SUPPORTING EXPLORATORY QUERIES IN DATABASES

Abhijit Kadlag Amol Wanjari Juliana Freire[1] Jayant R. Haritsa

Technical Report
TR-2003-02

Database Systems Lab
Supercomputer Education and Research Centre
Indian Institute of Science
Bangalore 560012, India

`http://dsl.serc.iisc.ernet.in`

[1]Computer Science and Engineering, Oregon Graduate Institute, Beaverton, Oregon 97006, USA.

Supporting Exploratory Queries in Databases

Abhijit Kadlag[†] Amol Wanjari[†] Juliana Freire[‡] Jayant R. Haritsa[†]

[†]Computer Science & Automation
Indian Institute of Science
Bangalore 560012, India

[‡]Computer Science Engineering
OGI/OHSU
Beaverton, Oregon 97006, USA

Abstract

Users of database applications, especially in the e-commerce domain, often resort to exploratory "trial-and-error" queries since the underlying data space is huge and unfamiliar, and there are several alternatives for search attributes in this space. For example, scouting for cheap airfares typically involves posing multiple queries, varying flight times, dates, and airport locations. Exploratory queries are problematic from the perspective of both the user and the server. For the database server, it results in a drastic reduction in effective throughput since much of the processing is duplicated in each successive query. For the client, it results in a marked increase in response times, especially when accessing the service through wireless channels.

In this paper, we investigate the design of automated techniques to minimize the need for repetitive exploratory queries. Specifically, we present SAUNA, a server-side query relaxation algorithm that, given the user's initial range query and a desired cardinality for the answer set, produces a relaxed query that is expected to contain the required number of answers. The algorithm incorporates a range-query-specific distance metric that is weighted to produce relaxed queries of a desired shape (e.g., aspect ratio preserving), and utilizes multi-dimensional histograms for query size estimation. A detailed performance evaluation of SAUNA over a variety of multi-dimensional data sets indicates that its relaxed queries can significantly reduce the costs associated with exploratory query processing. To improve the performance of SAUNA even further we have studied the wavelet techniques for query size estimation and found them to be better.

1 Introduction

An increasing number of Web applications are utilizing *database engines* as their backend information storage system. In fact, a recent survey [1] states that more than 200,000 Web sites generate content from databases containing 7500 terabytes of information, and they receive 50% more monthly traffic than other sites.

Users of database applications, especially in the e-commerce domain, often resort to exploratory "trial-and-error" queries since the underlying data space is huge and unfamiliar, and there are several alternatives for search attributes in this space [2]. Consider, for example, the query interface provided at Travelocity [3], a popular Web site for travel planning. Here, for each itinerary, users must select origin and destination airports, departure and return times, departure and return dates, and may optionally select airlines. Faced with this environment, users often pose a *sequence* of *range queries* while planning their travel schedule. For example, the first query could be:

```
SELECT * FROM FLIGHTS
WHERE DepartureTime BE-
TWEEN 10.00 A.M. AND 11.00 A.M. AND
DepartureDate BETWEEN 09-11-2003 AND 09-
12-2003 AND
Origin = "LAX" AND
Destination = "JFK" AND Class = "ECONOMY".
```

and if the result for this query proves to be unsatisfactory, it is likely to be followed by

```
SELECT * FROM FLIGHTS
WHERE DepartureTime BE-
TWEEN 08.00 A.M. AND 12.00 A.M. AND
```

```
DepartureDate BETWEEN 09-11-2003 AND 09-
13-2003 AND
Origin = "LAX" AND
Destination = "JFK" AND Class = "ECONOMY".
```

and so on, until a satisfactory result set is obtained.

Such trial-and-error queries are undesirable from the perspective of both the user and the database server. For the server, it results in a drastic reduction in effective throughput since much of the processing is duplicated in each successive query. For the client, it results in a marked increase in response times, as well as frustration from having to submit the query repeatedly. The problem is compounded for users who access the Web service through a handheld device (PDA, smart-phone, etc.) due to the high access latencies, cumbersome input mechanisms, and limited power supply.

Too Few Answers

A primary reason for the user dissatisfaction that results in repetitive queries is the *cardinality* of the answer set – the Web service may return *no* or insufficiently few answers, and worse, give no indication of how to alter the query to provide the desired number of answers [2]. (The complementary problem of "too many answers" has been previously addressed in the literature – see, for example [4, 5].)

Two approaches, both implemented on the *client-side*, have been proposed for the "too few answers" problem: The 64K Inc.[6] engine augments query results (if any) with statistical information about the underlying data distribution. Users are expected to utilize this information to rephrase their queries appropriately. However, it is unrealistic to expect that naïve Web users will be able (or willing) to perform the calculations necessary to rephrase their queries.

An alternative approach was proposed in Eureka [2]. In response to the initial user query, Eureka caches the relevant portion of the *database* at the client machine, allowing follow-up exploratory queries to be answered locally. A major drawback is that the user needs to install a customized software for each of the Web services that she wishes to access. In addition, this strategy may not be feasible for resource-constrained client devices which may be unable to host the entire database segment, or which are connected through a low-bandwidth network.

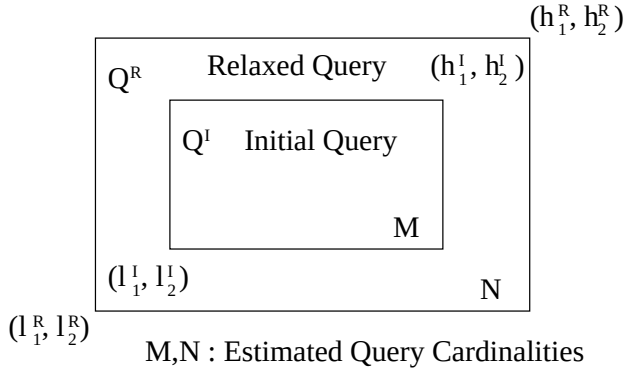


Figure 1: Range query relaxation in 2 dimensions

Finally, yet another possibility is to convert the user's range query into a point query (e.g., by replacing the box represented by the query with its centerpoint) and then to use one of the several Top-K algorithms available in the literature (e.g., [7]) with respect to this point. However, this approach is unacceptable since it runs the risk of not providing all the results that are part of the original user query. [1] Further, as discussed later in this paper, closeness to a point may not be equivalent to closeness to the query box.

The SAUNA Technique

In this paper, we propose SAUNA (Stretch A User query to get N Answers), a *server-side* solution for efficiently supporting exploratory queries. More formally, given an initial user query Q^I (which we expect to return M answers), and given the desired number of answers N, if $N > M$, SAUNA derives a new relaxed query Q^R which *contains* Q^I and is expected to have N answers. A pictorial representation of a SAUNA relaxation is shown in Figure 1 for a two-dimensional range query.

Note that a variety of relaxed queries, which may even be infinite in number, could be derived that obey

[1] For an unevenly shaped user query, the evenly shaped relaxed query box obtained by point-query relaxation may not enclose the original query.

the above constraints. In this solution space, SAUNA aims to deliver a relaxed query that (a) minimizes the distance of the additional answers with respect to the original query, that is, it aims to derive the closest $N - M$ answers, and (b) minimizes the data processing required to produce this set of answers. The first goal is predicated on defining a distance metric for points lying outside the original query – this issue is well understood for *point-queries* [7] but not for the *range* (or *box*) queries that we consider here. Therefore, SAUNA incorporates a box-query-specific distance metric that is suitably weighted to produce relaxed queries of a desired shape (e.g., aspect-ratio preserving with respect to the original query). To achieve the second goal, SAUNA utilizes multi-dimensional histograms as the tool for query size estimation. Histograms [8, 9, 10] are the de facto standard technique for maintaining statistical summaries in current database systems, and therefore our system is easily portable to these platforms. While uni-dimensional histograms are currently the norm, techniques for easily building and maintaining their multi-dimensional counterparts have recently appeared in the literature [11].

