Adaptive Sampling for Rapidly Matching Histograms

Stephen Macke†, Yiming Zhang‡, Silu Huang†, Aditya Parameswaran†
University of Illinois-Urbana Champaign
†{smacke,shuang86,adityagp}@illinois.edu | ‡ym_zhang@sjtu.edu.cn

ABSTRACT

In exploratory data analysis, analysts often have a need to identify histograms that possess a specific distribution, among a large class of candidate histograms, e.g., find countries whose income distribution is most similar to that of Greece. This distribution could be a new one that the user is curious about, or a known distribution from an existing histogram visualization. At present, this process of identification is brute-force, requiring the manual generation and evaluation of a large number of histograms. We present FastMatch: an end-to-end approach for interactively retrieving the histogram visualizations most similar to a user-specified target, from a large collection of histograms. The primary technical contribution underlying FastMatch is a probabilistic algorithm, HistSim, a theoretically sound sampling-based approach to identify the top-k closest histograms under ℓ_1 distance. While HistSim can be used independently, within FastMatch we couple HistSim with a novel system architecture that is aware of practical considerations, employing asynchronous block-based sampling policies, building on lightweight sampling engines developed in recent work [47]. FastMatch obtains near-perfect accuracy with up to $35\times$ speedup over approaches that do not use sampling on several real-world datasets.

1. INTRODUCTION

In exploratory data analysis, analysts often generate and peruse a large number of visualizations to identify those that *match desired criteria*. This process of iterative "generate and test" occupies a large part of visual data analysis [13, 33, 62], and is often cumbersome and time consuming, especially on very large datasets that are increasingly the norm. This process ends up impeding interaction, preventing exploration, and delaying the extraction of insights.

Example 1: Census Data Exploration. Alice is exploring a census dataset consisting of hundreds of millions of tuples, with attributes such as gender, occupation, nationality, ethnicity, religion, adjusted income, net assets, and so on. In particular, she is interested in understanding how applying various filters impacts the relative distribution of tuples with different attribute values. She might ask questions like *Q1:* Which countries have similar distributions of wealth to that of Greece? *Q2:* In the United States, which professions have an ethnicity distribution similar to the profession of doctor? *Q3:* Which (nationality, religion) pairs have a similar distribution of number of children to Christian families in France?

Example 2: Taxi Data Exploration. Bob is exploring the distribution of taxi trip times originating from various locations around Manhattan. Specifically, he plots a histogram showing the distribution of taxi pickup times for trips originating from various locations. As he varies the location, he examines how the histogram changes, and he notices that choosing the location of a popular nightclub skews the distribution of pickup times heavily in the range of 3am to 5am. He wonders *Q4:* Where are the other locations around Manhattan that have similar distributions of pickup times? *Q5:* Do they all have nightclubs, or are there different reasons for the late-night pickups?

Example 3: Sales Data Exploration. Carol has the complete history of all sales at a large online shopping website. Since users must enter birthdays in order to create accounts, she is able to plot the age distribution of purchasers for any given product. To enhance the website's recommendation engine, she is considering recommending products with similar purchaser age distributions. To test the merit of this idea, she first wishes to perform a series of queries of the form *Q6:* Which products were purchased by users with ages most closely following the distribution for a certain product—a particular brand of shoes, or a particular book, for example? Carol wishes to perform this query for a few test products before integrating this feature into the recommendation pipeline.

These cases represent scenarios that often arise in exploratory data analysis—finding matches to a specific distribution. The focus of this paper is to *develop techniques for rapidly exploring a large class of histograms to find those that match a user-specified target*.

Referring to *Q1* in the first example,a typical workflow used by Alice may be the following: first, pick a country. Generate the corresponding histogram. This could be done either using a language like R, Python, or Javascript, with the visualization generated in ggplot [73] or D3 [15], or using interactions in a visualization platform like Tableau [70]. Does the visualization look similar to that of Greece? If not, pick another, generate it, and repeat. Else, record it, pick another, generate it, and repeat. If only a select few countries have similar distributions, she may spend a huge amount of time sifting through her data, or may simply give up early.

The Need for Approximation. Even if Alice generates all of the candidate histograms (i.e., one for each country) in a single pass, programmatically selecting the closest match to her target (i.e., the Greece histogram), this could take unacceptably long. If the dataset is tens of gigabytes and every tuple in her census dataset contributes to some histogram, then any exact method must necessarily process tens of gigabytes—on a typical workstation, this can take tens of seconds even for in-memory data. Recent work suggests that latencies greater than 500ms cause significant frustration for end-users and lead them to test fewer hypotheses and potentially identify fewer insights [54]. Thus, in this work, we explore approximate techniques that can return matching histogram visualizations with accuracy guarantees, but much faster.

One tempting approach is to employ approximation using pre-computed samples [7, 6, 5, 10, 31, 28], or pre-computed sketches or other summaries [18, 60, 77]. Unfortunately, in an interactive exploration setting, pre-computed samples or summaries are not helpful, since the workload is unpredictable and changes rapidly, with

more than half of the queries issued one week completely absent in the following week, and more than 90% of the queries issued one week completely absent a month later [58]. In our case, based on the results for one matching query, Alice may be prompted to explore different (and arbitrary) slices of the same data, which can be exponential in the number of attributes in the dataset. Instead, we materialize samples on-the-fly, which doesn't suffer from the same limitations and has been employed for generating approximate visualizations incrementally [64], and while preserving ordering and perceptual guarantees [46, 8]. To the best of our knowledge, however, on-demand approximate sampling techniques have not been applied to the problem of evaluating a large number of visualizations for matches in parallel.

Key Research Challenges. In developing an approximation-based approach for rapid histogram matching we immediately encounter a number of theoretical and practical challenges.

1. Quantifying Importance. To benefit from approximation, we need to be able to quantify which samples are "important" to facilitate progress towards termination. It is not clear how to assess this importance: at one extreme, it may be preferable to sample more from candidate histograms that are more "uncertain", but these histograms may already be known to be rather far away from the target. Another approach is to sample more from candidate histograms at the "boundary" of top-k, but if these histograms are more "certain", refining them further may be useless. Another challenge is when we quantify the importance of samples: one approach would be to reassess importance every time new data become available, but this approach could be computationally costly.

2. Deciding to Terminate. Our algorithm needs to ascribe a confidence in the correctness of partial results in order to determine when it may safely terminate. This "confidence quantification" requires performing a statistical test. If we perform this test too often, we spend a significant amount of time doing computation that could be spent performing I/O, and we further lose statistical power since we are performing more tests; if we do not do this test often enough, we may end up taking many more samples than are necessary to terminate.

3. Challenges with Storage Media. When performing sampling from traditional storage media, the cost to fetch samples is locality-dependent; truly random sampling is extremely expensive due to random I/O, while sampling at the level of blocks is much more efficient, but is less random.

4. Communication between Components. It is crucial for our overall system to not be bottlenecked on any component. In particular, the process of quantifying importance (via the sampling manager) must not block the actual I/O performed; otherwise, the time for execution may end up being greater than the time taken by exact methods. As such, these components must proceed asynchronously, while also minimizing communication across them.

Our Contributions. In this paper, we have developed an end-to-end architecture for histogram matching, dubbed FastMatch, addressing the challenges identified above:

1. Importance Quantification Policies. We develop a sampling engine that employs a simple and theoretically well-motivated criterion for deciding whether processing particular portions of data will allow for faster termination. Since the criterion is simple, it is *easy to update* as we process new data, "understanding" when it has seen enough data for some histogram, or when it needs to take more data to distinguish histograms that are close to each other.

2. Termination Algorithm. We develop a statistics engine that repeatedly performs a lightweight "safe termination" test, based on the idea of performing multiple hypothesis tests for which simultaneous rejection implies correctness of the results. Our statistics engine further quantifies how often to run this test to ensure timely termination without sacrificing too much statistical power.

3. Locality-aware Sampling. To better exploit locality of storage media, FastMatch samples at the level of blocks, proceeding sequentially. To estimate the benefit of blocks, we leverage bitmap indexes in a cache-conscious manner, evaluating multiple blocks at a time in the same order as their layout in storage. Our technique minimizes the time required for the query output to satisfy our probabilistic guarantees.

4. Decoupling Components. Our system decouples the overhead of deciding which samples to take from the actual I/O used to read the samples from storage. In particular, our sampling engine utilizes a just-in-time lookahead technique that marks blocks for reading or skipping while the I/O proceeds unhindered, in parallel.

Overall, we implement FastMatch within the context of a bitmap-based sampling engine, which allows us to quickly determine whether a given memory or disk block could contain samples matching ad-hoc predicates. Such engines were found to effectively support approximate generation of visualizations in recent work [8, 46, 64].

We find that our approximation-based techniques working in tandem with our novel systems components ***lead to speedups ranging from*** $8\times$ ***to over*** $35\times$ over exact methods, and moreover, unlike less-sophisticated variants of FastMatch, whose performance can be highly data-dependent, FastMatch consistently brings latency to near-interactive levels.

Related Work. To the best of our knowledge, there has been no work on sampling to identify histograms that match user specifications. Sampling-based techniques have been applied to generate visualizations that preserve visual properties [8, 46], and for incremental generation of time-series and heat-maps [64]—all focusing on the generation of a single visualization. Similarly, Pangloss [57] employs approximation via the Sample+Seek approach [28] to generate a single visualization early, while minimizing error. One system uses workload-aware indexes called "VisTrees" [29] to facilitate sampling for interactive generation of histograms *without* error guarantees. M4 uses rasterization without sampling to reduce the dimensionality of a time-series visualization and generate it faster [43]. SeeDB [71] recommends visualizations to help distinguish between two subsets of data while employing approximation. However, their techniques are tailored to evaluating differences between pairs of visualizations (that share the same axes, while other pairs do not share the same axes). In our case, we need to compare one visualization versus others, all of which have the same axes and have comparable distances, hence the techniques do not generalize.

Recent work has developed zenvisage [67], a visual exploration tool, including operations that identify visualizations similar to a target. However, to identify matches, zenvisage does not consider sampling, and requires at least one complete pass through the dataset. FastMatch was developed as a back-end with such interfaces in mind to support rapid discovery of relevant visualizations.

Outline. Section 2 articulates the formal problem of identifying top-k closest histograms to a target. Section 3 introduces our HistSim algorithm for solving this problem, while Section 4 describes the system architecture that implements this algorithm. In Section 5 we perform an empirical evaluation on several real-world datasets. After surveying additional related work in Section 6, we describe several generalizations and extensions of our techniques in Appendix A.

2. PROBLEM FORMULATION

In this section, we formalize the problem of identifying histograms whose distributions match a reference.

Symbol(s)	Description
X, Z, V_X, V_Z, T	x-axis attribute, candidate attribute, respective value sets, and relation over these attributes, used in histogram-generating queries (see Definition 1)
$k, \delta, \varepsilon, \sigma$	User-supplied parameters (number of matching histograms to retrieve, error probability upper bound, approximation error upper bound, selectivity threshold (below which candidates may optionally be ignored)
$\mathbf{q}, \mathbf{r}_i, \mathbf{r}_i^*, (\bar{\mathbf{q}}, \bar{\mathbf{r}}_i, \bar{\mathbf{r}}_i^*)$	Visual target, candidate i's estimated (unstarred) and true (starred) histogram counts (normalized variants)
$d(\cdot, \cdot)$	Distance function, used to quantify visual distance (see Definition 2)
$n_i, n_i', \varepsilon_i, \delta_i, \tau_i$ (τ_i^*)	Quantities specific to candidate i during HistSim run: number of samples taken, estimated samples needed (see Section 4), deviation bound (see Definition 4), confidence upper bound on ε_i-deviation or rareness, and distance estimate from $\mathbf{q}$ (true distance from $\mathbf{q}$), respectively
$n_i^\partial, \mathbf{r}_i^\partial, \tau_i^\partial$	Quantities corresponding to samples taken in a specific round of HistSim stage 2: number of samples taken for candidate i in round, per-group counts for candidate i for samples taken in round, corresponding distance estimates using the samples taken in round, respectively
M, A	Set of matching histograms (see Definition 3) and non-pruned histograms, respectively, during a run of HistSim
$N_i, N, m, f(\cdot; N, N_i, m)$	Number of datapoints corresponding to candidate i, total number of datapoints, samples taken during stage 1, hypergeometric pdf

Table 1: Summary of notation.

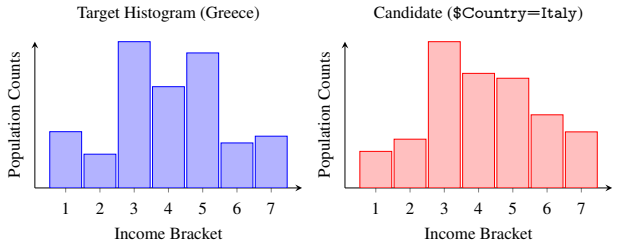


Figure 1: Example visual target and candidate histogram

2.1 Generation of Histograms

We start with a concrete example of the typical database query an analyst might use to generate a histogram. Returning to our example from Section 1, suppose an analyst is interested in studying how population proportions vary across income brackets for various countries around the world. Suppose she wishes to find countries with populations distributed across different income brackets most similarly to a specific country, such as Greece. Consider the following SQL query, where $COUNTRY is a variable:

```
SELECT income_bracket, COUNT(*) FROM census
WHERE country=$COUNTRY
GROUP BY income_bracket
```

This query returns a list of 7 (income bracket, count) pairs to the analyst for a specific country. The analyst may then choose to visualize the results by plotting the counts versus different income brackets in a histogram, i.e., a plot similar to the right side of Figure 1 (for Italy). Currently, the analyst may examine hundreds of similar histograms, one for each country, comparing it to the one for Greece, to manually identify ones that are similar.

In contrast, the goal of FastMatch is to perform this search automatically and efficiently. Conceptually, FastMatch will iterate over all possible values of country, generate the corresponding histograms, and evaluate the similarity of its distribution (based on some notion of similarity described subsequently) to the corresponding visualization for Greece. In actuality, FastMatch will perform this search all at once, quickly pruning countries that are either clearly close or far from the target.

Candidate Visualizations. Formally, we consider visualizations as being generated as a result of **histogram-generating queries**:

DEFINITION 1. *A* histogram-generating query *is a SQL query of the following type:*

SELECT X, COUNT(*) FROM T
WHERE $Z = z_i$ GROUP BY X

The table T and attributes X and Z form the query's template.

For each concrete value z_i of attribute Z specified in the query, the results of the query—i.e., the grouped counts—can be represented in the form of a vector $(r_1, r_2, \ldots, r_n)$, where $n = |V_X|$, the cardinality of the value set of attribute X. This n-tuple can then be used to plot a histogram visualization—in this paper, when we refer to a histogram or a visualization, we will be typically referring to such an n-tuple. For a given *grouping attribute* X and a *candidate attribute* Z, we refer to the set of all visualizations generated by letting Z vary over its value set as the set of ***candidate visualizations***. We refer to each distinct value in the grouping attribute X's value set as a *group*. In our example, X corresponds to `income_bracket` and Z corresponds to `country`.

For ease of exposition, we focus on candidate visualizations generated from queries according to Definition 1, having single categorical attributes for X and Z. Our methods are more general and extend naturally to handle *(i) predicates:* additional predicates on other attributes, *(ii) multiple and complex Xs:* additional grouping (i.e., X) attributes, groups derived from binning real-values (as opposed to categorical X), along with groups derived from binning multiple categorical X attribute values together (e.g., quarters instead of individual months), and *(iii) multiple and complex Zs:* additional candidate (i.e., Z) attributes, as well as candidate attribute values derived from binning real values (as opposed to categorical Z). The flexibility in specifying histogram-generating queries—exponential in the number of attributes—makes it impossible for us to precompute the results of all such queries.

Visualization Terminology. Our methods are agnostic to the particular method used to present visualizations. That is, analysts may choose to present the results generated from queries of the form in Definition 1 via line plots, heat maps, choropleths, and other visualization types, as any of these may be specified by an ordered tuple of real values and are thus permitted under our notion of a "candidate visualization". We focus on bar charts of frequency counts and histograms—these naturally capture aggregations over the categorical or binned quantitative grouping attribute X respectively. Although a bar graph plot of frequency counts over a categorical grouping attribute is not technically a histogram, which implies that the grouping attribute is continuous, we loosely use the term "histogram" to refer to both cases in a unified way.

