Workload-Aware Self-Tuning Histograms for the Semantic Web

Katerina Zamani[1], Angelos Charalambidis[1], Stasinos Konstantopoulos[1], Nickolas Zoulis[1,2], and Effrosyni Mavroudi[3]

[1] Institute of Informatics and Telecommunications, NCSR 'Demokritos', Athens, Greece
{kzam, acharal, konstant}@iit.demokritos.gr
[2] Computer Science Department, Athens University of Economics and Business, Greece
[3] School of Electrical and Computer Engineering, National Technical University of Athens, Greece

Abstract. Query processing systems typically rely on histograms, data structures that approximate data distribution, in order to optimize query execution. Histograms can be constructed by scanning the database tables and aggregating the values of the attributes in the table, or, more efficiently, progressively refined by analysing query results. Most of the relevant literature focuses on histograms of numerical data, exploiting the natural concept of a numerical range as an estimator of the volume of data that falls within the range. This, however, leaves Semantic Web data outside the scope of the histograms literature, as its most prominent datatype, the URI, does not offer itself to defining such ranges. This article first establishes a framework that formalises histograms over arbitrary data types and provides a formalism for specifying value ranges for different datatypes. This makes explicit the properties that ranges are required to have, so that histogram refinement algorithms are applicable. We demonstrate that our framework subsumes histograms over numerical data as a special case by using to formulate the state-of-the-art in numerical histograms. We then proceed to use the Jaro-Winkler metric to define URI ranges by exploiting the hierarchical nature of URI strings. This greatly extends the state of the art, where strings are treated as categorical data that can only be described by enumeration. We then present the open-source STRHist system that implements these ideas. We finally present empirical evaluation results using STRHist over a real dataset and query workload extracted from AGRIS, the most popular and widely used bibliographic database on agricultural research and technology.

1 Introduction

Query optimizers in query processing systems typically rely on *histograms*, data structures that approximate data distribution, in order to be able to apply their cost model. Histograms can be constructed by scanning the database tables and aggregating the values of the attributes in the table; and similarly maintained in the face of database updates.

This histogram lifecycle, however, cannot be efficiently applied to large-scale and frequently updated databases, such as, for example, stores of sensor data. An alternative approach is taken by *adaptive* query processing systems that update their histograms by observing and analysing the results of the queries that constitute the client-requested workload, as opposed to maintenance workload only for updating the histograms. The relevant databases literature focuses on numerical attributes, exploiting the concept of an *interval* as a description of a set of numerical values that is *succinct* and that has a *length* that can be used to estimate the cardinality of many different intervals that have roughly the same density.

In the work described here, we investigate how to extend adaptive query processing so that it can be applied to the domain of *strings*, typically treated as purely categorical symbols that can only be described by enumeration. This, however, disregards the fact that there are several classes of strings that have an internal structure and that can be handled in a more sophisticated manner. Specifically, we use string *prefixes* to expresses 'intervals', i.e., sub-spaces of the overall string space that are interesting from the point of view of providing query optimization statistics. Although weaker than regular expressions, prefixes can be very efficiently applied and can capture interesting ranges in hierarchically-structured string domains, such as that of URIs. We also experiment with describing a string range as a volume of strings similar to a central string, quantifying similarity in a way that favours similar prefixes.

This attention on URIs is motivated by their prominent position in the increasingly popular *Semantic Web* and *Linked Data* infrastructures for publishing data. In fact, these paradigms motivate adaptive query processing for a further reason besides the scale of the data: *distributed querying* engines often concentrate loose federations of publicly-readable remote data sources over which the distributed querying engine cannot effect that histograms are maintained and published. Furthermore, the URIs of large-scale datasets are not hand-crafted names but are automatically generated following naming conventions, usually hierarchical. These observations both motivate extending adaptive query processing to Semantic Web data stores and also present an opportunity for our string prefix extension.

In the remainder of this article, we first review self-tuning histograms (Section 2) where we identify STHoles as our starting point, a very successful algorithm for multi-dimensional histograms of numerical data. We proceed to formalize the key concepts in STHoles in a way that subsumes STHoles as its specialization for numerical intervals (Section 3) and to provide two alternatives for an extension that covers URI strings (Section 4). We then proceed to present experimental results using our prototype implementations (Section 5) and conclude (Section 6).

2 Background

In their simplest form, histograms describe an attribute a. The range of possible values of a is divided into non-overlapping *ranges*. A histogram is a set of *buckets* where each bucket is associated with a range and holds the number of tuples where the value for a is within the bucket's range. *Self-tuning* histograms are progressively refined from *query feedback* after each selection on a, using the actual result count to update the statistics in the bucket of a. In order to manage memory usage, some error is tolerated and buckets with similar statistics are merged into a single bucket with a wider range. In *workload-aware* self-tuning histograms, frequently used buckets (in a given workload) are split into narrower and more accurate buckets, while less frequently used buckets are more likely to be merged with more dissimilar buckets and produce a larger error when used.

Workload-aware self-tuning histograms have been successfully used in relational databases as a way to avoid the costly creation of static histograms of massive datasets. These techniques are memory efficient as they are focused towards the current workload, providing more accurate statistics for data regions that are being queried more frequently. Furthermore, they efficiently adapt to changes in the data distribution or the focus of the workload as they exploit query feedback collected from the production workload and do not impose any maintenance workload.

2.1 Histograms of numerical attributes

In one of their earliest instances [1], such one-dimensional histograms of numerical attributes were used to hold *statistics on intermediate tables (SIT)*, where each SIT corresponds to an intermediate node of the query plan. Adjacent buckets shared their ranges' edges and the ranges of all the buckets together covered the entire range of values of the attribute. In order to estimate the cardinality of arbitrary select-project-join (SPJ) queries, statistics are estimated for the individual patterns and then propagated through the query plan. Consider, for instance an SPJ query of the form:

$$(R.x = S.y\ )\ \text{AND}\ (S.a < 10)$$

The histograms of tables R and S are used to estimate the selectivity of $R \bowtie S$ ignoring $S.a < 10$ and then the histogram of $S.a$ is used to estimate the selectivity of $S.a < 10$ over the result of $R \bowtie S$.

To avoid the propagation of errors through a sequence of operators, SITs cat also match intermediate sub-expressions of the query. That is to say, we would use statistics that are built on the result of the query expression $R \bowtie S$ specifically on ranges of $S.a$ values, rather than estimates derived from the isolated statistics of $R.x$, $S.y$, and $S.a$. A workload-driven technique was used to identify the SITs that maximized the benefit to the query optimizer.

These ideas are relevant to the Learning Optimizer (LEO) framework [2] used in DB2. LEO monitors query execution and accordingly adjusts the cardinality estimates and statistics used by the query optimizer. By comparing estimated and actual cardinalities, LEO gives positive or negative feedback to the statistics and the cardinality model used. Correlations can be also detected when estimates for individual predicates are known to be accurate but some combination of them is not. LEO does not modify statistics, but saves separately adjustment factors such that the product of the adjustment factor and the estimated selectivity derived from the DB2 statistics yields the correct selectivity. Stillger et al. [2] demonstrated that LEO improves cardinality estimates by orders of magnitude, changing plans to improve performance by orders of magnitude, while adding less than 5% overhead to execution time when collecting query feedback.

STGrid [3] extends these ideas to multidimensional self-tuning histograms that use query workloads to refine a grid-based histogram structure. These self-tuning histograms are a low-cost alternative to traditional histograms with comparable accuracy. However, since the splitting (or merging) of each bucket entails the splitting (or merging) of several other buckets that could be far away from and unrelated to the original one, overall accuracy is degraded in order to satisfy the grid-partitioning constraint.

To alleviate the poor bucket layout problem of STGrid, STHoles [4] allows buckets to overlap. This more flexible data structure allows STHoles to exploit feedback in a truly multi-dimensional way and is adopted by many subsequent algorithms [5, 6], including the one presented here. STHoles allows for inclusion relationships between buckets, resulting in a tree-structured histogram where each node represents a bucket. Holes are sub-regions of a bucket with different tuple density and are buckets themselves. To refine an STHoles histogram, query results are used to count how many tuples fall inside each bucket of the current histogram. Each partial intersection of query results and a bucket can be used to refine the histogram by drilling new holes, whenever the query results diverge from the prediction made through the bucket's statistics.

