Fast Evaluation of Iceberg Pattern-Based Aggregate Queries

Zhian He[*] Petrie Wong[*] Ben Kao[†] Eric Lo[*] Reynold Cheng[†]

[*]The Hong Kong Polytechnic University
{cszahe, cskfwong, ericlo}@comp.polyu.edu.hk

[†]The University of Hong Kong
{kao, ckcheng}@cs.hku.hk

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. For example, the template $T = (X,Y,Y,X)$ consists of two variables X and Y. 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 identify and evaluate iceberg cells of s-cuboids. Experimental results show that our algorithms are orders of magnitude faster than existing approaches.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications-Data Mining

Keywords

OLAP; Iceberg; Probabilistic Algorithm

1. INTRODUCTION

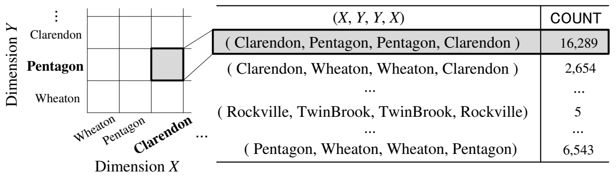
Sequence data is ubiquitous. Examples include workflow data, data streams and RFID logs. Techniques for processing various kinds of sequence data have been studied extensively in the literature (e.g., [12, 13, 10, 1, 3, 14]). Recently, issues related to warehousing and online analytical processing (OLAP) 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 resulting 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 [t_9; Wheaton; exit; 1.9] of passenger s623 indicates that the passenger exited Wheaton Station at time t_9 and paid $1.9 for the trip.

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

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 $\mathcal{A}$ of event records. The pattern symbols are instantiated by the values of $\mathcal{A}$ to generate various *patterns*. Data sequences are grouped based on

CIKM'13, Oct. 27–Nov. 1, 2013, San Francisco, CA, USA.
Copyright 2013 ACM 978-1-4503-2263-8/13/10 ...$15.00.
http://dx.doi.org/10.1145/2505515.2505726 .

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 cell[1]. 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: $\texttt{COUNT}(C(\text{(Clarendon, Pentagon, Pentagon, Clarendon)})) = 16{,}289$. We write $P \vdash T$ if pattern P is an instantiation of the template T. (e.g., (Clarendon, Pentagon, Pentagon, Clarendon) $\vdash (X, Y, Y, X)$). 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, consider 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). (More details on this list joining operation will be given in Section 2.)

In some cases, the computation of a full s-cuboid could be expensive. 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.

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.,* $\{(P, F(C(P))) \mid (P \vdash T) \wedge (F(C(P)) \geq \sigma)\}$.

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). 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 (by processing the small synopsis instead of scanning big data sequences or joining large inverted lists).

The rest of this paper is organized as follows. Section 2 reviews the basic algorithm for evaluating PBA queries based on inverted indices. Section 3 presents our system architecture. We discuss how sequence data, inverted indices, and a synopsis are stored. In Section 4, we discuss how a synopsis is constructed and present our synopsis-based algorithm for evaluating IPBA queries. Section 5 evaluates our methods through an experimental study. Section 6 discusses some related works, and finally, Section 7 concludes the paper.

2. INVERTED LISTS

In [9], two methods for computing s-cuboids are studied, namely, the counter-based method (CB) and the inverted-list method (II). It is shown that while CB is suitable for computing a full s-cuboid from scratch, the II method is more efficient if the system has already materialized and cached a significant number of inverted lists, or if only a portion of the s-cuboid cells have to be computed. As we have mentioned, our approach to answering an IPBA query is to identify iceberg cells and compute only them, the II method is more suitable. In this section we describe the inverted index structure and explain how to compute s-cuboids using inverted indices. For reference, symbols that are frequently used in our discussion are shown in Table 1.

An inverted index consists of a set of inverted lists. An inverted list, $L[P]$, is associated with a length-m pattern $P = (v_1, \ldots, v_m)$. Each element (v_i) in pattern P is a value of a chosen attribute's domain. The inverted list $L[P]$ is a list of *postings* which record the occurrences of P in the sequence dataset. Each posting is of the form $(s_i\colon\ p_1, \ldots, p_{f_i})$, where s_i is a sequence identifier, f_i

[1]A data sequence may contain more than one pattern. A sequence can, therefore, belong to multiple cells.