Visual Target Specification. Given our specification of candidate visualizations, a ***visual target*** is an n-tuple, denoted by $\mathbf{q}$ with entries $Q_1, Q_2, \ldots, Q_n$, that we need to match the candidates with. Returning to our flight delays example, $\mathbf{q}$ would refer to the visualization corresponding to Greece, with Q_1 being the count of individuals in the first income bracket, Q_2 the count of individuals in the second income bracket, and so on.

Samples. To estimate these candidate visualizations, we need to take *samples*. In particular, for a given candidate i for some attribute Z, a sample corresponds to a single tuple t with attribute value $Z = z_i$. The attribute value $X = x_j$ of t increments the jth entry of the estimate $\mathbf{r}_i$ for the candidate histogram.

Candidate Similarity. Given a set of candidate visualizations with estimated vector representations $\{\mathbf{r}_i\}$ such that the ith candidate is generated by selecting on $Z = z_i$, our problem hinges on finding the candidate whose distribution is most "similar" to the visual target $\mathbf{q}$ specified by the analyst. For quantifying visual similarity, we do not care about the absolute counts $r_1, r_2, \ldots, r_{|V_X|}$, and instead prefer to determine whether $\mathbf{r}_i$ and $\mathbf{q}$ are close in a *distributional*

sense. Using hats to denote normalized variants of $\mathbf{r}_i$ and $\mathbf{q}$, write

$$\bar{\mathbf{r}}_i = \frac{\mathbf{r}_i}{\mathbf{1}^T\mathbf{r}_i} \qquad \bar{\mathbf{q}} = \frac{\mathbf{q}}{\mathbf{1}^T\mathbf{q}}$$

With this notational convenience, we make our notion of similarity explicit by defining candidate distance as follows:

DEFINITION 2. *For candidate $\mathbf{r}_i$ and visual predicate $\mathbf{q}$, the **distance** $d(\mathbf{r}_i, \mathbf{q})$ between $\mathbf{r}_i$ and $\mathbf{q}$ is defined as follows:*

$$d(\mathbf{r}_i, \mathbf{q}) = ||\bar{\mathbf{r}}_i - \bar{\mathbf{q}}||_1 = ||\frac{\mathbf{r}_i}{\mathbf{1}^T\mathbf{r}_i} - \frac{\mathbf{q}}{\mathbf{1}^T\mathbf{q}}||_1$$

That is, after normalizing the candidate and target vectors so that their respective components sum to 1 (and therefore correspond to distributions), we take the ℓ_1 distance between the two vectors. When the target $\mathbf{q}$ is understood from context, we denote the distance between candidate $\mathbf{r}_i$ and $\mathbf{q}$ by $\tau_i = d(\mathbf{r}_i, \mathbf{q})$.

The Need for Normalization. A natural question that readers may have is why we chose to normalize each vector prior to taking the distance between them. We do this because the goal of FastMatch is to find visualizations that have similar distributions, as opposed to similar actual values. Returning to our example, if we consider the population distribution of Greece across different income brackets, and compare it to that of other countries, without normalization, we will end up returning other countries with similar population counts in each bin—e.g., other countries with similar overall populations—as opposed to those that have similar shape or distribution. To see an illustration of this, consider Figure 3. The overlaid histogram in goldenrod is identical to the blue one, but we are unable to capture this without normalization.

Choice of Metric Post-Normalization. A similar metric, using ℓ_2 distance between normalized vectors (as opposed to ℓ_1), has been studied in prior work [71, 28] and even validated in a user study in [71]. However, as observed in [12], the ℓ_2 distance between distributions has the drawback that it could be small even for distributions with disjoint support. The ℓ_1 distance metric over discrete probability distributions has a direct correspondence with the traditional statistical distance metric known as *total variation distance* [32] and does not suffer from this drawback.

Additionally, we sometimes observe that ℓ_2 heavily penalizes candidates with a small number of vector entries with large deviations from each other, even when they are arguably closer visually than those candidates closest in ℓ_2. Consider Figure 2, which depicts histograms generated by one of the queries on a FLIGHTS dataset we used in our experiments, corresponding to a histogram of departure time. The target is the Chicago ORD airport, and we are depicting the first non-ORD top-k histogram for both ℓ_1 and ℓ_2 (i.e., the 2nd ranked histogram for both metrics), among all airports. As one can see in the figure, the middle histogram is arguably "visually closer" to the ORD histogram on the left, but is not considered so by ℓ_2 due to the mismatch at about the 6th hour.

KL-divergence is another possibility as a distance metric, but it has the drawback that it will be infinite for any candidate that places 0 mass in a place where the target places nonzero mass, making it difficult to compare these (note that this follows directly from the definition: $KL(p||q) = -\sum_i p_i \log \frac{q_i}{p_i}$).

2.2 Guarantees and Problem Statement

Since FastMatch takes samples to estimate the candidate histogram visualizations, and therefore may return incorrect results, we need to enforce probabilistic guarantees on the correctness of the returned results.

First, we introduce some notation: we use $\mathbf{r}_i$ to denote the *estimate* of the candidate visualization, while $\mathbf{r}_i^*$ (with normalized version $\bar{\mathbf{r}}_i^*$) is the *true* candidate visualization on the entire dataset. Our formulation also relies on constants ε, δ, and σ, which we assume either built into the system or provided by the analyst. We further use N and N_i to denote the total number of datapoints and number of datapoints corresponding to candidate i, respectively.

GUARANTEE 1. *(SEPARATION) Any approximate histogram $\mathbf{r}_i$ with selectivity $\frac{N_i}{N} \geq \sigma$ that is in the true top-k closest (w.r.t. Definition 2) but* not *part of the output will be less than ε closer to the target than the furthest histogram that* is *part of the output. That is, if the algorithm outputs histograms $\mathbf{r}_{j_1}, \mathbf{r}_{j_2}, \ldots, \mathbf{r}_{j_k}$, then, for all i, $\max_{1 \leq l \leq k} \left\{d(\mathbf{r}_{j_l}^*, \mathbf{q})\right\} - d(\mathbf{r}_i^*, \mathbf{q}) < \varepsilon$, or $\frac{N_i}{N} < \sigma$.*

Note that we use "selectivity" as a number and not as a property, matching typical usage in database systems literature [66, 45]. As such, candidates with lower selectivity appear less frequently in the data than candidates with higher selectivity.

GUARANTEE 2. *(RECONSTRUCTION) Each approximate histogram $\mathbf{r}_i$ output as one of the top-k satisfies $d(\mathbf{r}_i, \mathbf{r}_i^*) < \varepsilon$.*

The first guarantee says that any ordering mistakes are relatively innocuous: for any two histograms $\mathbf{r}_i$ and $\mathbf{r}_j$, if the algorithm outputs $\mathbf{r}_j$ but not $\mathbf{r}_i$, when it should have been the other way around, then either $\left|d(\mathbf{r}_i^*, \mathbf{q}) - d(\mathbf{r}_j^*, \mathbf{q})\right| < \varepsilon$, or $\frac{N_i}{N} < \sigma$. The intuition behind the minimum selectivity parameter, σ, is that certain candidates may not appear frequently enough within the data to get a reliable reconstruction of the true underlying distribution responsible for generating the original data, and thus may not be suitable for downstream decision-making. For example, in our income example, a country with a population of 100 may have a histogram similar to the visual target but this would not be statistically significant. Overall, our guarantee states that we still return a visualization that is quite close to $\mathbf{q}$, and we can be confident that anything dramatically closer has relatively few total datapoints available within the data (i.e., N_i is small).

The second guarantee says that the histograms output are not too dissimilar from the corresponding true distributions that would result from a complete scan of the data. As a result, they form an adequate and accurate proxy from which insights may be derived. With these definitions in place, we now formally state our core problem:

PROBLEM 1. *(TOP-K-SIMILAR). Given a visual target $\mathbf{q}$, a histogram-generating query template, k, ε, δ, and σ, display k candidate attribute values $\{z_i\} \subseteq V_Z$ (and accompanying visualizations $\{\mathbf{r}_i\}$) as quickly as possible, such that the output satisfies Guarantees 1 and 2 with probability greater than $1 - \delta$.*

3. THE HISTSIM ALGORITHM

In this section, we discuss how to conceptually solve Problem 1. We outline an algorithm, named HistSim, which allows us to determine confidence levels for whether our separation and reconstruction guarantees hold. We rigorously prove in this section that when our algorithm terminates, it gives correct results with probability greater than $1 - \delta$ *regardless* of the data given as input. Many systems-level details and other heuristics used to make HistSim perform particularly well in practice will be presented in Section 4. Table 1 provides a description of the notation used.

3.1 Algorithm Outline

HistSim operates by sampling tuples. Each of these tuples contributes to one or more candidate histograms, using which HistSim constructs histograms $\{\bar{\mathbf{r}}_i\}$. After taking enough samples corresponding to each candidate, it will eventually be likely that $d(\mathbf{r}_i, \mathbf{r}_i^*)$ is "small", and that $|d(\mathbf{r}_i, \mathbf{q}) - d(\mathbf{r}_i^*, \mathbf{q})|$ is likewise "small", for each i. More precisely, the set of candidates will likely be in a state such that Guarantees 1 and 2 are both satisfied simultaneously.

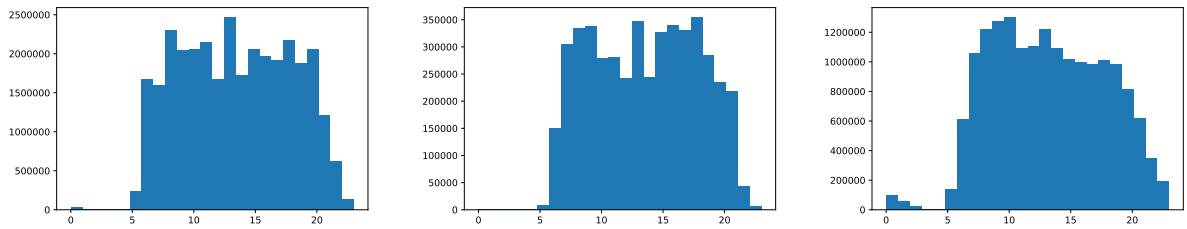


Figure 2: The target (departure hour histogram for ORD), second closest in normalized ℓ_1 (DAL) , second closest in normalized ℓ_2 (PHX)

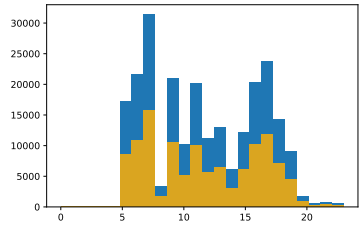


Figure 3: The goldenrod histogram is identical to the blue one post-normalization, but appears very far visually pre-normalization.

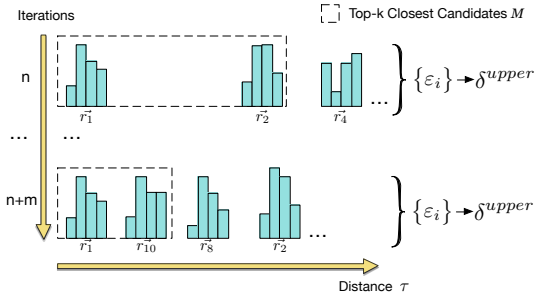


Figure 4: Illustration of HistSim.

Stages Overview. HistSim separates its sampling into three stages, each with an error probability of at most $\frac{\delta}{3}$, giving an overall error probability of at most δ:

- Stage 1 [Prune Rare Candidates]: Sample datapoints uniformly at random without replacement, so that each candidate is sampled a number of times roughly proportional to the number of datapoints corresponding to that candidate. Identify rare candidates that likely satisfy $\frac{N_i}{N} < \sigma$, and prune these ones.
- Stage 2 [Identify Top-k]: Take samples from the remaining candidates until the top-k have been identified reliably.
- Stage 3 [Reconstruct Top-k]: Sample from the estimated top-k until they have been reconstructed reliably.

This separation is important for performance: the pruning step (stage 1) often *dramatically reduces* the number of candidates that need to be considered in stages 2 and 3.

The first two stages of HistSim factor into phases that are pure I/O and phases that involve one or more statistical tests. The I/O phases sample tuples (lines 6 and 19 in Algorithm 1)—we will describe how in Section 4; our algorithm's correctness is independent of how this happens, provided that the samples are uniform.

Stage 1: Pruning Rare Candidates (Section 3.3). During stage 1, the I/O phase (line 6) takes m samples, for some m fixed ahead of time. This is followed by updating, for each candidate i, the number of samples n_i observed so far (line 7), and using the P-values $\{\delta_i\}$ of a test for underrepresentation to determine whether each candidate i is rare, i.e., has $\frac{N_i}{N} < \sigma$ (lines 7–9).

Stage 2: Identifying Top-k (Section 3.4). For stage 2, we focus on a smaller set of candidates; namely, those that we did not find to be rare (denoted by A). Stage 2 is divided into *rounds*. Each round

Algorithm 1: The HistSim algorithm

Input : Columns Z, X, visual target $\mathbf{q}$, parameters $k, \varepsilon, \delta, \sigma$
Output : Estimates M of the top-k closest candidates to $\mathbf{q}$, histograms $\{\mathbf{r}_i\}$

1
2 **Initialization.**
3 $n_i, n_i^\partial \leftarrow 0, \mathbf{r}_i, \mathbf{r}_i^\partial \leftarrow \mathbf{0}$ for $1 \le i \le |V_Z|$;
4
5 **stage 1:** $\delta^{upper} \leftarrow \frac{\delta}{3}$;
6 Repeat m times: uniformly randomly sample some tuple without replacement;
7 Update $\{n_i\}, \{\mathbf{r}_i\}, \{\tau_i\}$ based on the new samples;
8 $\Delta \leftarrow \{\delta_i\}$ where $\delta_i = \sum_{j=0}^{n_i} f(j; N, \lceil \sigma N \rceil, m)$ for $1 \le i \le |V_Z|$;
9 Perform a Holm-Bonferroni statistical test with P-values in Δ; that is:
10 $A \leftarrow \left\{ i : \delta_i \le \frac{\delta}{|V_Z|-i+1} \text{ and for all } j < i, \delta_j \le \frac{\delta}{|V_Z|-j+1} \right\}$;
11
12 **stage 2:** $\delta^{upper} \leftarrow \frac{\delta}{3}$;
13 **do**
14 $\quad \delta^{upper} \leftarrow \frac{1}{2}\delta^{upper}$;
15 $\quad n_i \mathrel{+}= n_i^\partial, \quad \mathbf{r}_i \mathrel{+}= \mathbf{r}_i^\partial, \quad \tau_i \leftarrow d(\mathbf{r}_i, \mathbf{q}) \quad$ for $i \in A$;
16 $\quad n_i^\partial \leftarrow 0, \quad \mathbf{r}_i^\partial \leftarrow \mathbf{0} \quad$ for $i \in A$;
17 $\quad M \leftarrow \{i \in A : \tau_i$ among k smallest$\}$;
18 $\quad s \leftarrow \frac{1}{2}(\max_{i \in M} \tau_i + \min_{j \in A \setminus M} \tau_j)$;
19 $\quad$ Repeat: take uniform random samples from any $i \in A$;
20 $\quad$ Update $\{n_i^\partial\}, \{\mathbf{r}_i^\partial\}$, and $\{\tau_i^\partial\}$ based on the new samples;
21 $\quad \varepsilon_i \leftarrow s + \frac{\varepsilon}{2} - \tau_i^\partial$ for $i \in M$;
22 $\quad \varepsilon_j \leftarrow \tau_j^\partial - (s - \frac{\varepsilon}{2})$ if $s - \frac{\varepsilon}{2} \ge 0$ else ∞ for $j \in A \setminus M$;
23 $\quad \Delta \leftarrow \{\delta_i\}$ where $\delta_i \ge \mathbb{P}\left(d(\mathbf{r}_i^\partial, \mathbf{r}_i^*) > \varepsilon_i\right)$ for $i \in A$;
24 **while** $\max(\Delta) > \delta^{upper}$;
25
26 **stage 3:** Sample until $n_i \ge \frac{2}{\varepsilon^2}\left(|V_X| \log 2 + \log \frac{3k}{\delta}\right)$, for all $i \in M$;
27 Update $\{\mathbf{r}_i\}$ based on the new samples;
28 **return** M, *and* $\{\mathbf{r}_i : i \in M\}$;

attempts to use existing samples to estimate which candidates are top-k and which are non top-k, and then draws new samples, testing how unlikely it is to observe the new samples in the event that its guess of the top-k is wrong. If this event is unlikely enough, then it has recovered the correct top-k, with high probability.