In order to maintain a constant number of buckets, buckets with close tuple densities are merged to make space for new holes. A penalty function measures the difference in approximation accuracy between the old and the new histogram to choose which buckets to merge. Parent-child merges are useful to eliminate buckets that become too similar to their parents; sibling merges are useful to extrapolate frequency distributions to yet unseen regions in the data domain and also to consolidate buckets with similar density that cover nearby regions.

ISOMER [5] is a more recent feedback-based algorithm for building and maintaining multidimensional histograms. ISOMER uses the histogram structure of STHoles and the information-theoretic principle of maximum entropy to refine the histogram based on *query feedback records (QFR)*. QFRs are $\langle q, N(q) \rangle$ records that match queries against the size of the query result. Once ISOMER obtains a consistent set of QFRs, the algorithm computes the 'simplest' (in terms of entropy) histogram that is consistent with all QFRs added so far. The result is a maximization problem under a system of constraints, solved with iterative scaling. Furthermore, to meet a space budget ISOMER discards QFRs merges buckets in a way similar to STHoles.

Except for storing cardinalities, another useful statistic for selectivity estimation of queries with equality or LIKE selection predicates is the number of distinct values. Kaushik and Suciu [7] presented the first self-tuning histogram modelling cardinalities and distinct value counts, which was based on the same *entropy maximization (EM)* principle as ISOMER but with a different probability space. Due to the computational complexity of the resulting EM problem, they minimize instead the squared distance between the histogram's estimates and the query feedback viewed as vectors. However, their method can only construct one-dimensional histograms on numerical or categorical data.

Markl et al. [8] address the problem of combining complementary selectivity estimations from multiple sources (estimations which are computed using ISOMER histograms) to obtain a consistent selectivity estimation using the idea of maximum entropy. Similar to the approach in ISOMER, this work exploits all available information and avoids biasing the optimizer towards plans for which the least information is known [9].

Khachatryan et al. [10] note that like traditional index structures such as R-Trees, STHoles fails in high-dimensional data spaces and is sensitive to the order of tree construction. As far as the latter is concerned, they argue that if the first few queries define a top-level bucket structure that is bad, the subsequent tuning is unlikely to correct it. They propose an initializing with subspace buckets which are derived from a subspace clustering algorithm. They use the MineClus cell clustering algorithm, which outputs a set of clusters with an assigned importance. Each cluster consists of tuples and has dimensions $d_1, d_2, ..., d_k$. The corresponding bucket is the minimal rectangle containing these points-tuples and spans the entire length of every dimension not in $d_1, d_2, ..., d_k$. They showed that the new initialization improves estimation quality and, in some situations, reduces the number of buckets.

2.2 Histograms of categorical attributes

STHoles and, in general, workload-aware self-tuning histograms have been successfully used in relational databases as a low-overhead alternative to statically re-scanning database tables. The resulting histogram is focused towards the current workload, providing more accurate statistics for data regions that are being queried more frequently. Furthermore, they are able to adapt to changes in data distribution and thus are well-suited for datasets with frequently changing contents. They are, however, for the most part targeting numerical attributes, since they exploit the idea that a value range is an indication of the size of the range. Turning our attention to the Semantic Web, the *Resource Description Framework (RDF)* is the dominant standard for expressing information. RDF information is a graph where *properties* (labelled edges) link two resources to each other or one resource to a *literal* (a concrete value). The relevance of this discussion to self-tuning histograms is that RDF uses URIs as abstract symbols that denote resources. Given this prominent role of URIs in RDF data, extending self-tuning histograms to string attributes can have a significant impact in optimizing querying of RDF datasets.

There has been relatively limited amount of work around string selectivity estimation in the field of relational databases. Chaudhuri et al. [11] proposed to collect multiple candidate identifying substrings of a string using, for example, a Markov estimator and build a regression tree as a combination function of their estimated selectivities, in order to alleviate the selectivity underestimation problem of queries involving string predicates in previous methods, which used independence and Markov assumptions. In 2005, Lim et al. [12] introduced CXHist, which is a workload-aware histogram for selectivity estimation supporting a broad class of XML string-based queries. CXHist is the first histogram based on classification that uses feature distributions to summarize queries and quantize their selectivities into buckets and a naive-Bayes classifier to capture the mapping between queries and their selectivity.

Within the Semantic Web community itself, the SWOOGLE search engine collects metadata, such as classes, class instances and properties for web documents and relations between documents [13]. LODStats computes several schema-level statistical values for large-scale RDF datasets using an approach based on *statement streams* [14]. More closely related to our work is RDFStats [15], which is a generator for statistics of RDF sources like SPARQL endpoints. They generate different statistical items such as instances per class and histograms. Unlike our approach, they generate different static histograms (i.e. that must be rebuilt to reflect any changes in the RDF source) per class, property and XML data type. For range estimations on strings, RDFStats mentions three possibilities: (a) one bucket for each distinct string, resulting in large histograms; (b) reducing strings to prefixes; or (c) using a hash function to reduce the number of distinct strings, although no appropriate general-purpose hash function has been identified. However, as Harth et al. [16] have also noted in relation to Q-Trees for indexing RDF triples, hashing URIs is a purely syntactic mapping from URIs to numerical coordinates and fails to take into account the semantic similarity between resources; and no universally good function has been identified.

As URIs are the most prominent datatype in the Semantic Web, the absence of an extension that can naturally handle URI strings leaves Semantic Web data outside the scope of many developments in self-tuning histograms.

3 Self-Tuning String Histograms

In this section we establish a new histogram structure that extends the structure of the STHoles algorithm with the ability to cover strings. We also present the algorithms that construct and refine this new structure.

In our treatment, we first defer defining how string ranges are specified. Instead, we construct a framework of preliminary definitions where we specify the properties that must be satisfied by any compliant definition of string ranges. We then proceed to construct two alternative string ranges: the first one is based on prefixes and is a slight re-formulation of previous work [17] so that it complies with this framework. The second definition of string ranges is based on string distance.

3.1 Preliminaries

Let D be a *dimension*, any subset of D be a *range* in D, and $\mathcal{P}(D)$ the set of all possible ranges in D. A range can be defined either implicitly by constraints over the values of D or explicitly by enumeration. Note that $D \in \mathcal{P}(D)$, meaning that a range does not *need* to impose a restriction but can also include the whole dimension. Let H be a histogram of n dimensions $D_1, \dots D_n$. Let $\mathrm{V}(H)$ be the set of all possible n-dimensional vectors $(r_1, \dots r_n)$ where $\forall i \in [1, n] : r_i \in \mathcal{P}(D_i)$

A histogram is represented as an inclusion hierarchy of *buckets*; we shall use B_H to denote the set of buckets of a histogram H.

Definition 1. *Each* bucket $b \in B_H$ *is an entity of histogram* H *such that:*

- *b is associated with a* $\mathrm{box}(b) \in V_H$*, the vector that specifies the set of tuples that the bucket describes.*
- *b is associated with a* $\mathrm{size}(b)$ *which indicates the number of tuples that match* $\mathrm{box}(b)$
- *b is associated with n values* $\mathrm{dvc}(b, D_i), i = 1 \dots n$ *which indicate the number of distinct values appearing in dimension D_i of the tuples that match* $\mathrm{box}(b)$

We define the density of a bucket b to be the quantity

$$\mathrm{density}(b) = \frac{\mathrm{size}(b)}{\prod\limits_{i : r_i \in \mathrm{box}(b)} \mathrm{dvc}(b, D_i)}$$

Definition 2. *Every histogram implicitly includes a bucket* $b_\top$ *such that* $\mathrm{box}(b_\top) \equiv (D_1, \dots D_n)$ *that is, the bucket that imposes no restrictions in any of the dimensions of H and includes all tuples. We call this the* top bucket $b_\top$.

