Answering Analytical Queries on Text Data with Temporal Term Histograms

Kai Lin
Univ. of California San Diego
9500 Gilman Drive
La Jolla, CA 92093
klin@sds.edu

Subhasis Dasgupta
Univ. of California San Diego
9500 Gilman Drive
La Jolla, CA 92093
sudasgupta@ucsd.edu

Amarnath Gupta
Univ. of California San Diego
9500 Gilman Drive
La Jolla, CA 92093
a1gupta@ucsd.edu

ABSTRACT
Temporal text, i.e., time-stamped text data are found abundantly in a variety of data sources like newspapers, blogs and social media posts. While today’s data management systems provide facilities for searching full-text data, they do not provide any simple primitives for performing analytical operations with text. This paper proposes the temporal term histograms (TTH) as an intermediate primitive that can be used for analytical tasks. We propose an algebra, with operators and equivalence rules for TTH and present a reference implementation on a relational database system.

CCS CONCEPTS
• Information systems → Data model extensions; Database query processing; Query optimization;

KEYWORDS
Query Processing, Text Analytics, Histogram, Temporal Term Histograms (TTH)

ACM Reference format:
Kai Lin, Subhasis Dasgupta, and Amarnath Gupta. 1997. Answering Analytical Queries on Text Data with Temporal Term Histograms. In Proceedings of ABC, ZZY, Jan 2019 (2018), 10 pages.
DOI: 10.475/123_4

1 INTRODUCTION
Today’s data management systems are designed to support data heterogeneity and analytical operations[3, 6]. In particular, all DBMSs today have support for full-text search for text-valued columns, and more complex aggregate operations. For example, PostgreSQL has two data types for full-text search - tsvector represents a document as a vector of terms, and tsquery represents a predicate over search terms where terms are combined by Boolean operators and a followedBy operator to construct phrases. On the analytical side, PostgreSQL supports a number of statistical operations, window operations, temporal operations of different granularities, sampling operations and so on. Other DBMSs like DB2, Oracle, Vertica, SQL Server have comparable set of facilities.

But how well do today’s DBMSs support text analytics, i.e., the need to perform analytical operations on textual data?
A number of application domains today, including computational social science applications are shifting to a paradigm called Text as Data [1, 2] whose goal is to use construct statistical models of diverse text-centric documents like financial news, legislative text, social media content and company filings, and perform statistical inferences based on these models[5, 8]. When the textual data is time-stamped, temporal analytical tasks include finding significant temporal trends in document features, or comparison between subsets of documents from two temporal intervals are similar.

Running Example Suppose, we have a relation of newspapers with the following schema:

articles(id:int, newspaper:string, title:string, section:string, city:string, pubDate:date, content:text)

Our target user is an analyst from the domain of social sciences. Let us consider the following analytical questions – we consider a question to be “analytical” if it needs any complex computation beyond data retrieval and standard aggregate functions provided by modern data management platforms.

Q1. Topic Co-occurrence. For all n-day intervals where "tax cut" was a dominant topic (i.e., within the top k key terms), what other topics were also dominant? How does this vary over cities?
Q2. Salience Detection Given a term set T is there a week i such that the set T is salient for that week? The set T is salient for a week if all terms in T are in the top k terms of that week and their rank distribution is significantly different with respect to every other week.
Q3. Synchronized Topics. Which newspapers show perfectly time-coordinated (same top k terms on the same day) articles in the politics section? This class of queries are used for detecting propaganda and censorship in state-sponsored newspapers [7].
Q4. Trendy Topics. Find documents dated today()−6 months that contain “trendy” topics. where terms are “trendy” in a time interval if they have a count above the average over all terms for that interval, and have a positive temporal gradient greater than a threshold θ.

Each of these questions hinges on the interplay between the temporal dimensions of the data, the frequency distributions of terms of the content field, and their co-variation with other attributes like city and section. However, they are hard to write as single queries and would require a complex database program requiring deep expertise in database technology, which might be beyond the expertise of our target user group.
Larger Context and Challenge. It is now recognized that supporting analytic workloads over relational data needs an analytics-friendly data algebra that can express a wide range of analytical operations needed, for example, for machine learning tasks. Linear algebra has been proposed as the new algebra for these workloads, and mappings between linear algebra and relational algebra are being defined. However, we are seeking an implementable intermediate abstraction that can support linear algebra like vector and matrix operations, and yet is implementable over today’s technology platforms. The representation we seek must be simple enough to be usable by our target users and yet be expressive enough to serve as a bridge to linear algebra processors. To address this gap, we propose an abstract data type, called the temporal term histogram as a database primitive that can be used at a higher-level than current database abstractions of text. We argue that this primitive captures a core abstraction of many computations underlying the Text as Data analysis paradigm.

Our Contributions. The contributions of this paper are as follows.