As we show in Section 6, a detailed performance evaluation of SAUNA over a variety of real and synthetic multi-dimensional data sets stored on a Microsoft SQL Server 2000 engine indicates that its relaxed queries can significantly reduce the costs associated with exploratory query processing, and in fact, often compare favorably with the optimal-sized relaxed query (obtained through off-line processing). Further, these improvements are obtained even when the memory budget for storing statistical information is extremely limited.

Organization

The remainder of this paper is organized as follows: The relaxation problem is formally defined in Section 2. Distance metrics for box queries are discussed in Section 3. The SAUNA query relaxation strategy is presented in Section 4. The wavelets technique of query size estimation is explained in Section 5. The performance model and the experimental results are highlighted in Section 6. Related work on query relaxation is reviewed in Section 7. Finally, in Section 8, we summarize the conclusions of our study and outline future research avenues.

2 Problem Definition

We assume that the data space is characterized by D dimensions and that the corresponding attribute set is $\{X_1, X_2, \ldots, X_D\}$. The domain of each attribute X_i may be either continuous, discrete, or categorical, and each domain has minimum value X_i^{min} and maximum value X_i^{max} (an ordering is imposed on categorical attributes as discussed later in Section 4.5). We assume that all domains are normalized to the range [0,1].

The initial query posed by the user is a D-dimensional hyper-rectangle defined by $Q^I = \{[l_1^I, h_1^I], [l_2^I, h_2^I], \ldots, [l_D^I, h_D^I]\}$ where each l_i^I and h_i^I denote the lower and upper limit of the query along the ith dimension (see Figure 1). That is, $X_i^{min} \leq l_i^I \leq h_i^I \leq X_i^{max}, \forall i\ 1 \leq i \leq D$. Here, some attributes will have *ranges* (i.e., $l_i^I < h_i^I$), some will be *points* (i.e., $l_i^I = h_i^I$), and some will be *don't-cares* (i.e., $l_i^I = X_i^{min}, h_i^I = X_i^{max}$). We assume that the user specifies the attributes that are *fixed* in that they should not be relaxed. In the absence of this information, for point attributes we introduce a small range variation, to avoid divide by zero errors that will arise with the *Aspect* and Inverse distance metrics explained in Section 3.

The relaxed query is denoted by $Q^R = \{[l_1^R, h_1^R], [l_2^R, h_2^R], \ldots, [l_D^R, h_D^R]\}$, with $Q^I \subseteq Q^R$ and $X_i^{min} \leq l_i^R \leq l_i^I$ and $X_i^{max} \geq h_i^R \geq h_i^I$, $\forall i\ 1 \leq i \leq D$. The differences $r_{il} = l_i^I - l_i^R$ and $r_{ih} = h_i^R - h_i^I$ $(r_{il}, r_{ih} \geq 0)$ are used to denote the relaxations w.r.t. the lower and upper limits of the original query along the ith dimension.

We assume that the user also provides N, the desired cardinality of the answer set. The estimated cardinalities of the original and relaxed queries are denoted by $M = \mid Q^I \mid$ and $N' = \mid Q^R \mid$, respectively. Relaxation is invoked only if $M < N$, and the goal of the relaxation system is to produce a relaxed query such that (a) $N' \geq N$, (b) $N' - N$ is minimized, (c) the additional $N - M$ answers returned to the user are the *closest neighbors* of Q^I, and (d) the data processing required to produce these additional answers is minimized. The

definition of closest neighbors is made precise in the next section.

Note that even in the absence of a definitive specification of N from the user, there may be some *default values* that could be effectively used by the system. It has been observed in user studies that a compact representation of results on fewer screens, and that reduce the need to scroll are more effective [12]. This implies that a good rule of thumb is to display a page-full of answers. In this paper, following the approach used in search engines such as Google, (`www.google.com`) we use 10 as the *ideal* target number of results. Note that this value can be easily changed in the framework to adapt to the *access characteristics* of diverse devices, such as PDAs and smartphones, whose displays typically have 10 to 12 lines at a low resolution.

3 Distance Metrics for Box Queries

Most distance functions used in practice are based on the general theory of *vector p-norms* [13], with $1 \leq p \leq \infty$. For example, $p = 2$ gives the classical Euclidean metric, $p = 1$ represents the Manhattan metric, and $p = \infty$ results in the Max metric. In the remainder of this paper, for ease of exposition, we assume that all distances, along each dimension, are measured with the Euclidean metric. Note, however, that the SAUNA relaxation algorithm can be easily adapted to any of the alternative metrics.

3.1 Reference Points

When computing the distances of database tuples with respect to *point* queries, it is clear that the distances are always to be measured (whatever be the metric) between the pair of points represented by the database tuple and the point query. However, when we come to *box* (range) queries, which is the focus of this paper, the issue is not so clear-cut since it is not obvious as to which point in the box should be treated as the reference point. In fact, it is even possible to think of distances being measured with respect to *a set* of reference points.

One obvious solution is to take some point inside the box (e.g., the center), treat the box as being represented by this point, and then resort to the traditional distance measurement techniques. However, this formulation appears highly unsatisfactory since the spatial structure of the box, which is representative of the user intentions, is completely ignored. Instead, we contend here that the user's specification of a box query implies that she would prefer answers that are *close to the periphery* of the box. To motivate this, consider the example situation shown in Figure 2, where point P is farther from the box center than point Q i.e., $r_2 > r_4$, but P's distance from the closest face of the box is smaller than the corresponding distance for Q i.e., $r_1 < r_3$. In this situation, we expect the user to prefer point P over Q since there is less deviation with respect to the complete box.

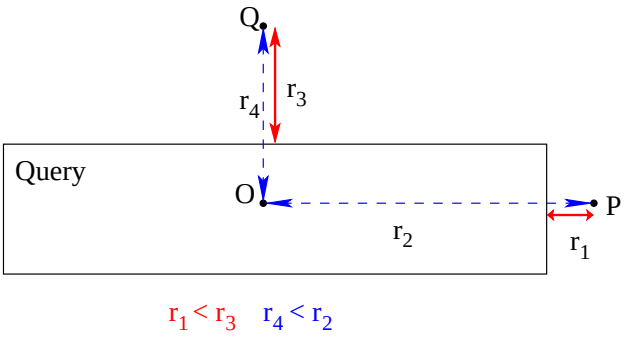


Figure 2: Measuring distance from periphery. P is closer to periphery than Q

The above observation can be formally captured by the following reference point assignment technique: For measuring the distance between a point P and a query box B, the reference point on B is the point of intersection of the perpendicular line drawn from P to the nearest face or corner of the box B.

We could, of course, have devised more complex reference point assignments – for example, compute the average of the distance between P and all corners of the box B, with the box corners operating as a universal set of reference points. However, we expect that the above simple formulation may be sufficient to express the expectation of a significant fraction of users of Web services, and further, more complex assignments can be directly accommodated, if required, in the SAUNA relaxation algorithm.