The implication of Definition 2 is that the overall size of the dataset and the number of distinct values in each dimension should be known (or at least approximated) regardless of what query feedback has been received. In our implementation we assume the *root bucket* (the top-most bucket of the hierarchy) as an approximation of the top.

Let $\mathcal{Q}_H$ be the set of all possible queries over the tables covered by H. Regardless of how they are syntactically expressed, we perceive $\mathcal{Q}_H$ as the set of all possible restrictions over the dimensions of H; thus:

Definition 3. *Each* query $q \in \mathcal{Q}_H$ *is an entity of histogram H such that:*

- *q is associated with a* $\mathrm{box}(q) \in V_H$*, the vector that specifies the restrictions expressed by the query*
- *q is associated with a* $\mathrm{size}(q)$ *which indicates the number of tuples that are returned by executing q*

As pointed out earlier, in our preliminary constructions ranges are simply defined as any subset of D, without making any requirements on how these are specified; in fact they may even be specified by enumeration. However, in order to realize the memory efficiency of workload-aware histograms, ranges should be specified intensionally, so that their representation consumes a memory unit regardless of how many elements match the specification. We shall present below the definitions we propose for string ranges; at this point, it suffices to define a range as follows:

Definition 4. *We define a* range $r \in \mathcal{P}(D)$ *of dimension* D *of histogram* H *to be an entity of* H *with the following properties:*

- *There is a* membership function $\text{member}_r : D \to \{true, false\}$ *that can consistently decide for any* $t \in D$ *whether it is or is not inside* r*.*
- *There is an* intersection function $\Cap : \mathcal{P}(D) \times \mathcal{P}(D) \to \mathcal{P}(D)$ *that returns the range resulting from the intersection of two ranges.*

It should be noted that we do not make any claims on the intersection function, although it is advantageous if such a function is approximately (if not exactly) the same as a function that would output a range that has as extension the intersection of the ranges' extensions. However, it should be possible to operate in instantiations of the framework where the (strict) intersection cannot be computed or syntactically represented for all pairs of ranges. In such instantiations, our relaxed definition of $\Cap$ provides the flexibility to define operators that roughly (but not exactly) correspond to producing a representation for the intersection of the extensions of its operands.

We use range intersection to also define multi-dimensional box intersection as follows:

Definition 5. *Given two boxes* $v_1, v_2 \in V_H$ *from the* n*-dimensional histogram* H*, let* $v_1 = (r_{1,1}, \dots r_{1,n})$ *and* $v_2 = (r_{2,1}, \dots r_{2,n})$*. We define* box intersection*:*

$$v_1 \Cap v_2 = (r_{1,1} \Cap r_{2,1}, \dots r_{1,n} \Cap r_{2,n})$$

Definition 6. *Given two boxes* $v_1, v_2 \in V_H$ *from the* n*-dimensional histogram* H*, let* $v_1 = (r_{1,1}, \dots r_{1,n})$ *and* $v_2 = (r_{2,1}, \dots r_{2,n})$*. We say that* v_1 encloses v_2 *iff* $\forall i \in [1, n]$ *at least one of the following holds:*

1. $r_{2,i} \subseteq r_{1,i} \subset D_i$*, that is, none of the ranges is the complete dimension and* $r_{2,i}$ *is contained within* $r_{1,i}$
2. $r_{2,i} = D_i$ *and* $r_{1,i} \subset D_i$*, that is, if one of the ranges is the complete dimension then it is enclosed by the one that is not.*
3. $r_{2,i} = r_{1,i} = D_i$*, that is, both ranges are the complete dimension.*

It should be noted that we have defined an unrestricted dimension as *being enclosed by* (rather than enclosing) a restriction. The rationale behind this will be explained in conjunction with *bucket merging* (Section 3.4).

Definition 7. *Given two boxes* $v_1, v_2 \in V_H$ *from histogram* H*,* v_1 tightly encloses v_2 *iff* v_1 encloses v_2 *and there is no* $u \in V_H$ *such that* $v_1 \Supset u \Supset v_2$

Definition 8. *Given a query $q \in \mathbb{Q}_H$, we associate with q the* best fit*, the set of buckets* $\mathrm{bf}(q) \subseteq B_H$ *such that*

$$\forall b \in \mathrm{bf}(q) : \mathrm{box}(b) \textit{ tightly encloses } \mathrm{box}(q)$$

Lemma 1. *For every query there is always a non-empty* best fit*.*

Proof. There is always at least one bucket that *encloses* any $\mathrm{box}(q)$, the *top bucket* $b_\top$ (Definition 2). If there is no other bucket that *encloses* $\mathrm{box}(q)$, then $b_\top$ *tightly encloses* $\mathrm{box}(q)$ (Definition 7) and thus $\mathrm{bf}(q) = \{b_\top\}$, which is non-empty. If there are other buckets that *enclose* $\mathrm{box}(q)$, then there is also at least one that *tightly encloses* $\mathrm{box}(q)$, so $\mathrm{bf}(q)$ is non-empty.

3.2 Cardinality Estimation

Being able to predict the size of querying results is important input for query execution optimizers, but the specifics of how this optimization is performed is outside the scope of this paper. We will here proceed to define metrics over the values associated with the buckets of H in order to predict $\mathrm{size}(q), q \in \mathcal{Q}_H$, the number of results returned by q.

In the literature, numerical intervals are used to succinctly define ranges and to efficiently decide if a query is enclosed by a bucket or not. The numerical difference between the interval's starting and ending value is sometimes used to define *range length* and, in multi-dimensional buckets, *bucket volume*: an estimator of the number of tuples in a bucket. We, accordingly, define range length as follows:

Definition 9. *Given a histogram dimension D and a range $r \in \mathcal{P}(D)$ we define the function* $\mathrm{length} : \mathcal{P}(D) \to \mathbb{R}$ *as follows:*

1. *Unrestricted ranges that span the whole dimension have length* 0*.*
2. *If r is an extensionally defined range of any type, then* $\mathrm{length}(r) = |r|$*, the number of distinct values in the range.*
3. *If r is a numerical range defined by an interval $[x, y]$, then* $\mathrm{length}(r) = y - x + 1$*.*

The addition of the unit term guarantees that the length cannot be zero even if $x = y$, i.e., even if the numerical range is a single point. This makes the third clause of the definition consistent with the second one, since for any number n we would expect the length of the singleton $\{n\}$ according to clause 2 to be the same as the length of the range $[n, n]$ that can also only include a single distinct value. It should also be noted that this is the only situation in which the length of a range can be 1. This property is important for Definition 10 below.

We will revisit this definition in Section 4 and complete it with the definition of length for URI ranges. Regardless of how length is defined for numerical, URI, or other types of ranges, we propose the following function as an estimator of the number of tuples that lie inside q, given a histogram:

Algorithm 1 Refinement of a histogram H given a set of queries W.

```
procedure REFINE($H$,$W$)
    for all queries $q \in W$ do
        if $q$ is not contained in $H$ then
            expand $H$'s root bucket so that it contains $q$
        for all buckets $b_i$ such that $q \Cap b_i \neq \emptyset$ do
            $(c_i, T_{c_i}, d_{c_i}) \leftarrow$ SHRINKBUCKET$(b_i, q)$
            if estimation is not accurate then
                DRILLHOLE$(b_i, c_i, T_{c_i}, d_{c_i})$
    while $H$ has too many buckets do
        Let $b_1$, $b_2$ in $H$ with the lowest penalty$_H(b_1, b_2)$
        MERGE$(b_1, b_2)$
```

Definition 10. *Given a histogram H and a query q, let* $\text{box}(q) = r_1, ...r_n$. *We define the function* $\text{est}_H : V_H \to \mathbb{R}$ *as follows:*

$$\text{est}_H(\text{box}(q)) = \sum_{b \in \text{bf}(q)} \frac{\text{size}(b)}{\prod_{i:length(r_i)=1} \text{dvc}(b, D_i)}$$

The intuition behind this definition is that we identify a best-fitting bucket (cf. Definition 8) and assume that tuples are uniformly distributed among the distinct values in each dimension. Since the query might have bindings for some of its dimensions, we use this assumption to apply simple division to estimate the fraction of the bucket's tuples that will be selected by the query dimensions that are unbinded variables. Naturally, this also assumes that the length of the range of the query's box can only be 1 for binded dimensions and is greater than 1 otherwise. This property is guaranteed by the definition of categorical and numerical length (Definition 9), and should also be observed by any extensions for other types.