- We present a new data model and algebra for temporal term histograms
- We present a number of query optimization rules over the above algebra
- We present an implementation scheme for temporal term histograms over a relational DBMS
- We present a number of experiments to demonstrate the factors that influence the query performance of the above algebra

2 THE TEMPORAL TIME HISTOGRAM

2.1 The Model

Let \( C \) be a collection of documents. A document \( d \) minimally has an ID denoted by \( d.id \), a timestamp of creation denoted by \( d.ts \), and textual content denoted by \( d.text \), which can in turn be considered a collection of terms, \( d.text.terms \).

Given the above minimal setting, our first component model is based on term distributions. For simplicity, we do not go into the definition of what makes a term. In practice, issues like whether stop words belong to terms, words after stemming should be considered a single term, or multi-word phrases should be considered a single term etc. do not affect the model and are left for the application. The collection of unique terms over all documents in \( C \) is called \( V(C) \), the Vocabulary of \( C \). In a vocabulary, every term \( t \) is represented by a unique ID \( t.id \). Henceforth, we will use the term ID to be the proxy for a term.

A document histogram is a tuple \( h(d.id, d.ts, d.hist) \) where \( d.ts \) is a discretized time interval (e.g. per day) and \( d.hist \) is a 2-column matrix – the first column has the term ID and the second column has the count of the term in the document.

The Minimal Temporal Term Histogram (MTTH) can be viewed as a 4-tuple \( MTTH(\text{term.ID}, ts, count, list(doc.id)) \) created by merging of multiple document histograms as shown in Fig. 1, where \( ts \) is a discretized time interval as before and \( count \) is the term count over all documents within the specific interval. The start and width of the time intervals are specified by the user during setup time, and are part of the configuration parameters of the model.

The MTTH can be slightly extended by introducing a set of auxiliary attributes \( c_1, c_2 \ldots c_k \). For the present paper, we will consider only attributes with discrete domains like \( \text{city}, \text{newspaper} \) and \( \text{session} \) in our running example. The extended model called the Temporal Term Histogram (TTH) is a \( 4 + k \)-tuple. If \( \text{dom}(i) \) is the active domain of auxiliary attribute \( c_i \) of TTH, each row of the projection \( \pi_{c_1,c_2\ldots c_k}(TTH) \) is unique. In other words, every combination of values from \( \text{dom}(1) \times \ldots \times \text{dom}(k) \) is represented only once in the projection.

2.2 Operators

One can view a temporal term histogram as a hybrid structure between an inverted index and a pure histogram model, because it uses both the vocabulary \( V \) and \( TTH \); the algebraic operators for the model are listed in Table 1. In this model we assume that every tuple of the TTH structure has a unique tupleID. In the following, we point out some salient features of this operator set.

- The TTH is a mostly a relational structure except for the docID list. So it admits extractive relational operations like \( \text{select} (\sigma) \) and \( \text{project} (\pi) \). The union operation is replaced by \( \text{merge} \), which requires the count variable to be added if the (term, ts) pair matches, and behave like a union otherwise (i.e., a bag union semantics). Similarly, the docIDs would be merged for the matching pairs. For example, the merge operation of two TTHs is shown in Figure 2.

- Since TTH can be viewed as a histogram over terms and time, the operators offer a number of histogram manipulation operations. Specifically, it permits \( \text{axis manipulation} \) using \( \text{coarsen} \), \( \text{sortByAxis} \) and \( \text{collapse} \) operations as well as built-in and user-defined aggregation operations that can be invoked using the \( \text{apply} \) operation. Operation \( \text{applyArg} \) is motivated by the commonly used \( \text{argmax} \) and \( \text{argmin} \) functions, and finds not only the value of the function (e.g., the peaks and troughs of the histogram) but also identifies the records (i.e., effectively the term, time and documents) for which this value occurs.

- Since the TTH is an inverted index, it allows index operations that are typically used for answering Boolean queries. Standard operations for \( \text{list intersection} \) and \( \text{list union} \) are enabled by the \( \text{IndexOp} \) operation. The \( \text{queryIndex} \) operation accepts a list of document IDs and finds term-time pairs in those documents. We recognize that like any inverted index, TTH does not obviate the need for a document-wise forward index. Consider the query "Find the term frequency of the term 'Trump' in all documents where 'Obama' also appears". The query would first fetch documents where both terms occur (e.g., by index intersection); then it would need to find the frequency of the term 'Trump' in the documents from the prior result. However, the TTH does not maintain a forward index from a document to the terms it contains. Thus, the system will need a separate forward index to answer this query properly.

- A primary use of the auxiliary attributes in the TTH is that they are primarily used as grouping variables. The \( \text{group} \) operation in the TTH algebra is used for simply constructing the groups, and a partitioned relation over the TTH tuples for each distinct value of the grouping variable(s) – no aggregation operations are associated with grouping. A partition is identified by the specific value of the grouping variable it corresponds to. Thus, if the \( \text{city} \) variable has three values \( \text{NY}, \text{LA}, \text{SF} \) the result of \( \gamma_{\text{city}(TTH)} \)
where the answer

we present query plans for these questions based on the algebraic

TTH enables us to express some analytical queries that cannot be

expressed through a full-text query enabled DBMS like PostgreSQL

or standard inverted index server like Apache Solr and can be

complex to formulated in RDBMS.