In summary, given a point $P = \{p_1, p_2, ..., p_D\}$ and a box-query B with lower and upper limits $l_i(B)$ and $h_i(B)$ respectively, we denote the component of distance on the i-th dimension as

$$\begin{aligned} d_i(P,B) &= p_i - h_i(B) \quad if \;\; p_i > h_i(B) \\ &= l_i(B) - p_i \quad if \;\; p_i < l_i(B) \\ &= 0 \qquad\qquad otherwise \end{aligned}$$

and the overall (Euclidean) distance between P and B as

$$dist(P,B) = \sqrt{\sum_{i=1}^{D} (d_i(P,B))^2} \tag{1}$$

Note that with this formulation, all points that lie *within* or *on* the box have an associated distance of zero.

3.2 Attribute Weighting

An implicit assumption in the above discussion was that relaxation on all dimensions was equivalent. However, it is quite likely that the user finds relaxation on some attributes more desirable than on others. For example, a business traveler may be time-conscious as compared to price, whereas a vacationer may have the opposite disposition. Therefore, we need to *weight* the distance on each dimension appropriately. That is, we modify Equation 1 to

$$dist(P,B) = \sqrt{\sum_{i=1}^{D} (d_i(P,B) * w_i)^2} \tag{2}$$

where $w_i,\ w_i \geq 0$ is the weight assigned to dimension i.

One option certainly is to explicitly acquire these weights from the user, and use them in the above equation. However, as a default in the absence of these inputs, we can resort to the following: *Use the box shape as an indicator of the user's intentions*. Specifically, we can assume that the user is willing to accept a relaxation on each range dimension that is *proportional* to the range size in that dimension, i.e., the user would prefer what we term as an *Aspect-Ratio-Preserving* relaxation. This metric preserves the aspect ratio of user-supplied query hence the name. This objective can be easily implemented by setting

$$w_i^{aspect} = \frac{1}{Asp\_ratio(i)} = \frac{Max_{i=1}^{D}(h_i(B) - l_i(B))}{h_i(B) - l_i(B)} \tag{3}$$

An alternative interpretation of the user's box-query structure could be that attributes should be relaxed in *inverse* proportion to their range sizes, since the user has already *built-in* relaxation into the larger ranges of her query. This can be implemented with the following distance function

$$w_i^{inverse} = Asp\_ratio(i) = \frac{h_i(B) - l_i(B)}{Max_{i=1}^{D}(h_i(B) - l_i(B))}$$

It should be noted that the notion of measuring distance from periphery as opposed to from the center holds even for this distance function. Figure 3 shows an example of the relaxed queries produced by using the *Aspect* and *Inverse* metrics, respectively. Given a constant k and relaxation units a and b (in the x and y axes, respectively), we see in these figures that the locus of points equidistant from the original query is not hyper-rectangular in the corners. Since relational databases can execute only hyper-rectangular queries, we approximate the relaxed queries by their *Minimum Bounding (Hyper)-Rectangles* (*MBR*s). We refer to the area enclosed within the locus as the *core* region and the area between the *core* region and the MBR rectangle as the *extended region*.

If our goal is to produce the *closest set* of answers to the query box, then we need to explicitly prune the extended region points. This is because there may be a point lying just outside the relaxed query box, whose distance is less than that of one of the points from the extended region. We term this as a *distance-preserving* relaxation. However, if minor deviations from the optimal set of answers is acceptable, then we can settle for a *box-preserving* relaxation instead, wherein answers from the extended region are also included in the answer set. Our experimental results indicate little performance difference for these alternative relaxations – therefore, we assume a box-preserving relaxation in the remainder of this paper.

As a final point, note that if the user has specified a point query as opposed to a box query, then the

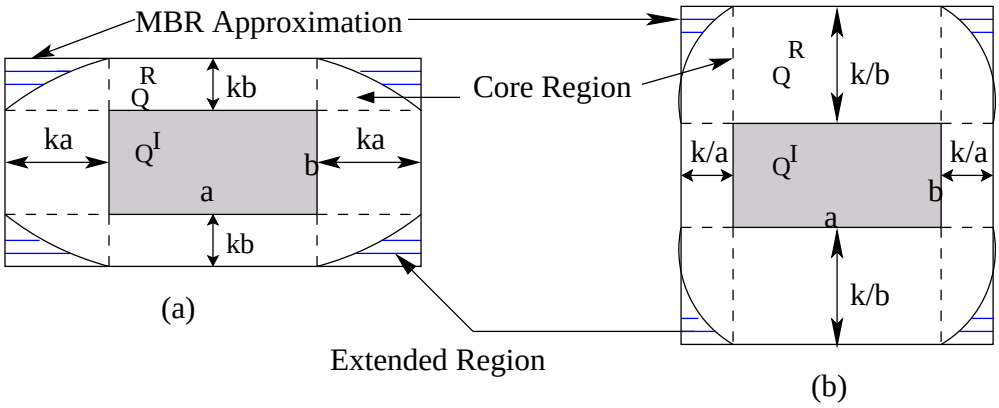


Figure 3: Distance Metrics and Relaxation regions: (a) Aspect (b) Inverse

above formulation degenerates to a traditional Top-N query [7], where the goal is to find the nearest N neighbors to the query point.

4 The SAUNA Relaxation Algorithm

We propose SAUNA, a simple query relaxation mechanism that attempts to ensure the desired cardinality and quality of answers while simultaneously trying to reduce the cost of relaxed query execution. Specifically, our algorithm generalizes to box queries the approach taken for point queries in [7, 14].

Our relaxation strategy leverages histograms for query size estimation. Histograms are the de facto standard technique for maintaining statistical summaries in current database systems, and therefore SAUNA is easily portable to these platforms. In particular, we use multi-dimensional histograms for the experiments reported in this study. These include multi-dimensional equidepth histograms and EQUIDWAV hybrid histogram which will be covered in Section 5. Although multi-dimensional histograms have been touted as being resource-intensive to create and maintain, recent work [11] has addressed this problem by proposing an online adaptive mechanism for easily building and maintaining multi-dimensional histograms, the so-called self-tuning histograms.

Due to their summary nature, histograms can provide only estimates, and not the exact values. Therefore, when relaxing a query to produce N answers, there is always a risk of either under-estimation or over-estimation of the cardinality of the answer set. While under-estimation results in inefficiency due to accessing more database tuples than necessary, over-estimation requires the query to be relaxed further and submitted again – a *restart* in the terminology of [14].

Estimation strategies possible in this environment include a conservative approach that completely eliminates restarts at the risk of getting many more tuples than necessary, and an optimistic approach that trades restarts for improved efficiency. These *No-Restarts* and *Restarts* approaches were implemented in [14] by assuming that all database tuples in a histogram bucket are at the maximum or minimum distance, respectively, with respect to the point query. Note that for a point query, there is always a unique location on a histogram bucket which is at a minimum (maximum) distance from the point query. However, when we consider box-queries in conjunction with the periphery-based distance metric described in the previous section, there is a *set* of points on the histogram bucket that are all at the same minimum (maximum) distance from the box query. In Figure 4, we present the MinDist and MaxDist algorithms to find these minimum and maximum distances, respectively. Both these algorithms are linear in the number of query attribute dimensions. We describe be-

low the various relaxation strategies for box queries that are based on these distance computations.

4.1 Box-Restarts Strategy

In this approach, all tuples inside a histogram bucket are assumed to be present on a locus of *minimum* distance from the query box. Since both the query box and the histogram bucket are D-dimensional yper-rectangles, the minimum distance between them is the minimum distance between any pair of their $D-1$ dimensional hyper-rectangle surfaces. We use the *MinDist* algorithm (Figure 4(a)) to compute this minimum distance. MinDist locates one of the points at minimum distance on the bucket and then computes the distance of that point from the query box. In the algorithm, b_i^l and b_i^h are the lower and upper bounds of the bucket in the *i*-th dimension, while q_i^l and q_i^h are the corresponding lower and upper bounds of the box query. It should be noted here that the identification of the nearest point in the *MinDist* algorithm is *independent* of the specific distance metric (including attribute weighting) chosen for computing the minimum distance.