3.3 Histogram Construction and Refinement

The construction of the histogram follows the same high level steps as the STHoles algorithm. In particular, we start with an empty histogram. For each query q in the workload, we identify *candidate buckets* b_i that intersect with q. For each candidate bucket b_i we compute $b_i \Cap q$ and these intersections constitute *candidate holes* c_i. We then shrink each candidate hole to the largest sub-region that does not intersect with the box of any other bucket, we count the exact number of tuples from the result stream that lie inside the shrunk hole and the distinct values count. Then, we determine whether the current density of the candidate bucket is close to the actual density of the candidate hole. If not, we 'drill' the candidate hole as a new histogram bucket and we move all children of b_i that are enclosed by c_i to the new bucket (Algorithms 1 and 2).

A point of divergence from STHoles is when shrinking candidate holes. Let X be the set of all buckets that partially intersect with a candidate hole c_i.

Algorithm 2 Drilling a hole in bucket b, given a candidate hole c and the counted cardinality T_c and distinct values $D_c(i)$ for each dimension D_i.

```
procedure DrillHole($b$, $c$, $T_c$, $d_c(\cdot)$)
    if box($b$) = box($c$) then
        size($b$) ← $T_c$
        dvc($b, D_i$) ← $D_c(i)$ $\forall i \in$ attributes
    else
        Add a new child $b_n$ of $b$ to the histogram
        box($b_n$) ← $c$
        size($b_n$) ← $T_c$
        dvc($b_n, D_i$) ← $d_c(i)$ $\forall i \in$ attributes
        Migrate all children of $b$ that are enclosed by $c$
        so they become children of $b_n$
```

Algorithm 3 Shrink a bucket that is enclosed by the intersection of b and q and does not partially intersect any other bucket.

```
function ShrinkBucket($b$, $q$)
    $c \leftarrow \text{box}(q) \cap \text{box}(b)$
    $\mathcal{P} \leftarrow \{b_i \in children(b) \mid c \cap \text{box}(b_i) \neq \emptyset \wedge \text{box}(b_i) \not\subseteq c\}$
    while $\mathcal{P} \neq \emptyset$ do
        Get first bucket $b_i \in \mathcal{P}$ and dimension $j$
        such that shrinking $c$ along $j$ by excluding $b_i$ results
        in the smallest reduction of $c$.
        Shrink $c$ along $j$
        $\mathcal{P} \leftarrow \{b_i \in children(b) \mid c \cap \text{box}(b_i) \neq \emptyset \wedge \text{box}(b_i) \not\subseteq c\}$
    Count from the result the number of tuples in $c$, $T_c$
    for all attributes $i$ do
        Count from the result the number of
        distinct values of the ith attribute in $c$, $d_c(i)$.
    return $(c, T_c, d_c(\cdot))$
```

STHoles selects at each step the pair $\langle x, j \rangle$ that comprises bucket $x \in X$ and dimension j such that shrinking c_i along j by excluding x has as a result the smallest reduction of c_i. Instead of checking for the optimal $\langle x, j \rangle$ our method selects the first pair where shrinking c_i along j by excluding x results in the smallest *relative* reduction of c_i's *length* in that dimension, the intuition being that often excluding x will give similar relative reduction along all dimensions. We then shrink c_i, we update participants and repeat the procedure until there are no participants left (Algorithm 3). This may result in a suboptimal shrink, but we avoid examining all possible combinations at each step. Furthermore, in STHoles the number of tuples in this shrunk subregion is estimated assuming uniformity; instead, we measure exactly the number of tuples and distinct values per dimension.

3.4 Bucket Merging

In order to limit the number of buckets and memory usage, buckets are *merged* to make space for drilling new holes. Following STHoles, our method looks for *parent-child* or *sibling* buckets that can be merged with minimal impact on the cardinality estimations. We diverge from STHoles when computing the box, size, and dvc associated with the merged bucket as well as in the *penalty* measure that guides the merging process towards merges that have the smallest impact on estimation accuracy.

Let b_1, b_2 be two buckets in the n-dimensional histogram H and let H' be the histogram after the merge and b_m the bucket in H' that replaces b_1 and b_2. In the *parent-child* case, the parent bucket, let that be b_1, *tightly encloses* the child bucket. In this case, we merge b_2 into b_1, so that $\text{box}(b_m) \equiv \text{box}(b_1)$. Any children that b_2 had become children of b_m.

In *sibling-sibling* merges, let b_p be the common parent bucket that *tightly encloses* both siblings b_1 and b_2. The merged bucket b_m is a child of b_p and the parent of all children of b_1 and b_2. The box of b_m must be such that it encloses the boxes of b_1 and b_2, without partially overlapping with any further siblings. Different implementations might achieve this either by defining checks that block sibling merges or by defining the box of b_m in such a way that it also encloses any further siblings that partially overlap with the extended box that encloses b_1 and b_2.

The size of b_m is estimated by adding the sizes of b_1 and b_2; the distinct values count of b_m is estimated by the maximum distinct values count among the merged buckets:

1. $\text{box}(b_p)$ *tightly encloses* $\text{box}(b_m)$
2. $\text{box}(b_m)$ *tightly encloses* both buckets b_1, b_2
3. $\text{box}(b_m)$ *tightly encloses* the boxes of all children of b_p that partially intersect either of b_1, b_2. That is, $\text{box}(b_m)$ encloses $\text{box}(b_c)$ for all b_c such that:
   (a) b_p *tightly encloses* b_c; and
   (b) $\text{box}(b_1)$ *partially overlaps* $\text{box}(b_c)$ or $\text{box}(b_2)$ *partially overlaps* $\text{box}(b_c)$
4. $\text{size}(b_m) = \sum_{k=1,2,c_1,\ldots} \text{size}(b_k)$
5. $\text{dvc}(b_m) = \max_{k=1,2,c_1,\ldots} \text{dvc}(b_k)$

It should be noted that the procedure that constructs the merged bucket b_m is deterministic and thus b_m can be uniquely determined by b_1 and b_2. In Point 3 above, it should be stressed that the partially intersecting buckets b_c are *not* merged into b_m, but that the latter is expanded so that it can assume b_c as its children. This is because in some algorithms (including STHoles), $\text{box}(b_m)$ can become larger than $\text{box}(b_1) \cup \text{box}(b_2)$ in order to have a succinct description with a single interval in each dimension. As a result, it might cut across other buckets; $\text{box}(b_m)$ should then be extended so as to subsume those as children. In order to avoid, however, dropping informative restrictions, STHoles only extends $\text{box}(b_m)$ along dimensions where the boxes of b_c do have a restriction. In order to capture this, we have defined the *encloses* relation (Definition 6) in a way that makes unrestricted dimensions *enclosed by* (rather than enclosing) restrictions.

