Optimizing Candidate Check Costs for Bitmap Indices

Doron Rotem, Kurt Stockinger, Kesheng Wu
Computational Research Division
Lawrence Berkeley National Laboratory
University of California
Berkeley, California, USA
{D_Rotem, KStockinger, KWu}@lbl.gov

ABSTRACT

In this paper, we propose a new strategy for optimizing the placement of bin boundaries to minimize the cost of query evaluation using bitmap indices with binning. For attributes with a large number of distinct values, often the most efficient index scheme is a bitmap index with binning. However, this type of index may not be able to fully resolve some user queries. To fully resolve these queries, one has to access parts of the original data to check whether certain candidate records actually satisfy the specified conditions. We call this procedure the *candidate check*, which usually dominates the total query processing time. Given a set of user queries, we seek to minimize the total time required to answer the queries by optimally placing the bin boundaries. We show that our dynamic programming based algorithm can efficiently determine the bin boundaries. We verify our analysis with some real user queries from the Sloan Digital Sky Survey. For queries that require significant amount of time to perform *candidate check*, using our optimal bin boundaries reduces the *candidate check* time by a factor of 2 and the total query processing time by 40%.

Categories and Subject Descriptors

E.2 [**Data**]: Data Storage Representations; H.3 [**Information Systems**]: Information Storage and Retrieval—*Indexing Methods*

General Terms

Algorithms Theory Experimentation Performance

Keywords

Bitmap Index, Query Processing, Dynamic Programming

1. INTRODUCTION

Bitmap indexing is a common technique for indexing high-dimensional data in data warehousing, OLAP and scientific applications. In these applications, it is common to have complex, multi-dimensional ad-hoc queries against read-only data. Bitmap indices are very efficient at supporting Boolean logic operations at the bit level to perform predicate evaluation at increased machine speeds. In addition, COUNT and SUM queries, whose responses are derivable through index scans without searching the database, can benefit considerably from this technology. For these reasons bitmap indices have been introduced into several commercial DBMS products by database vendors including Red Brick Systems, Sybase, IBM and Oracle. Although efficient for low cardinality attributes, query processing can be rather costly for high-cardinality attributes due to the large storage requirements for the bitmap indices. Typical approaches for reducing the storage complexity of bitmap indices are compression, bitmap encoding and binning. In this paper we focus on binning strategies.

Simple bitmap indices represent each distinct attribute value by one bitmap vector. Binning, on the other hand, partitions the attribute values into a small number of ranges, called bins, and uses bitmap vectors to represent bins (attribute ranges) rather than distinct values. Although binning typically reduces storage costs for high-cardinality attributes, it may increase the access costs of queries that do not fall on exact bin boundaries (edge bins). For this kind of queries the original data values associated with edge bins must be accessed, in order to check them against the query constraints.

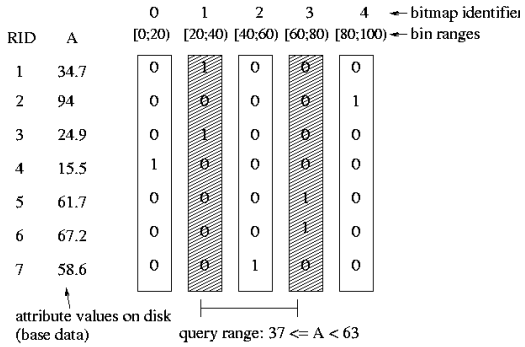

Figure 1: Range query $37 \leq A < 63$ on a bitmap index with binning.