T	A pattern template.
$P, C(P)$	A pattern and its s-cuboid cell.
$P \vdash T$	Pattern P is an instantiation of template T.
I^T	Inverted index for template T.
$L[P]$	Inverted list of pattern P.
$S, \tilde{S}$	The set of data sequences and its synopsis.
x	$\|\tilde{S}\|/\|S\|$.
$\tilde{L}[P]$	The inverted list of pattern P w.r.t. synopsis $\tilde{S}$.
k_P, U_{k_P}	$\|\tilde{L}[P]\|$ and the largest hash value of sequences in $\tilde{L}[P]$, respectively.
$D_P, \tilde{D}_P$	The true count and the estimated count of cell $C(P)$.
σ	Iceberg threshold.
θ	Significance level requirement of hypothesis testings.
ϵ	Error tolerance threshold.
α	Confidence threshold of aggregate value estimation.

Table 1: Frequently used symbols

$I^{(X,Y,Y)}$
l_1: L[(Clarendon, Pentagon, Pentagon)] = {(s2:1,4), (s27:3), (s34:1)}
⋮
l_3: L[(Pentagon, Rockville, Rockville)] = {(s4:5,8)}
l_4: L[(Pentagon, Suitland, Suitland)] = {(s15:3)}
⋮
l_5: L[(Wheaton, Pentagon, Pentagon)] = {(s23:7), (s26:1,4,12,16)}

$I^{(Y,X)}$
l_6: L[(Clarendon, Pentagon)] = {(s2:1,4), (s27:1,3,19), (s34:1)}
⋮
l_7: L[(Pentagon, Glenmont)] = {(s16:2,7,9), (s25:6), (s18:6,12)}
l_8: L[(Pentagon, Clarendon)] = {(s1:1,3), (s27:5), (s34:16), (s40:6)}
⋮
l_9: L[(Glenmont, Pentagon)] = {(s6:1), (s7:4,11), (s8:2,6,12), (s50:7)}
l_{10}: L[(Glenmont, Wheaton)] = {(s19:4,14), (s21:8)}

$I^{(X,X)}$
l_{11}: L[(Clarendon, Clarendon)] = {(s3:2,8,12), (s4:5,22), (s16:19,32)}
l_{12}: L[(Deanwood, Deanwood)] = {(s5:2,9), (s7:2,12,17)}
⋮
l_{13}: L[(Pentagon, Pentagon)] = {(s2:2,5), (s27:2,4,16), (s34:2,19)}
⋮
l_{14}: L[(Wheaton, Wheaton)] = {(s67:7,14), (s86:2,8,11)}

Figure 3: Inverted indices

is the number of occurrences of P in s_i, and $p_1, \ldots, p_{f_i}$ are the starting positions at which pattern P occurs in s_i. Given a pattern template T, the inverted index I^T is the set of inverted lists $L[P]$ such that $P \vdash T$. Figure 3 shows three example inverted indices: $I^{(X,Y,Y)}$, $I^{(Y,X)}$ and $I^{(X,X)}$. For instance, the first inverted list l_1 in $I^{(X,Y,Y)}$ indicates that sequences $s2$, $s27$ and $s34$ all contain the pattern (Clarendon, Pentagon, Pentagon)[2]. The first posting ($s2$:1,4) indicates that there are two occurrences of the pattern in sequence $s2$ at positions [1..3] and [4..6].

The answer of a PBA query "T, F" is an s-cuboid, which can be obtained from the inverted index I^T by applying the aggregate function F to each inverted list $L[P]$ in I^T. For example, given the set of inverted indices shown in Figure 3 and the PBA query "(Y, X), COUNT", the value of the cell C((Pentagon, Clarendon)) is 4 because the inverted list l_8 contains four sequences. Evaluating an s-cuboid cell of a pattern P thus requires the inverted list $L[P]$.

[2]Symbols in a pattern template are unbound variables. So, the inverted indices $I^{(X,Y)}$ and $I^{(Z,U)}$ are the same. Multiple occurrences of the same variable in a template, however, have the same binding. So, $I^{(X,X)}$ and $I^{(X,Y)}$ are different.