At the start of each round, HistSim accumulates any samples taken during the previous round (lines 15–16). It then determines the current top-k candidates and a separation point s between top-k and non top-k (lines 17–18), as this separation point determines a set of hypotheses to test. Then, it begins an I/O phase and takes samples (line 19). The samples taken each round are used to generate the number of samples taken per candidate, $\{n_i^\partial\}$, the estimates $\{\mathbf{r}_i^\partial\}$, and the distance estimates $\{\tau_i^\partial\}$ (line 20). These statistics are computed from fresh samples each round (i.e., they do not reuse samples across rounds) so that they may be used in a statistical test (lines 20–23), discussed in Section 3.4. After computing the P-values for each null hypothesis to test (line 23), HistSim determines whether it can reject all the hypotheses with type 1 error (i.e., probability of mistakenly rejecting a true null hypothesis) bounded by δ^{upper} and break from the loop (line 24). If not, it repeats with new samples and a smaller δ^{upper} (where the $\{\delta^{upper}\}$ are chosen so that the probability of error across *all* rounds is at most $\frac{\delta}{3}$).

Stage 3: Reconstructing Top-k (Section 3.5). Finally, stage 3

ensures that the identified top-k, M, all satisfy $d(\mathbf{r}_i, \mathbf{r}_i^*) \leq \varepsilon$ for $i \in M$ (so that Guarantee 2 holds), with high probability.

Figure 4 illustrates HistSim stage 2 running on a toy example in which we compute the top-2 closest histograms to a target. At round n, it estimates $\mathbf{r}_1$ and $\mathbf{r}_2$ as the top-2 closest, which it refines by the time it reaches round $n + m$. As the rounds increase, HistSim takes more samples to get better estimates of the distances $\{\tau_i\}$ and thereby improve the chances of termination when it performs its multiple hypothesis test in stage 2.

Choosing where to sample and how many samples to take. The estimates M and $\{\tau_i\}$ allow us to determine which candidates are "important" to sample from in order to allow termination with fewer samples; we return to this in Section 4. Our HistSim algorithm is agnostic to the sampling approach.

Outline. We first discuss the Holm-Bonferroni method for testing multiple statistical hypotheses simultaneously in Section 3.2, since stage 1 of HistSim uses it as a subroutine, and since the simultaneous test in stage 2 is based on similar ideas. In Section 3.3, we discuss stage 1 of HistSim, and prove that upon termination, all candidates i flagged for pruning satisfy $\frac{N_i}{N} < \sigma$ with probability greater than $\frac{\delta}{3}$. Next, in Section 3.4, we discuss stage 2 of HistSim, and prove that upon termination, we have the guarantee that any non-pruned candidate mistakenly classified as top-k is no more than ε further from the target than the furthest true non-pruned top-k candidate (with high probability). The proof of correctness for stage 2 is the most involved and is divided as follows:

- In Section 3.4.1, we give lemmas that allow us to relate the reconstruction of the candidate histograms from estimates $\{\mathbf{r}_i^\partial\}$ to the separation guarantee via multiple hypothesis testing;
- In Section 3.4.2, we describe a method to select appropriate hypotheses to use for testing in the lemmas of Section 3.4.1;
- In Section 3.4.3, we prove a theorem that enables us to use the samples per candidate histogram to determine the P-values associated with the hypotheses.

In Section 3.5, we discuss stage 3 and conclude with an overall proof of correctness.

3.2 Controlling Family-wise Error

In the first two stages of HistSim, the algorithm needs to perform multiple statistical tests simultaneously [17]. In stage 1, HistSim tests null hypotheses of the form "candidate i is high-selectivity" versus alternatives like "candidate i is not high-selectivity". In this case, "rejecting the null hypothesis at level δ^{upper}" roughly means that the probability that candidate i is high-selectivity is at most δ^{upper}. Likewise, during stage 2, HistSim tests null hypotheses of the form "candidate i's true distance from $\mathbf{q}$, τ_i^*, lies above (or below) some fixed value s." If the algorithm correctly rejects every null hypothesis while controlling the family-wise error [50] at level δ^{upper}, then it has correctly determined which side of s every τ_i^* lies, a fact that we use to get the separation guarantee.

Since stages 1 and 2 test multiple hypotheses at the same time, HistSim needs to control the family-wise type 1 error (false positive) rate of these tests simultaneously. That is, if the family-wise type 1 error is controlled at level δ^{upper}, then the probability that one or more rejecting tests in the family should not have rejected is less than δ^{upper} — during stage 1, this intuitively means that the probability one or more high-selectivity candidates were deemed to be low-selectivity is at most δ^{upper}, and during stage 2, this roughly means that the probability of selecting some candidate as top-k when it is non top-k (or vice-versa) is at most δ^{upper}.

The reader may be familiar with the Bonferroni correction, which enforces a family-wise error rate of δ^{upper} by requiring a significance level $\frac{\delta^{upper}}{|V_Z|}$ for each test in a family with $|V_Z|$ tests in total. We instead use the Holm-Bonferroni method [36], which is uniformly more powerful than the Bonferroni correction, meaning that it needs fewer samples to make the same guarantee. Like its simpler counterpart, it is correct regardless of whether the family of tests has any underlying dependency structure. In brief, a level δ^{upper} test over a family of size $|V_Z|$ works by first sorting the P-values $\{\delta_i\}$ of the individual tests in increasing order, and then finding the minimal index j (starting from 1) where $\delta_j > \frac{\delta^{upper}}{|V_Z|-j+1}$ (if this does not exist, then set $j = |V_Z|$). The tests with smaller indices reject their respective null hypotheses at level δ^{upper}, and the remaining ones do not reject.

3.3 Stage 1: Pruning Rare Candidates

One way to remove *rare* (i.e. low-selectivity) candidates from processing is to use an index to look up how many tuples correspond to each candidate. While this will work well for some queries, it unfortunately does not work in general, as candidates generated from queries of the form in Definition 1 could have arbitrary predicates attached, which cannot all be indexed ahead-of-time. Thus, we turn to sampling.

To prune rare candidates, we need some way to determine whether each candidate i satisfies $\frac{N_i}{N} < \sigma$ with high probability. To do so, we make the simple observation that, after drawing m tuples without replacement uniformly at random, the number of tuples corresponding to candidate i follows a hypergeometric distribution [42]. The number of samples to take, m, is a parameter; we observe in our experiments that $m = 5 \cdot 10^5$ is an appropriate choice.[1] That is, if candidate i has N_i total corresponding tuples in a dataset of size N, then the number of tuples n_i for candidate i in a uniform sample without replacement of size m is distributed according to $n_i \sim \text{HypGeo}(N, N_i, m)$. As such, we can make use of a well-known test for underrepresentation [50] to accurately detect when candidate i has $\frac{N_i}{N} < \sigma$. The null hypothesis is that candidate i is not underrepresented (i.e., has $N_i \geq \sigma N$), and letting $f(\,\cdot\,; N, \lceil \sigma N \rceil, m)$ denote the hypergeometric pdf in this case, the P-value for the test is given by

$$\sum_{j=0}^{n_i} f(j; N, \lceil \sigma N \rceil, m)$$

where n_i is the number of observed tuples for candidate i in the sample of size m. Roughly speaking, the P-value measures how surprised we are to observe n_i or fewer tuples for candidate i when $\frac{N_i}{N} \geq \sigma$ — the lower the P-value, the more surprised we are.

If we reject the null hypothesis for some candidate i when the P-value is at most δ_i, we are claiming that candidate i satisfies $\frac{N_i}{N} < \sigma$, and the probability that we are wrong is then at most δ_i. Of course, we need to test *every* candidate for rareness, not just a given candidate, which is why HistSim stage 1 uses a Holm-Bonferroni procedure to control the *family-wise* error at any given threshold. We note in passing that the joint probability of observing n_i samples for candidate i across *all* candidates is a multivariate hypergeometric distribution for which we could perform a similar test without a Holm-Bonferroni procedure, but the CDF of a multivariate hypergeometric is extremely expensive to compute, and we can afford to sacrifice some statistical power for the sake of computational efficiency since we only need to ensure that the candidates pruned are actually rare, without necessarily finding all the rare candidates — that is, we need high precision, not high recall.

We now prove a lemma regarding correctness of stage 1.

LEMMA 1 (STAGE 1 CORRECTNESS). *After* HistSim *stage 1 completes, every candidate i removed from A satisfies $\frac{N_i}{N} < \sigma$, with probability greater than* $1 - \frac{\delta}{3}$

[1] Our results are not sensitive to the choice of m, provided m is not too small (so that the algorithm fails to prune anything) or too big (i.e., a nontrivial fraction of the data).

PROOF. This follows immediately from the above discussion, in conjunction with the fact that the P-values generated from each test for underrepresentation are fed into a Holm-Bonferroni procedure that operates at level $\frac{\delta}{3}$, so that the probability of pruning one or more non-rare candidates is bounded above by $\frac{\delta}{3}$. □

3.4 Stage 2: Identifying Top-K Candidates

HistSim stage 2 attempts to find the top-k closest to the target out of those remaining after stage 1. To facilitate discussion, we first introduce some definitions.

DEFINITION 3. *(*MATCHING CANDIDATES*) A candidate is called* matching *if its distance estimate $\tau_i = d(\mathbf{r}_i, \mathbf{q})$ is among the k smallest out of all candidates remaining after stage 1.*

We denote the (dynamically changing) set of candidates that are matching during a run of HistSim as M; we likewise denote the true set of matching candidates out of the remaining, non-pruned candidates in A as M^*. Next, we introduce the notion of *ε_i-deviation*.

DEFINITION 4. *(ε_i-*DEVIATION*) The empirical vector of counts $\mathbf{r}_i$ for some candidate i has ε_i-deviation if the corresponding normalized vector $\bar{\mathbf{r}}_i$ is within ε_i of the exact distribution $\bar{\mathbf{r}}_i^*$. That is, $d(\mathbf{r}_i, \mathbf{r}_i^*) = ||\bar{\mathbf{r}}_i - \bar{\mathbf{r}}_i^*||_1 < \varepsilon_i$*

Note that Definition 4 overloads the symbol ε to be candidate-specific by appending a subscript. In Section 3.4.3, we provide a way to quantify ε_i given samples.

If HistSim reaches a state where, for each matching candidate $i \in M$, candidate i has ε_i-deviation, and $\varepsilon_i < \varepsilon$ for all $i \in M$, then it is easy to see that the Guarantee 2 holds for the matching candidates. That is, in such a state, if HistSim output the histograms corresponding to the matching candidates, they would look similar to the true histograms. In the following sections, we show that ε_i-deviation can also be used to achieve Guarantee 1.

Notation for Round-Specific Quantities. In the following subsections, we use the superscript "Δ" to indicate quantities corresponding to samples taken during a particular round of HistSim stage 2, such as $\{\mathbf{r}_i^\partial\}$ and $\{\tau_i^\partial\}$. In particular, these quantities are *completely independent* of samples taken during previous rounds.

3.4.1 Deviation-Bounds Imply Separation

In order to reason about the separation guarantee, we prove a series of lemmas following the structure of reasoning given below:

- We show that when a carefully chosen set of null hypotheses are all false, M contains valid top-k closest candidates.
- Next, we show how to use ε_i-deviation to upper bound the probability of rejecting a *single* true null hypothesis.
- Finally, we show how to reject *all* null hypotheses while controlling the probability of rejecting *any* true ones.

LEMMA 2 (FALSE NULLS IMPLY SEPARATION). *Consider the set of null hypotheses $\{H_0^{(i)}\}$ defined as follows, where $s \in \mathbb{R}^+$:*

$$H_0^{(i)} = \begin{cases} \tau_i^* \geq s + \frac{\varepsilon}{2}, \text{ for } i \in M \\ \tau_i^* \leq s - \frac{\varepsilon}{2}, \text{ for } i \in A \setminus M \end{cases}$$

When $H_0^{(i)}$ is false for every $i \in A$, then M is a set of top-k candidates that is correct with respect to Guarantee 1.

PROOF. When all the null hypotheses are false, then $\tau_i^* < s + \frac{\varepsilon}{2}$ for all $i \in M$, and $\tau_j^* > s - \frac{\varepsilon}{2}$ for all $j \in A \setminus M$. This means that

$$\max_{i \in M} \tau_i^* - \min_{j \in A \setminus M} \tau_j^* < \varepsilon$$

and thus M is correct with respect to the separation guarantee. □

Intuitively, Lemma 2 states that when there is some reference point s such that all of the candidates in M have their τ_i^* smaller than $s - \frac{\varepsilon}{2}$, and the rest have their τ_i^* greater than $s + \frac{\varepsilon}{2}$, then we have our separation guarantee.

Next, we show how to compute P-values for a single null hypothesis of the type given in Lemma 2. Below, we use "$\mathbb{P}_H$" to denote the probability of some event when hypothesis H is true.

LEMMA 3 (DISTANCE DEVIATION TESTING). *Let $x \in \mathbb{R}^+$. To test the null hypothesis $H_0^{(i)} : \tau_i^* \geq x$ versus the alternative $H_A^{(i)} : \tau_i^* < x$, we have that, for any $\varepsilon_i > 0$,*

$$\mathbb{P}_{H_0^{(i)}}\left[x - \tau_i^\partial > \varepsilon_i\right] \leq \mathbb{P}\left(d(\mathbf{r}_i^\partial, \mathbf{r}_i^*) > \varepsilon_i\right)$$

Likewise, for testing $H_0^{(i)} : \tau_i^* \leq x$ versus the alternative $H_A^{(i)} : \tau_i^* > x$, we have

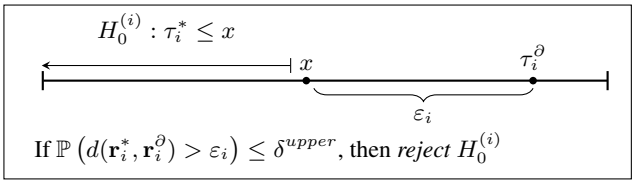
$$\mathbb{P}_{H_0^{(i)}}\left[\tau_i^\partial - x > \varepsilon_i\right] \leq \mathbb{P}\left(d(\mathbf{r}_i^\partial, \mathbf{r}_i^*) > \varepsilon_i\right)$$

PROOF. We prove the first case; the second is symmetric. Suppose candidate i satisfies $\tau_i^* \geq x$ for some $x \in \mathbb{R}^+$. Then, if we take n_i^∂ samples from which we construct the random quantities $\mathbf{r}_i^\partial$ and τ_i^∂, we have that

$$\begin{aligned} \mathbb{P}_{H_0^{(i)}}\left[x - \tau_i^\partial > \varepsilon_i\right] &\leq \mathbb{P}\left(\tau_i^* - \tau_i^\partial > \varepsilon_i\right) \\ &= \mathbb{P}\left(||\bar{\mathbf{r}}_i^* - \bar{\mathbf{q}}|| - ||\bar{\mathbf{q}} - \bar{\mathbf{r}}_i^\partial|| > \varepsilon_i\right) \\ &\leq \mathbb{P}\left(||\bar{\mathbf{r}}_i^* - \bar{\mathbf{r}}_i^\partial|| > \varepsilon_i\right) \\ &= \mathbb{P}\left(d(\mathbf{r}_i^*, \mathbf{r}_i^\partial) > \varepsilon_i\right) \end{aligned}$$

Each step follows from the fact that increasing the quantity to the left of the ">" sign within the probability expression can only increase the probability of the event inside. The first step follows from the assumption that $\tau_i^* \geq x$, and the third step follows from the triangle inequality. □

We use Lemma 3 in conjunction with Lemma 2 by using $s \pm \frac{\varepsilon}{2}$ for the reference x of Lemma 3, for a particular choice of s (discussed in Section 3.4.2). For example, Lemma 3 shows that when we are testing the null hypothesis for $i \in M$ that $\tau_i^* \geq s + \frac{\varepsilon}{2}$ and we observe τ_i^∂ such that $0 < \varepsilon_i = s + \frac{\varepsilon}{2} - \tau_i^\partial$, we can use (any upper bound of) $\mathbb{P}\left(d(\mathbf{r}_i^*, \mathbf{r}_i^\partial) > \varepsilon_i\right)$ as a P-value for this test. That is, consider a tester with the following behavior, illustrated pictorially:



In the above picture, the tester assumes that τ_i^* is smaller than x, but it observes a value τ_i^∂ that exceeds x by ε_i. When the true value $\tau_i^* \leq x$ for any reference x, then the observed statistic τ_i^∂ will only be ε_i or larger than x (and vice-versa) when the reconstruction $\mathbf{r}_i^\partial$ is also bad, in the sense that $\mathbb{P}\left(d(\mathbf{r}_i^*, \mathbf{r}_i^\partial) > \varepsilon_i\right)$ is very small. If the above tester rejects $H_0^{(i)}$ when $\mathbb{P}\left(d(\mathbf{r}_i^*, \mathbf{r}_i^\partial) > \varepsilon_i\right) \leq \delta^{upper}$, then Lemma 3 says that it is guaranteed to reject a true null hypothesis with probability at most δ^{upper}. We discuss how to compute an upper bound on $\mathbb{P}\left(d(\mathbf{r}_i^*, \mathbf{r}_i^\partial) > \varepsilon_i\right)$ in Section 3.4.3.