A small example of the problem we are considering here is given in Figure 1. In this example we assume that an

CIKM'05, October 31–November 5, 2005, Bremen, Germany.

attribute A has values between 0 and 100. The values of the attribute A are given in the second leftmost column. The range of possible values of A is partitioned into five sub-ranges [0,20),[20,40) etc. with a bin allocated to each sub-range. A "1-bit" indicates the attribute value falls into the range, and "0-bit" otherwise. Assume that we want to evaluate the query "Count the number of rows where $37 \leq A < 63$". The correct result should be 2 (rows 5 and 7). We know that rows qualifying for this query have "1-bits" in the bitmaps corresponding to bins 1, 2, and 3. However, performing an OR operation on the bitmaps of bins 1, 2 and 3 will produce the bit vector "1010111" with a count of 5 "1-bits". The reason for this over-estimation is of course due to the fact that bins 1 and 3, also called edge bins, contain both qualifying and non-qualifying values. A correct response to this query involves checking the original attribute values corresponding to each of the four "1-bits" (rows 1,3, 5 and 6) in these two bins. Such a check may involve more additional accesses to disk pages depending on how the attribute values are stored. As we can see from this small example, the cost of performing a *candidate check* on an edge bin is related to the number of "1-bits" in that bin. In this example only one of the four records qualifies, namely, the value 61.7 in row 5. We call this additional step the *candidate check*.

The work presented in this paper is motivated by our bitmap indexing technology which is used in production for large-scale high-energy physics experiments [18]. We demonstrated that our technique significantly speeds up real interactive analysis processes. The data sets of these experiments are read-only and have similar characteristics to typical data warehouses. Due to the increased performance gain of our technology, we recently integrated the bitmap indices directly into the ROOT analysis framework [14] that has a user community of some 10,000 scientists around the world. Based on the experience we gained with the integration of our software and the feedback from the user community, we identified the *candidate check costs* as the main bottleneck of query processing. As we will show in the next section, an efficient way to tackle this issue is to optimally place the bin boundaries of the bitmap indices based on data distribution and query workloads (access patterns).

1.1 Factors Affecting Binning Strategies

Due to disk storage constraints, bitmap indexing systems that use binning must limit the number of bins that are allowed per attribute. Such constraints are still applicable even when bitmap compression is effectively used. Effective binning strategies attempt to compute bin boundaries that minimize the I/O cost incurred by the *candidate check* step subject to total index storage constraints.

It turns out that an optimal binning strategy must be sensitive to both *query distribution* as well as *data distribution*. *Query distribution*, in terms of location of query endpoints and popularity of queries, may affect bin boundary locations as the number of edge bins may be minimized by attempting to align bin boundaries with query endpoints. In addition, more bins can be allocated to data regions that are heavily hit by queries. *Data distribution* affects the binning strategy as one can allocate more bins to densely populated regions of the data to avoid costly *candidate check* operations on edge bins with many values.

Figures 2 and 3 illustrate, by a small example, the effect of data and query distribution on the optimal binning strategy. In both cases we show optimal binning into 12 bins (produced by our software) for an attribute using 100 simulated range queries. The horizontal segments represent generated range queries and the dashed vertical lines represent optimal bin boundaries computed by our software.

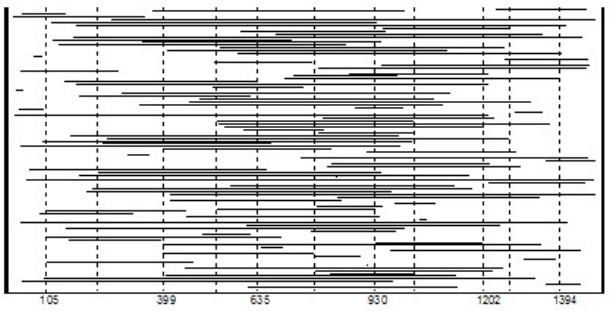

Figure 2: Uniformly distributed data with uniform range queries.

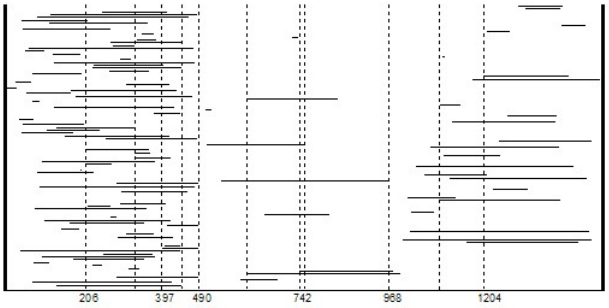

Figure 3: Normally distributed data with heavy left region queries.