In order to decide which is the optimal merge at any stage of histogram refinement, we need to balance between merges of buckets with similar statistics (minimizing the error introduced by discarding the statistics held in the merged buckets) and buckets with similar boxes (minimizing the error introduced by generalizing boxes beyond what was warranted by hole drilling, i.e., query feedback). To achieve the latter, we first define a *distance function* that evaluates

Definition 11. *Given a histogram H and any two of its boxes v_1 and v_2, we define the* distance *between v_1 and v_2 as any function* $\mathrm{distance}_H : V_H \times V_H \to \mathbb{R}$ *that has the following properties:*

– $\mathrm{distance}(v_1, v_1) = 0$
– *If v_1 encloses v_2 then* $\mathrm{distance}(v_1, v_2) = 0$

We can now define the *penalty function* that evaluates a possible merge:

Definition 12. *Given a histogram H and any three of its buckets b_1, b_2 and b_m, we define the* penalty function $\mathrm{penalty}_H : B_H \times B_H \to \mathbb{R}$ *of merging b_1 and b_2 into b_m as follows:*

$$\begin{aligned}
&\mathrm{penalty}_H(b_1, b_2) = \\
&\quad \frac{1}{2}\left(\frac{|\mathrm{density}(b_1) - \mathrm{density}(b_m)|}{\mathrm{density}(b_1) + \mathrm{density}(b_m)} + \frac{|\mathrm{density}(b_2) - \mathrm{density}(b_m)|}{\mathrm{density}(b_2) + \mathrm{density}(b_m)}\right) \\
&\quad + \sum_i \left(\frac{|\mathrm{dvc}(b_1, i) - \mathrm{dvc}(b_m, i)|}{\mathrm{dvc}(b_1, i) + \mathrm{dvc}(b_m, i)} + \frac{|\mathrm{dvc}(b_2, i) - \mathrm{dvc}(b_m, i)|}{\mathrm{dvc}(b_2, i) + dvc(b_m, i)}\right) \\
&\quad + \mathrm{distance}\left(\mathrm{box}\left(b_1\right), \mathrm{box}\left(b_2\right)\right)
\end{aligned}$$

The first two terms of this function represent the error in the statistics introduced by the merge while the third term increases the penalty for bucket pairs that are more distant as defined in Definition 11. Therefore, a sibling-sibling merge must have a small enough statistics-based penalty to be preferred over a parent-child merge, so that it can counter the fact that parent-child merges always have 0 distance-based penalty (since a child is always enclosed by its parent).

This penalty function allows us to rank the candidate bucket pairs and select the one with the minimum penalty. It should be noted though that not every bucket pair can be candidate for merging. The following merging constraints apply:

– The new $box(b_m)$ should not intersect with any other box, otherwise we would result in an inconsistent histogram
– The new $box(b_m)$ should not cover more than the half volume of its parent. This constraint is significant in order to control over-generalization in the early stages of an histogram when distant siblings might not be blocked from merging by the previous clause

– If the new $box(b_m)$ encloses the boxes of other buckets, b_m assumes these buckets as as its children.

The specifics of how to calculate the box of the merged bucket are left to be defined for each dimension type.

3.5 Extending for further types

We have deliberately avoided binding the discussion so far to specific data types, in order to define a general framework for histograms. The only exception is that the *length* of numerical ranges is already defined (Definition 9), in order to ensure backwards compatibility with numerical ranges in STHoles.

In order to specify the histograms of a new data type, which we shall here call *newtype*, one needs to provide the following:

1. A function *newtype member* that satisfies the definition of the generic *member* function (Definition 4).
2. A function *newtype intersection* that satisfies the definition of the generic *intersection* function (Definition 4).
3. A function *newtype length* that satisfies the definition of the generic *length* function (Definition 9).
4. A function *newtype distance* that satisfies the definition of the generic *distance* function (Definition 11).
5. A procedure for calculating the box of the resulting bucket in sibling merging. This procedure must satisfy the merging constraints in Section 3.4.

In the following section we will proceed to present two alternative specifications for URI histograms within this framework.

4 URI Ranges

As a first approach to expressing ranges of URIs, we have looked at prefixes. Prefixes can naturally express ranges of semantically related resources given the natural tendency to group together relevant items in hierarchical structures such as pathnames and URIs. We have also experimented with exploiting a geometrical analogy where we express a range as the volume around a central URI; again, we have defined distance in a way that prefixes weigh more, in order to preserve the bias towards hierarchical structures but offering more flexibility by comparison to exact prefix matching.

4.1 Prefix Ranges

In this approach we assume string prefixes as the description language for implicitly defining string ranges.

Definition 13. *Let H be a histogram and D be a string dimension of H. We define a* prefix range *r of D to be a set of strings, denoted as* $\mathrm{Pref}(r)$*. The strings in* $\mathrm{Pref}(r)$ *are to be interpreted as the prefixes of the elements of D that are in r. For any string $s \in D$ we define* prefix membership *as follows:*

$$\mathrm{member}_r(s) = \begin{cases} true, & \exists p \in \mathrm{Pref}(r) : s \textit{ starts with } p \\ false, & otherwise \end{cases}$$

In order to satisfy the requirements set in Section 3.5, we need to define the functions *prefix intersection*, *prefix length*, and *prefix distance* over prefix ranges, as well as the procedure for sibling merging.

Definition 14. *Let H be a histogram, D a string dimension of H, and $r_1, r_2 \in \mathcal{P}(D)$ two prefix ranges over D. The* range intersection *$r_1 \Cap r_2$ is defined as:*

1. *If r_1, r_2 are string ranges defined by sets of prefixes, then $r_1 \Cap r_2 = \{p | (p_1, p_2) \in \mathrm{Pref}(r_1) \times \mathrm{Pref}(r_2) \wedge (p = p_1 = p_2 \vee$ one of p_1, p_2 is a prefix of the other and p is the longest (more specific) of the two$)\}$*
2. *If one of the ranges is a string range defined by sets of prefixes (say r_1 without loss of generality) and the other is an explicit set of strings (say r_2), then $r_1 \Cap r_2 = \{v | v \in r_2 \wedge \exists p \in r_1 : p$ is a prefix of $v\}$*
3. *In any other case, $r_1 \Cap r_2 = r_1 \cap r_2$*

Definition 15. *Given a histogram dimension D and a range $r \in \mathcal{P}(D)$ we define the function* $\mathrm{length} : \mathcal{P}(D) \to \mathbb{R}$ *as follows:*

1. *Unrestricted ranges that span the whole dimension have length* 0.
2. *If r is an extensionally defined range of any type, then* $\mathrm{length}(r) = |r|$*, the number of distinct values in the range.*
3. *If r is a numerical range defined by an interval $[x, y]$, then* $\mathrm{length}(r) = y - x + 1$.
4. *If r is a string range defined by a set of prefixes* $\mathrm{Pref}(r)$*, then* $\mathrm{length}(r) = 1 + |\mathrm{Pref}(r)|$

It should be noted that no prefix range can ever be guaranteed to be equivalent to an extensional singleton range, since any valid URI prefix can be extended into a longer valid URI subsumed by the prefix. Therefore, all and only extensional singleton ranges can have a length of 1, which satisfies Requirement 3.

Definition 16. *Let r_1 and r_2 be prefix ranges. We define the* prefix distance *between r_1 and r_2 to be a constant* 0 *for any r_1, r_2.*

That is to say, in this setup there is no bias in sibling merges towards more similar prefixes and candidate merges are evaluated only on the basis of the similarity of the statistics in the buckets.

Box of merged siblings Suppose that sibling buckets b_1 and b_2 are to be merged. The box of the merged bucket b_m is calculated as the union of the prefixes in each:

$$\mathrm{Pref}(\mathrm{box}(b_m)) = \mathrm{Pref}(\mathrm{box}(b_1)) \cup \mathrm{Pref}(\mathrm{box}(b_2))$$

4.2 Similarity Ranges

In this approach we use the Jaro-Winkler similarity metric [18] to define the distance between two strings. This metric is suitable for URI comparison since it provides preference to the strings that match exactly at the beginning. Based on this, we define URI ranges as spherical volumes around a characteristic central URI, so that a range is specified by a URI (the center) and the radius around it that is within the range.

Definition 17. *Let H be a histogram and D be a string dimension of H. Let* $\mathrm{JW} : D \times D \rightarrow [0, 1]$ *be the Jaro-Winkler metric that assigns a similarity to an unordered pair of strings from D. We define* similarity range r_d *as a tuple* $r_d = \langle c, R\rangle$ *where c is a string called the* center *of r denoted as* $\mathrm{center}(r)$ *and* $R \in \mathbb{R}$ *is called the* radius *of r and denoted as* $\mathrm{radius}(r)$*. For any string $s \in D$ we define* similarity membership *as follows:*

$$\mathrm{member}_r(s) = \begin{cases} true, & \textit{if}\ 1 - \mathrm{JW}(s, center(r)) \leq radius(r) \\ false, & otherwise \end{cases}$$

In order to satisfy the requirements set in Section 3.5, we need to define the functions *similarity intersection*, *similarity length*, and *similarity distance* over similarity ranges, as well as the procedure for sibling merging.