If $L[P]$ is not previously computed (and thus is not available), it can be materialized by joining the inverted lists of shorter patterns. For example, the inverted list $L[(v_1, v_2, v_2, v_1)]$ can be obtained by joining $L[(v_1, v_2, v_2)]$ and $L[(v_2, v_1)]$. More specifically, if a sequence s appears in a posting of $L[(v_1, v_2, v_2)]$ with a starting position i and in another posting of $L[(v_2, v_1)]$ with a starting position $i+2$, then s is recorded in a posting of $L[(v_1, v_2, v_2, v_1)]$ with a starting position i. We use $L[(v_1, v_2, v_2, v_1)] = L[(v_1, v_2, v_2)] \bowtie L[(v_2, v_1)]$ to denote this join operation. For example, the inverted list L[(Clarendon, Pentagon, Pentagon, Clarendon)] can be obtained by considering l_1 and l_8. From the lists, we see that $s27$ at position 3 is in l_1 and $s27$ at position 5 is in l_8, so the posting ($s27$:3) is added to $L[(v_1, v_2, v_2, v_1)]$.

3. 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 operations[3]. 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 overflows[4]. 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 $\tilde{S}$ in main memory, which is a small sample of the sequence dataset S. Accompanying $\tilde{S}$ is the synopsis' 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 $\tilde{S}$. We use $\tilde{L}[P]$ to denote such a list of the pattern P. In other words, a posting $(s_i : p_1, \ldots, p_{f_i})$ is in $\tilde{L}[P]$ iff the sequence s_i contains P at starting positions $p_1, \ldots, p_{f_i}$ and $s_i \in \tilde{S}$. 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.

[3]Readers are referred to [9] for a discussion of the six S-OLAP operations for s-cuboids manipulations. These S-OLAP operations are analogous to traditional OLAP operations such as slice and dice.

[4]A study on various II store replacement policies can be found in [4].

Given an IPBA query "T, F, σ", for each $P \vdash T$, 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 ϵ and the estimation has to exceed a confidence threshold α.

We remark that (θ, ϵ, α) could be system-wise parameters or could be specified by the user of each IPBA query. We call $\mathcal{R}$ = (σ, θ, ϵ, α) the *statistical requirement* of an IPBA query. An interesting feature of our system is that given $\mathcal{R}$, we can mathematically determine how big $\tilde{S}$ 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.

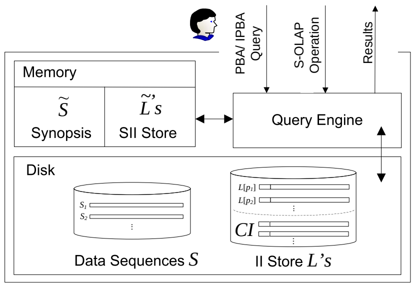


Figure 4: Architecture

4. IPBA QUERY EVALUATION

In this section we first discuss how to obtain a random sample $\tilde{S}$ 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 $\mathcal{R}$'s.

4.1 Sampling

We obtain the synopsis $\tilde{S}$ 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 $\tilde{S}$. First, we randomly pick a hash function $h : S \mapsto [0, 1]$ from a family of universal hash functions $\mathcal{H}$. Let $\{s_1, \ldots, s_{|S|}\}$ be the set of all data sequence id's. The hash values, $h(s_1), \ldots, h(s_{|S|})$, 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 $\tilde{S}$ all sequences in S whose ids' hash values are $\leq x$, where $x = B/|S|$. That is, $\tilde{S} = \{s_i \in S | h(s_i) \leq x\}$. By expectation, $|\tilde{S}| = B$ and so $x = |\tilde{S}|/|S|$.

4.2 The SBA Algorithm

Given an IPBA query "T, F, σ" and its statistical requirement $\mathcal{R}$ = (σ, θ, ϵ, α), our algorithm, SBA, needs to return the aggregate values of the iceberg cells of an s-cuboid. Also, the reported results should satisfy $\mathcal{R}$. Given a pattern $P \vdash T$, SBA uses $\tilde{L}[P]$, which is the inverted list of P for the synopsis $\tilde{S}$, 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 $\tilde{L}[P]$. For those cells $C(P)$'s whose lists $\tilde{L}[P]$'s are not found in the SII store, the small synopsis $\tilde{S}$ is scanned once to build the missing $\tilde{L}[P]$'s. In the following discussion, we assume that $\tilde{L}[P]$ has been made available (either by retrieval from the SII store or by construction from $\tilde{S}$). 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.