2.3 Query Examples

In Section 1, we introduced a set of analytical questions. Here we present query plans for these questions based on the algebraic operations in Table 1, broken up into phases. Our goal in this section is to write the query plans in terms of TTH and standard relational operations. These plans are not optimized. We assume that the default temporal granularity of TTH is 1 day.

Q1. Topic Co-occurrence. Since the default temporal granularity is 1 day and the desired granularity is 5 days, we first apply the coarsen operation. Then we filter non the term “tax cut” and sort the results on the count field in descending order, and pick the top 20 records. From these records, we project the timestamp field to find the times when the selected term is most used.

\[
\pi_{\text{ts}}(\sigma_{\text{top}(20)}(\text{sort}_{\text{count},d}(\sigma_{\text{term}=\text{taxcut}}(\text{coarsen}(\text{5days})))),(\text{TTH} - \text{term}(\text{taxcut}))))
\]

where the d parameter performs a sort in descending order.

Next, we select the top 21 distinct terms from TTH in the same time period as above and then remove the term “tax cut” from the result.

\[
\pi_{\text{term}}(\sigma_{\text{top}(21)}(\text{sort}_{\text{count},d}(\sigma_{\text{ts}} \in \bullet)(\text{coarsen}(\text{5days})))((\text{TTH} - \text{term}(\text{taxcut}))'\text{taxcut}'))
\]

where • indicates the result of the first step and \(\sigma^D\) refers to a select distinct operation.

The second part of the query (“how does it vary by city”), we perform the above two steps except that we add a group operation on the variable city as follows.

\[
\pi_{\text{ts}}(\sigma_{\text{top}(20)}(\text{sort}_{\text{count},d}(\gamma_{\text{city}}((\sigma_{\text{term}=\text{taxcut}}(\text{coarsen}(\text{5days}))))))((\text{TTH} - \text{term}(\text{taxcut}))))
\]

The result of the query will produce a set of timestamps per city.

\[
\pi_{\text{term}}(\sigma_{\text{top}(21)}(\text{sort}_{\text{count},d}(\gamma_{\text{city}}((\sigma_{\text{term}=\text{taxcut}}(\text{coarsen}(\text{5days}))))))((\text{TTH} - \text{term}(\text{taxcut}))))
\]

The innermost grouping creates a city-based partition over the coarsened data. The selection operation over timestamp and every upstream operation is performed per partition, leading to an set of terms per city.

Q2. Salience Detection. Since the query is on weekly term ranking, one way to formulate the query is create a TTH view that transforms the default count-based TTH to a rank-based TTH, called TTH\(\rho\) here. For practical purposes, the view would be computed only on weekly aggregate data for which the doc.count exceeds a given threshold \(ct\), and the doc.list column is omitted. We first
Table 1: Algebraic Operators over the Vocabulary and the Temporal Term Histogram Index. In this table, we omit some straightforward operations like getRecords, that fetches the records given their tupleIDs, and sort.

| Operator       | Description                                                                 | Use                                                                 |
|----------------|-----------------------------------------------------------------------------|----------------------------------------------------------------------|
| lookup: term → termID | Given a term returns its ID                                                 | Usually the first step of a search                                    |
| project (π): [attrib] → bindings | The standard relational projection operation on the TTH structure. It returns the bindings on the projected attributes | Find only the terms from the TTH, which may be the result of another operation, typically select. |
| select: predicate → TTH | the predicate is over all four properties of a TTH structure, returns a subset of the TTH structure matching the predicate | Return the temporal histogram of documents where count('ObamaCare') > 3 & count('Trump') > 0 & ts > ('06/01/2017') & doc.id ∈ [201 – 399] |
| indexOp: (Op, Idx1, Idx2) → [doc.id] | Op can be intersection, difference or union of two document index lists. | The exploration process may want to expand the result set for a query by adding the results of another query – this needs a union of indexes. |
| queryIndex: ([qDocID], timeStart?, timeEnd?) → [termID, ts] | Given qDocID, a list of query document IDs, returns a set of term-timestamp combinations where the terms occur the query documents. Optionally, the operation is restricted to a specified time interval | Which terms are included in documents [4000-8000], provided the term document combination occurred in 201? |
| coarsen: (newWidth, timeStart?, timeEnd?) → TTH | The operation reduces the time granularity of an existing TTH to the specified newWidth, which must be a multiple of the current width. The count value will be recomputed based on the newWidth. Optionally, this operation would be carried out only on a specified time interval marked by the start and/or end parameters. | Return the temporal term histogram aggregating data by the week instead of by the day. |
| merge: (TTH₁, TTH₂) → TTH | The operation merges two TTHs that are temporally aligned. The merged histogram has terms from both TTHs and adds the count of the common term, while performing a union of their doc.Idx. | This operation will be used to merge the TTHs obtained from two different queries on the same base data. |
| group: (TTH, <grouping-vars>) → <group-value>[TTH] | The operation creates an array of TTHs indexed by values of the grouping variables. The size of the array is that of the Catersian product of the active domains of the grouping variables. | Grouping will mostly be used for the auxiliary attributes. |
| apply: function(<parameters>) → [value] | The apply operation calls a histogram function like min, max, findModes, findMoments(k). The nature of the value depends on the function called. The function can be multi-valued e.g., modes of the histogram. | Apply can be used to find peaks and troughs in the 3D histogram |
| applyArg: function(<parameters>) → [(value, [recordID])] | This is a convenient variant of the apply function that returns the result of the function together with set of records associated with each returned value (when the function is multivalued) | Find time intervals where there is burst of terms and identify the corresponding terms |
| sortByAxis: (axis) → TTH | The operation on a single TTH and rearranges it by sorting it by the axis parameter which can either be term or count. The time axis is not impacted by the operation. | Depending on the application, the distance-function can be simple as a Euclidean distance or complex as a KL divergence |
| distance: (TTH₁, TTH₂, distance-function) → float | Given two TTH structure, the operation applies the distance-function to the frequency distributions and returns numerical measure of their distance | When time axis is collapsed, the result can be used to construct a term-document matrix that can be used to compute topic models. |
| collapse: (axis) → 1D histogram | In this operation, the axis can be term or ts – it produces a marginal histogram over only the time axis (i.e., over all terms) or over the term axis. The doc.id list gets recomputed as part of the operation. | |
| extractAxis: (axis) → Vector(axis-Value) | Given an axis (i.e., term or ts), it extracts the values of the axis as a vector | Given a selection query which produces a reduced TTH, this operation may be used to get the term vector corresponding to the result. |

