data to reduce energy consumption in smart homes: Applying machine learning to save energy without lowering comfort of inhabitants* (pp. 1123–1129). IEEE.

Shin, J. H. (2018). Novel Selectivity Estimation Strategy for Modern DBMS. https://arxiv.org/abs/1806.08384

To, H., Chiang, K., & Shahabi, C. (2013). October). Entropy-based histograms for selectivity estimation. *In Proceedings of the 22nd ACM international conference on Information & Knowledge Management* (pp. 1939–1948). ACM..

van der Aalst, W. M., Reijers, H. A., Weijters, A. J., van Dongen, B. F., De Medeiros, A. A., Song, M., & Verbeek, H. M. W. (2007). Business process mining: An industrial application. *Information Systems, 32*(5), 713–732.

Wang, J., Han, J., & Pei, J. (2003). August). Closet+: Searching for the best strategies for mining frequent closed itemsets. *In Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 236–245). ACM..

Wang, J., & Han, J. (2004, March). BIDE: Efficient mining of frequent closed sequences. In Proceedings. 20th international conference on data engineering (pp. 79-90). IEEE.

Yan, X., Han, J., & Afshar, R. (2003). Mining closed sequential patterns in large databases. *SDM SIAM.*.

Yang, Y., Zhang, W., Zhang, Y., Lin, X., & Wang, L. (2019). Selectivity Estimation on Set Containment Search. *Data Science and Engineering, 4*(3), 254–268.

Yao, B., Li, F., Hadjieleftheriou, M., & Hou, K. (2010, March). Approximate string search in spatial databases. In 2010 IEEE 26th International Conference on Data Engineering (ICDE 2010)(pp. 545-556). IEEE.

Zaki, M. J., & Hsiao, C. J. (2000). An efficient algorithm for closed itemset mining. In 0-Porc. SIAM Int. Conf. Data Mining, Arlington, VA.

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