In the Box-Restarts relaxation strategy, we compute the minimum distances of all histogram buckets from the query box, and then sort these buckets in increasing order of these distances. We assume that relaxing the query up to the minimum distance of some bucket implies that the relaxed query includes all tuples in that bucket. Hence we choose the largest distance from the set of bucket distances such that the relaxed query is expected to contain N tuples. Since the underlying assumption that all points in a bucket are as close as possible to the query box is optimistic, the Box-Restarts strategy does not guarantee that the relaxed query will indeed return N tuples.

4.2 Box-NoRestarts Strategy

In this approach, all tuples inside a histogram bucket are assumed to be present on a locus of *maximum* distance from the query box. We use the *MaxDist* algorithm (Figure 4(b)) to compute this maximum distance. The process we follow for finding the Box-NoRestarts relaxation distance is the same as that for the Box-Restarts approach outlined above. Since the relaxed query is guaranteed to cover all the histogram buckets at a distance less than or equal to relaxation distance, the Box-NoRestarts strategy guarantees that the relaxed query shall return at least N answers. This guarantee is obtained at the cost of efficiency in that many more tuples than strictly necessary may have to be processed to find the desired answer set.

To make the above discussion concrete, sample points chosen by the MinDist and MaxDist algorithms are shown in Figures 5(a) and 5(b), respectively. In these figures, Q is the query box, b_1 through b_8 are the histogram buckets in the 2-dimensional space, and p_1 through p_8 are the points chosen by the algorithms. Note that while minimum distance points can be located on the query box itself (e.g., p_5 in Figure 5(a)), the maximum distance points always have to be on the corners of the histogram bucket (all p_i in Figure 5(b)).

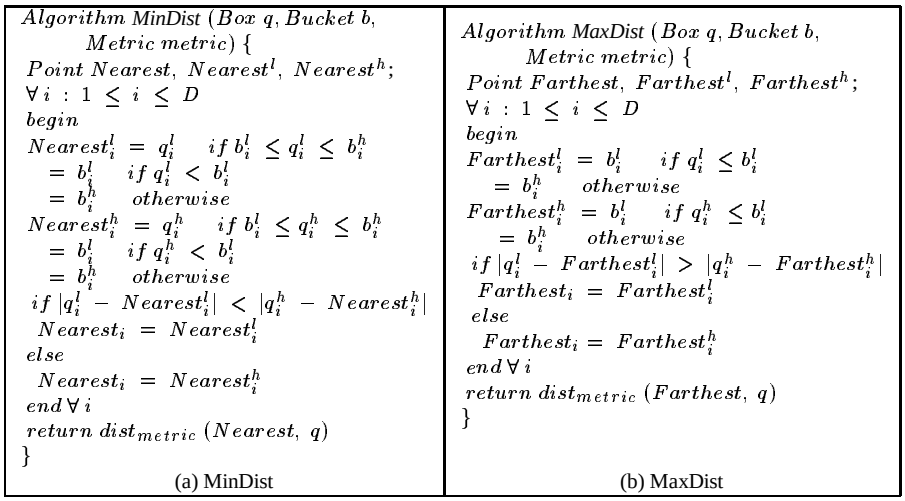
4.3 Box-Dynamic Strategy

Since Box-Restarts and Box-NoRestarts represent extreme solutions, an obvious question is whether an intermediate solution that provides the best of both worlds can be devised? For this, we adopt the dynamic workload-based mapping strategy of [7], which attempts to find the relaxation distance that minimizes the expected number of tuples retrieved for a set of queries while ensuring a reduced number of restarts. This is implemented as follows: Given α as a parameter such that

$$d_q(\alpha) = d_q^{BR} + \alpha \left(d_q^{BNR} - d_q^{BR} \right)$$

where d_q^{BR} and d_q^{BNR} are the *Box-Restarts* and *Box-NoRestarts* distances for query q, we need to find the value of $d_q(\alpha)$ that minimizes the average number of tuples retrieved for a given query workload. Since $d_q(\alpha)$ is a unidimensional function of α, the *golden search* algorithm [15] can be utilized to estimate this optimal value of α. Note that this approach requires an initial "training workload" to determine a suitable value of α, which can then be used in the subsequent "production workloads".

In the remainder of this paper, we present results only for the Box-Dynamic strategy since we found that it consistently outperformed the extreme strategies.



```
Algorithm MinDist (Box q, Bucket b,
        Metric metric) {
Point Nearest, Nearest^l, Nearest^h;
∀ i : 1 ≤ i ≤ D
begin
Nearest_i^l = q_i^l    if b_i^l ≤ q_i^l ≤ b_i^h
  = b_i^l    if q_i^l < b_i^l
  = b_i^h    otherwise
Nearest_i^h = q_i^h    if b_i^l ≤ q_i^h ≤ b_i^h
  = b_i^l    if q_i^h < b_i^l
  = b_i^h    otherwise
if |q_i^l − Nearest_i^l| < |q_i^h − Nearest_i^h|
 Nearest_i = Nearest_i^l
else
 Nearest_i = Nearest_i^h
end ∀ i
return dist_metric (Nearest, q)
}
```

(a) MinDist

```
Algorithm MaxDist (Box q, Bucket b,
        Metric metric) {
Point Farthest, Farthest^l, Farthest^h;
∀ i : 1 ≤ i ≤ D
begin
Farthest_i^l = b_i^l    if q_i^l ≤ b_i^l
  = b_i^h    otherwise
Farthest_i^h = b_i^l    if q_i^h ≤ b_i^l
  = b_i^h    otherwise
if |q_i^l − Farthest_i^l| > |q_i^h − Farthest_i^h|
 Farthest_i = Farthest_i^l
else
 Farthest_i = Farthest_i^h
end ∀ i
return dist_metric (Farthest, q)
}
```

(b) MaxDist

Figure 4: Algorithms for computing distances

4.4 Relaxation Algorithm

While the Box-Dynamic strategy does reduce the likelihood of restarts, it does not completely eliminate them. To ensure that we do not get into a situation where there are repeated restarts of a given query, we follow the strategy that if the Box-Dynamic strategy happens to fail for a particular query, then we immediately resort to the conservative Box-NoRestarts strategy – that is, all queries are relaxed with at most one restart. The complete set of steps of the SAUNA relaxation algorithm is shown in Figure 6.

In Section 5 ahead, we introduce a new relaxation function *relaxBoxWavelet*. The SAUNA relaxation algorithm requires replacement of *relaxBoxDynamic* by *relaxBoxWavelet* only to utilize the power of the new histogram type introduced.

4.5 Handling Categorical Attributes

An implicit assumption in the discussion so far was that all attributes are either continuous or discrete with inherent ordering among the values. In practice, however, some of the dimensions may be *categorical* in nature (e.g., color in an automobile database), without a natural ordering scheme. We now discuss how to integrate categorical attributes into our relaxation algorithm.

In the prior literature, we are aware of two techniques that address the problem of clustering in categorical spaces – the first approach is based on *similarity* [16] and the second is based on *summaries*[17]. While both techniques can be used in our framework to calculate distances, we restrict our attention to the former in this paper.

The similarity approach works as follows: Greater weight is given to "uncommon feature-value matches" in similarity computations. For example, consider a categorical attribute whose domain has two possible values, a and b. Let a occur more frequently than b in the dataset. Further, let i and j be tuples in the database that contain a, and let p and q be tuples that contain b. Then the pair p, q is considered to be more similar than the pair i, j, i.e., $Sim(p, q) > Sim(i, j)$; in essence, tuples that match on less frequent values are considered more similar.