write the view definition for TTHₚₐₙ₉₉.

\[
\pi_{\text{term}, \text{ts}, \rho(\text{rank})}, \text{rank}, \text{sort}(\text{count}, d)
\]  
\[
(\sigma_{\text{doc.count} > 0} \land \text{all}(\text{terms} \in T)(\text{coarsen}'7\text{days}')(\text{TTH}))
\]  

where T is the list of terms given by the user, the rank() function reports the sort order of a tuple, and ρ is the rename operation. Further, the predicate all(terms ∈ T) is universally quantified, i.e.
the predicate only selects weeks in which all terms appear in some document.
Now we need to choose a week for which the aggregate rank of the terms in T is the highest using any suitable aggregate function. We use the sum of their ranks $R_{sum}$ as an example. Once we have the $R_{sum}$ values for the highest-rated week:

$$π_{ts, sum}(max(sum(σ_{term∈T,·}(rank))))(TTH_R)$$

With the weekly $R_{sum}$ values, we can directly use it for a statistical test like the Mann Whitney U Test [4], where the $R_{sum}$ is directly compared with that of every other week. Q3. Synchronized Topics.

**Q4. Trendy Terms.** We can formulate the query in the following steps.

a) select a part of the TTH for the specified time period and whose count value is greater than AvegCount, which is the average of the count of all terms in the same time period, computed separately.

$$σ_{ts>today()−months(i)} ∧ count>ArvgCount(TTH)$$

b) Next, we would apply a function that extracts the maximum slope from the count value along the time axis and identify the corresponding records. The \(●\) designates the partial result from the previous step. Select the results where the value part of the result exceeds the threshold.

$$σ_{value>θ}(σ_{value>θ}(count, ts, ●))$$

where the user-defined findMaxSlope function accepts the axis \(ts\) and the variable \(count\) over which the function would be computed.

c) Project the record IDs (2nd variable) from the results of the filter, and extract their docIDs. Then, find the docIDs for these records.

$$π_{doc,d}(getRecords(π_{ts, sum}(σ_{ts>today()−months(i)}) ∧ count>ArvgCount(TTH)))$$

If in the process of exploration we want to apply the findMaxSlope function on the time granularity of a week instead of a day, we will use the \(\text{coarse}\) operator. In that case, the second segment of the query will be modified to:

$$σ_{value>θ}(σ_{value>θ}(count, ts, \text{coarse}●))$$

**Algebraic Optimization.** Many mathematical properties are satisfied by Term Temporal Histogram operations. The following are some example properties between the coarsen and merge operations:

1. The associativity of the merge operation:

$$\text{thi} \text{merge}(\text{thi} \text{merge}(X, Y), Z) = \text{thi} \text{merge}(X, \text{thi} \text{merge}(Y, Z))$$

2. The commutativity between the merge and coarsen operation:

$$\text{thi} \text{coarsen}(\text{thi} \text{merge}(X, Y), D) = \text{thi} \text{merge}(\text{thi} \text{coarsen}(X, D), \text{thi} \text{coarsen}(Y, D))$$

3. The quasi-idempotence of the coarsen operation:

$$\text{thi} \text{coarsen}(\text{thi} \text{coarsen}(X, D1), D2) = \text{thi} \text{coarsen}(X, D2)$$

Using these properties can optimize a complicated histogram expression. For example, the left hand side of the commutativity above requires merging two very large histograms X and Y, which could be very slow, and then convert the merging result into a new histogram with larger data ranges; however, the right hand side first converts two large histograms into two smaller histograms and then merge them. Using the expression of the right hand side to calculate the left hand side could have better performance.