Finally, notice that Lemma 3 provides a test which controls the type 1 error of an individual $H_0^{(i)}$, but we only know that the separation guarantee holds for $i \in M$ when *all* the hypotheses $\{H_0^{(i)}\}$

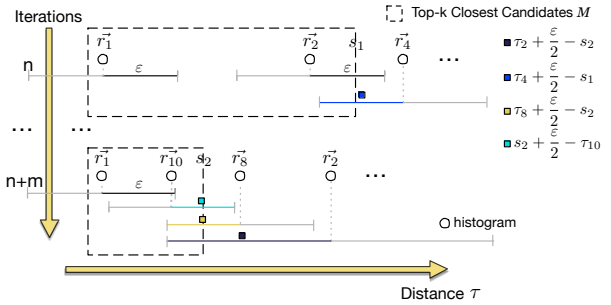


Figure 5: Illustration of HistSim choosing the split point s when testing whether the separation and reconstruction guarantees hold.

are false. Thus, the algorithm requires a way to control the type 1 error of a procedure that decides whether to reject every $H_0^{(i)}$ simultaneously. In the next lemma, we give such a tester which controls the error for any upper bound δ^{upper}.

LEMMA 4 (SIMULTANEOUS REJECTION). *Consider any set of null hypotheses $\{H_0^{(i)}\}$, and consider a set of P-values $\{\delta_i\}$ associated with these hypotheses. The tester given by*

$$\textit{Decision} = \begin{cases} \textit{reject every } H_0^{(i)}, & \textit{when } \max_i \delta_i \leq \delta^{upper} \\ \textit{reject no } H_0^{(i)}, & \textit{otherwise} \end{cases}$$

rejects ≥ 1 true null hypotheses with probability $\leq \delta^{upper}$.

PROOF. Consider the set of true null hypotheses and call it $\{H_0^{(t)}\}$ — suppose there are $T \geq 1$ in total (if $T = 0$, we have nothing to prove), and index them using t from 1 to T. Then

$$\begin{aligned} \mathbb{P}\left(\exists t : \text{reject } H_0^{(t)}\right) &= \mathbb{P}\left(\forall t : \text{reject } H_0^{(t)}\right) \\ &= \prod_{t=1}^{T} \mathbb{P}\left(\text{reject } H_0^{(t)} \mid \text{reject } H_0^{(1,\ldots,t-1)}\right) \\ &= \delta_1 \prod_{t=2}^{T} \mathbb{P}\left(\text{reject } H_0^{(t)} \mid \text{reject } H_0^{(1,\ldots,t-1)}\right) \\ &\leq \delta_1 \cdot 1 \\ &\leq \delta^{upper} \end{aligned}$$

The first step follows since null hypotheses are only rejected when they are all rejected. The second to last step follows since probabilities are at most 1, and the last step follows since the tester only rejects when all the P-values are at most δ^{upper}, including δ_1. □

Discussion of Lemma 4. At first glance, the multiple hypothesis tester given in Lemma 4, which compares all P-values to the same δ^{upper}, seems to be even more powerful than a Holm-Bonferroni tester, which compares P-values to various fractions of δ^{upper}. In fact, although based on similar ideas, they are not comparable: a Holm-Bonferroni tester may allow for rejection of a subset of the null hypotheses, wheres the tester of Lemma 4 is "all or nothing". In fact, the tester of Lemma 4 is essentially the union-intersection method formulated in terms of P-values; see [17] for details.

3.4.2 Selecting Each Round's Tests

Each round of HistSim stage 2 constructs a family of tests to perform whose family-wise error probability is at most δ^{upper}. At round t (starting from $t = 1$), δ^{upper} is chosen to be $\frac{\delta/3}{2^t}$, so that the error probability across *all* rounds is at most $\sum_{t\geq 1} \frac{\delta/3}{2^t} = \frac{\delta}{3}$ via a union bound (see Lemma 5 for details).

There is still one degree of freedom: namely, how to choose the split point s used for the null hypotheses in Lemma 2. In line 18, it is chosen to be $s \leftarrow \frac{1}{2}(\max_{i\in M} \tau_i + \min_{j\in A\setminus M} \tau_j)$. The intuition for this choice is as follows. Although the quantities $\mathbf{r}_i^{\partial}$ and τ_i^{∂} are generated from fresh samples in each round of HistSim stage 2, the quantities $\mathbf{r}_i$ and τ_i are generated from samples taken across *all* rounds of HistSim stage 2. As such, as rounds progress (i.e., if the testing procedure fails to simultaneously reject multiple times), the estimates $\mathbf{r}_i$ and τ_i become closer to $\mathbf{r}_i^*$ and τ_i^*, the set M becomes more likely to coincide with M^*, and the null hypotheses $\{H_0^{(i)}\}$ chosen become less likely to be true *provided* an s chosen somewhere in $[\max_{i\in M} \tau_i, \min_{j\in A\setminus M} \tau_j]$, since values in this interval are likely to correctly separate M^* and $A \setminus M^*$ as more and more samples are taken. In the interest of simplicity, we simply choose the midpoint halfway between the furthest candidate in M and the closest candidate in $A \setminus M$. For example, at iteration n in Figure 5, s lies halfway between candidates $\mathbf{r}_2$ and $\mathbf{r}_4$. In practice, we observe that $\max_{i\in M} \tau_i$ and $\min_{j\in A\setminus M} \tau_j$ are typically very close to each other, so that the algorithm is not very sensitive to the choice of s, so long as it falls between M and $A \setminus M$.

Figure 5 illustrates this choice of s and the $\{H_0^{(i)}\}$ on our toy example. As in Figure 4, the boundary of M is represented by the dashed box. The split point s is located at the rightmost boundary of the dashed box.The $\{\varepsilon_j\}$ (i.e., the amounts by which the $\{\tau_j^{\partial}\}$ deviate from $s \pm \frac{\varepsilon}{2}$) determine the P-values associated with the $\{H_0^{(i)}\}$ which ultimately determine whether HistSim stage 2 can terminate, as we discuss more in the next section.

3.4.3 Deviation-Bounds Given Samples

The previous section provides us a way to check whether the rankings induced by the empirical distances $\{\tau_i\}$ are correct with high probability. This was facilitated via a test which measures our "surprise" for measuring $\{\tau_i^{\partial}\}$ if the current estimate M is not correct with respect to Guarantee 1, which in turn used a test for how likely some candidate's $d(\mathbf{r}_i^*, \mathbf{r}_i^{\partial})$ is greater than some threshold ε_i after taking n_i samples. We now provide a theorem that allows us to infer, given the samples taken for a given candidate, how to relate ε_i with the probability δ_i with which the candidate can fail to respect its deviation-bound ε_i. The bound seems to be known to the theoretical computer science community as a "folklore fact" [27]; we give a prooffor the sake of completeness. Our proof relies on repeated application of the method of bounded differences [56] in order to exploit some special structure in the ℓ_1 distance metric. The bound developed is *information-theoretically optimal*; that is, it takes asymptotically the fewest samples required to guarantee that an empirical distribution estimated from the samples will be no further than ε_i from the true distribution.

THEOREM 1. *Suppose we have taken n_i samples with replacement for some candidate i's histogram, resulting in the empirical estimate $\mathbf{r}_i$. Then $\mathbf{r}_i$ has ε_i-deviation with probability greater than $1 - \delta_i$ for $\varepsilon_i = \sqrt{\frac{2}{n_i}\left(|V_X| \log 2 + \log \frac{1}{\delta_i}\right)}$. That is, with probability $> 1 - \delta_i$, we have: $||\bar{\mathbf{r}}_i - \bar{\mathbf{r}}_i^*||_1 < \varepsilon_i$.*

In fact, this theorem also holds if we sample without replacement; we return to this point in Section 4.

PROOF. For $j \in [|V_X|]$, we use r_j to denote the number of occurrences of attribute value j among the n_i samples, and the normalized count $\bar{r}_j$ is our estimate of $\bar{r}_j^*$, the true proportion of tuples having value j for attribute X. Note that we have omitted the candidate subscript i for clarity.

We need to introduce some machinery. Consider functions of the form $f : [|V_X|] \to \{+1, -1\}$. Let $\{f_m\}$ be the set of all such functions, where $m \in [2^{|V_X|}]$, since there are $2^{|V_X|}$ such functions.

For any $m \in [2^{|V_X|}]$, consider the random variable

$$Y_m = \sum_{j=1}^{|V_X|} f_m(j)(\bar{r}_j - \bar{r}_j^*)$$

By linearity of expectation, it's clear that $\mathbb{E}[Y_m] = 0$, since $f_m(j)$ is constant and $\mathbb{E}[\bar{r}_j] = \bar{r}_j^*$ for each j. Since each $\bar{r}_j$ is a function of the samples taken $\{s_k : 1 \leq k \leq n_i\}$, each Y_m is likewise uniquely determined from samples, so we can write $Y_m = g_m(s_1, \ldots, s_{n_i})$, where each sample s_k is a random variable distributed according to $s_k \sim \bar{\mathbf{r}}^*$. Note that the function g_m satisfies the Lipschitz property

$$|g_m(s_1, \ldots, s_k, \ldots, s_{n_i}) - g_m(s_1, \ldots, s_k', \ldots, s_{n_i})| \leq \frac{2}{n_i}$$

for any $j \in ||V_X||$ and $s_1, \ldots, s_{n_i}$. For example, this will occur with equality if $f_m(s_k) = -f_m(s_k')$; that is, if f_m assigns opposite signs to s_k and s_k', then changing this single sample moves $1/n_i$ of the empirical mass in such a way that it does not get canceled out. We may therefore apply the method of bounded differences [56] to yield the following McDiarmid inequality—a generalization of the standard Hoeffding's inequality:

$$\mathbb{P}(Y_m \geq \mathbb{E}[Y_m] + \varepsilon_i) \leq \exp\left(-\varepsilon_i^2 n_i/2\right)$$

Recalling that $\mathbb{E}[Y_m] = 0$, this actually says that

$$\mathbb{P}(Y_m \geq \varepsilon_i) \leq \exp\left(-\varepsilon_i^2 n_i/2\right)$$

This holds for any $m \in [2^{|V_X|}]$. Union bounding over all such m, we have that

$$\mathbb{P}(\exists m : Y_m \geq \varepsilon_i) \leq 2^{|V_X|} \exp\left(-\varepsilon_i^2 n_i/2\right)$$

If this does not happen (i.e., for every Y_m, we have $Y_m < \varepsilon_i$), then we have that $||\bar{\mathbf{r}}_i - \bar{\mathbf{r}}_i^*||_1 < \varepsilon_i$, since for any attribute value j, $|\bar{r}_j - \bar{r}_j^*| = \max_{t_j \in \{+1,-1\}} t_j(\bar{r}_j - \bar{r}_j^*)$. But if $Y_m < \varepsilon_i$ for all m, this means that we must have some m such that

$$\varepsilon_i > \sum_j f_m(j)(\bar{r}_j - \bar{r}_j^*) = \sum_j |\bar{r}_j - \bar{r}_j^*| = ||\bar{\mathbf{r}}_i - \bar{\mathbf{r}}_i^*||_1$$

As such $\mathbb{P}(\exists m : Y_m \geq \varepsilon_i)$ is an upper bound on $\mathbb{P}(||\bar{\mathbf{r}}_i - \bar{\mathbf{r}}_i^*||_1 \geq \varepsilon_i)$. The desired result follows from noting that

$$\delta_i \leq 2^{|V_X|} \exp\left(-\varepsilon_i^2 n_i/2\right)$$
$$\iff \varepsilon_i \leq \sqrt{\frac{2}{n_i}\left(|V_X| \log 2 + \log \frac{1}{\delta_i}\right)} \quad \square$$

Optimality of the bound in Theorem 1. If we solve for n_i in Theorem 1, we see that we must have $n_i = \frac{|V_X| \log 4 + 2\log(1/\delta_i)}{\varepsilon_i^2}$. That is, $\Omega\left(\frac{|V_X|}{\varepsilon_i^2}\right)$ samples are necessary guarantee that the empirical discrete distribution $\bar{\mathbf{r}}_i$ is no further than ε_i from the true discrete distribution $\bar{\mathbf{r}}_i^*$, with high probability. This matches the information theoretical lower bound noted in prior work [12, 20, 26, 72].

Generating P-values from Theorem 1. We use the above bound to generate P-values for testing the null hypotheses in Lemma 2. From the discussion in that lemma, a tester which rejects $H_0^{(i)}$ for $i \in M$ when it observes $s + \frac{\varepsilon}{2} - \tau_i^{\partial} > \varepsilon_i$, for fixed ε_i, has a type 1 error bounded above by $\delta_i = 2^{|V_X|} \exp\left(-\varepsilon_i^2 n_i/2\right)$. Since we want to bound the type 1 error rate by an amount δ^{upper}, this induces a particular ε_i against which we can compare $s + \frac{\varepsilon}{2} - \tau_i^{\partial}$, but because δ_i and ε_i are monotonically related, we can take

$$\delta_i = 2^{|V_X|} \exp\left(-(s + \frac{\varepsilon}{2} - \tau_i^{\partial})^2/2\right)$$

and compare with δ^{upper} directly, allowing us to use this δ_i as a P-value for use with the tester in Lemma 4.

3.4.4 Stage 2 Correctness

We can now show correctness of HistSim stage 2.

LEMMA 5 (STAGE 2 CORRECTNESS). *After* HistSim *stage 2 completes, each candidate $i \in M$, satisfies $\tau_i^* - \tau_j^* \leq \varepsilon$ for every $j \in A \setminus M$ with probability greater than $1 - \frac{\delta}{3}$.*

PROOF. First, show that if HistSim stage 2 terminates after iteration t, then the probability of an error is at most $\frac{\delta/3}{2^t}$. Next, show that the probability of an error after terminating at *any* iteration is at most $\frac{\delta}{3}$ by union bounding over iterations.