Quantitatively, similarity values are normalized to the range [0,1]. The similarity is zero if two tuples have different values for the categorical attribute. If they have the same value v, then the similarity is computed as follows:

$$Sim(v) = 1 - \sum_{l \in MoreSim(v)} \frac{f_l(f_l - 1)}{n(n-1)}$$

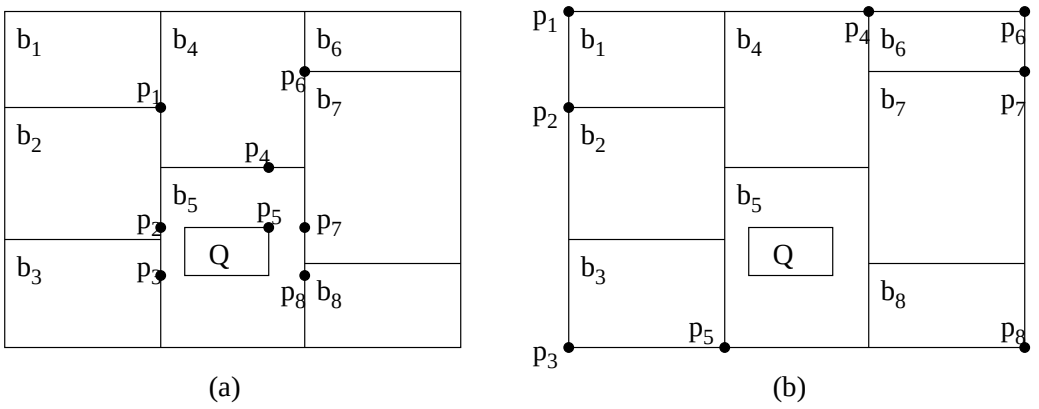


Figure 5: Box query relaxation strategies. (a) *Box-Restarts* (b) *Box-NoRestarts*.

where f_l is frequency of occurrence of value l, n is the number of tuples in the database, and MoreSim(v) is the set of all values in the categorical attribute domain that are more similar or equally similar as the value v (i.e., they have smaller frequencies).

We cannot directly use the above in our framework since our goal is to measure *distance*, not similarity. At first glance, the obvious choice might seem to be to set *distance* $= 1-$ *similarity*. But this has two problems: Firstly, tuples with different values in the categorical attribute will have a distance of 1. Secondly, tuples with identical values will have a non-zero distance. Both these contradict our basic intuition of distance.

Therefore, we set the definition of distance as follows: If two tuples have the same attribute value, then their distance is zero. Tuples with different values will have distances based on the frequencies of their attribute values. The more frequent the values, the less is the distance. For example, if the categorical attribute has values a, b and c in decreasing order of frequencies, $DIST(a,c) < DIST(b,c)$, since a is more frequent than b. In general, given tuples with values v_1 and v_2, we can quantitatively define

$$\begin{aligned} DIST(v_1, v_2) &= 1 - Sim(v_1) * Sim(v_2) \text{ if } v_1 \neq v_2 \\ &= 0 \quad otherwise \end{aligned}$$

5 Wavelet based histograms

The accuracy of the SAUNA relaxation algorithm depends on the accuracy of the query size estimation technique used. Histograms, particularly the equidepth histograms are the de facto choice in commercial databases. However a novel technique using wavelets was introduced in [18] for query size estimation and shown to be working better than traditional histogram types.We explain the technique in brief ahead and provide our comparative results for errors in recomputing the distribution made by wavelet histograms and equidepth histograms in Section 6.

5.1 Wavelets

Wavelets are mathematical tool for hierarchically decomposing functions. Wavelets represent a function in terms of a coarse overall shape, plus details that range from broad to narrow. The data distribution function can be represented using the wavelets. Wavelets offer an elegant technique for representing the various levels of detail of the function in a space-efficient manner.

```
Algorithm SAUNA Relaxation (Query Q^I, Integer N)
{
1  M = estimateCardinality(Q^I);
2  if M < N
3       Q^R = relaxBoxDynamic(Q^I);
4       numAnswers = execute(Q^R);
5       if numAnswers ≥ N return the N nearest answers;
6       else
7          Q^R' = relaxNoRestart(Q^I);
8          execute(Q^R');
9  else
10       numAnswers = execute(Q^I);
11       if numAnswers ≥ N return all answers;
12       else
13          M = numAnswers;
14          go to Step 7;
}
```

Figure 6: SAUNA relaxation algorithm

5.1.1 Wavelet Decomposition and Histogram Construction

The goal of wavelet decomposition step is to represent the frequency distribution at hierarchical levels of detail. Signal compression techniques employing wavelets can be used to reduce the space complexity of representing the underlying data distribution within a histogram bucket. For this purpose we need to choose an appropriate basis function. We chose Haar wavelets basis function, because they are the easiest to implement and fastest to compute. For detailed treatment on Haar wavelets we refer the reader to [18].

The wavelet decomposition step involves applying the wavelet transform i.e., wavelet decomposion on the cumulative frequency distribution of the data. The transform results in N wavelet coefficients, equal in number to number of values in the frequency distribution. For space efficiency reasons we store only a few of, say m, wavelet coefficients along with their positions in the wavelet transform. After wavelet transformation is done, most of the wavelet coefficients are either very small or zero in magnitude. The common practice is to store the m coefficients highest in magnitude.

5.1.2 Reconstruction of frequency distribution

To reconstruct the frequency distribution we take the m stored wavelet coefficients and reconstruct the wavelet transform by assuming other coefficients to be zero. An inverse wavelet transform on this set of N coefficients, then gives the approximate cumulative frequency distribution. The accuracy of this approximate distribution depends on the number of actual wavelet coefficients that were stored.

5.1.3 Query Size Estimation

Query size estimation using the approximated cumulative frequency distribution is straightforward now. For a range query $a \leq x \leq b$ on attribute X, the query size is estimated to be $f(b) - f(a)$, where $f(b)$ and $f(a)$ are the cumulative frequency counts upto b and a respectively.

5.1.4 Relaxation Algorithm

Use of wavelets as estimation technique demands for a different way of computing the relaxation distance which was earlier done using Box-Dynamic strategy. We suggest a simple binary search algorithm which tries to find out iteratively the relaxation distance that will impart desired selectivity to relaxed query. The algorithm *relaxBoxWavelet* is given ahead. It should be noted that the the overheads for this algorithm are not high, as all the computation is done in memory using few coefficients only. The lo and hi values are 0 and $domain\_size$ respectively because the wavelets histogram is equivalent to a single bucket equidepth histogram in which all the wavelet coefficients are stored. The performance of the algorithm can be further improved by search techniques like golden search, however it does not affect the accuracy of the results.