In Figure 2 both data and queries are uniformly distributed where the values of the data fall in the range [0, 1500]. As expected, in this case we note that the bin widths are of approximately equal size and spread evenly over the entire range.

In Figure 3 the values of the attributes are normally distributed (truncated in the range [0, 1500]) with mean 750 and standard deviation 230. The generated query distribution is skewed in the following way. The region 0 to 1500 is divided into three equal subregions of size 500. Queries are generated over the three regions in the ratio of 6:1:2. We see that the region 0 to 500, which is heavily hit by queries, is allocated 5 of the 12 bins whereas the region 1000 to 1500 gets only 3 bins to account for this skewed query distribution (fewer queries falling into this region).

Although query and data distributions are not always known ahead of time, in many scientific applications, simulation and experimental data follow some known distributions and scientists are interested in specific regions of the data. This observation is also supported by our analysis of real application query logs that reveal very distinct patterns in the way scientists submit queries to an astrophysics database. As we will show in this paper, knowledge of data and query distribution, even if only approximate, can be used to sig-

nificantly improve the performance of the bitmap indexing system.

In this paper, we focus on optimizing the costs involved in the *candidate check*. Our experience with real application data shows that the I/O costs of this step dominate all other costs involved in answering a query such as scanning the bitmap index and performing the necessary Boolean operations. In fact, as shown later in Figure 6, the I/O costs for the *candidate check* can be significantly higher than the costs of the index scan.

1.2 Outline and Contributions

The main contributions of this paper are as follows:

- We propose a novel dynamic programming algorithm for optimal partitioning of attribute values into bins that takes into consideration range query access patterns as well as data distribution statistics. The algorithm also accounts for realistic I/O costs in terms of disk page accesses.
- We greatly reduce the complexity of the algorithm by proofing that only query endpoints need to be considered as potential locations for bin boundaries rather than all possible values of the attribute as in [8].
- We tested our algorithm on synthetic and real application data sets, based on real query workloads from a large astrophysics application. The tests are performed on a bitmap indexing system used in production for scientific experiments [18]. The results show that the optimal partitioning achieved by our algorithm leads to a significant improvement in the access costs of bitmap indexing systems for high-cardinality attributes.

The rest of the paper is organized as follows. Section 2 revises the related work on bitmap indices. Section 3 provides definitions and introduces a formal model for calculating optimal bin boundary locations for bitmap indices based on a dynamic programming approach. In Section 4 we evaluate our algorithm against synthetic and real queries with real data from a large astrophysics application. Section 5 concludes the paper and raises open issues for future work.

2. RELATED WORK

Bitmap indices are used for speeding up complex, multidimensional queries for On-Line Analytical Processing and data warehouses [5] as well as for scientific applications [15]. The first commercial implementation was Model 204 [10]. Improvements on this approach were discussed in [11]. In [4] three bitmap encoding strategies are introduced: *equality, range* and *interval* encoding.

The authors of [20] represented attribute values in binary form that yields indices with only $\lceil log_2 |A| \rceil$ bitmaps, where $|A|$ is the attribute cardinality. The advantage of this encoding scheme is that the storage overhead is even smaller than for *interval encoding*. However, in most cases query processing is more efficient with *interval encoding* since in the worst case only two bitmaps need to be read whereas with binary encoding always all bitmaps have to be read.

Various bitmap compression schemes were studied in [7, 1]. The authors demonstrated that the scheme named Byte-aligned Bitmap Code (BBC) [2] shows the best overall performance characteristics. More recently a new compression scheme called Word-Aligned Hybrid (WAH) [17] was introduced. This compression algorithm significantly reduces the overall query processing time compared to BBC. The key reason for the efficiency of WAH is that it uses a much simpler compression algorithm.