**3 A REFERENCE RELATIONAL IMPLEMENTATION**

We have implemented the TTH model on top of a relational database (in our case, PostgreSQL). We have assumed that the documents in question are text-valued attributes of relations. A relation may have multiple text attributes to be indexed by the TTHI structure. In the following, we outline the steps to construct and use a temporal term histogram.

**Mapping Declaration.** To construct a Temporal Term Histogram from a relational schema, the user must first identify the schema elements within the purview of the index. This is performed through a set of annotated mapping statements. Consider the schema 

```sql
newspapers(name, province, publisher)
newsArticle(id, newspaper, title, date, author, page, content)
```

where the data is extracted from CSV files, the following mapping statements would identify text and temporal attributes in `newsArticle`.

```sql
@CSV(corpus = "chinese-newspaper", entity = "Newspaper")
@CSVColumn(mapto = /quotes.Varnewspaper.name/quotes)
@Temporal(TimeType.DATE, format="YYYY-MM-DD")
Date date;
@TermIndex(mapto = "title", language="Chinese")
String title;
@TermIndex(mapto = "content", language="Chinese")
String content;
```

The first statement declares that a corpus called “chinese-newspaper” is a CSV file and is henceforth called by the name “Newspaper”. By default all attributes actually used in constructing the TTHI are referred to as CSVColumn. Since the attribute `newsArticle.author` is not relevant for the TTH, it is not captured in any mapping statement. Some of the columns have a special role in the TTH. An attribute annotated as TermIndex with data type String is interpreted as a text field that will be indexed on TTH construction algorithm. Similarly, the attribute that serves as the time indicator is annotated as @Temporal. If the TTH is constructed over multiple tables, one can also specify foreign keys between tables. For example, the reference from `newsArticle` to `newspapers` name is stated as:

```sql
@CSVColumn(mapto = "newspaper.name")
String newspaper;
```

The mapping specification also allows the specification of attribute attributes, attributes for which we would like to create additional axes for specifying the histogram. To declare `province` as a category, we state:

```sql
@Category(name=province)
String province;
```

In this case, the term count for the histogram will be based on for each value of the cross product of date, term(from content, and separately, from title) and province.
Relational Implementation. Given the mapping declaration, the system constructs a set of base tables that store the histogram for each TermIndex column for each time-unit (day in the above specification) and category column. Algorithm 1 shows the process of constructing these tables.

Algorithm 1: Create Database Tables for A Corpus (PostgreSQL version)

```
for each entity ent in the corpus do
    create a table tab with the specified name or the entity name;
    for each attribute att of ent do
        if the type of att is an entity ent’ then
            create a column in tab with the name of att and the postfix _id and with the type of the @Id attribute of ent’;
        else
            create a column in tab with the name and correspondent type of att;
        if ent is the main entity of the corpus then
            for each term index tid do
                create a tsvector column with the name of tid and tsv;
            for each term index tid do
                create a table tab with the name of tid and the postfix _doc_frequency;
                for each attribute of tid do
                    create a string column in tab with the name of tid;
                create a column doc_id in tab as the foreign key of the main entity;
                create an int column count in tab;
            for each category of cat do
                create a table tab using the concatenation of the name of tid, cat and the postfix _base histogram as the table name;
                create a string column in tab with the name of tid;
                create a date column in tab with the name date;
                for each attribute att of cat do
                    create a string column in tab with the name of att;
                create an int column in tab with the tid name and term count as the column name;
                create an int column in tab with the name doc_count;
```

Thus, for our example case, Algorithm 1 produces the following.

1. For any combination of document attributes that are considered as a histogram object, a new column with the type tsvector is added into the document table and a trigger is setup to save the tokens from the columns of these attributes into the tsvector column whenever an insert event occurs on the document table.

2. A table term_doc_frequency is created to record the term frequency in each document for the selected terms in the study. A trigger is attached to the update event on the tsvector column of the document table to calculate the term frequencies in the tsvector column using the built-in function ts_stat and save the result into the term_doc_frequency table.

3. A query-specific histogram table is generated on the fly as a materialized view of a query against the document table, the term_doc_frequency table and other metadata tables. For example, a user can create "a monthly histogram for the newspaper articles that are published on Beijing Daily in 2017 and contain the term ‘Xi Jianping’ at least 5 times” by automatically constructing the view definition in Figure 4. Once the view is defined, subsequent queries can be asked on it. For example, the query

```
SELECT *
FROM Histogram(NewspaperArticle, '2017-01-01', '2017-12-31', 10, province, content_term)
WHERE NewspaperArticle.page = '3'
AND province = 'Beijing'
AND content_term @@ 'Trump'
```

seeks the TTH from news papers with predicated on the auxiliary attributes province and page, and on the term of interest (Trump) for the specified date range. Algorithm 2 specifies how the above query will be translated to an equivalent query against the histogram created with the materialized view.

The declaration MATERIALIZED TERM_TEMPORAL_HISTOGRAM VIEW generates a histogram with columns term, date_range, count and doc_list, where count gives the frequency of the term in the date_range. The array doc_list includes all document ids containing the term in the date_range.