In the *Iterative Relaxation algorithm* the $ExpandQuery$ function expands query q by radius rad and using metric $metric$ to generate a relaxed query $\hat{q}$. $SizeEst$ function estimates the cardinality of query $\hat{q}$. Note that we need to replace the call to *relaxBoxDynamic* in the SAUNA algorithm with a call to *relaxBoxWavelet* function.

```
Algorithm relaxBoxWavelet (Box q, int NumAns,
                           Metric metric)
{
 int lo, hi, σ, rad;
 Box q̂;
 lo = 0;
 hi = domain_size;
 while (hi > lo)
 {
  rad = (lo + hi)/2;
  q̂ = ExpandQuery(q, rad, metric);
  σ = SizeEst(q̂);
  if (σ < NumAns)
      lo = rad;
  else if (σ > NumAns)
      hi = rad;
  else return q̂;
  }
 return q̂;
}
```

Figure 7: Iterative Relaxation algorithm

6 Experimental Results

6.1 Experimental Settings

We used a variety of synthetic and real-world data sets to evaluate SAUNA – these datasets are the same as those used in [7]. The real-world data sets consisted of the US census data set ($199,523$ tuples) and the Forest data set ($581,012$ tuples) obtained from [19]. We selected from these data sets the same set of attributes as [7]. The synthetic data consisted of the *Gauss* and *Array* data sets, each containing $500,000$ tuples. The *Gauss* data sets [15] were generated using predetermined number of overlapping multidimensional gaussian bells. Each bell was parameterized by the variance and zipfian parameter. The *Array* data sets were generated using zipfian distribution [20] for frequency of data values along each attribute. The value sets of each attribute were generated independently. The values of zipfian parameter for both these data sets were chosen to be $0.5, 1, 1.5$ and 2.

All the experiments were performed using multidimensional equidepth histograms [8], as they are both accurate and simple to implement. Further, an N-dimensional unclustered concatenated-key B^+-tree multidimensional index covering all the query attributes was built over each data set.

The query workload consists of queries with the number of range dimensions varying from 2 to 4, which is typical of many Web applications. The specific queries were generated by moving a query template over the entire domain space, returning a set of 100 queries. This query density was sufficient to ensure that most queries suffered from the problem of too few answers and therefore required relaxation. All results we report are averages for this set of hundred queries.

Besides different datasets, we also evaluated the performance of SAUNA with respect to (a) varying the number of buckets in the histogram; (b) varying N, the desired result cardinality; (c) varying the skew in the data; and, (d) varying the distance metric. To serve as comparative yardsticks for SAUNA's performance, we used two benchmarks:

Sequential (SEQ) : In this strategy, a sequential scan of the database is made in order to produce a sorted list of the tuples w.r.t. their distance from the query box, after which the top N tuples are returned.

Optimal (OPT) : This strategy refers to a hypothetical optimal relaxation strategy which produces the *minimally relaxed query* that contains the desired answer set. Note that the minimum bounding hyper-rectangle enclosing the N nearest tuples of a query box is not guaranteed to return N answers only and often returns more than N answers. Further, it is not possible for any relaxation technique, without observing the actual data tuples, to retrieve tuples equal to OPT tuples. In our experiments, the answers for OPT were found through an offline complete scan of all the data tuples.

The terminology used in the following experimental descriptions is explained in Table 1. For all the results, unless otherwise mentioned, the default settings were zipfian parameter $z = 1$, number of dimensions $= 3$, number of desired answers $N = 10$, *Aspect* distance metric and number of histogram buckets $= 256$. Finally, the *Box-dynamic* strategy (see Section 4.3) is used for SAUNA relaxation in all the experiments presented here. Our experiments were conducted on a Pentium IV machine running the Windows 2000 operating system.

Term	Meaning
Dim	No. of Dimensions
Strat	Relaxation strategy
cen	census dataset
cov	cover dataset
arr	Array dataset
opt	Optimal relaxation strategy
B-dyn	Box-dynamic strategy

Table 1: Terminology

6.2 Experiment 1: Basic SAUNA performance

The performance of SAUNA and OPT on the various datasets for the default parameter settings is shown in Figure 8 with respect to the number of tuples retrieved (note that the Y-axis is shown on a *log scale*). The first point to observe here is that for all the datasets, SAUNA requires processing less than 4% of the tuples – in fact, for the *census* and *array* datasets they are less than 1%. Secondly, note that there is quite a substantial difference between the optimal performance and that of SAUNA. This is due to the fact that SAUNA has to depend on statistical information that is limited by a tight memory budget (only 256 histogram buckets, consuming around 5KB memory, were used in this experiment). It can be seen that the *gz* and *cover* datasets consistently perform worse than the other datasets, but for *array* and *census* datasets the performance is closer to optimal. We attribute this to the dense and clustered nature of the *gz* and *cover* datasets which results in retrieval of too many tuples even from a small space. Again this is largely dependent on the quality of histograms available.

In Figure 9, we show the running times of SAUNA and OPT strategy(excluding the time required to find the optimal relaxed query), normalized to the execution time of SEQ, for the various datasets. The first point to note here is that the SAUNA execution times are below 10% of the sequential scan time for all the datasets. Secondly, for the *census* and *array* datasets the SAUNA times are close to that of OPT, and even for the other datasets the difference is not much. The number of query restarts were found to be negligible for *census* and *array* datasets. The *gz* and *cover* datasets showed

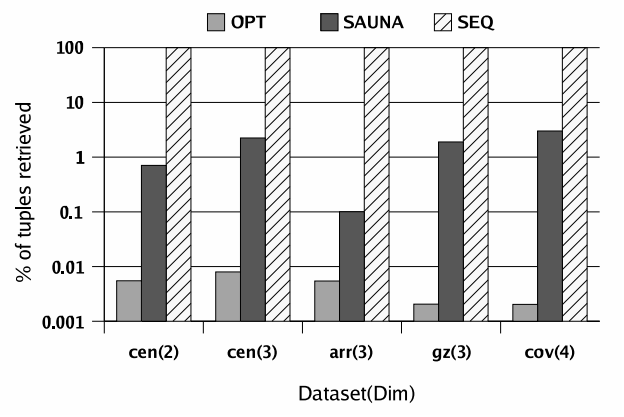


Figure 8: Percentage of tuples retrieved

around 10% query restarts. The restarts were particularly observed for the queries nearer to the void regions of the datasets which are very small in other datasets.

The execution time figures clearly indicate the efficiency of SAUNA w.r.t. the optimal strategy. Again, it should be noted that it is not the relaxation algorithm, but the quality of the histograms (the type and number of buckets) that affect the efficiency of SAUNA as compared to the optimal in terms of number of tuples retrieved or the execution time. By increasing histogram sizes we expect that SAUNA would perform closer to the optimal.

6.3 Experiment 2: Varying Number of Histogram Buckets

In this experiment, we investigated the performance improvements that could be obtained if our tight memory budget for statistical information was somewhat relaxed. In particular, we varied the memory budget from the default 5 KB to about 100 KB.

The results of this experiment are shown in Figure 10. It can be clearly seen here that the number of tuples retrieved decreases steadily with increasing number of buckets. The decrease in number of tuples retrieved is almost linear with the histogram sizes and at small size of 40KB the number of tuples are well within 0.5% of total number of tuples for all datasets except *cover*. This supports our claim that SAUNA is limited