The bitmap indices discussed so far encode each distinct attribute value as one bitmap vector. This technique is very efficient for data values with low attribute cardinalities. However, scientific data is often based on floating point values with high attribute cardinalities. The work presented in [15] demonstrated that bitmap indices with binning can significantly speed up multi-dimensional queries on high-cardinality attributes.

A further bitmap index with binning, called *range-based* bitmap indexing, was introduced in [19]. The idea is to evenly distribute skewed attribute values onto various bins in order to achieve uniform search times for different queries. The authors demonstrated that the algorithm efficiently redistributes highly skewed data. However, performance results about query response times were not discussed.

The work in [8] focuses on point (equality) queries rather than range queries discussed in this paper. We extend the work of [8] by analyzing range queries. We reduce the problem complexity significantly by showing that only *query endpoints* need to be considered rather then all possible attribute values. In addition, it turns out that the range query problem cannot be translated directly into the point query problem of [8] by considering all possible query endpoints as point data. The reason for this is that the two endpoints of a range query form a single unit and are dependent on each other as far as candidate check costs are concerned. More specifically, if two endpoints belonging to the same query fall into a single edge bin, the candiate check procedure reads this bin's data values only once. If, on the other hand, the two endpoints belong to two different range queries, the values of edge bin will be read twice.

To the best of our knowledge, previous dynamic programming for optimal binning is based only on synthetic data and synthetic queries. We, however, evaluate our optimal binning algorithm on real data and real query workloads taken from the Sloan Digital Sky Survey [13, 16].

Bitmap indices could also be used to provide histogram information. The optimal construction of histograms for range queries also uses binning algorithms and is discussed in [9, 6]. The main difference is that for bitmap indices precise answers are required and therefore the objective is to minimize disk access costs to edge bins. However, in the histogram case, some statistical techniques can be used to estimate errors without actual access to the original data on disk.

3. OPTIMAL BINNING ALGORITHM

3.1 Preliminaries

In this section we will formulate the *OptBin* problem for attributes where the set of queries is known. Most of the common notation used through the paper is summarized in Table 1. Assume attribute A has N values that occupy P disk pages. For simplicity we will assume that each value is an integer in the domain range $[1, n]$. We are also given a collection of range queries Q such that each $q \in Q$ defines a range $q = [l_q, u_q)$ open on the right (i.e., it includes the points $l_q, l_q + 1, ..., u_q - 1$) and is as-

Notation	Explanation
N	Total number of attribute values
P	Total number of pages on disk for values of an attribute
$q = [l_q, u_q)$	A range query q with endpoints l_q and u_q, the range is open on the right
Q	A set of range queries
x_i	A bin boundary point
$b_i = [x_{i-1}, x_i)$	A bin defines a sub-range open on the right
n_i	Number of values in bin b_i
k	Constraint on number of bins
$B =< b_1, b_2, ..., b_k >$	A partitioning into k bins
$E(b)$	The set of queries having bin b as an edge bin
$Cost(Q, B)$	Candidate check I/O cost associated with binning B and query set Q
e_j	The j^{th} smallest query endpoint
$EP(Q) = \langle e_1, ..., e_r \rangle$	Ordered set of distinct query endpoints
r	Number of distinct query endpoints
$\Pi(k, n)$	All possible binnings of the range 1 to n into k bins
$[1, n]$	Range of possible attribute values
B_{opt}	Optimal binning
$R(Q, j)$	The set of queries in Q with a right endpoint on the right of e_j
$B_{opt}(e_j, l)$	Optimal binning of the sub-region from e_j to n using l bins
$b_{i,j}$	A bin defined over the range between query endpoints e_i and e_j, i.e., $b_{i,j} = [e_i, e_j)$

Table 1: Notation used throughout the paper.

sociated with a probability p_q reflecting its relative popularity. The points $l_q \in [1, n]$ and $u_q \in [2, n+1]$ are called endpoints of query q**.** A bitmap index on A is built by partitioning the range $[1, n]$ into bins. Each bin is represented by one bitmap (see Figure 1). In order to limit the size of the bitmap index, we introduce an integer constraint k that specifies the maximum number of bins allowed, i.e., it is required to partition the range $[1, n]$ into k successive sub-ranges (bins) $B =< b_1, b_2, ..., b_k >$. This is done by choosing $k-1$ integer bin boundary points x_i where $1 < x_1 < x_2 < ... < x_{k-1} < n+1$.