Definition 18. *Given two similarity ranges of the same dimension* $r_1, r_2 \in \mathcal{P}(D)$ *their* similarity intersection *is defined as* $r_1 \Cap r_2 = \langle c', R'\rangle$ *where:*

$$c' = center(r_i)\ \textit{where}\ i = \mathrm{argmax}_{i=1,2}\, \mathrm{radius}(r_i)$$
$$R' = \max\{0, \mathrm{radius}(r_1) + \mathrm{radius}(r_2) - \mathrm{distance}_H(r_1, r_2)\}$$

Definition 19. *Given a histogram dimension D and a range $r \in \mathcal{P}(D)$ we define the* similarity length *function* $\mathrm{length} : \mathcal{P}(D) \rightarrow \mathbb{R}$ *as follows:*

1. *Unrestricted ranges that span the whole dimension have length* 0*.*
2. *If r is an extensionally defined range of any type then* $\mathrm{length}(r) = |r|$*, the number of distinct values in the range.*
3. *If r is a numerical range defined by an interval $[x, y]$, then* $\mathrm{length}(r) = y - x + 1$*.*
4. *If r is a similarity range then* $\mathrm{length}(r) = 1 + \mathrm{radius}(r)$

It should be noted that range $\langle u, 0\rangle$ has u as its single member and is equivalent to the extensional singleton range u. The similarity range length is 1 in both cases, which satisfies Requirement 3.

Definition 20. *Let r_1 and r_2 be similarity ranges. We define the* similarity distance *between r_1 and r_2 using the Jaro-Winkler similarity of their centers:*

$$\mathrm{distance}_H(r_1, r_2) = 1 - \mathrm{JW}(\mathrm{center}(r_1), \mathrm{center}(r_2))$$

Box of merged siblings Suppose that sibling buckets b_1 and b_2 are to be merged. The box of the merged bucket b_m is calculated for each dimension i that is URI dimension, where r_1 is the range of b_1 in dimension i, r_2 is the range of b_2 in dimension i, and r_m is the range of b_m in dimension i. We assume that for every range r_i we can assign consistently an id and without loss of generality let r_1 be the range with the smallest id.

1. If $radius(r_1) = 0$ and $radius(r_2) = 0$, then :

$$\begin{aligned} center(r_m) &= center(r_1) \\ radius(r_m) &= \text{distance}_H(r_1,\ r_2) \end{aligned}$$

2. If $radius(r_1) \neq 0 \wedge radius(r_2) \neq 0$, then :

$$center(r_m) = \begin{cases} center(r_1), \text{ if } radius(r_1) \geq radius(r_2) \\ center(r_2), \text{ otherwise} \end{cases}$$

$$radius(r_m) = \begin{cases} \text{distance}_H(r_1,\ r_2) + radius(r_2), \text{ if } radius(r_1) \geq radius(r_2) \\ \text{distance}_H(r_1,\ r_2) + radius(r_1), \text{ otherwise} \end{cases}$$

3. otherwise,

$$center(r_m) = \begin{cases} center(r_1), \text{ if } radius(r_1) \neq 0 \\ center(r_2), \text{ otherwise} \end{cases}$$

$$radius(r_m) = \text{distance}_H(r_1,\ r_2)$$

That is, the center of the merged range is that of the range with the greater radius, and the radius of the merged range is large enough so that the merged range also encloses the range with the smaller radius. The intuition behind this definition is that by assuming the larger of the two ranges as the basis for the merged range, a smaller expansion will be needed in order to enclose the other range, reducing the risk of over-generalizing.

4.3 Discussion

We have defined a multi-dimensional histogram over numerical, string, and categorical data. The core added value of this work is that we introduce the notion of *descriptions* in string dimensions, akin to intervals for numerical dimensions. This has considerable advantages for RDF stores and, more generally, in the Semantic Web and Linked Open Data domain, where URIs have a prominent role and offer the opportunity to exploit the hierarchical structure of their string representation.

Initially, we propose *prefixes* as the formalism for expressing string ranges, motivated by its applicability to URI structure. We then relax this formalism, using *similarity ranges* to describe string ranges based on string distances. This is no loss of generality, since it is straightforward to use more expressive pattern formalisms (such as regular expressions) without altering the core method but at a considerable computational cost. The only requirement is that *membership*, *intersection* and some notion of *length* can be defined. Length, in particular, can be used in the way STHoles uses it as an indication of a bucket's size relative to the size of its parent bucket. If a metric of *distance* or *dissimilarity* can be defined, this is also exploited to introduce bias towards merging similar ranges, but this is not required.

What allows us to relax the definition of length by comparison to STHoles, is that for range queries we return the statistics of the bucket that more tightly encloses the query, instead of returning an estimation based on the ratio of the volume occupied by the query to the volume of the overall bucket. In other words, we use *length* more as a metric of the size of description, rather than a metric of the bucket size (the number of tuples that fit this description). To compensate, we exactly measure in query results (rather than estimate) bucket size when shrinking buckets, compensating for the extra computational time by avoiding examining all combinations of buckets $\times$ dimensions (cf. Section 3.3). For point queries (with unit length), we also take into account statistics about distinct value counts in a bucket, increasing the accuracy of the estimation.

A limitation of our algorithm is that when we merge two sibling buckets we assign to the resulting bucket the sum of the sizes of the merged buckets and of the children of the resulting bucket, which is an overestimation of the real size. Furthermore, we also assign as distinct value count the maximum of the distinct value counts of these buckets, which is an underestimation of the real distinct value count. These estimations will persist until subsequent workload queries effect an update of merged bucket's statistics and will be used in cardinality estimations. We try to compensate for these possibly inaccurate estimations by carefully selecting buckets for sibling-sibling merging and defining a sibling-sibling merge penalty which favours the merging of buckets which not only have similar statistics, i.e. densities and distinct value counts, but their central strings are also similar. Besides empirically testing and tuning these estimators, we are also planning to extend the theoretical framework so that estimated values are represented as ranges or distributions, and subsequent calculations take into account the whole range or the distribution parameters rather than a single value.

In general, and despite these limitations, our framework is an accurate theoretical account of STHoles, a state-of-the-art algorithm for self-tuning multi-dimensional numerical histograms, and an extension to heterogeneous numerical/string histograms that is backwards-compatible with STHoles.

5 Experiments

To empirically validate our approach, the algorithm presented above has been implemented in Java as the *STRHist* module of the *Semagrow Stack* [19], an optimized distributed querying system for the Semantic Web.[4] The execution

[4] *STRHist* is available at `https://github.com/semagrow/strhist`
For more details on Semagrow, please see `http://semagrow.github.io`

flow of the Semagrow Stack starts with client queries, analysed to build an optimal *query plan*. The optimizer relies on cardinality statistics (produced by STRHist) in order to provide an execution plan for the Semagrow *Query Execution Engine*. This engine, besides joining results and serving them to the client application, also forwards to STRHist measurements collected during query execution. STRHist analyses these query feedback logs in batches to maintain the histogram that is used by the optimizer. The histogram is persisted in RDF stores using the Sevod vocabulary [20], which expresses the in-memory tree of *bucket* objects that is the internal representation of STRHist.

5.1 Experimental Setup

We applied STRHist to the AGRIS bibliographic database on agricultural research and technology maintained by the *Food and Agriculture Organization of the UN*. AGRIS comprises approximately 23 million RDF triples describing 4 million distinct publications with standard bibliographic attributes.[5] AGRIS consolidates data from more than 150 institutions from 65 countries. Bibliography items are denoted by URIs that are constructed following a convention that includes the location of the contributing institution and the date of incorporation into AGRIS. As scientific output increases through the years and since there is considerable variation in the scientific output of different countries, there are interesting generalizations to be captured by patterns over publication URIs.