If stage 2 terminates at iteration t, then the probability of rejecting one or more null hypotheses is at most $\frac{\delta/3}{2^t}$ by Lemma 4 and by Theorem 1. Each $H_0^{(i)}$ for $i \in M$ says that $\tau_i^* > s + \frac{\varepsilon}{2}$, and each $H_0^{(j)}$ for $j \in A \setminus M$ says that $\tau_i^* < s - \frac{\varepsilon}{2}$ – if all of these are false, then by Lemma 2 we have that M and $A \setminus M$ induce a separation of the candidates that is correct with respect to Guarantee 1, so the only way an error *could* occur is if one or more nulls are true. We just established that the probability of rejecting one or more true nulls at iteration t is at most $\frac{\delta/3}{2^t}$, which means that the probability of an incorrect separation between M and $A \setminus M$ is also at most $\frac{\delta/3}{2^t}$.

Finally, by union bounding over iterations, we have that

$$\mathbb{P}(\cup_{t \geq 1} \text{mistake at iteration } t) \leq \sum_{t \geq 1} \mathbb{P}(\text{mistake at iteration } t)$$
$$< \sum_{t \geq 1} \frac{\delta/3}{2^t} = \delta/3$$

Thus, when stage 2 terminates, M is correct (with respect to Guarantee 1) with probability greater than $1 - \frac{\delta}{3}$ $\square$

3.5 Stage 3 and Overall Proof of Correctness

Stage 3 of HistSim, discussed in our overall proof of correctness, consists of taking samples from each candidate in M to ensure they all have ε-deviation with high probability (using Theorem 1). This proof is given next, and proceeds in four steps:

- Step 1: HistSim stage 1 incorrectly prunes one or more candidates meeting the selectivity threshold σ with probability at most $\frac{\delta}{3}$ (Lemma 1).
- Step 2: The probability that stage 2 incorrectly (with respect to Guarantee 1) separates M and $A \setminus M$ is at most $\frac{\delta}{3}$.
- Step 3: The probability that the set of candidates M violates Guarantee 2 after stage 3 runs is at most $\frac{\delta}{3}$.
- Step 4: The union bound over any of these bad events occurring gives an overall error probability of at most δ.

THEOREM 2. *The k histograms returned by Algorithm 1 satisfy Guarantees 1 and 2 with probability greater than $1 - \delta$.*

PROOF. From Lemma 1, the probability that high-selectivity candidates were pruned during stage 1 is upper bounded by $\frac{\delta}{3}$. From Lemma 5, the probability that the algorithm chooses M such that there exists some $i \in M$ and $j \in M^* \setminus M$ with $\tau_i^* - \tau_j^* > \varepsilon$ is at most $\frac{\delta}{3}$. Union bounding over these events, the probability of either occurring is at most $\frac{2\delta}{3}$. Since Guarantee 1 cannot be violated

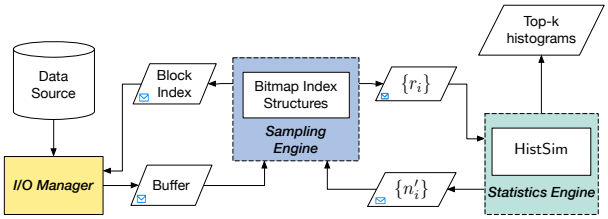


Figure 6: FastMatch system architecture

when neither of these events occur, the algorithm violates this guarantee also with probability at most $\frac{2\delta}{3}$. Finally, using Theorem 1, HistSim stage 3 line 26 takes a number of samples for each candidate $i \in M$ such that the probability that a given candidate fails to be reconstructed with error ε or less (that is, $d(\mathbf{r}_i, \mathbf{r}_i^*) > \varepsilon$) is at most $\frac{\delta}{3k}$. Union bounding over all candidates in M, and noting that $|M| = k$, the probability that one or more candidates does not have ε_i-deviation is at most $\frac{\delta}{3}$. Union bounding with the upper bound on the probability that Guarantee 1 is violated, the probability that either Guarantee 1 or Guarantee 2 is violated is at most $\frac{2\delta}{3} + \frac{\delta}{3} = \delta$, and we are done. □

Computational Complexity. Stage 1 of Algorithm 1 shares computation between candidates when computing P-values induced by the hypergeometric distribution, and thus makes at most $\max_{i \in V_Z} n_i$ calls to evaluate a hypergeometric pdf (we use Boost's implementation [1]); this can be done in $\mathcal{O}\left(\max_{i \in V_Z} n_i\right)$. To facilitate the sharing, stage 1 requires sorting the candidates in increasing order of n_i, which is $\mathcal{O}\left(|V_Z| \cdot \log |V_Z|\right)$. Next, each iteration of HistSim stage 2 requires computing distance estimates τ_i and τ_i^{∂} for every $i \in A$, which runs in time $\mathcal{O}\left(|A| \cdot |V_X|\right)$. Each iteration of stage 2 further uses a sort of candidates in A by τ_i to determine M and s, which is $\mathcal{O}\left(|A| \cdot \log |A|\right)$. HistSim stage 2 almost always terminates within 4 or 5 iterations in practice. Overall, we observe that the computation required is inexpensive compared to the cost of I/O, even for data stored in-memory.

4. THE FASTMATCH SYSTEM

This section describes FastMatch, which implements the HistSim algorithm. We start by presenting the high-level components of FastMatch. We then describe the challenges we faced while implementing FastMatch and describe how the components interact to alleviate those challenges, while still satisfying Guarantees Guarantee 1 and Guarantee 2. While design choices presented in this section are heuristics with practicality in mind, the algorithm implemented is still theoretically rigorous, with results satisfying our probabilistic guarantees. In the following, each time we describe a heuristic, we will clearly point it out as such.

4.1 FastMatch Components

FastMatch has three key components: the I/O Manager, the Sampling Engine, and the Statistics engine. We describe each of them in turn; Figure 6 provides an architecture diagram—we will revisit the interactions within the diagram at the end of the section.

I/O Manager. In FastMatch, requests for I/O are serviced at the granularity of *blocks*. The I/O manager simply services requests for blocks in a synchronous fashion. Given the location of some block, it synchronously processes the block at that location.

Sampling Engine. The sampling engine is responsible for deciding which blocks to sample. It uses bitmap index structures (described below) in order to determine the types of samples located at a given block. Given the current state of the system, it prioritizes certain candidates over others for sampling.

Statistics Engine. The statistics engine implements most of the logic in the HistSim algorithm. The only substantial difference between the actual code and the pseudocode presented in Algorithm 1 is that the statistics engine does not actually perform any sampling, instead leaving this responsibility to the sampling engine. The reason for separating these components will be made clear later on.

Bitmap Index Structures. FastMatch runs on top of a bitmap-based sampling system used for sampling on-demand, as in prior work [8, 47, 46, 64]. These papers have demonstrated that bitmap indexes [19] are effective in supporting sampling for incremental or early termination of visualization generation. Within FastMatch, bitmap indexes help the sampling engine determine whether a given block contains samples for a given candidate. For each attribute A, and each attribute value A_v, we store a bitmap, where a '0' at position p indicates that the corresponding block at position p contains no tuples with attribute value A_v, and a '1' indicates that block p contains one or more tuples with attribute value A_v. Candidate visualizations are generated by attribute values(or a predicate of ANDs and ORs over attribute values; see Appendix A), so these bitmaps allow the sampling engine to rapidly test whether a block contains tuples for a given candidate histogram. Bitmaps are amenable to significant compression [74, 75], and since we are further only requiring a single bit per block per attribute value, our storage requirements are orders-of-magnitude cheaper than past work that requires a bit per tuple [8, 46, 64]. Notice also that our techniques also apply for continuous candidate attributes; please see Appendix A for details.

4.2 Implementation Challenges

So far, we have designed HistSim without worrying about how sampling actually takes place, with an implicit assumption that there is no overhead to taking samples randomly across various candidates. While implementing HistSim within FastMatch, we faced several non-trivial challenges, outlined below:

- **Challenge 1: Random sampling at odds with performance characteristics of storage media.** The cost to fetch data is locality-dependent when dealing with real storage devices. Even if the data is stored in-memory, tuples (i.e., samples) that are spatially closer to a given tuple may be cheaper to fetch, since they may already be present in CPU cache.
- **Challenge 2: Deciding how many samples to take between rounds of HistSim.** The HistSim algorithm does not specify how many samples to taken in between rounds of stage 2; it is agnostic to this choice, with correctness unaffected. If the algorithm takes many samples, it may spend more time on I/O than is necessary to terminate with a guarantee. If the algorithm does not take enough samples, the statistical test on line 24 will probably not reject across many rounds, decaying δ^{upper} and making it progressively more difficult to get enough samples to meet stage 2's termination criterion.
- **Challenge 3: Non-uniform cost/benefit of different candidates.** Tuples for some candidates can be over-represented in the data and therefore take less time to sample compared to underrepresented candidates. At the same time, the benefit of sampling tuples corresponding to different candidate histograms is non-uniform: for example, those histograms which are "far" from the target distribution are less useful (in terms of getting HistSim to terminate quickly) than those for which HistSim chooses small values for ε_i.
- **Challenge 4: Assessing benefit to candidates depends on data seen so far.** The "best" choice of which tuples to sample for getting HistSim to terminate quickly can be most accurately estimated from *all* the data seen so far, including the most recent data. However, computing this estimate after processing every tuple and blocking I/O until the "best" decision can be made is prohibitively expensive.

We now describe our approaches to tackling these three challenges.

Challenge 1: Randomness via Data Layout

To maximize performance benefits from locality, we randomly permute the tuples of our dataset as a preprocessing step, and to "sample" we may then simply perform a linear scan of the shuffled data starting from any point. This matches the assumptions of stage 1 of HistSim, which requires samples to be taken without replacement. Although the theory we developed in Section 3 for HistSim stage 2 was for sampling with-replacement, as noted in [35, 11], it still holds now that we are sampling without replacement, as concentration results developed for the with-replacement regime may be transferred automatically to the without-replacement regime. This approach of randomly permuting upfront is not new, and is adopted by other approximate query processing systems [76, 63, 78].

Challenge 2: Deciding Samples to Take Between Rounds

The HistSim algorithm leaves the number of samples to take during a given round of stage 2 lines 19 unspecified; its correctness is guaranteed regardless of how this choice is made. This choice offers a tradeoff: take too many samples, and the system will spend a lot of time unnecessarily on I/O; take too few, and the algorithm will never terminate, since the "difficulty" of the test increases with each round, as we set $\delta^{upper} \leftarrow \delta^{upper}/2$.

To combat this challenge, we employ a simple heuristic. To estimate the number of samples we need to take for candidate i, we assume that $\tau_i = \tau_i^*$, so that we need to learn $\mathbf{r}_i^{\partial}$ to within ε_i' of $\mathbf{r}_i^*$ for a given round's statistical test to successfully reject, where $\varepsilon_i' = s + \frac{\varepsilon}{2} - \tau_i$ for $i \in M$ and $\varepsilon_i' = \tau_i - (s - \frac{\varepsilon}{2})$ for $i \in A \backslash M$. (Recall that we use ε_i-deviation to upper bound the P-values.) For this setting of $\{\varepsilon_i'\}$, we thus choose to take samples for each candidate by solving for n_i in the bound of Theorem 1. This yields

$$n_i' = 2\left(|V_X| \log 2 - \log \delta^{upper}\right) / \left(\varepsilon_i'\right)^2 \qquad (1)$$

Each round of stage 2 of our FastMatch implementation of HistSim thus continues to take samples until $n_i^{\partial} \geq n_i'$ for every candidate i. It then performs the multiple hypothesis test on lines 20–23. If it rejects, the algorithm terminates and the system gives the output to the user; otherwise, it once again estimates each n_i' using Equation (1) (plugging in $\{\varepsilon_i'\}$ from updated $\{\tau_i\}$) and repeats.

Challenge 3: Block Choice Policies

Deciding which blocks to read during stage 1 of HistSim is simple since we are only trying to detect low-selectivity candidates — in this case we just scan each block sequentially. Deciding which blocks to read during stage 2 of HistSim is more difficult due to the non-uniform cost (i.e., time) and benefit of samples for each candidate histogram. If either cost or benefit were uniform across candidates, matters would be simplified significantly: if cost were uniform, we could simply read in the blocks with the most beneficial candidates; if benefit were uniform, we could simply read in the lowest cost blocks (for example, those closest spatially to the current read position). To address these concerns, we developed a simple policy which we found worked well in practice for getting HistSim to terminate quickly.

AnyActive **block selection policy.** Recall that the end of each iteration of stage 2 of HistSim estimates the number of samples $\{n_i'\}$ necessary from each candidate so that the next iteration is more likely to terminate. Note that if each candidate satisfied $n_i = n_i'$ at the time HistSim performed the test for termination and *before* it computed the $\{n_i'\}$, then HistSim would be in a state where it can safely terminate. Those candidates for whom $n_i < n_i'$ we dub *active candidates*, and we employ a very simple block selection policy, dubbed the AnyActive block selection policy, which is to *only read blocks which contain at least one tuple corresponding to some active candidate*. The bitmap indexes employed by FastMatch allow it to rapidly test whether a block contains tuples for a given candidate visualization, and thus to rapidly apply the AnyActive block selection policy. Overall, our approach is as follows: we read blocks in sequence, and if blocks satisfy our AnyActive criterion, then we read all of the tuples in that block, else, we skip that block. We discuss how to make this approach performant below.

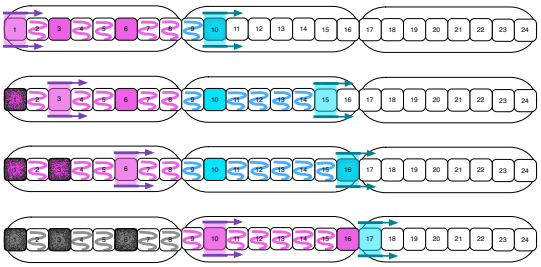


Figure 7: While the I/O manager processes magenta blocks, the sampling engine selects blue blocks ahead of time, using `lookahead`. Blocks with solid color = read, blocks with squiggles = skip.

A naive variant of this policy is presented in Algorithm 2, for which we describe improvements below.

Challenge 4: Asynchronous Block Selection

From the previous discussion, the sampling engine employs an AnyActive block selection policy when deciding which blocks to process. Ideally, the $\{n_i\}$ and $\{n_i'\}$ (number of samples taken for candidate i and estimated number of samples needed for candidate i, respectively) used to assign active status to candidates should be computed from the freshest possible counts available to the sampling engine. That is, in an ideal setting, each candidate's active status would be updated immediately after each block is read, and the potentially new active status should be used for making decisions about immediately subsequent blocks. Unfortunately, this requirement is at odds with real system characteristics. Employing it exactly implies leaving the I/O manager idle while the sampling engine determines whether each block should be read or skipped. To prevent this issue, we relax the requirement that the sampling thread employ AnyActive with the freshest $\{n_i\}$ available to it. Instead, given the current $\{n_i\}$ and fresh set of $\{n_i'\}$, it precomputes the active status for each candidate and "looks ahead", marking an entire batch of blocks for either reading or skipping, and communicates this with the I/O manager. The batch size, or the `lookahead` amount, is a system parameter, and offers a trade-off between freshness of active states used for AnyActive and degree to which the I/O manager must idle while waiting for instructions on which block to read next. We evaluate the impact of this parameter in our experimental section. The `lookahead` process is depicted in Figure 7 for a value of `lookahead` $= 8$. While the I/O manager processes a previously marked batch of magenta-colored `lookahead` blocks, the sampling engine's `lookahead` thread marks the next batch in blue. It waits to mark the next batch until the I/O manager "catches up".

Employing `lookahead` allows us to prevent two bottlenecks. First, the sampling engine need not wait for each candidate's active status to update after a block is read before moving on to the next block, effectively decoupling it from the I/O manager.