Note that there are $\binom{n-1}{k-1}$ possible ways of choosing the bin boundary points which makes it impractical to exhaustively check all possibilities.

The sub-ranges associated with bins b_i are all open on the right and defined as follows:

$$\begin{aligned} b_1 &= [1, x_1) \\ b_i &= [x_{i-1}, x_i) \qquad \text{for } 2 \le i \le k \\ b_k &= [x_{k-1}, n+1) \end{aligned}$$

A bin $b \in B$ is an edge bin for query q if the range defined by the query q overlaps some part of the range bin b but not its whole range, i.e., $q \cap b \neq \emptyset$ and $q \cap b \neq b$. In general, a query may have 0, 1, or 2 edge bins.

In Figure 4 a set of 10 range queries and a binning into 4 bins is shown. In this example query q_3 has no edge bins since both its endpoints fall on bin boundaries. Each of the queries q_4, q_5, q_6, q_7, q_{10} has 1 edge bin and each of the queries q_1, q_2, q_8, q_9 has 2 edge bins.

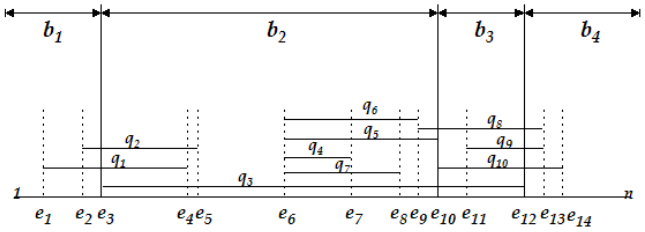

Figure 4: Query endpoints and bin boundaries. Horizontal lines represent query ranges. Dotted vertical lines mark query endpoints.

As explained earlier, a significant fraction of the query I/O costs are related to the number of data pages needed to read in order to perform the *candidate check* on each of its edge bins. For a given bin b, let $E(b)$ denote the set of queries that have bin b as an edge bin. For example, in Figure 4 $E(b_1) = \{q_1, q_2\}$; $E(b_2) = \{q_1, q_2, q_4, q_5, q_6, q_7, q_8\}$; $E(b_3) = \{q_9\}$; $E(b_4) = \{q_8, q_9, q_{10}\}$. Let n_b denote the number of data values that fall into the range defined by b, this is also the number of "1-bits" in the bitmap corresponding to b.

Our algorithm uses a formula to compute the number of disk pages that must be retrieved to check all n_b data values. Based on the usual assumption that records are distributed uniformly across pages and recalling that the total number of pages occupied by attribute A is P, the expected number of disk pages that contain data values that fall in the range defined by bin b denoted by P_b, satisfies [11]:

$$P_b = P(1 - (1 - 1/P)^{n_b}) \approx P(1 - e^{-n_b/P}) \qquad (1)$$

It is important to note that our algorithm's correctness does not depend on the above formula as it can take alternative expressions for P_b in case the *uniform* assumption about the distribution of records on pages does not hold. This often happens when attribute values exhibit some type of *physical clustering* across disk pages due to the way the data is entered into the system (see Section 4.2.3). Assuming that an attribute range is partitioned by the set of bins B, the expected *candidate check* cost $Cost(Q, B)$ of answering the queries in Q is defined as

$$Cost(Q, B) = \sum_{b \in B} P_b \sum_{q \in E(b)} p_q \qquad (2)$$

The inner sum computes the total probability of all the queries that use a given bin b as an edge bin, this is then multiplied by the I/O cost of the bin (expected number of pages) and summed over all bins.

3.2 Bin Boundaries and Query Endpoints

The problem we wish to solve, namely *OptBin*, is defined as follows:

Given a set of range queries Q on an attribute in the domain range [1,n], find the (k-1) bin boundary points that partition the attribute values into k bins such that the I/O cost for the candidate check is minimized.

