Volume No:3, Issue No:9 (February-2018) ISSN No : 2454-423X (Online)

International Journal of Research in Advanced Computer Science Engineering

A Peer Reviewed Open Access International Journal

www.ijracse.com

Efficient Pattern Based Aggregation on Sequence Data

Gorle Rohit
Department Computer Science and Engineering,
Gitam (Deemed to be University)
Vishakhapatnam, Andhra Pradesh - 530045, India.

Panuganti Mahendra Amaranth
Department Computer Science and Engineering,
Gitam (Deemed to be University)
Vishakhapatnam, Andhra Pradesh - 530045, India.

Nirman Jajjari
Department Computer Science and Engineering,
Gitam (Deemed to be University)
Vishakhapatnam, Andhra Pradesh - 530045, India.

Abstract:

A Sequence OLAP (S-OLAP) system provides a platform on which pattern-based aggregate (PBA) queries on a sequence database are evaluated. In its simplest form, a PBA query consists of a pattern template T and an aggregate function F. A pattern template is a sequence of variables, each is defined over a domain. Each variable is instantiated with all possible values in its corresponding domain to derive all possible patterns of the template. Sequences are grouped based on the patterns they possess. The answer to a PBA query is a sequence cuboid (s-cuboid), which is a multidimensional array of cells. Each cell is associated with a pattern instantiated from the query's pattern template. The value of each s-cuboid cell is obtained by applying the aggregate function F to the set of data sequences that belong to that cell. Since a pattern template can involve many variables and can be arbitrarily long, the induced s-cuboid for a PBA query can be huge. For most analytical tasks, however, only iceberg cells with very large aggregate values are of interest. This paper proposes an efficient approach to identifying and evaluating iceberg cells of s-cuboids. Experimental results show that our algorithms are orders of magnitude faster than existing approaches.

INTRODUCTION

Sequence data is ubiquitous. Examples include workflow data, data streams and RFID logs. Techniques for processing various kinds of sequence data have been studied extensively in the literature (e.g., [12, 13, 10, 1, 3, 14]). Recently, issues related to ware- housing and online analytical processing (OLAP) of archived sequence data (e.g., stock ticks archive, passenger traveling histories) have received growing attentions [7, 6, 9]. In particular, [9] developed a sequence OLAP system (called S-OLAP) that efficiently supports various kinds of pattern-based aggregate queries.

While traditional OLAP systems group data tuples based on their attribute values, an S-OLAP system groups sequences based on the patterns they possess. Common aggregate functions such as COUNT/SUM/AVG can then be applied to each group. The result- ing aggregate values form the cells of a so-called sequence data cuboid, or s-cuboid.

Since an s-cuboid displays the aggregate values of sequences that are grouped by the patterns they possess, one can view an s-cuboid as the answer to a pattern-based aggregate (PBA) query. To illustrate PBA queries and s-cuboids, let us consider the sequence data set shown in Figure 1. The dataset models a collection of passenger traveling records registered by the Washington DC's metro system. The records are captured electronically by SmarTrip, which is an RFID-card-based stored-value e-payment system.

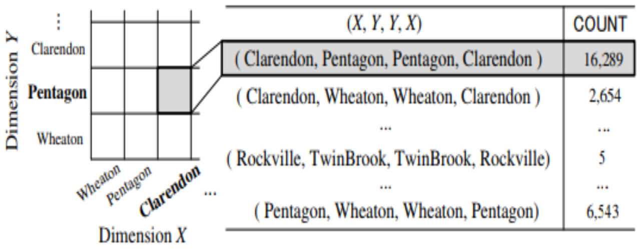
Each row in Figure 1 shows a sequence of passenger events. An event consists of a number of attributes, such as Time, Station, Action and Amount. For example, the event [t9; Wheaton; exit; 1.9] of passenger s623 indicates that the passenger exited Wheaton Station at time t9 and paid $1.9 for the trip.

Cite this article as: Gorle Rohit, Panuganti Mahendra Amaranth & Nirman Jajjari, "Efficient Pattern Based Aggregation on Sequence Data", International Journal of Research in Advanced Computer Science Engineering, Volume 3 Issue 9, 2018, Page 1-7.