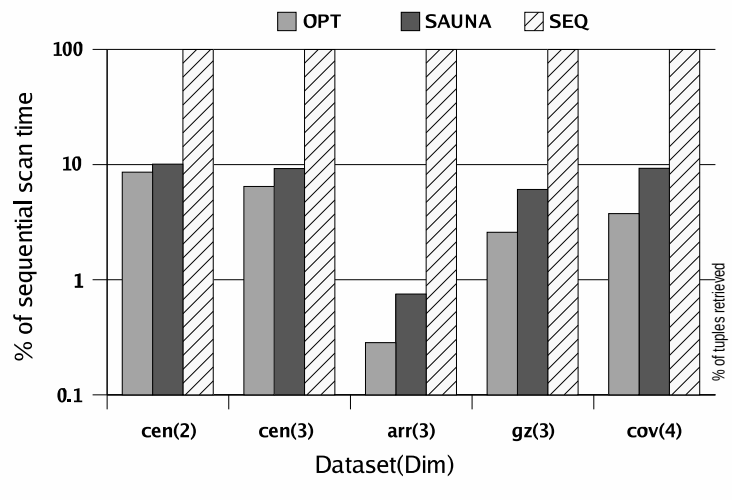


Figure 9: Execution time of SAUNA relative to SEQ

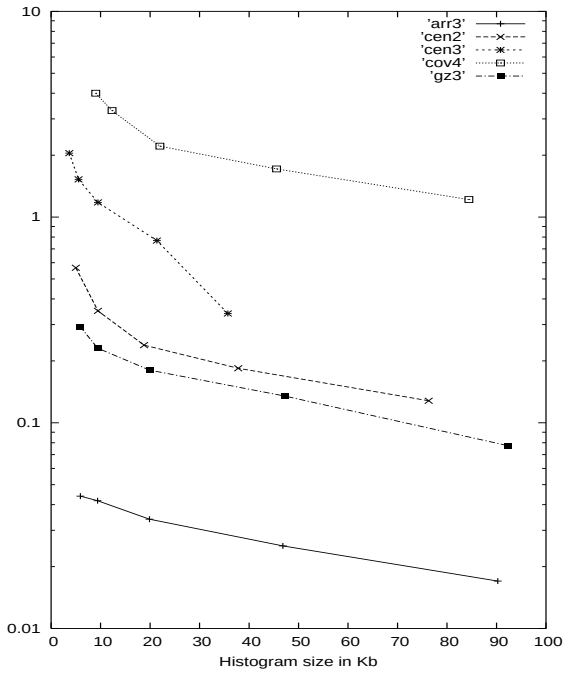


Figure 10: Varying Histogram Sizes

by the quality of histogram statistics only.

6.4 Experiment 3: Varying N

We now move on to evaluating the effect of the choice of N, the desired answer cardinality, on the performance of SAUNA. The performance for values of $N = 10, 50, 250$ is shown in Figure 11. We see here that, in most cases, the cost does not increase considerably with increasing values of N. This is because as N increases, the effective accuracy of the histogram becomes better and better, and therefore there is lesser wasted effort. It can also be observed that the ratio of tuples retrieved by SAUNA versus optimal tuples decreases by almost one order of magnitude with each increased values of N. Thus in environments where higher number of answers are expected (e.g. in a banking application where the manager wants to see a list of 250 customers with balance more than $\$100,000$.) we expect SAUNA to perform even better.

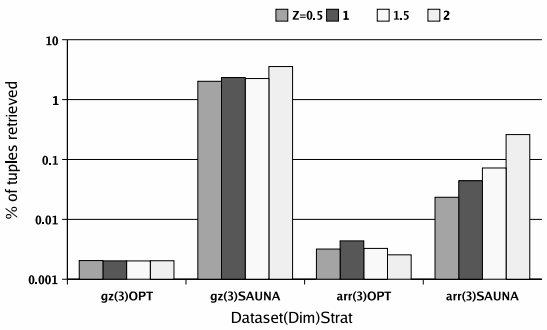


Figure 12: Percentage of tuples retrieved:Varying Skew

6.5 Experiment 4: Varying the data skew

In our next experiment, we considered the effect of varying the skew in the dataset contents. The results of this experiment are shown in Figure 12, and we observe here that, in most cases, the number of tuples retrieved is relatively robust with regard to the skew. Further, note that even with heavy skew (z = 2), the absolute number of tuples retrieved is well below 3% of the total number of tuples.

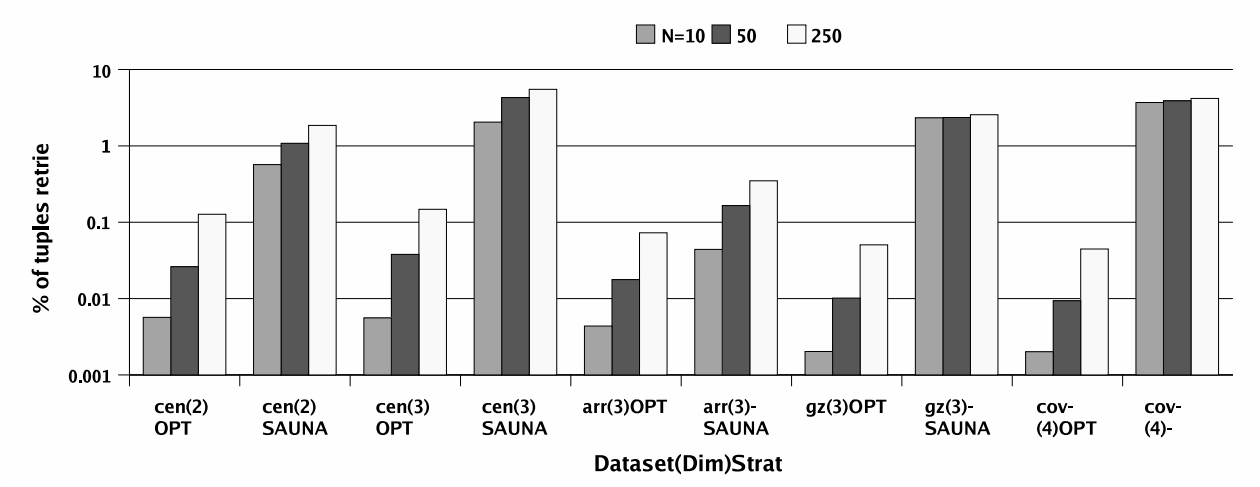


Figure 11: Percentage of tuples retrieved: Varying N

6.6 Experiment 5: Varying the Distance Metric

In our last experiment, we report the effects of changing the distance metric to Inverse on SAUNA's performance. The results of this experiment are shown in Figure 13. We see here that the performance characteristics are very similar to those of the Aspect metric (Figure 8).

We observed similar behavior for our experiments with other vector p-norm distance metrics also – the details are omitted here due to space limitations.

Overall, the above experiments show that SAUNA, despite being constrained by the limited memory resources, robustly and efficiently provides automated query relaxation. When more memory is provided, the performance improves accordingly.

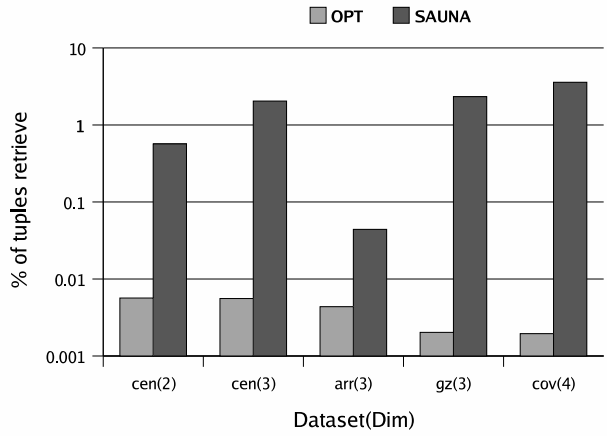


Figure 13: Tuples retrieved for Inverse metric

6.7 Comparison of wavelets and Histograms

For a fair comparison of the wavelets technique and histograms, alculated the number of wavelet coefficients and histogram buckets that could be accomodated in a given space and constructed both the kinds of histograms. We found the relative error made by both of them in approximating the cumulative frequency distribution of the data. We found out the error in the approximation of the original frequency distribution that each of the techniques made. Figure 14 shows the results of our study for single dimensional data. The errors clearly indicate that wavelet histogram outperforms equidepth histogram.