We define a 3-dimensional histogram over subject, predicate and object variables. Subject URIs are represented as strings[6] while predicate URIs are treated as categorical values, since there is always a small number of distinct predicates. Each bucket is composed of a 3-dimensional subject/predicate/object bounding box, a size indicating the number of triples contained in the bucket, and the number of distinct subjects, predicates and objects.

We experiment on a real query workload extracted from the logs of the user evaluation of the Semagrow Stack [21]. We separated the workload into a training set that is used to refine of a histogram H over D and an evaluation set that is used to compare the statistics reported by the histogram against the actual dataset. Specifically, we measure the *average absolute estimation error* and the *root mean square error* of histogram H on the respective workload W:

$$\begin{aligned}
\mathrm{err}^{ABS}_{H,D}(W) =& \frac{1}{|W|} \sum_{q \in W} |\mathrm{est}_H(q) - \mathrm{act}_D(q)| \\
\mathrm{err}^{RMS}_{H,D}(W) =& \frac{1}{|W|} \sqrt{\sum_{q \in W} \left(\mathrm{est}_H(q) - \mathrm{act}_D(q)\right)^2}
\end{aligned}$$

[5] Please see `http://agris.fao.org` for more details on AGRIS. The AGRIS site mentions 7 million distinct publications, but this includes recent additions that are not in end-2013 data dump used for these experiments.

[6] We use the *canonical string representation* of URIs as defined in Section 2, IETF RFC 7320 (`http://tools.ietf.org/html/rfc7320`)

Training	Similarity ranges					Prefix ranges				
	Error		Merges			Error		Merges		
Batch	RMS	Abs	PC	SS	Total	RMS	Abs	PC	SS	Total
01	0.283	2.14	0	0	0	0.283	2.14	0	0	0
02	0.414	2.58	3	3	6	0.457	2.67	5	12	17
03	1.728	9.26	23	6	29	1.562	6.61	8	30	38
04	1.758	9.84	19	8	27	2.350	11.55	11	28	39
05	0.899	7.89	13	6	19	2.711	15.13	12	30	42
06	4.483	40.84	9	13	22	5.856	26.36	9	23	32
07	4.691	44.66	31	0	31	6.844	32.58	11	28	49
08	4.762	46.08	44	1	45	6.724	38.20	5	44	49
09	4.735	45.58	31	21	52	6.911	41.52	5	42	47
10	4.787	46.57	20	4	24	7.968	46.96	11	28	39
11	4.794	47.07	25	3	28	10.444	60.59	11	28	39
12	4.814	47.07	15	6	21	12.153	70.67	13	27	40
13	4.814	43.56	23	6	29	13.883	81.95	13	28	41
14	4.608	43.56	23	8	31	14.201	85.07	12	27	39
15	4.608	47.58	28	6	34	14.201	85.07	11	28	39
16	4.841	47.58	29	4	33	19.365	110.09	14	28	42
17	4.841	47.58	35	4	39	23.147	131.65	14	28	42
18	4.841	47.58	24	5	29	23.415	134.37	13	27	40
19	4.841	47.58	24	4	28	23.792	137.85	10	28	38
20	4.841	47.58	41	1	42	23.792	137.85	15	28	43
21	4.841	47.58	32	5	37	27.048	157.13	13	28	41
22	4.841	47.58	14	2	16	27.048	157.13	14	28	42
23	4.841	47.58	27	2	29	27.567	162.20	13	28	41
24	4.841	47.58	14	1	15	27.567	162.20	2	8	10

Table 1. Estimation error (RMS and absolute) versus training batch and merges (parent-child (PC) and sibling-sibling (SS) merges) using prefixes and similarity ranges. Configured for a maximum of 50 buckets.

where $\mathrm{est}_H(q)$ is the cardinality estimation for query q and $\mathrm{act}_D(q)$ is the actual number of tuples in D that satisfy q.

The expected behaviour of the algorithm is to improve estimates by adding buckets that punch holes and add sub-buckets in areas where there is a difference between the actual statistics and the histogram estimates. Considering how client applications access some 'areas' more heavily than others, the algorithm zooms into such critical regions to provide more accurate statistics. Naturally, the more interesting observations relate to the effect of merges as soon as the available space is exhausted, so we have allocated to STRHist unrealistically small memory (50 and 100 buckets).

5.2 Results

The AGRIS workload queries follow the same template: Both subjects and predicate URIs are defined by the query, leaving the object dimension unrestricted.

As it represents a real scenario, we may have duplicate queries in the workload. To generate the workload we randomly select a set of queries for refinement and another set for evaluation. Therefore, we create 24 batches of 55 training queries, totalling 1320 training queries, followed by a set of 100 evaluation queries used to compare the estimations against the actual size of the query results and the estimated ones. We experiment with different system configurations. Specifically, we set a maximum of 100 and 50 buckets. Moreover, we evaluate both reported representations for string ranges (i.e. prefix ranges and similarity ranges). Tables 1, 2 depict the average errors of the evaluation queryset and the number of merges performed during each training batch.

One can note that the similarity range approach produced more accurate estimations, especially when the maximum number of buckets is very limited causing more merges. Using this observation we can infer that the similarity range approach makes better merging decisions than the prefix range one. The reason that prefixes cannot create as good merged buckets as in the similarity ranges is that (a) prefixes as a succinct description is more restrictive and (b) the AGRIS URIs have a hierarchical structure, but this structure is not that deep that it would make strict prefixes expressive. Notice that the total merges performed per batch are fewer in the similarity range case. This is due to the fact that more training queries are already accurately estimated and thus the histogram refinement algorithm discards them without drilling new holes. Moreover, this observation is also consistent even after considerable merges have been applied to the histogram, deducing that merged buckets are not introducing significant error to the estimations.

The histogram stabilizes after a certain number of training batches, as evidenced by the fact that the error remains constant. A significant difference can be seen in the type of merging preferred by the two approaches: the number of the parent-child merges is higher in similarity range approach, while the prefix range approach prefers the sibling merging. This demonstrates the bias towards parent-child merges encoded by the distance-based penalty in similarity merging.

6 Conclusions

In this article we have presented an algorithm for building and maintaining multi-dimensional histograms exploiting query feedback. Our algorithm is based on STHoles algorithm, but extends it to also handle URIs. One significant contributions of the article is that it establishes a framework that formalises histograms over arbitrary data types and identifies the specification of a language for specifying data ranges as a key element of histograms. Building upon this, we have identified the properties that any such language should have for histogram refinement algorithms to be applicable.

This led to the second major contribution, that of proposing the Jaro-Winkler similarity metric as an appropriate basis upon which to build a formalization of histograms over URI strings. This metric has the advantage of accommodating the hierarchical nature of URI strings by placing more importance on the be-

Training	Similarity ranges					Prefix ranges				
	Error		Merges			Error		Merges		
Batch	RMS	Abs	PC	SS	Total	RMS	Abs	PC	SS	Total
01	0.283	2.14	0	0	0	0.283	2.14	0	0	0
02	0.259	1.73	0	0	0	0.259	1.73	0	0	0
03	0.259	1.73	0	0	0	0.259	1.73	0	0	0
04	0.408	2.56	10	4	14	0.259	1.73	7	8	15
05	1.688	8.88	20	10	30	0.259	1.73	6	30	36
06	1.768	10.94	15	9	24	0.259	1.73	10	24	34
07	4.581	32.96	24	4	28	0.259	1.73	13	28	41
08	5.886	40.53	23	11	34	0.472	2.48	9	33	42
09	8.236	76.94	37	19	56	1.919	5.91	1	52	53
10	8.236	76.94	22	3	25	2.687	8.87	13	29	42
11	6.654	50.75	17	5	22	4.624	12.11	11	27	38
12	6.136	43.52	12	9	21	4.960	13.79	13	28	41
13	5.921	40.84	18	5	23	5.528	15.23	12	28	40
14	5.530	35.94	13	3	26	5.537	15.53	13	27	40
15	5.740	35.92	7	5	12	5.537	15.93	13	28	41
16	5.740	35.94	9	4	13	5.806	16.92	13	27	40
17	6.190	41.42	16	4	20	5.955	17.72	10	28	38
18	5.623	34.68	13	5	18	5.955	17.72	11	27	38
19	5.623	34.68	10	2	12	9.658	25.56	8	22	30
20	5.623	34.68	6	0	6	10.846	28.44	15	28	43
21	5.623	34.65	12	7	19	10.846	28.44	11	27	38
22	6.102	37.50	12	11	23	12.453	33.24	11	28	39
23	6.182	38.47	21	0	21	12.872	35.16	13	27	40
24	10.137	98.79	11	0	11	12.876	35.56	8	17	25

Table 2. Estimation error (RMS and absolute) versus training batch and merges (parent-child (PC) and sibling-sibling (SS) merges) using prefixes and similarity ranges. Configured for a maximum of 100 buckets.

ginning of the string, while being more flexible than strict prefix matching. This gives our system a great advantage over the state of the art, where ranges are only defined over numerical data and strings are treated as categorical data that can only be described by enumeration: by having the ability to succinctly describe ranges of related URI strings, finer (and thus more accurate) histograms can fit a given amount of memory.