Volume No:3, Issue No:9 (February-2018) ISSN No : 2454-423X (Online)
International Journal of Research in Advanced Computer Science Engineering
A Peer Reviewed Open Access International Journal
www.ijracse.com

Passenger (Sequence) ID	*Event Sequence*
...	...
s28	⟨ [t_4;Clarendon;enter;0], [t_7;Pentagon;exit;1.9], [t_9;Pentagon;enter;0],[t_{10};Clarendon;exit;1.9] ⟩
...	...
s623	⟨ [t_1;Pentagon;enter;0], [t_9;Wheaton;exit;1.9] ⟩
...	...

Figure 1: An example sequence data set — Passenger traveling log of Washington DC's metro



(X, Y, Y, X)	COUNT
(Clarendon, Pentagon, Pentagon, Clarendon)	16,289
(Clarendon, Wheaton, Wheaton, Clarendon)	2,654
...	...
(Rockville, TwinBrook, TwinBrook, Rockville)	5
...	...
(Pentagon, Wheaton, Wheaton, Pentagon)	6,543

Figure 2: A PBA query and its s-cuboid

Figure 2 shows a PBA query "(X, Y, Y, X), COUNT" and a few cells of the resulting s-cuboid. A PBA query "T , F " consists of two components: A pattern template T (e.g., (X, Y, Y, X)) and an aggregate function F (e.g., COUNT). A pattern template is a sequence of pattern symbols (e.g., X, Y) defined over an attribute of event records. The pattern symbols are instantiated by the values of to generate various patterns. Data sequences are grouped based on the patterns. Finally, the function F is applied to each sequence group to derive aggregate values.

For example, the pattern template (X, Y, Y, X) defined on the Station attribute specifies that passenger sequences are grouped together if they have traveled round-trip between stations X and Y (i.e., he first entered station X and exited station Y in his first trip, and then entered station Y and come back to station X in the next). The symbols X and Y are instantiated with various station names to form patterns, such as (Clarendon, Pentagon, Pentagon, Clarendon). Data sequences that possess a given pattern are grouped into a cell1. Each data sequence gives a value (or measure) to be aggregated. For example, a passenger sequence could be associated with the amount of fare paid, or simply '1' if we only care about the cardinality of a cell. The aggregate function F is then applied to the values of the sequences of each cell to obtain an aggregate value of the cell. In this paper we use C(P) to denote the cell of a pattern P (i.e., C(P) = the set of sequences containing pattern P), and we use F (C(P)) to denote the aggregate value of the cell. For example, Figure 2 shows that there are 16,289 sequences that possess the pattern (Clarendon, Pentagon, Pentagon, Clarendon). In our notation: COUNT(C((Clarendon, Pentagon, Pentagon, Clarendon))) = 16,289. We write P T if pattern P is an instantiation of the template T . (e.g., (Clarendon, Pentagon, Pentagon, Clarendon) (X, Y, Y, X)). An s-cuboid consists of all the aggregate values of the cells derived from all possible instantiations of the pattern template. A PBA query (e.g., "(X, Y, Y, X), COUNT") is evaluated by computing all the cells of the corresponding s-cuboid (e.g., all the cells and their counts listed in Figure 2).

In [9], a basic implementation of an S-OLAP system is presented. In that study, data sequences were indexed by inverted lists. Given a pattern P , its inverted list L[P] is a list of sequence id's such that each sequence s listed in L[P] contains the pattern P . An s-cuboid cell C(P) can thus be represented by the inverted list L[P]. The inverted list of a pattern P can be obtained by either (1) scanning the data sequences and checking which sequences contain P , or (2) joining the lists of P 's sub-patterns. For example, con- sider the query pattern template (X, Y, Z, X). To materialize an s-cuboid cell, say, C((a, b, c, a)), one can intersect (or "join") the inverted lists L[(a, b, c)] and L[(c, a)] (if these lists are available). This is because a sequence that contains the pattern (a, b, c, a) must contain the sub-patterns (a, b, c) and (c, a).