The second bottleneck prevented by `lookahead` is more subtle. To illustrate it, consider the pseudocode in Algorithm 2, implementing the AnyActive block policy. The AnyActive block policy algorithm works by considering each candidate in turn, and querying a bitmap index for that candidate to determine whether the current block contains tuples corresponding to that candidate. Querying a bitmap actually brings in surrounding bits into the cache of the CPU performing the query, and evicts whatever was previously

Algorithm 2: Naive AnyActive block processing

Input : unpruned candidate set A, block index i
Output : A value indicating whether to :read or :skip block i

```
for each active cand ∈ A do
    // cache inefficient index lookup
    // evicts bits from previous candidate's bitmap index
    if cand.index_lookup(i) then
        return :read;
    end
end
return :skip;
```

Algorithm 3: AnyActive block selection with lookahead

Input : lookahead amount, start block, unpruned candidate set A
Output : An array mark indicating whether to :read or :skip blocks

```
// Initialization
mark[i] ← :skip for 0 ≤ i < lookahead;
for each active cand ∈ A do
    for 0 ≤ i < lookahead do
        if mark[i] == :read then
            continue;
        else if cand.index_lookup(start + i) then
            mark[i] ← :read;
        end
    end
end
return mark
```

Dataset	Size	#Tuples	#Attributes	Replications
FLIGHTS	32 GiB	606 million	7	5×
TAXI	36 GiB	679 million	7	4×
POLICE	34 GiB	448 million	10	72×

Table 2: Descriptions of Datasets

in the cache line. If blocks are processed individually, then only a single bit in the bitmap is used each time a portion is brought into cache. This is quite wasteful and turns out to hurt performance significantly as we will see in the experiments. Instead, applying AnyActive selection to lookahead-size chunks instead of individual blocks is a better approach. This simply adds an extra inner loop to the procedure shown in Algorithm 2 (depicted in Algorithm 3). This approach has much better cache performance, since it uses an entire cache-line's worth of bits while employing AnyActive.

We verify in our experiments that these optimizations allow FastMatch to terminate more quickly via AnyActive block selection with *fresh-enough* active states without significantly slowing any single component of the system.

4.3 System Architecture

FastMatch is implemented within a few thousand lines of C++. It uses pthreads [59] for its threading implementation. FastMatch uses a column-oriented storage engine, as is common for analytics tasks. We can now complete our description of Figure 6. When the I/O manager receives a request for a block at a particular block index from the sampling engine (via the "block index" message), it eventually returns a buffer containing the data at this block to the sampling engine (via the "buffer" message). Once the I/O phase of stage 1 or 2 of HistSim completes, the sampling engine sends the current per-group counts for each candidate, $\{\mathbf{r}_i\}$, to the statistics engine. After running a test for whether to move to stage 2 (performed in stage 1) or to terminate (performed in stage 2), the statistics engine either posts a message of updated n' (in stage 1) or $\{n'_i\}$ (stage 2) that the sampling engine uses to determine when to complete the I/O phase of each HistSim stage, as well as how to perform block selection during stage 2.

5. EXPERIMENTAL EVALUATION

The goal of our experimental evaluation is to test the accuracy and runtime of FastMatch against other approximate and exact approaches on a diverse set of real datasets and queries. Furthermore, we want to validate the design decisions that we made for FastMatch in Section 4 and evaluate their impact.

5.1 Datasets and Queries

We evaluate FastMatch on publicly available real-world datasets summarized in Table 2 — flight records [2], taxi trips [3], and police road stops [4]. The replication value indicates how many times each dataset was replicated to create a larger dataset. In preprocessing these datasets, we eliminated rows with "N/A" or erroneous values for any column appearing in one or more of our queries.

FLIGHTS Dataset. Our FLIGHTS dataset, representing delays measured for flights at more than 350 U.S. airports from 1987 up to 2008, is available at [2]; we used 7 attributes (for origin / destination airports, departure / arrival delays, day of week, day of month, and departure hour).

TAXI Dataset. Our TAXI dataset summarizes all Yellow Cab trips in New York in 2013 [3]. The subset of data we used corresponds with the urls ending in "yellow_tripdata_2013" in the file raw_data_urls.txt. We extracted some time-based discrete attributes, two attributes based on passenger count, and one attribute based on area, for 7 columns total. In particular, the "Location" attribute was generated by binning the pickup location into regions of 0.01 longitude by 0.01 latitude. As with our FLIGHTS data, we discarded rows with missing values, as well as rows with outlier longitude or latitude values (which did not correspond to real locations). The taxi data stressed our algorithm's ability to deal with low-selectivity candidates, since more than 3000 candidates have fewer than 10 total datapoints.

POLICE Dataset. Our POLICE dataset summarizes more than 8 million police road stops in Washington state [4]. We extracted attributes for county, two gender attributes, two race attributes, road number, violation type, stop outcome, whether a search was conducted, and whether contraband was found, for 10 attributes total.

Queries and Query Format. We evaluate several queries on each dataset, whose templates are summarized in Table 3. We had four queries on FLIGHTS, FLIGHTS-q1-q4, two on TAXI, TAXI-q1-q2, and three on POLICE, POLICE-q1-q3. For simplicity, in all queries we test, the x-axis is generated by grouping over a single attribute (denoted by "X" in Table 3), and the different candidates are likewise generated by grouping over a single (different) attribute (signified by "Z"). For each query, the visual target was chosen to correspond with the closest distribution (under ℓ_1) to uniform, out of all histograms generated via the query's template, except for q1, q2, and q3 of FLIGHTS. Our queries spanned a number of interesting dimensions: *(i) frequently-appearing top-k candidates:* FLIGHTS-q1, POLICE-q1 and q2, *(ii) rarely-appearing top-k candidates:* FLIGHTS-q2 and q3, *(iii) high-cardinality candidate attribute Z:* TAXI-q1 and q2 ($|V_Z| = 7641$), POLICE-q3 ($|V_Z| = 2110$), and *(iv): high-cardinality grouping attribute X:* FLIGHTS-q4 ($|V_X| = 351$). The taxi queries in particular stressed our algorithm's ability to deal with low-selectivity candidates, since more than 3000 locations have fewer than 10 total datapoints.

5.2 Experimental Setup

Dataset	Query	Z ($\|V_Z\|$)	X ($\|V_X\|$)	k	target
FLIGHTS	q_1	Origin (347)	DepartureHour (24)	10	Chicago ORD
	q_2	Origin (347)	DepartureHour (24)	10	Appleton ATW
	q_3	Origin (347)	DayOfWeek (7)	5	[0.25, 0.125, 0.125, 0.125, 0.125, 0.125, 0.125]
	q_4	Origin (347)	Dest (351)	10	closest candidate to uniform
TAXI	q_1	Location (7641)	HourOfDay (24)	10	closest candidate to uniform
	q_2	Location (7641)	MonthOfYear (12)	10	closest candidate to uniform
POLICE	q_1	RoadID (210)	ContrabandFound (2)	10	closest candidate to uniform
	q_2	RoadID (210)	OfficerRace (5)	10	closest candidate to uniform
	q_3	Violation (2110)	DriverGender (2)	5	closest candidate to uniform

Table 3: Summary of queries

Query		Avg Speedup over Scan (raw time in (s))		
	Scan(s)	ScanMatch	SyncMatch	FastMatch
F-q1	12.26	27.74× (0.44)	25.53× (0.48)	**37.52**× (0.33)
F-q2	12.29	3.17× (3.87)	2.73× (4.51)	**10.11**× (1.21)
F-q3	11.62	4.76× (2.44)	3.14× (3.70)	**8.72**× (1.33)
F-q4	13.97	5.93× (2.36)	5.76× (2.43)	**8.15**× (1.71)
T-q1	13.09	4.89× (2.68)	0.32× (40.95)	**15.93**× (0.82)
T-q2	13.09	6.48× (2.02)	0.37× (35.60)	**17.38**× (0.75)
P-q1	8.57	5.72× (1.50)	5.14× (1.67)	**13.34**× (0.64)
P-q2	8.49	14.31× (0.59)	15.48× (0.55)	**36.11**× (0.24)
P-q3	8.65	9.25× (0.93)	1.53× (5.66)	**33.26**× (0.26)

Table 4: Summary of average query speedups and latencies

Approaches. We compare FastMatch against a number of less sophisticated approaches that provide the same guarantee as FastMatch. All approaches are parametrized by a minimum selectivity threshold σ, and all approaches except Scan are additionally parametrized by ε and δ and satisfy Guarantees 1 and 2 with probability greater than $1 - \delta$.

- SyncMatch($\varepsilon, \delta, \sigma$). This approach uses FastMatch, but the AnyActive block selection policy is applied without `lookahead`, synchronously and for each individual block. *By comparing this method with* FastMatch*, we quantify how much benefit we may ascribe to the* `lookahead` *technique.*
- ScanMatch($\varepsilon, \delta, \sigma$). This approach uses FastMatch, but without the AnyActive block selection policy. Instead, no blocks are pruned: it scans through each block in a sequential fashion until the statistics engine reports that HistSim's termination criterion holds. *By comparing this with* SyncMatch*, we quantify how much benefit we may ascribe to* AnyActive *block selection.*
- Scan(σ). This approach is a simple heap scan over the entire dataset and always returns correct results, trivially satisfying Guarantees 1 and 2. It exactly prunes candidates with selectivity below σ. *By comparing* Scan *with our above approximate approaches, we quantify how much benefit we may ascribe to the use of approximation.*

Environment. Experiments were run on single Intel Xeon E5-2630 node with 125 GiB of RAM and with 8 physical cores (16 logical) each running at 2.40 GHz, although we use at most 2 logical cores to run FastMatch components. The Level 1, Level 2, and Level 3 CPU cache sizes are, respectively: 512 KiB, 2048 KiB, and 20480 KiB. We ran Linux with kernel version 2.6.32. We report results for data stored in-memory, since the cost of main memory has decreased to the point that most interactive workloads can be performed entirely in-core. Each run of FastMatch or any other approximate approach was started from a random position in the shuffled data. We report both wall clock times and accuracy as the average across 30 runs with identical parameters, with the exception of Scan, whose wall clock times we report as the average over 5 runs. Where applicable, we used default settings of $m = 5 \cdot 10^5$, $\delta = 0.01$, $\varepsilon = 0.04$, $\sigma = 0.0008$, and `lookahead` $= 1024$. We set the block size for each column to 600 bytes, which we found to perform well; our results are not too sensitive to this choice.

5.3 Metrics

We use several metrics to compare FastMatch against our baselines in order to test two hypotheses: one, that FastMatch does indeed provide accurate answers, and two, that the system architecture developed in Section 4 does indeed allow for earlier termination while satisfying the separation and reconstruction guarantees.

Wall-Clock Time. Our primary metric evaluates the end-to-end time of our approximate approaches that are variants of FastMatch, as well as a scan-based baseline.

Satisfaction of Guarantees Guarantee 1 and Guarantee 2. Our δ parameter ($\delta = 0.01$), serves as an upper bound on the probability that either of these guarantees are violated. If this bound were tight, we would expect to see about one run in every hundred fail to satisfy our guarantees. We therefore count the number of times our guarantees are violated relative to the number of queries performed.

Total Relative Error in Visual Distance. In some situations, there may be several candidate histograms that are quite close to the analyst-supplied target, and choosing any one of them to be among the k returned to the analyst would be a good choice. We define the *total relative error in visual distance* (denoted by Δ_d) between the k candidates returned by FastMatch and the true k closest visualizations as: $\Delta_d(M, M^*, \mathbf{q}) = \frac{\sum_{i \in M} d(\mathbf{r}_i, \mathbf{q}) - \sum_{j \in M^*} d(\mathbf{r}_j^*, \mathbf{q})}{\sum_{j \in M^*} d(\mathbf{r}_j^*, \mathbf{q})}$ Note that here, M^* is computed by Scan and only considers candidates meeting the selectivity threshold. Since FastMatch and our other approximate variants have no recall requirements with respect to identifying low-selectivity candidates (they only have precision requirements), it is possible for $\Delta_d < 0$.

5.4 Empirical Results

Speedups and Error of FastMatch over others.

Summary. All FastMatch variants we tested show significant speedups over Scan for at least one query, but only FastMatch shows consistently excellent performance, typically beating other approaches and bringing latencies for all queries near interactive levels; with an overall speedup ranging between **8× and 35×** over Scan. Further, the output of FastMatch **and all approximate variants satisfied Guarantees 1 and 2 across all runs for all queries.**

Average run times of FastMatch and other approaches, for all queries as well as speedups over Scan, are summarized in Table 4. We used default settings for all runs. The reported speedups are the ratio of the average wall time of Scan with the average wall time of each approach considered. Scan was generally slower than approximate approaches because it had to examine all the data. Then, we typically observed that ScanMatch and SyncMatch were pretty evenly matched, with ScanMatch usually performing slightly better, except in some pathological cases where it performed very poorly due to poor cache usage. FastMatch had better performance than either SyncMatch or ScanMatch, thanks to `lookahead` paired with AnyActive block selection. Overall, we observed that each of FastMatch's key innovations: the termination criterion, the block selection, and `lookahead`, all led to substantial performance improvements, with an overall speedup of up to **35**× over Scan.

Queries with high candidate cardinality (TAXI-q*, POLICE-q3), displayed particularly interesting performance differences. For these, FastMatch shows greatly improved performance over ScanMatch.

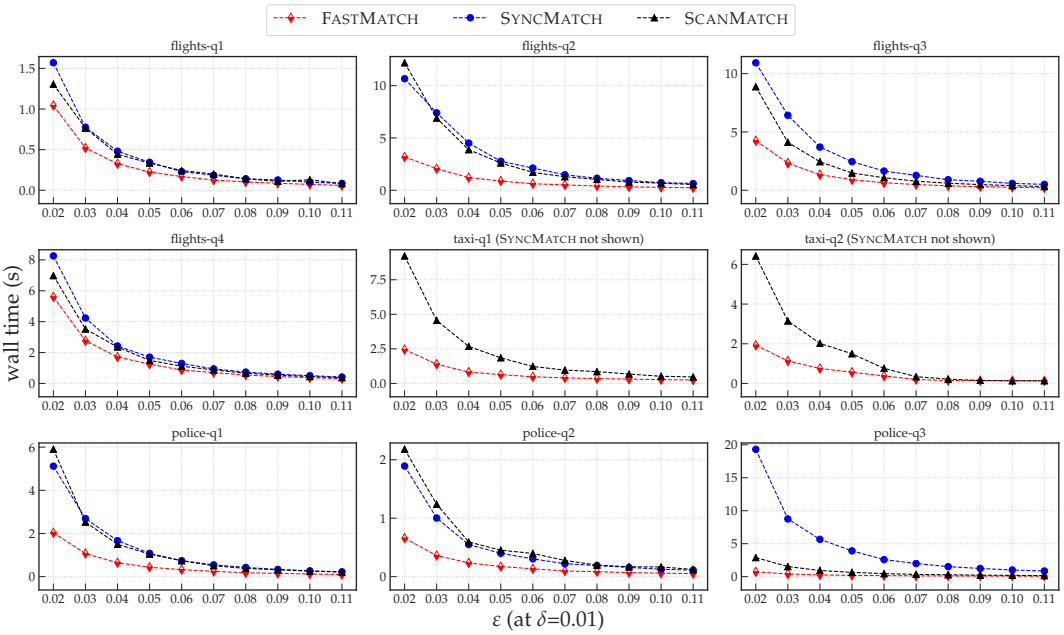


Figure 8: Effect of ε on query latency

It also scales much better to the large number of candidates than SyncMatch, which performs extremely poorly due to poor cache utilization and takes around $3\times$ longer than a simple non-approximate Scan. In this case, the `lookahead` technique of FastMatch is necessary to reap the benefits of AnyActive block selection.

Additionally, we found that the output of FastMatch *and all approximate variants satisfied Guarantees 1 and 2 across all runs for all queries.* This suggests that the parameter δ may be a loose upper bound for the actual failure probability of FastMatch.

Effect of varying ε.

> ***Summary.*** In almost all cases, increasing the tolerance parameter ε leads to reduced runtime and accuracy, but **on average, Δ_d was never more than 5% larger than optimal for any query, even for the largest values of ε used.**

Figures 8 and 9 depict the effect of varying ε on the wall clock time and on Δ_d, respectively, using $\delta = 0.01$ and `lookahead` = 1024, averaged over 30 runs for each value of ε. Because of the extremely poor performance of SyncMatch on the TAXI queries, we omit it from both figures.