More formally, let $\Pi(k, n)$ denote the set of all possible binnings of the range [1, n] into k bins. We wish to find a binning B_{opt} such that $B_{opt} \in \Pi(k, n)$ and

$$Cost(Q, B_{opt}) \leq Cost(Q, B) \text{ for all } B \in \Pi(k, n) \quad (3)$$

Let $EP(Q) = \langle e_1, ..., e_r \rangle$ denote the ordered set of distinct query endpoints of queries in Q, i.e., $e_i \in EP(Q)$ implies that for at least one query $q \in Q$, $e_i = l_q$ or $e_i = u_q$. For example, Figure 4 shows the distinct endpoints of 10 range queries.

The following lemma shows that an optimal solution exists such that each of its bin boundary points are in $EP(Q)$. In practice $|EP(Q)|$ is much smaller than n as only a few of the points in the range serve as query endpoints. We will use this result to speed up the dynamic programming solution presented in Section 3.3.

LEMMA 1. *If any binning $B \in \Pi(k, n)$ uses a boundary point x_i that is not in $EP(Q)$ then a binning of equal or smaller cost can be found by replacing x_i with some point in $EP(Q)$.*

The proof of this lemma can be found in [12].

3.3 Dynamic Programming Formulation

The previous lemma showed that the boundary points of an optimal binning are taken from the set of query endpoints $EP(Q)$. Assuming that the number of distinct query endpoints is much larger than the desired number of bins, i.e., $|EP(Q)| = r \gg k$, we present here a dynamic programming algorithm that chooses $k-1$ bin boundary points from the r elements of the set $EP(Q)$ in $O(kr^2)$ time and obtains a minimum cost binning.

Let us recall that the elements of $EP(Q)$ are sorted in increasing order, where e_j denotes the j^{th} smallest member of $EP(Q)$. For the purpose of describing the "principle of optimality" in our problem [3], we need to describe the cost of an optimal solution on sub-ranges of $[1, n]$. This requires looking at a subset of the queries in Q falling on the right of some potential boundary point. To this end we need some definitions. Let $R(Q, j)$ denote the queries in Q that have a right endpoint greater than e_j, formally, $R(Q, j) = \{q \in Q : u_q > e_j\}$. For two query endpoints, $e_i, e_j \in EP(Q)$, let $b_{i,j}$ represent a potential bin defined over the range $[e_i, e_j)$ (see Figure 5). In case this bin is eventually used in any binning B, its contribution to $Cost(Q, B)$ is $Cost(b_{i,j}) = P_{b_{i,j}} \sum_{q \in E(b_{i,j})} p_q$, where as before, $P_{b_{i,j}}$ denotes the total number of disk pages that hold values of the attribute falling in the range $[e_i, e_j)$ and $E(b_{i,j})$ denotes the set of queries having $b_{i,j}$ as a bin, i.e., overlapping the range defined by the bin but not containing it.

To simplify our notation we assume a dummy endpoint $e_0 = 1$ and let $B_{opt}(e_i, l)$ represent an optimal binning of the range $[e_i, n]$ using l bins defined over the queries in $R(Q, i)$ (see Figure 5). Using the above notation, our goal is to find $B_{opt}(e_0, k)$.

THEOREM 2. *Given a set of queries Q with an ordered set of distinct endpoints $EP(Q) =< e_0, e_1, e_2, .., e_r >$ consisting of r distinct endpoints then*

$$Cost(B_{opt}(e_i, l)) = \min_{i<j\leq r-(l-2)} (Cost(b_{i,j}) + Cost(B_{opt}(e_j, l-1))) \quad (4)$$

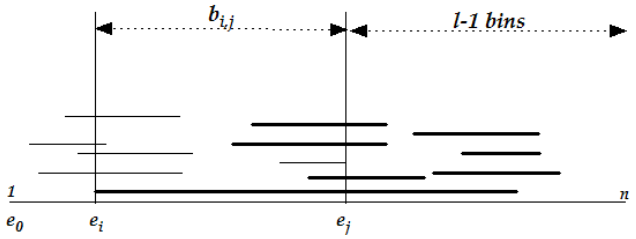

Figure 5: Dynamic programming, queries in $R(Q, j)$ are emphasized using thick lines.