The auxiliary query dr generates a list of the first day of each month from 2017-01-01 (included) to 2018-01-01 (excluded). The query date ranges creates month based data ranges from 2017-01-01 to 2018-01-01 by joining dr with the one month shifted dr.

A typical histogram usually can be created by joining the document table with the table term_doc_frequency and the result of the auxiliary query date ranges. In the example above, the table newspaper_articles is the document table. The condition newspaper_articles.date::date <= date_range gets all the documents whose publish dates are in the date_range, and the predicate on newspaper_articles.province limits the newspaper to Beijing Daily. The last condition gets all the documents where the term Xi Jianping occurs at least 5 times.

Operator Implementation. We describe the implementation of some of the more complex TTH operations.

Coarsen. The coarsen operation generates a new histogram from an existing histogram by using a larger date range. For example, the following statement creates a quarterly based histogram for the newspaper articles that are published on Beijing Daily in 2017 and contain the term ‘Xi Jianping’ at least 5 times:

```
SELECT *
FROM tthi_coarsen(beijing_xi_monthly_2017, '2017-01-01',
```
CREATE MATERIALIZED TERM_TEMPORAL_HISTOGRAM VIEW
beijing_xi_monthly_2017 AS
WITH dr AS
  (SELECT generate_series as date_start,
   row_number() OVER () as num
   FROM generate_series('2017-01-01'::date,
   '2018-01-01'::date,
   '1 month'::interval)),
  dateranges AS (SELECT daterange(X.date_start::date,
   Y.date_start::date)
   as date_range
   FROM dr X, dr Y
   WHERE X.num + 1 = Y.num)
SELECT term_doc_frequency.term,
   dateranges.date_range,
   sum(term_doc_frequency.frequency) as count,
   array_agg(term_doc_frequency.doc_id) as doc_list
FROM newspaper_articles,
    term_doc_frequency,
    dateranges
WHERE newspaper_articles.id = term_doc_frequency.doc_id
  AND newspaper_articles.date::date <@ date_range
  AND newspaper_articles.newspaper = 'Beijing Daily'
  AND term_doc_frequency.doc_id IN
    (SELECT term_doc_frequency.doc_id
     FROM term_doc_frequency
     WHERE term_doc_frequency.term = 'Xi Jianping'
     AND term_doc_frequency.frequency >= 5)
GROUP BY term_doc_frequency.term, date_range;

Figure 3: The system automatically generates materialized views

Algorithm 2: Query Translation

1. If conditions_for_main_entry and conditions_for_term_index are empty then
   1.1. If TimeInterval == 1 then
       translate the query conditions into a direct query on the base histogram for the category and the term index;
   1.2. Else
       translate the query conditions into a grouping query on the base histogram for the category and the term index;
2. Else
   translate the query conditions into a grouping query on the join of the main entity table and relevant tables and the doc_frequency table;

'2018-01-01',
'3 month');

The coarsen operator first validates the compatibility of the date ranges of the new histogram with the date ranges in the base histogram. The data range is valid if each date range in the new histogram is aligned with several consecutive date ranges in the base histograms. The following query is used in the validation for the statement above:

WITH tmp_tthi_1 AS (SELECT generate_series AS date_start,
   row_number() OVER () AS num
   FROM generate_series('2017-01-01'::date,
   '2018-01-01'::date,
   '3 month'::interval)),
tmp_tthi_2 AS (SELECT daterange(X.date_start::date,
   Y.date_start::date) AS date_range
   FROM tmp_tthi_1 X, tmp_tthi_1 Y
   WHERE X.num + 1 = Y.num),
tmp_tthi_3 AS (SELECT * FROM beijing_xi_monthly_2017)
SELECT count(*)
FROM
  (SELECT DISTINCT date_range FROM tmp_tthi_3)
tmp_tthi_4,
  (SELECT DISTINCT date_range FROM tmp_tthi_2)
tmp_tthi_5
WHERE NOT tmp_tthi_4.date_range = tmp_tthi_5.date_range
  AND isempty(tmp_tthi_4.date_range *
  tmp_tthi_5.date_range) = false
  AND NOT tmp_tthi_4.date_range *
  tmp_tthi_5.date_range = tmp_tthi_4.date_range

The main idea behind this validation algorithm is that if a date range in the new histogram is overlapping with a date range in the base histogram, then the date range in the base histogram must be a sub-date range of the data range in the new histogram. If the validation fails, the execution of the operation will stop with an error message, otherwise the following query result will be returned as the new histogram:

SELECT *
FROM (WITH tmp_tthi_7 AS
    (SELECT generate_series AS date_start,
     row_number() OVER () AS num
     FROM generate_series('2017-01-01'::date,
     '2018-01-01'::date,
     '3 month'::interval)),
   tmp_tthi_8 AS (SELECT daterange(X.date_start::date,
   Y.date_start::date) AS date_range
   FROM tmp_tthi_7 X, tmp_tthi_7 Y
   WHERE X.num + 1 = Y.num),
tmp_tthi_9 AS (SELECT * FROM beijing_xi_monthly_2017)
SELECT *
FROM tmp_tthi_8
WHERE tmp_tthi_9.date_range <@ tmp_tthi_8.date_range
GROUP BY term, tmp_tthi_8.date_range
ORDER BY term, tmp_tthi_8.date_range) AS tmp_tthi_10
Merge. The merge operation generates a new histogram from two existing base histograms. For example, the following statement creates new histogram from two existing histograms:

```sql
SELECT * FROM tthi_merge(beijing_xi_monthly_2017, jiefang_xi_monthly_2017);
```

where `jiefang_xi_monthly_2017` is the histogram for the newspaper articles that are published on Jiefang Daily in 2017 and contain the term 'Xi Jianping' at least 5 times.

The merge operation requires a validation step too before creating the new histogram. The validation uses the following query to check whether the date ranges in the two existing histograms are aligned:

```sql
SELECT count(*) FROM (SELECT DISTINCT date_range FROM beijing_xi_monthly_2017) tmp_tthi_1,
(SELECT DISTINCT date_range FROM jiefang_xi_monthly_2017) tmp_tthi_2
WHERE NOT tmp_tthi_1.date_range = tmp_tthi_2.date_range
AND isempty(tmp_tthi_1.date_range * tmp_tthi_2.date_range) = false
```

The validation makes sure that any different data ranges from the two histograms are not overlapping. If the validation is passed, the following SQL statement will be executed to get the new histogram:

```sql
SELECT * FROM (WITH tmp_tthi_5 AS
(SELECT CASE WHEN tmp_tthi_3.term IS NOT NULL THEN tmp_tthi_3.term
WHEN tmp_tthi_4.term IS NOT NULL THEN tmp_tthi_4.term
END AS term,
CASE WHEN tmp_tthi_3.date_range IS NOT NULL THEN tmp_tthi_3.date_range
WHEN tmp_tthi_4.date_range IS NOT NULL THEN tmp_tthi_4.date_range
END AS date_range,
CASE WHEN tmp_tthi_3.count IS NULL THEN tmp_tthi_4.count
WHEN tmp_tthi_4.count IS NULL THEN tmp_tthi_3.count
ELSE tmp_tthi_3.count + tmp_tthi_4.count
END AS count,
CASE WHEN tmp_tthi_3.doc_list IS NULL THEN tmp_tthi_4.doc_list
WHEN tmp_tthi_4.doc_list IS NULL THEN tmp_tthi_3.doc_list
ELSE tmp_tthi_3.doc_list | tmp_tthi_4.doc_list
END AS doc_list
FROM beijing_xi_monthly_2017 AS tmp_tthi_3
FULL JOIN jiefang_xi_monthly_2017 AS tmp_tthi_4
ON (tmp_tthi_3.term = tmp_tthi_4.term
AND tmp_tthi_3.date_range = tmp_tthi_4.date_range)),
tmp_tthi_6 AS
(SELECT tmp_tthi_3.term,
  tmp_tthi_3.date_range,
  tmp_tthi_3.doc_list &
  tmp_tthi_4.doc_list
FROM beijing_xi_monthly_2017 AS tmp_tthi_3,
  jiefang_xi_monthly_2017 AS tmp_tthi_4
WHERE tmp_tthi_3.date_range = tmp_tthi_4.date_range
AND tmp_tthi_3.term = tmp_tthi_4.term
AND array_length (tmp_tthi_3.doc_list &
  tmp_tthi_4.doc_list, 1) > 0),
tmp_tthi_7 AS (SELECT term, date_range, doc_id
FROM tmp_tthi_6, unnest(tmp_tthi_6.doc_list) doc_id),
tmp_tthi_8 AS
(SELECT tmp_tthi_7.term, tmp_tthi_7.date_range,
sum(word_doc_frequency.frequency) AS count,
array_agg(tmp_tthi_7.doc_id) AS doc_list
FROM tmp_tthi_7, word_doc_frequency
WHERE tmp_tthi_7.term = word_doc_frequency.word
AND tmp_tthi_7.doc_id = word_doc_frequency.doc_id
GROUP BY tmp_tthi_7.term, tmp_tthi_7.date_range)
SELECT tmp_tthi_5.term, tmp_tthi_5.date_range,
CASE WHEN tmp_tthi_8.count IS NULL THEN tmp_tthi_5.count
WHEN tmp_tthi_8.count IS NOT NULL THEN tmp_tthi_5.count - tmp_tthi_8.count
END AS count,
tmp_tthi_5.doc_list
FROM tmp_tthi_5
LEFT JOIN tmp_tthi_8
ON (tmp_tthi_5.term = tmp_tthi_8.term
AND tmp_tthi_5.date_range = tmp_tthi_8.date_range)
ORDER BY tmp_tthi_5.term, tmp_tthi_5.date_range
AS tmp_tthi_9

where the auxiliary query `tmp_tthi_5` simply adds the term counts and union the document id arrays from the two base histograms for the same term and get the union of the base histograms. However the two document id arrays from the two base histograms may overlap, i.e., they may contain same document ids in their document arrays and count the term multiple times. The auxiliary query `tmp_tthi_6`, `tmp_tthi_7` and `tmp_tthi_8` calculate the intersection of the base histograms. The main statement subtracts the intersection from the union to compute the merge result.

Histogram Distance. The TTH implementation allows a number of histogram comparison operations, including a pointwise Euclidean distance. In the following example, the temporal term histograms of for two newspapers are compared based on a monthly granularity. Such a comparison shows the relative importance of terms between the two newspapers.