In some cases, the computation of a full s-cuboid could be ex- pensive. This is especially true when the pattern template is long with many pattern symbols, which results in a high-dimensional s- cuboid with large numbers of cells. We note that in many cases, computing the full s-cuboid is not necessary. More often, a user is interested in only those cells of an s-cuboid that return very large aggregate values. For example, a marketing manager of the Metro company may be interested in the pairs of stations for which most people commute roundtrip in order to design a fare and discount structure strategically. As another example, an online store manager may want to know what products X, Y , Z give high visiting counts of the product webpage visiting pattern (X, Y, Z, X). This pattern reveals that a customer interested in product X is likely to compare it against products Y and Z, but will eventually commit to X [2-8].

Given a pattern template T , an aggregate function F , and a user- specified threshold σ, our objective is to compute the iceberg cells, which are those whose aggregate values exceed the threshold σ.

We call the query "T , F , σ" an Iceberg Pattern-Based Aggregate Query (or IPBA query). Formally,

DEFINITION 1. (IPBA Query) The answer to the IPBA query "T , F , σ" is the set of all iceberg cells and their aggregate values, i.e., {(P, F (C(P))) | (P € T) ∧ (F (C(P)) ≥ σ)}.
One straightforward way to answer an IPBA query is to compute the full s-cuboid of the PBA query and return only the iceberg cells. In [9], two methods for computing full s-cuboids, namely, the counter-based method (CB) and the inverted-list method (II), are studied. The CB method scans the relevant sequences in the database to compute the cells' aggregate values in batch, while the II method computes the s-cuboid using list joining. Both of these methods could be expensive for very large sequence databases. For example, computing an inverted list requires I/O (to retrieve sub-patterns' inverted lists) and CPU processing (to join the sub- patterns' lists) [10]. Yet, most of these costs are wasted since the majority of cells are non-iceberg ones. In this paper, we propose statistical estimation techniques that allow very efficient identification and computation of iceberg cells. Our idea is to retain a very small synopsis of the database in main memory. Through statistical tests, the synopsis allows us to decide whether a cell is iceberg or not and whether the decision meets a given significance level requirement.

For the identified iceberg cells, we estimate their aggregate values based on the small synopsis and check whether the estimated values satisfy an accuracy requirement with a high confidence requirement. Through this mechanism, we show that we are highly confident that the reported cells are all and only iceberg cells and their reported aggregate values are highly accurate. We remark that our approach results in a very efficient method of answering IPBA queries. This is because we avoid heavy I/O (by not accessing disk-resident data) and reduce CPU processing [3] (by processing the small synopsis instead of scanning big data sequences or joining large inverted lists).

EXISTING SYSTEM:

The algorithms thus perform disk-based list-joining to compute the exact counts of those few cells. The exact number of cells that require such disk-based joining depends on how many cells whose counts are close to the threshold. It thus varies from case to case.

The curves for thus go up and down. However, in general, for a skewed dataset (such that a few cells have high populations and most cells have low populations) there are more cells with low counts. So when decreases, the chances of having *some* cells close to the threshold *σ* increase.

DISADVANTAGE:

A big advantage of using the sampling technique given in is that it allows very efficient updates (inserts and deletes of sequences). For example, consider that a set D of sequences is inserted into the sequence database.

The system architecture of our S-OLAP implementation for answering IPBA queries. A general SOLAP system should be able to answer general PBA queries and to support a set of S-OLAP operations.

PROPOSED SYSTEM:

In that preliminary version, we proposed a system architecture and a synopsis based algorithm (SBA) [11] to answer IPBA queries [14]. The core idea is to retain a very small synopsis of the database in main memory.

Through statistical tests, the synopsis allows us to decide whether a cell is iceberg or not for a given significance level θ. For each potential iceberg cell, we estimate its aggregate value based on the synopsis and check whether the estimate is accurate to within a small error tolerance threshold with a confidence exceeding a confidence threshold

ADVANTAGE

The theorems also allow us to estimate a synopsis size with which a given accuracy requirement is guaranteed to be satisfied. From this observation, we devised two algorithms, namely, SBA and SBA+. While both are orders of magnitude more efficient than previous approaches, SBA+ has the advantage of reducing I/O and CPU costs in cases where the accuracy test cannot be satisfied by some iceberg cells.

Also, SBA+ estimates a synopsis size that is sufficient to guarantee accuracy. Through this estimate, SBA+ avoids excessive synopsis processing when the synopsis is set too large. Overall, SBA+ provides the better performance compared with SBA [14].