Proof Assume an optimal binning, $B_{opt}(e_i, l)$ is given with all its boundary points in $EP(Q)$. The right boundary of the leftmost bin must be some $e_j \in EP(Q)$ such that $j > i$ and there are $l-1$ additional bins on its right (see Figure 5). Therefore e_j is a candidate only if j satisfies $i < j \leq r-(l-2)$. The total cost of this binning can be broken into the contribution of the first bin, $Cost(b_{i,j})$, and the contribution of the last $l-1$ bins. For any choice of e_j, the partition into $l-1$ bins on its right must be optimal with respect to the set of queries $R(Q, j)$ and therefore equal to $Cost(B_{opt}(e_j, l-1))$, otherwise $Cost(B_{opt}(e_i, l))$ can be reduced contradicting the optimality of $B_{opt}(e_i, l)$. □

Theorem 2 allows us to devise an efficient dynamic programming algorithm as it expresses the optimal cost of binning into l bins starting from an arbitrary end point e_i, $B_{opt}(e_i, l)$.

Optimizing for k bins requires $O(kr^2)$ total time. The details are worked out in [12]. As mentioned earlier, in realistic applications, r (the number of distinct query endpoints), is much smaller than n (the total possible values of the attribute). More precise lower and upper bounds on the values of r are provided in the following lemma.

LEMMA 3. *Let r denote the number of distinct query endpoints in Q (elements in $EP(Q)$), i.e, $r = |EP(Q)|$ then*

$$\left\lfloor \frac{1}{2}\left(\sqrt{1+8\,|Q|}-1\right)\right\rfloor \leq r \leq \min(2\,|Q|\,, n) \quad (5)$$

Proof The first inequality in Equation 5 follows by observing that r endpoints can define at most $r(r+1)/2$ distinct queries, the result then follows by solving the inequality $|Q| \geq (r^2+r)/2$. The second inequality in Equation 5 follows since each point in *EP(Q)* is a distinct element of $[1, n]$ and each query in Q can contribute at most 2 members to *EP(Q)*. □

4. EXPERIMENTAL RESULTS

We ran a set of experiments for evaluating our optimal binning algorithm on synthetic and real data with real query workloads. The goal of these experiments is to compare the performance of bitmap indices using the optimal binning strategy against bitmap indices using more conventional binning strategies such as *equi-width* and *equi-depth binning*. The experiments were carried out on a 2.8 GHz Intel Pentium IV with 2 GB RAM. The I/O subsystem is a hardware RAID with 4 SCSI disks. For all our experiments we flushed the disk cache before performing each query. This ensures

that the query response time includes the full costs of disk I/O.

Equi-width binning partitions the bins into equally spaced ranges by retrieving v_{min} and v_{max}, the minimum and maximum value of a specific attribute, and dividing the attribute range by the number of bins. This binning strategy is straightforward to implement and has shown to perform well in some cases. However, in the worst case, the *candidate check* is as expensive as sequential scan. *Equi-depth binning*, on the other hand, chooses the bin boundaries in such a way, that each bin contains approximately the same number of entries. This makes each *candidate check* equally expensive for all query ranges and thus reduces the worst-case response time.

4.1 Synthetic Data

We generated synthetic data following a Zipf distribution[1] with three different parameters z=0, z=0.5 and z=1. We also used two different attribute cardinalities. The first three data sets have an attribute cardinality of 10^6, the second three data sets have an attribute cardinality of 10^8. Next we generated 5,000 queries that follow a Zipf distribution with parameter z=1. Finally, we built bitmap indices with 1,000 bins based on the three binning strategies discussed above. The goal is to compare the performance of these strategies for different data distributions and different attribute cardinalities. Out of these 5,000 queries we have randomly chosen 100 queries.