```sql
SELECT *
FROM tth_eu_distance(beijing_xi_monthly_2017, jiefang_xi_monthly_2017);
```

TF-IDF Histogram. The TF-IDF score is normally defined for documents. With a minor abuse of definition, we define the TF-score of a term within a given time interval as the product of the term frequency divided by the log of the proportion of documents containing the term within the time interval. The practical use of this histogram ranking function is to ameliorate the skew of term
We used a corpus of 2500 articles from Chinese newspapers containing 6 million distinct terms. The relational implementation of the TTH prototype was created on PostgreSQL 10.1.

**Construction Cost for Term-Document Frequency Table.** Figure 5 shows that the time to construct the term document frequency table is linear with document and term sizes. The cost of 300s is linear and within interactive response limits. Figure 7 (left) is the size of the merged histogram. The operation is linear and within interactive response limits.

**Cost of the coarsen Operation.** The coarsen operation is a temporal condensation operation over TTHI tuples that reduces the number of tuples. Figure 7 (right) shows that the operation is linear with respect to the size of the un-coarsened data.

These experiments demonstrate that temporal term histograms are very useful for analytical queries, are not expensive to compute, and are therefore a feasible primitive for temporal text data.

## 5 CONCLUSION

In this paper, we presented the temporal term histogram as an intermediate level data object that can be used for text data analytics. While we showed how a number of analytical queries can be expressed and evaluated with TTH structures, there are indeed limitations. For example, the materialized view shown in Figure 4 can be automatically constructed only when the construction does not have arbitrary functions. For example, a query to create a materialized temporal term histogram view over all documents where the number of mentions of term t1 is at least 10% higher than that of t2 cannot be generated automatically. However, if the materialized view is manually constructed for such cases, the rest of the ad hoc histogram construction and query operations can work without change. Our future work includes a more efficient implementation of the TTH by implementing the TTH functionality through the User-defined aggregate (UDA) facilities of a DBMS, and separately, by using compression techniques to reduce the size of the histogram.

## ACKNOWLEDGMENTS

This work has been partly funded by grant number 1738411 from the National Science Foundation.

## REFERENCES

[1] Matthew Gentzkow, Bryan T Kelly, and Matt Taddy. 2017. Text as data. Technical Report. National Bureau of Economic Research.
[2] Justin Grimmer and Brandon M Stewart. 2013. Text as data: The promise and pitfalls of automatic content analysis methods for political texts. Political analysis 21, 3 (2013), 267–297.
[3] F. Heimerl, S. Lohmann, S. Lange, and T. Ertl. 2014. Word Cloud Explorer: Text Analytics Based on Word Clouds. In 2014 4th Hawaii International Conference on System Sciences. 1833–1842. https://doi.org/10.1109/HICSS.2014.231
[4] Myles Hollander, Douglas A Wolfe, and Eric Chicken. 2013. Nonparametric statistical methods. Vol. 751. John Wiley & Sons.
[5] Miliadis D. Lytras, Vijay Raghavan, and Ernesto Damiani. 2017. Big Data and Data Analytics Research: From Metaphors to Value Space for Collective Wisdom in Human Decision Making and Smart Machines. Int. J. Semantic Web Inf. Syst. 13, 1 (2017), 1–10. https://doi.org/10.4018/IJSWIS.2017010101
[6] P. Parzadeh, M. Carey, and T. Westmann. 2017. A performance study of big data analytics platforms. In 2017 IEEE International Conference on Big Data (Big Data). 2911–2920. https://doi.org/10.1109/BigData.2017.8258260
[7] Margaret E Roberts. 2018. Censored: Distraction and Diversion Inside Chinas Great Firewall. Princeton University Press.
[8] Tariq Siddiqui, John Lee, Albert Kim, Edward Xue, Xiaofo Yu, Sean Zou, Lijin Guo, Changfeng Liu, Chaoran Wang, Karrie Karahalios, and Aditya G. Parameswaran. 2017. Fast-Forwarding to Desired Visualizations with Zenvisage. In CIDR 2017, 8th Biennial Conference on Innovative Data Systems Research, Chaminade, CA, USA, January 8-11, 2017, Online Proceedings. http://cidrdb.org/cidr2017/papers/p43-siddiqui-cidr17.pdf
Figure 5: The construction of the frequency table is linear with respect to the number of documents and terms.

Figure 6: The ad hoc TTH creation cost is within interactive response limits.

Figure 7: (Left) The merge operation over two roughly equal sized TTHs. (Right) The cost of the coarsen operation.