SYSTEM ARCHITECTURE

Figure 4 shows the system architecture of our S-OLAP implementation for answering IPBA queries. We remark that a general S-OLAP system should also be able to answer general PBA queries and to support a set of S-OLAP operations3 [2]. Since we focus on evaluating IPBA queries in this paper, only components that are relevant to IPBA query processing are shown in Figure 4.

In our system, a set of data sequences S is stored on secondary storage. As the S-OLAP system operates and answers queries (PBA or IPBA), certain inverted indices and inverted lists are materialized. These materialized indices (lists) are stored in an inverted index store (II store). The II store serves as a disk-resident cache of the previously materialized lists. A replacement policy is employed by the system to control the II store's content when the II store overflows4. To compute an s-cuboid cell C(P) (for example, in answering a PBA/IPBA query), the query engine could consult the II store and check if the inverted list L[P] is present (i.e., materialized). If so, L[P] could be retrieved for computing the aggregate value F (C(P)). If L[P] is not present in the II store, it is materialized by joining the lists of P 's sub-patterns. These sub- patterns' lists are retrieved from the II store if they are present, or are recursively constructed otherwise. We assume that the indices of all length-2 pattern templates are materialized in the II store. We call this set of inverted indices the core index CI, which is always present in the II store. This approach is similar to bigram indexing in document retrieval systems and is shown to be effective in PBA query processing [9].

To speed up IPBA query processing, we should avoid disk accesses, such as in retrieving lists from the II store. We achieve this by maintaining a synopsis S˜ in main memory, which is a small sample of the sequence dataset S. Accompanying S˜ is the synop- sis' II store (SII store), also stored in main memory. The SII store is similar to the disk-resident II store except that inverted lists in the SII store contain the id's of only those sequences found in the synopsis S˜. We use L˜[P] to denote such a list of the pattern P .

In other words, a posting (si : p1, . . . , pfi) is in L˜[P] iff the sequence si contains P at starting positions p1, . . ., pfi and si S˜. Moreover, while the core index CI of the dataset S must be present in the disk-resident II store, we do not assume the presence of any particular inverted lists in the SII store. The SII store is simply a fixed-size temporary cache of previously materialized inverted lists of the sequences in the synopsis.

Given an IPBA query "T , F , σ", for each P T , we estimate the aggregate value F (C(P)) by processing the synopsis and the SII store. Our objectives are:
1. Derive statistical tests that decide whether C(P) is or is not an iceberg cell. The decision of the tests has to satisfy a significance level threshold θ.
2. For each cell C(P) declared iceberg by the tests, we estimate F (C(P)). The estimate has to be accurate to within an error tolerance threshold g and the estimation has to exceed a confidence threshold α.
We remark that (θ, g, α) could be system-wise parameters or could be specified by the user of each IPBA query. We call = (σ, θ, g, α) the statistical requirement of an IPBA query. An interesting feature of our system is that given , we can mathematically determine how big S˜ should be in order to meet the requirement. This information is very useful in designing the S-OLAP system because it allows us to decide how much memory the system needs to store an effective synopsis.

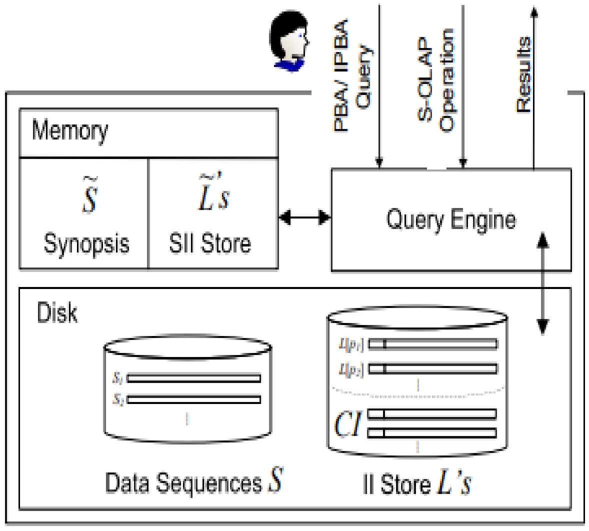


Figure 4: Architecture

IPBA QUERY EVALUATION

In this section we first discuss how to obtain a random sample S˜ from the sequence dataset S. Next, we describe our synopsis- based algorithm (SBA), which answers IPBA queries with results that satisfy the queries' statistical requirements R's.