In general, as we increased ε, wall clock time decreased and Δ_d increased. In some cases, ScanMatch latencies matched that of Scan until we made ε large enough. This sometimes happened when it needed more refined estimates of the (relatively infrequent) top-k candidates, which it achieved by scanning most of the data, picking up lots of superfluous (in terms of achieving safe termination) tuples along the way.

Effect of varying `lookahead`.

> ***Summary.*** When the number of candidates $|V_Z|$ is not large, performance is relatively stable as `lookahead` varies. For large $|V_Z|$, more `lookahead` helps performance, but is not crucial.

For most queries, we found that latency was relatively robust to changes in `lookahead`. Figure 10 depicts this effect. The queries with high candidate cardinalities (TAXI-q*, POLICE-q3) were the exceptions. For these queries, larger `lookahead` values led to increased utilization at all levels of CPU cache. Past a certain point, however, the performance gains were minor. Overall, we found the default value of 1024 to be acceptable in all circumstances.

Effect of varying δ. In general, we found that increasing δ led to slight decreases in wall clock time, leaving accuracy (in terms of Δ_d) more or less constant. We believe this behavior is inherited from our bound in Theorem 1, which is not sensitive to changes in δ. Figure 11 shows the effect of varying δ on wall clock time. For the values of δ we tried, we did not observe any meaningful trends in Δ_d and have omitted the plot.

Query	$\frac{\lvert M^*(\ell_1)\cap M^*(\ell_2)\rvert}{k}$	Relative distance difference
FLIGHTS-q_1	0.9	0.01
FLIGHTS-q_2	0.7	0.04
FLIGHTS-q_3	0.6	0.03
FLIGHTS-q_4	0.8	0.01

Table 5: Comparison of top-closest histograms for ℓ_1 and ℓ_2

When approximation performs poorly. In order to achieve the competitive results presented in this section, the initial pruning of low-selectivity candidates during stage 1 of HistSim ended up being critical for good performance. With a selectivity threshold of $\sigma = 0$, stages 2 and 3 of HistSim are forced to consider many extremely rare candidates. For example, in the taxi queries, nearly half of candidates have fewer than 10 corresponding datapoints. In this case, ScanMatch performs the best (essentially performing a Scan with a slight amount of additional overhead), but it (necessarily) fails to take enough samples to establish Guarantees 1 and 2. SyncMatch and FastMatch likewise fail to establish guarantees, but additionally have the issue of being forced to consider many rare candidates while employing AnyActive block selection, which can slow town query processing by a factor of $100\times$ or more.

Comparing results for ℓ_1 and ℓ_2 metrics. So far, we have not validated our choice of distance metric (normalized ℓ_1); prior work has shown that normalized ℓ_2 is suitable for assessing the "visual" similarity of visualizations [71], so here, we compare our top-k with the top-k using the normalized ℓ_2 metric, for the FLIGHTS queries. In brief, we found that the relative difference in the total ℓ_1 distance of the top-k using the two metrics never exceeded 4% for any query, and that roughly 75% of the top-k candidates were common across the two metrics. Thus, ℓ_1 can serve as a suitable replacement for ℓ_2, while further benefiting from the advantages we described in Section 2. Table 5 summarizes our full results.

6. RELATED WORK

We now briefly cover work that is related to FastMatch from a number of different areas.

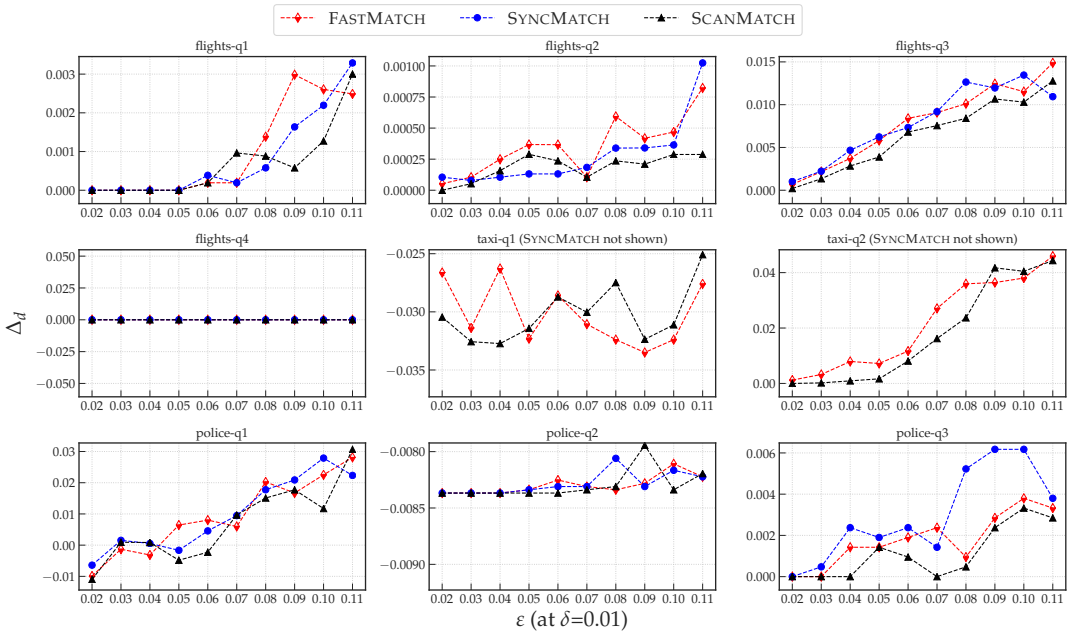


Figure 9: Effect of ε on Δ_d

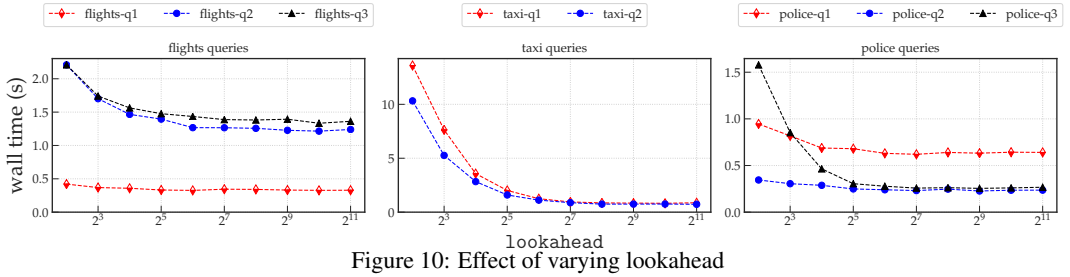


Figure 10: Effect of varying lookahead

Approximate Query Processing (AQP). Offline AQP involves computing a set of samples offline, and then using these samples when queries arrive e.g., [40, 21, 5, 9, 7], with systems like BlinkDB [7] and Aqua [6]. These techniques crucially rely on the availability of a workload. On the other hand, online approximate query processing, e.g., [34, 37, 53], performs sampling on-the-fly, typically using an index to facilitate the identification of appropriate samples. Our work falls into the latter category; however, none of the prior work has addressed a similar problem of identifying relevant visualizations given a query.

Top-K or Nearest Neighbor Query Processing. There is a vast body of work on top-k query processing [38]. Most of this work relies on exact answers, as opposed to approximate answers, and has different objectives. As an example, Bruno et al. [16] exploit statistics maintained by a RDBMS in order to quickly find top-k tuples matching user-specified attribute values. Some work tries to bridge the gap between top-k query processing and uncertain query processing [69, 65, 68, 25, 61, 23, 49, 14], but does not need to deal with the concerns of where and when to sample to return answers quickly, but approximately. Some of this work [69, 65, 49, 14] develops efficient algorithms for top-k or nearest neighbors in a uncertain databases setting—here, the sampling is restricted to monte-carlo sampling, which is very different in behavior. Silberstein et al. [68] retain samples of past sensor readings to avoid maintaining joint probability distributions in a sensor network. Cohen et al. [25] develops techniques to bound the probability of a given set of items being part of the top-k. Pietracaprina et al. [61] develops sampling schemes tailored to finding top-k frequent itemsets. Chen et al. [23] employ sampling to determine the bounds of a minimum bounding rectangle for top-k nearest neighbor queries. Zhang et al. [79] performs top-k similarity search efficiently in a network context.

Scalable Visualizations. There has been some limited work on scalable approximate visualizations, targeting the generation of a single visualization, while preserving certain properties [46, 60, 64]. In our setting, the space of sampling is much larger—as a result the problem is more complex. Furthermore, the objectives are very different. Fisher et al. [30] explores the impact of approximate visualizations on users, adopting an online-aggregation-like [34] scheme. As such, these papers show that users are able to interpret and utilize approximate visualizations correctly. Some work uses pre-materialization for the purpose of displaying visualizations quickly [44, 51, 55]; however, these techniques rely on in-memory data cubes. We covered other work on scalable visualization via approximation [28, 57, 43, 60, 77, 71] in Section 1.

Histogram Estimation for Query Optimization. A number of related papers [22, 39, 41] are concerned with the problem of sampling for histogram estimation, usually for estimating attribute value selectivities [52] and query size estimation (see [24] for a recent example). While some of the theoretical tools used are similar, the problem is fundamentally different, in that the aforementioned line of work is concerned with estimating one histogram per table or view for query optimization purposes with low error, while we are concerned with comparing histograms to a specific target.

Sublinear Time Algorithms. HistSim is related to work on sublinear time algorithms—the most relevant ones [12, 20, 72] fall under the setting of distribution learning and analysis of *property*

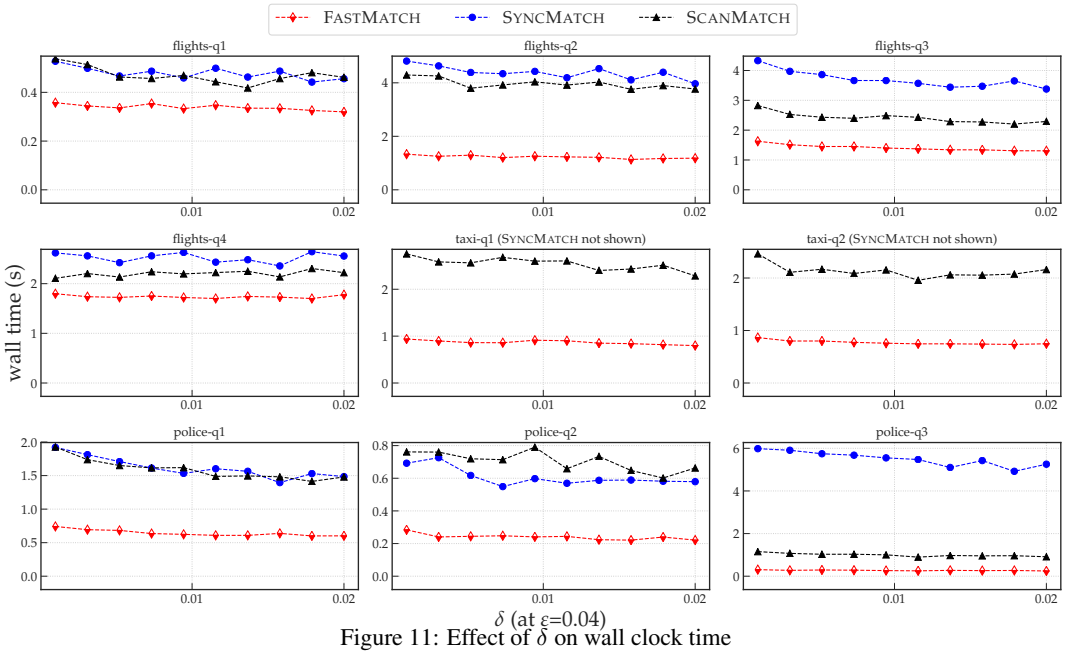


Figure 11: Effect of δ on wall clock time

testers for whether distributions are close under ℓ_1 distance. Although Chan et al. [20] develop bounds for testing whether distributions are ε-close in the ℓ_1 metric, property testers can only say when two distributions p and q are equal or ε-far, and cannot handle $||p - q||_1 < \varepsilon$ for $p \neq q$, a necessary component of this work.

7. CONCLUSION AND FUTURE WORK

We developed sampling-based strategies for rapidly identifying the top-k histograms that are closest to a target. We designed a general algorithm, HistSim, that provides a principled framework to facilitate this search, with theoretical guarantees. We showed how the systems-level optimizations present in our FastMatch architecture are crucial for achieving near-interactive latencies consistently, leading to speedups ranging from $8\times$ to $35\times$ over baselines. While this work suggests several possible avenues for further exploration, we are particularly interested in exploring the impact of our systems architecture in supporting general interactive analysis.

8. REFERENCES

[1] Boost Statistical Distributions and Functions. https://www.boost.org/doc/libs/1_67_0/libs/math/doc/html/dist.html, 2006.

[2] Flight Records. http://stat-computing.org/dataexpo/2009/the-data.html, 2009.

[3] NYC Taxi Trip Records. https://github.com/toddwschneider/nyc-taxi-data/, 2015.

[4] WA Police Stop Records. https://stacks.stanford.edu/file/druid:py883nd2578/WA-clean.csv.gz, 2017.

[5] S. Acharya, P. B. Gibbons, and V. Poosala. Congressional samples for approximate answering of group-by queries. In *ACM SIGMOD Record*, volume 29, pages 487–498. ACM, 2000.

[6] S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy. The aqua approximate query answering system. In *ACM Sigmod Record*, volume 28, pages 574–576. ACM, 1999.

[7] S. Agarwal, B. Mozafari, A. Panda, H. Milner, S. Madden, and I. Stoica. Blinkdb: Queries with bounded errors and bounded response times on very large data. In *EuroSys*, pages 29–42, New York, NY, USA, 2013. ACM.

[8] D. Alabi and E. Wu. Pfunk-h: approximate query processing using perceptual models. In *HILDA@ SIGMOD*, page 10, 2016.

[9] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. In *STOC*, pages 20–29. ACM, 1996.

[10] B. Babcock, S. Chaudhuri, and G. Das. Dynamic sample selection for approximate query processing. In *SIGMOD*, New York, New York, USA, 2003.

[11] R. Bardenet, O.-A. Maillard, et al. Concentration inequalities for sampling without replacement. *Bernoulli*, 21(3):1361–1385, 2015.

[12] T. Batu, L. Fortnow, R. Rubinfeld, W. D. Smith, and P. White. Testing that distributions are close. In *FOCS*, 2000.

[13] J. T. Behrens. Principles and procedures of exploratory data analysis. *Psychological Methods*, 2(2):131, 1997.

[14] G. Beskales, M. A. Soliman, and I. F. Ilyas. Efficient search for the top-k probable nearest neighbors in uncertain databases. *Proceedings of the VLDB Endowment*, 1(1):326–339, 2008.

[15] M. Bostock, V. Ogievetsky, and J. Heer. D^3 data-driven documents. *IEEE TVCG*, 17(12):2301–2309, 2011.

[16] N. Bruno, S. Chaudhuri, and L. Gravano. Top-k selection queries over relational databases: Mapping strategies and performance evaluation. *ACM TODS*, 27(2):153–187, June 2002.

[17] G. Casella and R. L. Berger. *Statistical inference*, volume 2. Duxbury Pacific Grove, CA, 2002.

[18] K. Chakrabarti, M. Garofalakis, R. Rastogi, and K. Shim. Approximate query processing using wavelets. *The VLDB Journal*, 10(2-3):199–223, Sept. 2001.

[19] C.-Y. Chan and Y. E. Ioannidis. Bitmap index design and evaluation. In *ACM SIGMOD Record*, volume 27, pages 355–366. ACM, 1998.

[20] S.-O. Chan, I. Diakonikolas, G. Valiant, and P. Valiant. Optimal algorithms for testing closeness of discrete distributions. In *SODA*, pages 1193–1203, 2014.

[21] S. Chaudhuri, G. Das, M. Datar, R. Motwani, and V. Narasayya. Overcoming limitations of sampling for aggregation queries. In *ICDE*, pages 534–542. IEEE, 2001.

[22] S. Chaudhuri, R. Motwani, and V. Narasayya. Random sampling for histogram construction: How much is enough? In *ACM SIGMOD Record*, volume 27, pages 436–447. ACM, 1998.