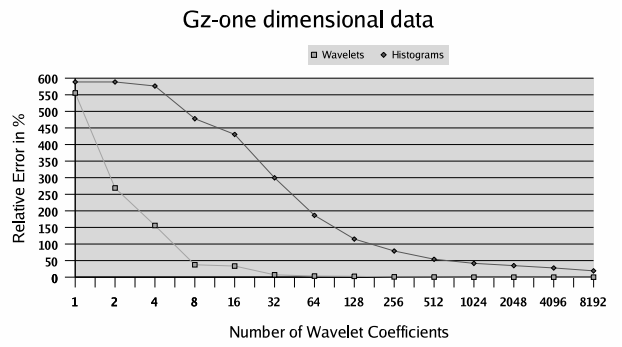


Figure 14: Wavelets Vs. Histograms

7 Related Work

The problems of dealing with *too many* and *too few* answers have been addressed in many different contexts. In the information retrieval literature, various techniques have been proposed to both relax and constrain keyword-based queries (see e.g., [21]). Many proposals for dealing with these problems for more structured queries can be found in the database literature [22, 23, 4, 5, 24, 14, 7].

Recently, significant attention has been devoted to the evaluation of Top-*N* queries. Top-*N* queries arise in many applications where users are willing to accept non-exact matches that are close to their specification. The answers to such queries consists of a ranked set of the *N* tuples in the database that best match the selection condition.

Chaudhuri et al [14] discuss the problem of evaluating Top-*N* equality selection queries that return too few answers. They propose distance metrics for equality selection queries and present histogram-based query relaxation strategies to automatically relax such queries and return the desired number of answers. They carry forward their work in [7], where they introduce a dynamic workload-aware strategy for processing Top-*N* equality queries. Their work differs from ours essentially in the type of queries they support – whereas their work is limited to equality selection queries, SAUNA supports the more general class of range queries. Chen and Ling [25] handle the same problem as [14], but using sampling as an estimation technique. They show that, unlike histograms, sampling is quite efficient and effective when the number of dimensions is large.

8 Conclusions

In this paper, we proposed SAUNA, a novel server-based framework for automated query relaxation that improves the efficiency and efficacy of query exploration over large and unknown data spaces. Unlike previous approaches that are limited to point queries, SAUNA is able to relax multi-dimensional range queries. Through the use of an intuitive range-query-specific distance metric, SAUNA returns high-quality answers that are *closest* to the user-specified query box. In addition, since histograms are used for query size estimation, the SAUNA framework can be easily integrated with commercial RDBMS that support histograms. We also showed how categorical attributes can be naturally integrated into this framework. We also proposed a novel estimation technique EQUIDWAV that combines simple equidepth histograms with wavelet histograms.

Our experimental results indicate that SAUNA significantly reduces the costs associated with exploratory query processing, and in fact, often compare favorably with the optimal-sized relaxed query (obtained through off-line processing). Further, these improvements are obtained even when the memory budget for storing statistical information is extremely limited. Specifically, we found that even with as low a memory budget as 5 KB, SAUNA was able to provide satisfactory relaxation retrieving less than 10% of the tuples in the database and taking less than 10% of the time taken by sequential scan. We also showed how it provides significant benefits of up to an order of magnitude in execution time as compared to user-driven manual relaxation.

There are two main directions we intend to pursue in future work:

- Since SAUNA relies on query cardinality estimations to perform relaxation, its effectiveness is highly dependent on the estimation mechanism. Although the current implementation uses multi-dimensional equidepth histograms, we would like to experiment with other strategies, e.g., [10, 9,

18].

- Currently, when a restart is required, relaxation is applied and the new relaxed query is executed. Note that this leads to redundant work, as all answers for previous query are derived again. For future work, we intend to investigate query splitting techniques (see e.g., [26]) to try and execute only the *difference* query.

References

[1] M. K. Bergman. The deep web: Surfacing hidden value (white paper). *Journal of Electronic Publishing*, 7(1), August 2001.

[2] John C. Shafer and Rakesh Agrawal. Continuous querying in database-centric web applications. *Computer Networks*, 33(1-6):519–531, 2000.

[3] Travelocity. `http://www.travelocity.com`.

[4] M. Carey and D. Kossmann. On saying "enough already!" in sql. In *Proc. of SIGMOD*, pages 219–230, 1997.

[5] M. Carey and D. Kossmann. Reducing the braking distance of an sql query engine. In *Proc. of VLDB*, pages 158–169, 1998.

[6] 64K Inc. DBGuide introduction and technology overview, 1997.

[7] Nicolas Bruno, Surajit Chaudhuri, and Luis Gravano. Top-k selection queries over relational databases: Mapping strategies and performance evaluation. *ACM TODS*, 27(2), 2002.

[8] M. Muralikrishna and David J. DeWitt. Equi-depth histograms for estimating selectivity factors for multi-dimensional queries. In *Proc. of SIGMOD*, pages 28–36, 1998.

[9] V. Poosala and Y. Ioannidis. Selectivity estimation without the attribute value independence assumption. In *Proc. of VLDB*, pages 486–495, 1997.

[10] V. Poosala, Y. Ioannidis, P. Haas, and E. Shekita. Improved histograms for selectivity estimation of range predicates. In *Proc. of SIGMOD*, pages 294–305, 1996.

[11] Ashraf Aboulnaga and Surajit Chaudhuri. Self tuning histograms: Building histograms without looking at data. In *Proc. of SIGMOD*, pages 181–192, 1999.

[12] B. Shneiderman, D. Byrd, and B. Croft. Clarifying search: A user interface framework for text searches. DLib Magazine, January 1997.

[13] I. Gradshteyn and I. Ryzhik. *Tables of Integrals, Series, and Products*. Academic Press, 2000.

[14] Surajit Chaudhuri and Luis Gravano. Evaluating top-k selection queries. In *Proc. of VLDB*, pages 397–410, 1999.

[15] William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling. *Numerical Recipes in C : The Art of Scientific Computing*. Cambridge University Press, 1993.

[16] Cen Li and Gautam Biswas. Unsupervised learning with mixed numeric and nominal data. *IEEE Trans. on Knowledge and Data Eng.*, 14(4):673–690, 2002.

[17] Venkatesh Ganti, Johannes Gehrke, and Raghu Ramakrishnan. Cactus - clustering categorical data using summaries. In *Proc. of KDD*, pages 73–83, 1999.

[18] Y. Matias, J.S. Vitter, and M. Wang. Wavelet-based histograms for selectivity estimation. In *Proc. of SIGMOD*, pages 448–459, 1998.

[19] UCI knowledge discovery in databases archive. `http://kdd.ics.uci.edu/summary.data.type.html`.

[20] G.K. Zipf. *Human behaviour and the principle of least effort*. Addison-Wesley, 1949.

[21] Ricardo A. Baeza-Yates and Berthier A. Ribeiro-Neto. *Modern Information Retrieval*. ACM Press/Addison-Wesley, 1999.

[22] T. Gaasterland, P. Godfrey, and J. Minker. Relaxation as a platform for cooperative answering. *Journal of Intelligent Information Systems*, 1(3-4):293–321, 1992.

[23] A. Motro. VAGUE: A user interface to relational databases that permits vague queries. *ACM Transactions on Office Information Systems*, 6(3):187–214, 1988.

[24] Ronald Fagin. Combining fuzzy information from multiple systems. In *Proc. of PODS*, pages 216–226, 1996.

[25] Chung-Min Chen and Yibei Ling. A sampling-based estimator for top-k query. In *Proc. of ICDE*, pages 617–627, 2002.

[26] K. Tan, C.H. Goh, and B.C. Ooi. On getting some answers quickly, and perhaps more later. In *Proc. of ICDE*, pages 32–39, 1999.