Sampling

We obtain the synopsis S˜ by drawing uniform random samples from S. Given an amount of memory for storing the synopsis, we determine a budget B, which is the number of sequences in the synopsis that the memory can hold. For example, if 100 million sequences occupy 250GB of disk space, then 500MB of memory for the synopsis gives a budget B of 200,000 sequences. We adopt the sampling technique proposed in [8] to obtain S˜. First, we randomly pick a hash function h : S ›→ [0, 1] from a family of universal hash functions H. Let {s1, . . . , s|S|} be the set of all data sequence id's. The hash values, h(s1), . . . , h(s|S|), form an i.i.d. sequence of the uniform distribution over the range [0,1]. To obtain a size-B sample of S, we collect into S˜ all sequences in S whose ids' hash values are ≤ x, where x = B/|S|. That is, S˜ = {si ∈ S|h(si) ≤ x}. By expectation, |S˜| = B and so x = |S˜|/|S|.

The SBA Algorithm

Given an IPBA query "T , F , σ" and its statistical requirement R = (σ, θ, g, α), our algorithm, SBA, needs to return the aggregate values of the iceberg cells of an s-cuboid. Also, the reported results should satisfy R. Given a pattern P ∈ T , SBA uses L˜[P], which is the inverted list of P for the synopsis S˜, to decide if the cell C(P) is iceberg and if so, to compute the cell's aggregate value F (C(P)). In this process, SBA accesses the SII store (Figure 4) to retrieve L˜[P]. For those cells C(P)'s whose lists L˜[P]'s are not found in the SII store, the small synopsis S˜ is scanned once to build the missing L˜[P]'s. In the following discussion, we assume that L˜[P] has been made available (either by retrieval from the SII store or by construction from S˜). To simplify our discussion, we only consider the COUNT aggregate function in this paper. Other functions, such as SUM and AVG, can be similarly handled [12].

Let DP be the count of the cell C(P), i.e., DP = L[P] . SBA computes an estimate D˜P of DP based on L˜[P]. SBA then con- ducts three tests to evaluate if the cell

C(P) is iceberg, and if so, whether the estimate D˜P is accurate enough. Figure 5 abstracts SBA's logic. It involves the following steps: (1) Test if we can reject the hypothesis "H1: C(P) is an iceberg cell," with a significance level θ. If so, discard the cell. (2) Otherwise, test if we can reject the hypothesis "H2: C(P) is not an iceberg cell," with a significance level θ. If we cannot reject H2 (and because we failed to reject H1), the synopsis is insufficient for us to determine if C(P) is iceberg or not. In this case, SBA computes the exact count of C(P) by reverting to the disk-based list-joining algorithm (see Section 2). (3) If H2 is rejected, then SBA tests if the estimate D˜P satisfies the error bound g with confidence α. If so, C(P) is reported as an iceberg cell with count D˜P . If the error tolerance requirement is not met, SBA again reverts to the disk-based algorithm to compute the exact count.