[23] C.-M. Chen and Y. Ling. A sampling-based estimator for top-k selection query. In *Data Engineering, 2002. Proceedings. 18th International Conference on*, pages 617–627. IEEE, 2002.

[24] Y. Chen and K. Yi. Two-level sampling for join size estimation. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 759–774. ACM, 2017.

[25] E. Cohen, N. Grossaug, and H. Kaplan. Processing top-k queries from samples. *Computer Networks*, 52(14):2605–2622, 2008.

[26] C. Daskalakis, I. Diakonikolas, R. ODonnell, R. A. Servedio, and L.-Y. Tan. Learning sums of independent integer random variables. In *FOCS*, pages 217–226. IEEE, 2013.

[27] I. Diakonikolas. Personal communication, 2017.

[28] B. Ding, S. Huang, S. Chaudhuri, K. Chakrabarti, and C. Wang. Sample + seek: Approximating aggregates with distribution precision guarantee. In *SIGMOD*, 2016.

[29] M. El-Hindi, Z. Zhao, C. Binnig, and T. Kraska. Vistrees: fast indexes for interactive data exploration. In *Proceedings of the Workshop on Human-In-the-Loop Data Analytics*, page 5. ACM, 2016.

[30] D. Fisher, I. Popov, S. Drucker, and m.c. Schraefel. Trust me, i'm partially right. In *CHI*, page 1673, New York, New York, USA, may 2012. ACM Press.

[31] V. Ganti, M.-L. Lee, and R. Ramakrishnan. Icicles: Self-tuning samples for approximate query answering. In *VLDB*, volume 176, 2000.

[32] A. L. Gibbs and F. E. Su. On choosing and bounding probability metrics. *International statistical review*, 70(3):419–435, 2002.

[33] P. Hanrahan. Analytic database technologies for a new kind of user: The data enthusiast. In *SIGMOD*, pages 577–578, New York, NY, USA, 2012. ACM.

[34] J. M. Hellerstein, P. J. Haas, and H. J. Wang. Online aggregation. *ACM SIGMOD Record*, 26(2):171–182, jun 1997.

[35] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American statistical association*, 58(301):13–30, 1963.

[36] S. Holm. A simple sequentially rejective multiple test procedure. *Scandinavian journal of statistics*, pages 65–70, 1979.

[37] W.-C. Hou, G. Ozsoyoglu, and B. K. Taneja. Processing aggregate relational queries with hard time constraints. In *ACM SIGMOD Record*, volume 18, pages 68–77. ACM, 1989.

[38] I. F. Ilyas, G. Beskales, and M. A. Soliman. A survey of top-*k* query processing techniques in relational database systems. *ACM Comput. Surv.*, 40(4), 2008.

[39] Y. E. Ioannidis and V. Poosala. Balancing histogram optimality and practicality for query result size estimation. In *ACM SIGMOD Record*, volume 24, pages 233–244. ACM, 1995.

[40] Y. E. Ioannidis and V. Poosala. Histogram-based approximation of set-valued query-answers. In *VLDB*, volume 99, pages 174–185, 1999.

[41] H. V. Jagadish, N. Koudas, S. Muthukrishnan, V. Poosala, K. C. Sevcik, and T. Suel. Optimal histograms with quality guarantees. In *VLDB*, volume 98, pages 24–27, 1998.

[42] N. L. Johnson, A. W. Kemp, and S. Kotz. *Univariate discrete distributions*, volume 444. John Wiley & Sons, 2005.

[43] U. Jugel, Z. Jerzak, G. Hackenbroich, and V. Markl. M4: a visualization-oriented time series data aggregation. *PVLDB*, 7(10):797–808, 2014.

[44] S. Kandel, R. Parikh, A. Paepcke, J. M. Hellerstein, and J. Heer. Profiler: Integrated statistical analysis and visualization for data quality assessment. In *AVI*, pages 547–554. ACM, 2012.

[45] M. S. Kester, M. Athanassoulis, and S. Idreos. Access path selection in main-memory optimized data systems: Should i scan or should i probe? In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 715–730. ACM, 2017.

[46] A. Kim, E. Blais, A. Parameswaran, P. Indyk, S. Madden, and R. Rubinfeld. Rapid sampling for visualizations with ordering guarantees. *PVLDB*, 8(5):521–532, Jan. 2015.

[47] A. Kim, S. Madden, and A. Parameswaran. Needletail: A system for browsing queries (demo). Technical report, Available at: `http://i.stanford.edu/~adityagp/ntail-demo.pdf`, 2014.

[48] A. Kim, L. Xu, T. Siddiqui, S. Huang, S. Madden, and A. Parameswaran. Optimally leveraging density and locality to support limit queries. Technical report, Available at: `https://arxiv.org/pdf/1611.04705.pdf`, 2016.

[49] H.-P. Kriegel, P. Kunath, and M. Renz. Probabilistic nearest-neighbor query on uncertain objects. *Advances in databases: concepts, systems and applications*, pages 337–348, 2007.

[50] E. L. Lehmann and J. P. Romano. *Testing statistical hypotheses*. Springer Science & Business Media, 2006.

[51] L. D. Lins, J. T. Klosowski, and C. E. Scheidegger. Nanocubes for real-time exploration of spatiotemporal datasets. *IEEE TVCG*, 19(12):2456–2465, 2013.

[52] R. J. Lipton, J. F. Naughton, and D. A. Schneider. *Practical selectivity estimation through adaptive sampling*, volume 19. ACM, 1990.

[53] R. J. Lipton, J. F. Naughton, D. A. Schneider, and S. Seshadri. Efficient sampling strategies for relational database operations. *Theoretical Computer Science*, 116(1):195–226, 1993.

[54] Z. Liu and J. Heer. The effects of interactive latency on exploratory visual analysis. *IEEE TVCG*, 20(12):2122–2131, 2014.

[55] Z. Liu, B. Jiang, and J. Heer. immens: Real-time visual querying of big data. In *CGF*, volume 32, pages 421–430. Wiley Online Library, 2013.

[56] C. McDiarmid. On the method of bounded differences. *Surveys in combinatorics*, 141(1):148–188, 1989.

[57] D. Moritz, D. Fisher, B. Ding, and C. Wang. Trust, but verify: Optimistic visualizations of approximate queries for exploring big data. In *CHI*, pages 2904–2915. ACM, 2017.

[58] B. Mozafari. Approximate query engines: Commercial challenges and research opportunities. In *SIGMOD*, pages

521–524. ACM, 2017.

[59] B. Nichols, D. Buttlar, and J. Farrell. *Pthreads programming: A POSIX standard for better multiprocessing*. " O'Reilly Media, Inc.", 1996.

[60] Y. Park, M. Cafarella, and B. Mozafari. Visualization-aware sampling for very large databases. In *ICDE*, pages 755–766. IEEE, 2016.

[61] A. Pietracaprina, M. Riondato, E. Upfal, and F. Vandin. Mining top-k frequent itemsets through progressive sampling. *Data Mining and Knowledge Discovery*, 21(2):310–326, 2010.

[62] P. Pirolli and S. Card. The sensemaking process and leverage points for analyst technology as identified through cognitive task analysis. In *Proceedings of international conference on intelligence analysis*, volume 5, pages 2–4, 2005.

[63] C. Qin and F. Rusu. Pf-ola: a high-performance framework for parallel online aggregation. *Distributed and Parallel Databases*, 32(3):337–375, 2014.

[64] S. Rahman, M. Aliakbarpour, H. K. Kong, E. Blais, K. Karahalios, A. Parameswaran, and R. Rubinfield. I've seen "enough": Incrementally improving visualizations to support rapid decision making. In *VLDB*, 2017.

[65] C. Ré, N. N. Dalvi, and D. Suciu. Efficient top-k query evaluation on probabilistic data. In *ICDE*, pages 886–895, 2007.

[66] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price. Access path selection in a relational database management system. In *Proceedings of the 1979 ACM SIGMOD international conference on Management of data*, pages 23–34. ACM, 1979.

[67] T. Siddiqui, A. Kim, J. Lee, K. Karahalios, and A. Parameswaran. Effortless data exploration with zenvisage: an expressive and interactive visual analytics system. *PVLDB*, 10(4):457–468, 2016.

[68] A. S. Silberstein, R. Braynard, C. Ellis, K. Munagala, and J. Yang. A sampling-based approach to optimizing top-k queries in sensor networks. In *Data Engineering, 2006. ICDE'06. Proceedings of the 22nd International Conference on*, pages 68–68. IEEE, 2006.

[69] M. A. Soliman, I. F. Ilyas, and K. C. Chang. Top-k query processing in uncertain databases. In *ICDE*, pages 896–905, 2007.

[70] C. Stolte, D. Tang, and P. Hanrahan. Polaris: A system for query, analysis, and visualization of multidimensional relational databases. *IEEE TVCG*, 8(1):52–65, 2002.

[71] M. Vartak, S. Rahman, S. Madden, A. Parameswaran, and N. Polyzotis. Seedb: efficient data-driven visualization recommendations to support visual analytics. *PVLDB*, 8(13):2182–2193, 2015.

[72] B. Waggoner. L p testing and learning of discrete distributions. In *ITCS*, pages 347–356. ACM, 2015.

[73] H. Wickham. *ggplot2: elegant graphics for data analysis*. Springer, 2016.

[74] K. Wu, E. Otoo, and A. Shoshani. Compressed bitmap indices for efficient query processing. *Lawrence Berkeley National Laboratory*, 2001.

[75] K. Wu, K. Stockinger, and A. Shoshani. Breaking the curse of cardinality on bitmap indexes. In *Scientific and Statistical Database Management*, pages 348–365. Springer, 2008.

[76] S. Wu, B. C. Ooi, and K.-L. Tan. Continuous sampling for online aggregation over multiple queries. In *SIGMOD*, pages 651–662. ACM, 2010.

[77] Y. Wu, B. Harb, J. Yang, and C. Yu. Efficient evaluation of object-centric exploration queries for visualization. *PVLDB*, 8(12):1752–1763, 2015.

[78] K. Zeng, S. Agarwal, A. Dave, M. Armbrust, and I. Stoica. G-ola: Generalized on-line aggregation for interactive analysis on big data. In *SIGMOD*, pages 913–918. ACM, 2015.

[79] J. Zhang, J. Tang, C. Ma, H. Tong, Y. Jing, J. Li, W. Luyten, and M.-F. Moens. Fast and flexible top-k similarity search on large networks. *ACM Transactions on Information Systems (TOIS)*, 36(2):13, 2017.

APPENDIX

A. EXTENSIONS

A.1 Generalizing Problem Description

A.1.1 SUM *aggregations*

While we do not consider it explicitly in this paper, in [28], the authors describe how to perform SUM aggregations with ℓ_2 distributional guarantees via *measure-biased sampling*. Briefly, a measured-biased sample for some attribute Y involves sampling each tuple t in T, where the probability of inclusion in the sample is proportional t's value of Y. FastMatch can also leverage measure-biased samples in order to match bar graphs generated via the following types of queries:

SELECT X, SUM(Y) FROM T
WHERE $Z = z_i$ GROUP BY X

As in Definition 1, Z is the candidate attribute and X is the grouping attribute for the x-axis. One measure-biased sample must be created per measure attribute Y the analyst is interested in, so if there are n such attributes, we require an additional n complete passes over the data for preprocessing. When matching bar graphs generated according to the above template, FastMatch would simply use the measure-biased sample for Y and pretend as if it were matching visualizations generated according to Definition 1; that is, it would use COUNT instead of SUM. There is nothing special about the ℓ_2 metric used in [28], and the same techniques may be used by FastMatch to process queries satisfying Guarantees 1 and 2.

A.1.2 Candidates based off arbitrary boolean predicates

In order to support candidates based off boolean predicates such as $Z^{(1)} = z_i^{(1)} \wedge Z^{(2)} = z_j^{(2)}$, FastMatch needs a way to estimate the number of active tuples in a block for the purposes of applying AnyActive block selection. In this case, simple bitmap indexes with one bit per block are not enough. We may instead opt to use the slightly costlier *density maps* from [48]. We refer readers to that paper for a description of how to estimate the number of tuples in a block satisfying an arbitrary boolean predicate. Even if different candidates share some of the same tuples, our guarantees still hold since HistSim uses a Holm-Bonferroni procedure to get joint guarantees across different candidates at a given iteration, a method which is agnostic to any dependency structure between candidates.

A.1.3 Multiple attributes in GROUP BY *clause*

In the case where the analyst wishes to use multiple attributes $X^{(1)}, X^{(2)}, \ldots, X^{(n)}$ to generate the support of our histograms generated via Definition 1, all of the same methods apply, but we estimate the support $|V_X|$ as

$$|V_{X^{(1)}}| \cdot |V_{X^{(2)}}| \cdot \ldots \cdot |V_{X^{(n)}}|$$

This may be an overestimate if two attribute values, say $x_i^{(1)}$ and $x_j^{(2)}$, never occur together. Our guarantees still hold in this case — overestimating the size of the support can only make the bound in Theorem 1 looser than it could be, which does not cause any correctness issues.

A.1.4 Handling continuous X attributes via binning

If the analyst wishes to use a continuous X, she must simply provide a set of non-overlapping bin ranges, or "buckets" in which to collect tuples. Everything else is still the same. In fact, FLIGHTS-q1 and FLIGHTS-q2 used this technique, since the DepartureHour attribute was actually a continuous attribute we placed into 24 bins (although we presented it as a discrete attribute for simplicity).

A.1.5 Handling an Unknown Candidate Domain

If the candidate domain is unknown at query time, for example if we do not have any bitmap index structures over the attribute(s) used to generate candidates, it is still possible to use a variant of our methods. First of all, we may still employ ScanMatch, creating state for new candidates as they are discovered. During stage 1 of HistSim, in which rare candidates are pruned, we must also account for any potential candidates for which HistSim has not yet seen any tuples. In this case, we may simply add one additional "dummy" candidate which matches against all the tuples for any unseen candidates. We add an additional test to the Holm-Bonferroni procedure for this dummy candidate — if the test rejects, and if U represents the indices of the unseen candidates, then we can be sure that $\frac{\sum_{j \in U} N_j}{N} < \sigma$, which in turn implies that $\frac{N_j}{N} < \sigma$ for each $j \in U$.

A.1.6 Handling Continuous Candidates

If one or more of the attributes used to group candidates is continuous, then, as in the case of continuous X, candidates may be "grouped" by placing different real-values into bins. We can also construct bitmaps for continuous attributes at some predetermined finest level of granularity of binning, which can then be used to induce bitmaps for any coarser granularity that may be needed. Even if the finest granularity available is too coarse to isolate different candidates, as long as it isolates some subsets of candidates, it may still be useful for pruning the blocks that need to be considered for AnyActive block selection. Even if there is no index available, one may still use ScanMatch.

A.2 Different Types of Guarantees

A.2.1 Allowing Distinct ε_1 and ε_2 for Guarantees 1 and 2

If the analyst believes one of Guarantees 1 and 2 is more important than the other, she may indicate this by providing separate ε_1 for Guarantee 1 and ε_2 for Guarantee 2; HistSim generalizes in a very straightforward way in this case. For example, if Guarantee 2 is more important than Guarantee 1, the analyst may provide ε_1 and ε_2 with $\varepsilon_2 < \varepsilon_1$.

A.2.2 Allowing other distance metrics

We can extend HistSim to work for any distance metric for which there exists an analogue to Theorem 1. For example, there exist such bounds for ℓ_2 distance [28, 72].

A.2.3 Allowing a range of k in input

In some cases, the analyst may not care about the exact number of matching candidates. For example, the analyst may be fine with finding anywhere between 5 and 10 of the closest histograms to a target. In this case, she may specify a range $[k_1, k_2]$, and FastMatch may return some number $k \in [k_1, k_2]$ of histograms matching the target, where k is automatically picked to make it as easy as possible to satisfy Guarantees 1 and 2. For example, in the case $[k_1, k_2] = [5, 10]$, there may be a very large separation between the 7th- and 8th-closest candidates, in which case HistSim can automatically choose $k = 7$, as this likely provides a small δ^{upper} as soon as possible.