Let D_P be the count of the cell $C(P)$, i.e., $D_P = |L[P]|$. SBA computes an estimate $\tilde{D}_P$ of D_P based on $\tilde{L}[P]$. SBA then conducts three tests to evaluate if the cell $C(P)$ is iceberg, and if so, whether the estimate $\tilde{D}_P$ is accurate enough. Figure 5 abstracts SBA's logic. It involves the following steps: (1) Test if we can reject the hypothesis "H_1: *$C(P)$ is an iceberg cell*," with a significance level θ. If so, discard the cell. (2) Otherwise, test if we can reject the hypothesis "H_2: *$C(P)$ is not an iceberg cell*," with a significance level θ. If we cannot reject H_2 (and because we failed to reject H_1), 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 H_2 is rejected, then SBA tests if the estimate $\tilde{D}_P$ satisfies the error bound ϵ with confidence α. If so, $C(P)$ is reported as an iceberg cell with count $\tilde{D}_P$. If the error tolerance requirement is not met, SBA again reverts to the disk-based algorithm to compute the exact count.

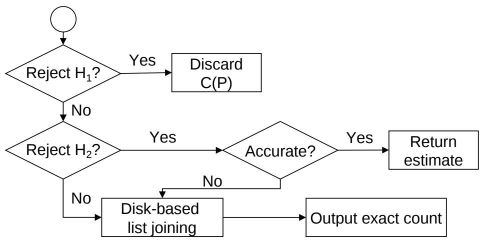


Figure 5: Algorithm flowchart

Our estimator is derived based on [2]:

THEOREM 1. *Given a pattern P, let U_r be the r-th smallest hash value of the sequences in $\tilde{L}[P]$. The quantity $(r-1)/U_r$ is an unbiased estimate of D_P. And the estimation variance decreases when r increases.*

With Theorem 1, we shall use the largest r made available by the list $\tilde{L}[P]$ for the most accurate estimation. Hence, we use $\tilde{D}_P = (k_P - 1)/U_{k_P}$, where $k_P = |\tilde{L}[P]|$ and U_{k_P} is the largest hash value in $\tilde{L}[P]$, as the estimator for D_P[5]. We consider the hypothesis:

$$H_1 : D_P \geq \sigma.$$

Given a uniform hash function h, each sequence in $L[P]$ has the same probability x (= $|\tilde{S}|/|S|$) of being included in the synopsis' inverted list $\tilde{L}[P]$ (see Section 4.1). Let k be the random variable that represents the number of sequences in $\tilde{L}[P]$. To reject H_1, we consider the following one-tail p-value:

$$PH_1(P) = P(k \leq k_P) = \sum_{k=0}^{k_P} \binom{D_P}{k} x^k (1-x)^{D_P - k}.$$

[5]We define $\tilde{D}_P$ = 0 and 1 for k_P = 0 and 1, respectively.

It can be showed that for $D_P \geq \sigma$,

$$PH_1(P) \leq \sum_{k=0}^{k_P} \binom{\sigma}{k} x^k (1-x)^{\sigma-k} = \overline{PH}_1(P). \qquad (1)$$

Note that $\overline{PH}_1(P)$ can be computed from k_P (the observed size of $\tilde{L}[P]$) and σ. Given a significance level θ, we reject hypothesis H_1 if

$$\text{Pruning Test: } \overline{PH}_1(P) \leq 1-\theta. \qquad (2)$$

We call this test the *pruning test* because if it holds we have strong evidence that the cell $C(P)$ is not iceberg. In this case, the cell is *pruned*. Otherwise, we consider another hypothesis:

$$H_2 : D_P < \sigma.$$

and the following p-value:

$$PH_2(P) : P(k \geq k_P) = \sum_{k=k_P}^{D_P} \binom{D_P}{k} x^k (1-x)^{D_P-k}.$$

It can be shown that for $D_P < \sigma$,

$$PH_2(P) \leq \sum_{k=k_P}^{\sigma} \binom{\sigma}{k} x^k (1-x)^{\sigma-k} = \overline{PH}_2(P).$$

Similar to $\overline{PH}_1(P)$, $\overline{PH}_2(P)$ can be computed from k_P and σ. We reject hypothesis H_2 if

$$\text{Accepting Test: } \overline{PH}_2(P) \leq 1-\theta. \qquad (3)$$

We call this the *accepting test* because if it holds, we have strong evidence that the cell $C(P)$ is iceberg. In this case, the cell is *accepted* and we proceed to estimate the accuracy of the estimate $\tilde{D}_P$.