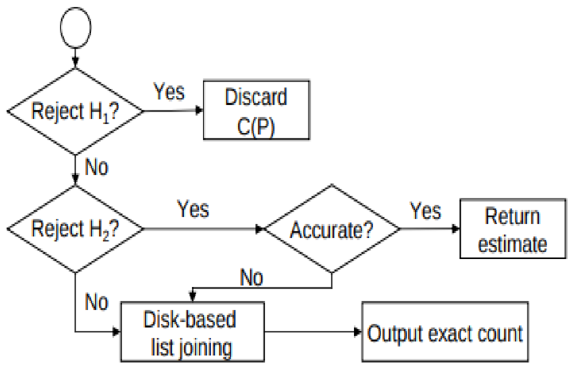


Figure 5: Algorithm flowchart

Algorithm 1 The SBA algorithm

```
Input: IPBA query "$T, F, \sigma$" with requirement $\mathcal{R} = (\sigma, \theta, \epsilon, \alpha)$
 1: for all $P \vdash T$ do
 2:     obtain $\tilde{L}[P]$ either from the SII store or by scanning $\tilde{S}$
 3: end for
 4: for all $P \vdash T$ do
 5:     if $\overline{PH}_1(P) \leq 1 - \theta$ then
 6:         Continue
 7:     else if $(\overline{PH}_2(P) \leq 1 - \theta) \wedge (\Upsilon'_P \geq \alpha)$ then
 8:         Output $C(P)$ and its estimated count $\tilde{D}_P = \frac{k_P - 1}{U_{k_P}}$
 9:     else
10:         Compute the true count $D_P$ by disk-based list joining
11:         if $D_P \geq \sigma$ then
12:             Output $C(P)$ and $D_P$
13:         end if
14:     end if
15: end for
```

CONCLUSION

In this paper we studied the problem of answering iceberg pattern- based aggregate (IPBA) queries. We put forward a synopsis-based solution, which samples and stores a small synopsis of a sequence database in main memory. We devised three statistical tests that process the synopsis to confidently classify a cell as iceberg or non- iceberg, and to confidently compute aggregate estimates of the ice- berg cells. Experimental study shows that our proposed algorithm outperforms the existing algorithms in order of magnitude.

REFERENCES

[1]B. Babcock, et al. Models and issues in data stream systems. In PODS, 2002.

[2]K. Beyer, P. J. Haas, B. Reinwald, Y. Sismanis, and R. Gemulla. On synopses for distinct-value estimation under multiset operations. In SIGMOD, pages 199–210, 2007.

[3]J. Chen, et al. NiagaraCQ: a scalable continuous query system for internet databases. SIGMOD Rec., 2000.

[4]C. K. Chui, B. Kao, E. Lo, and R. Cheng. I/O-efficient algorithms for answering pattern-based aggregate queries in a sequence OLAP system. In CIKM, pages 1619–1628, 2011.

[5]M. Fang, N. Shivakumar, H. Garcia-Molina, R. Motwani, and J. D. Ullman. Computing iceberg queries efficiently. In VLDB, pages 299–310, 1998.

[6]H. Gonzalez, J. Han, and X. Li. FlowCube: Constructing RFID FlowCubes for Multi-Dimensional Analysis of Commodity Flows. In VLDB, 2006.

[7]H. Gonzalez, J. Han, X. Li, and D. Klabjan. Warehousing and Analyzing Massive RFID Data Sets. In ICDE, 2006.

Volume No:3, Issue No:9 (February-2018) ISSN No : 2454-423X (Online)

International Journal of Research in Advanced Computer Science Engineering

A Peer Reviewed Open Access International Journal

www.ijracse.com

[8]M. Hadjieleftheriou, X. Yu, N. Koudas, and D. Srivastava. Hashed samples: selectivity estimators for set similarity selection queries. PVLDB, 1(1):201–212, 2008.

[9]E. Lo, B. Kao, W.-S. Ho, C.-K. Chui, and D. Cheung. OLAP on sequence data. In SIGMOD, pages 649–660, 2008.

[10]R. Ramakrishnan, D. Donjerkovic, A. Ranganathan, K. S. Beyer, and M. Krishnaprasad. SRQL: Sorted Relational Query Language. In SSDBM, 1998.

[11]R. Sadri, C. Zaniolo, A. Zarkesh, and J. Adibi. Optimization of sequence queries in database systems. In PODS, 2001.

[12]P. Seshadri, M. Livny, and R. Ramakrishnan. Sequence query processing. In SIGMOD, 1994.

[13]P. Seshadri, M. Livny, and R. Ramakrishnan. The design and implementation of a sequence database system. In VLDB, 1996.

[14]F. Wang, et al. Temporal management of RFID data. In VLDB, 2005.

Author Details

Gorle Rohit
Department Computer Science and Engineering,
Gitam (Deemed to be University)
Vishakhapatnam, Andhra Pradesh - 530045, India.
rohit.rohitg17@gmail.com

Panuganti Mahendra Amaranth
Department Computer Science and Engineering,
Gitam (Deemed to be University)
Vishakhapatnam, Andhra Pradesh - 530045, India.
p.amar560@gmail.com

Nirman Jajjari
Department Computer Science and Engineering,
Gitam (Deemed to be University)
Vishakhapatnam, Andhra Pradesh - 530045, India.
nirman1071997@gmail.com