As future work, we will experiment with a more sophisticated estimation of the size of a bucket based from the radius of its box. One idea would be to dynamically adapt a conversion ratio parameter to the observed query feedback, so as to better fit each given dimension in a dataset. This will improve multi-dimensional volume calculations, since it will lift the assumption that the breadth of a URI description and the size of the data that fits the description grow uniformly. An even more ambitious goal is to define the length of URI string ranges in a way that it can be combined with numerical range length, so that multi-dimensional and heterogeneous (strings and numbers) buckets can be assigned a meaningful volume.

Another strain of future research will experiment with finer representations of clusters of URIs than the radius around a single central URI. This would allow us to improve sibling merging, as our current approach is prone to overgeneralizing and making the histogram sensitive to the query feedback it receives when it is first constructed.

With respect to the software development, we plan to develop a more scalable implementation of the algorithm which will be able to efficiently serve histograms from databases and not from in-memory Java objects. Although the unavoidable delay is not critical for the refinement phase, it can be unacceptable for the run-time usage of the histogram by query optimizers. To keep such delays manageable, a caching mechanism will need to be integrated in the implementation so that the most frequent accesses to the histogram are served from a memory cache.

Acknowledgements The research leading to these results has received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement No. 318497. For more details about the SemaGrow project please see `http://www.semagrow.eu` and about the Semagrow system please see `http://semagrow.github.io`

References

1. Bruno, N., Chaudhuri, S.: Exploiting statistics on query expressions for optimization. In: Proceedings of the 2002 ACM International Conference on Management of Data (SIGMOD '02), New York, NY, USA, ACM (2002) 263–274
2. Stillger, M., Lohman, G.M., Markl, V., Kandil, M.: LEO - DB2's LEarning optimizer. In: Proceedings of the 27th International Conference on Very Large Data Bases. VLDB '01, San Francisco, CA, USA, Morgan Kaufmann Publishers Inc. (2001) 19–28

3. Aboulnaga, A., Chaudhuri, S.: Self-tuning histograms: Building histograms without looking at data. In: Proceedings of the 1999 ACM International Conference on Management of Data (SIGMOD '99), New York, NY, USA, ACM (1999) 181–192
4. Bruno, N., Chaudhuri, S., Gravano, L.: STHoles: a multidimensional workload-aware histogram. In: Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data (SIGMOD '01. (2001) 211–222
5. Srivastava, U., Haas, P.J., Markl, V., Kutsch, M., Tran, T.M.: ISOMER: Consistent histogram construction using query feedback. In: Proceedings of the 22nd International Conference on Data Engineering (ICDE '06), Washington, DC, USA, IEEE Computer Society (2006)
6. Roh, Y.J., Kim, J.H., Chung, Y.D., Son, J.H., Kim, M.H.: Hierarchically organized skew-tolerant histograms for geographic data objects. In: Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data. SIGMOD '10, New York, NY, USA, ACM (2010) 627–638
7. Kaushik, R., Suciu, D.: Consistent histograms in the presence of distinct value counts. Proc. VLDB Endow. **2** (2009) 850–861
8. Markl, V., Haas, P.J., Kutsch, M., Megiddo, N., Srivastava, U., Tran, T.M.: Consistent selectivity estimation via maximum entropy. The VLDB Journal **16** (2007) 55–76
9. Bruno, N., Chaudhuri, S., Weikum, G.: Database tuning using online algorithms. In Liu, L., Özsu, M.T., eds.: Encyclopedia of Database Systems. Springer US (2009) 741–744
10. Khachatryan, A., Müller, E., Stier, C., Böhm, K.: Sensitivity of self-tuning histograms: Query order affecting accuracy and robustness. In: Proceedings of the 24th International Conference on Scientific and Statistical Database Management (SSDBM '12), Berlin, Heidelberg, Springer-Verlag (2012) 334–342
11. Chaudhuri, S., Ganti, V., Gravano, L.: Selectivity estimation for string predicates: Overcoming the underestimation problem. In: Proceedings of the 20th International Conference on Data Engineering (ICDE '04), Washington, DC, USA, IEEE Computer Society (2004)
12. Lim, L., Wang, M., Vitter, J.S.: CXHist: An on-line classification-based histogram for XML string selectivity estimation. In: Proceedings of the 31st International Conference on Very Large Data Bases (VLDB 2005), Trondheim, Norway, 30 August – 2 September 2005. (2005) 1187–1198
13. Ding, L., Finin, T., Joshi, A., Pan, R., Cost, R.S., Peng, Y., Reddivari, P., Doshi, V., Sachs, J.: Swoogle: A search and metadata engine for the semantic web. In: Proceedings of the Thirteenth ACM International Conference on Information and Knowledge Management. CIKM '04, New York, NY, USA, ACM (2004) 652–659
14. Auer, S., Demter, J., Martin, M., Lehmann, J.: LODStats – an extensible framework for high-performance dataset analytics. In: Proceedings of the 18th International Conference on Knowledge Engineering and Knowledge Management (EKAW '12), Berlin, Heidelberg, Springer-Verlag (2012) 353–362
15. Langegger, A., Wöss, W.: RDFStats – an extensible RDF statistics generator and library. In: 23rd International Workshop on Database and Expert Systems Applications, Los Alamitos, CA, USA, IEEE Computer Society (2009) 79–83
16. Harth, A., Hose, K., Karnstedt, M., Polleres, A., Sattler, K.U., Umbrich, J.: Data summaries for on-demand queries over linked data. In: Proceedings of the 19th International World Wide Web Conference (WWW 2010), Raleigh, NC, USA, 26-30 April 2010. (2010)

17. Zoulis, N., Mavroudi, E., Lykoura, A., Charalambidis, A., Konstantopoulos, S.: Workload-aware self-tuning histograms on string data. In: Proceedings of the 26th International Conference on Database and Expert System Applications (DEXA 2015), Valencia, Spain, 1–4 September 2015. (2015)
18. Winkler, W.E.: String comparator metrics and enhanced decision rules in the Fellegi-Sunter model of record linkage. Technical report, Proceedings of the Section on Survey Research Methods, American Statistical Association, pp. 354–359 (1990)
19. Charalambidis, A., Troumpoukis, A., Konstantopoulos, S.: SemaGrow: Optimizing federated SPARQL queries. In: Proceedings of the 11th International Conference on Semantic Systems (SEMANTiCS 2015), Vienna, Austria, 15-18 September 2015. (2015)
20. Charalambidis, A., Konstantopoulos, S., Karkaletsis, V.: Dataset descriptions for optimizing federated querying. In: Companion Proceedings of the 24th International World Wide Web Conference Companion Proceedings (WWW 2015), Poster Session, Florence, Italy, 18-22 May 2015. (2015)
21. Celli, F., Keizer, J., Jaques, Y., Konstantopoulos, S., Vudragović, D.: Discovering, indexing and interlinking information resources. F1000Research **4** (2015) Version 2; referees: 3 approved.