Let Υ_P be the probability that the relative error of our estimate, i.e., $|(\tilde{D}_P - D_P)|/D_P$, is less than or equal to ϵ. Based on [2], we can prove the following theorem[6]:

THEOREM 2. *$\forall 0 < \epsilon < 1$ and $k_P \geq 1$,*

$$\begin{aligned}\Upsilon_P &= P(|\tilde{D}_P - D_P| \leq \epsilon D_P) \\ &= I_{u(D_P,k_P,\epsilon)}(k_P, D_P - k_P + 1) \qquad (4) \\ &\quad - I_{l(D_P,k_P,\epsilon)}(k_P, D_P - k_P + 1),\end{aligned}$$

where

$$u(D_P,k_P,\epsilon) = \frac{k_P - 1}{(1-\epsilon)D_P}, \quad l(D_P,k_P,\epsilon) = \frac{k_P - 1}{(1+\epsilon)D_P}, \text{ and}$$

$$I_d(a,b) = \left(\int_0^d t^{a-1}(1-t)^{b-1}dt\right) \Big/ \left(\int_0^1 t^{a-1}(1-t)^{b-1}dt\right)$$

is the regularized incomplete beta function.

Note that Υ_P depends on D_P, which is unknown. It is shown in [2] that the probability can be practically approximated by substituting D_P by $\tilde{D}_P$ in the beta function. Hence, we approximate Υ_P by,

$$\Upsilon'_P = I_{u(\tilde{D}_P,k_P,\epsilon)}(k_P, \tilde{D}_P - k_P + 1) - I_{l(\tilde{D}_P,k_P,\epsilon)}(k_P, \tilde{D}_P - k_P + 1).$$

Now, we report the cell $C(P)$ and its estimated count $\tilde{D}_P$ if it satisfies the *accuracy test*:

$$\text{Accuracy Test: } \Upsilon'_P \geq \alpha. \qquad (5)$$

In case $\Upsilon'_P < \alpha$, we do not have sufficient confidence in the accuracy of the estimation. In this case, the exact value of D_P is determined by the disk-based list-joining procedure. Algorithm 1 summarizes the SBA algorithm.

[6]Due to space limitation, we skip the proof of the theorem in this paper.

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
```

5. EXPERIMENT

In this section we evaluate SBA through an experimental study. The algorithms were implemented in Python and the experiments were conducted on a machine with a 2.5GHz Pentium dual core CPU and 8GB RAM running Ubuntu 11.04.

In order to study the algorithms' performance under various data characteristics, we generate synthetic data. This allows us to control various parameters, from database size to pattern selectivities. Our synthetic sequence data generator follows that of [4]. The default synopsis size is 100,000 sequences. In the experiment, the II store (Figure 4) contains only the core index *CI* and the SII store is empty. We remark that the algorithms' performance is better if more materialized inverted lists are cached in the stores. The results shown in this section are thus conservative ones of our algorithms. Below shows the parameters and their default values. We also show in the table the default settings of the query requirement.

parameter	notation	default value
Number of sequences	N	2,000,000
Average number of events per sequence	L	300
Domain size of pattern dimension	M	100
Data skewness	β	0.5
Iceberg selectivity	ξ_σ	0.005
Significance level	θ	0.95
Error tolerance	ϵ	0.1
Confidence	α	0.95
Synopsis size	$\lvert\tilde{S}\rvert$	100,000

We have conducted experiments using various pattern templates T. In this paper we show the results of two templates T_1 = (X,Y,Y,X) and T_2 = (X,Y,Z,X). As we have mentioned in the introduction, the first template is related to the round-trip query for metro data while the second template is related to the comparison-shopping query for an online store's web log data. The results of these two templates are illustrative of the algorithms' performance — the general observations drawn from these queries are found to be consistent with those of other templates we tested on the system. For reference, we have also implemented the disk-based CB algorithm [9], which computes the s-cuboid in full to identify iceberg cells. The CB algorithm generally takes more than an hour to execute on our 2-million-sequence dataset. In contrast, SBA take only seconds to a couple of minutes across our experiment settings. We do not explicitly show CB's performance in this section because it is at least an order of magnitude slower than SBA.

Varying ξ_σ. The first experiment studies the effect of the iceberg threshold σ as controlled by the iceberg selectivity $\xi_\sigma = \sigma/|S|$. Figures 6(a) and 6(b) show the execution times of SBA for the templates T_1 and T_2, respectively. Note that since the s-cuboid for T_2 is of a higher dimension than that of T_1, there are more cells to compute for T_2. The execution times shown in Figure 6(b) are therefore generally larger than those shown in Figure 6(a). Also, patterns derived from T_2 are less restrictive than those from T_1, therefore, the s-cuboid cells of T_2 are of much higher counts. To stay focus on the iceberg cells, we use a range of ξ_σ of larger values for T_2.

From the figures, we see that in general, as ξ_σ increases, the execution times decrease. This is because a larger ξ_σ gives a larger iceberg threshold σ. Hence, (1) there are more non-iceberg cells and (2) the gaps between their counts and the iceberg threshold are generally larger. These translate into (1) *more pruning opportunities* and (2) *more effective pruning* (i.e., more likely that the pruning test is satisfied), respectively. Moreover, for a larger σ, the count of an iceberg cell $C(P)$ has to be larger for $C(P)$ to be classified as iceberg. Therefore, $C(P)$ has to have *a bigger presence* in the synopsis, i.e., $\tilde{L}[P]$ has to be larger. This allows D_P to be more accurately estimated and so the accuracy test is easier to satisfy. In summary, a bigger ξ_σ gives us more effective tests and so there are fewer cases for which we need to compute the cells' exact counts.

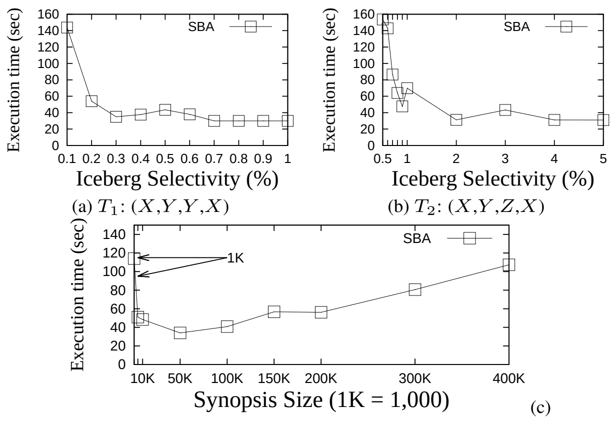


Figure 6: Running time vs ξ_σ and synopsis size $|\tilde{S}|$

Varying $|\tilde{S}|$. Next, we study the effect of the synopsis size $|\tilde{S}|$. Figure 6(c) shows the algorithms' execution times for template T_1 when $|\tilde{S}|$ varies from 1K to 400K. Let us first look at the performance of SBA. From the figure, we see that SBA's execution time first drops then rises as $|\tilde{S}|$ increases. There are two effects of the synopsis size: (1) A bigger synopsis results in longer time spent on processing the synopsis (e.g., to compute the lists $\tilde{L}[P]$'s). Hence execution time rises. (2) A bigger synopsis results in more effective pruning/accepting/accuracy tests. It is thus less likely that SBA has to resort to disk-based list-joining. Hence, execution time drops. The net effect of these two factors is a U-shaped performance curve for SBA. In our future work, we will try to automatically decide the optimal sample size for each query.

6. RELATED WORK

Before PREDATOR [13], traditional database systems do not formally support sequence data. PREDATOR stores sequence data based on the object-relational model. The DEVise system [10] supports sequence data processing using the relational model. To query sequence data, Sadri et al. [11] develop an extension to SQL, called SQL-TS, to express various kinds of pattern-based queries. These systems do not directly support OLAP operations or the processing of pattern-based aggregate queries

Iceberg query on relational data was first studied by Fang et al. [5]. The computational issue addressed in their work is the difficulty of housing a large multidimensional array in memory for effective computation of cuboids (and thus iceberg cells). Two techniques, namely, sampling and coarse counting are devised to identify candidate iceberg cells. Full scans of the disk-resident data is needed to eliminate false positives and false negatives in the answer. In contrast, our approach is to compute the iceberg cells of an *s-cuboid* via statistical tests. As we have shown in the experiments, disk accesses are mostly avoided, resulting in very fast processing.

7. 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 iceberg cells. Experimental study shows that our proposed algorithm outperforms the existing algorithms in order of magnitude.

8. ACKNOWLEDGEMENT

This work is partly supported by the Research Grants Council of Hong Kong (GRF PolyU 525009, 521012, 520413), Amazon Research Grant, and Microsoft Hong Kong Gift.

9. 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.
[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.