Figure 6 shows the average query processing time for 100 queries with different binning strategies. In total we ran 900 queries on data with different Zipf distributions (z=0, 0.5, 1). The cardinality of the data set is 10^8. The total query processing time shown in Figure 6 consists of the time for the index scan and the *candidate check* for attribute. The index scan is the time spent on reading the corresponding bitmap vectors and performing Boolean operations on them. The time for the *candidate check* is the difference between the total time and the time for the index scan. We see that for uniformly distributed and moderately skewed data (z=0, and z=0.5), the candidate check costs are a large portion of the total query processing time. For highly skewed data (z=1), the relative costs for the candidate check decreases. For all 900 queries the ratio of index scan to candidate check is 1:6. This suggests that by reducing the costs of the candidate check, the total query response time can also be significantly reduced.

The average performance speedup of our novel strategy *opt-binning* over the other two binning strategies is shown in Figure 7. We see that for data that follows a Zipf distribution with parameter z=0 the performance speedup of *opt-binning* is more than a factor of 5. For higher skewed data the performance speedup is on the average a factor of 1.5.

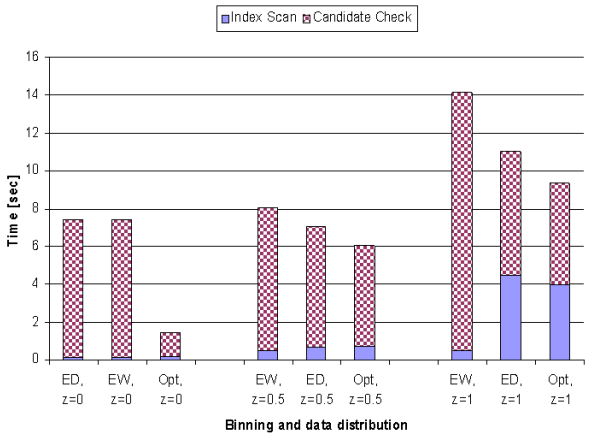

Figure 6: Average response time of queries on data with different Zipf distributions. EW = equi-width binning; ED = equi-depth binning; z = parameter of Zipf distribution for synthetic data values.

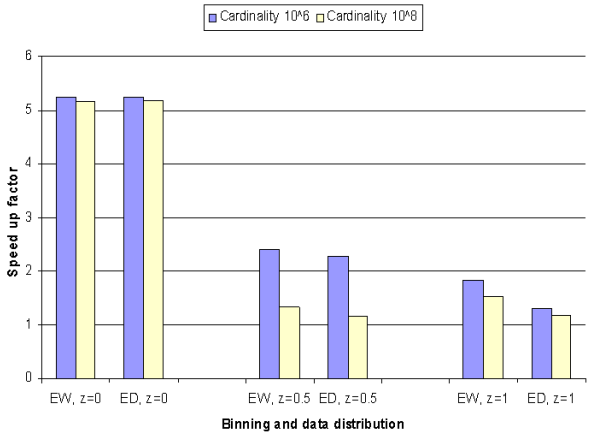

Figure 7: Performance speedup of opt-binning over traditional binning strategies. EW = equi-width binning; ED = equi-depth binning; z = parameter of Zipf distribution for synthetic data values.

4.2 Real Data

4.2.1 Query Workloads and Data Distribution

The next set of experiments is based on a large real data set from the Sloan Digital Sky Survey (SDSS), Data Release 1 [13]. SDSS is an astronomical survey project that maps one quarter of the entire sky in order to determine the positions and absolute brightnesses of more than 100 million celestial objects. The survey also measures the distances to more than a million galaxies and quasars.

The data set of Data Release 1 consists of 168 million records and some 500 attributes. We selected a representative subset of attributes for studying the query performance of our optimized bitmap index. For this purpose we did an extensive study of real query workloads from astronomers of the SDSS collaboration over a few weeks. We extracted 100,000 queries and identified three attributes that were by far the most commonly used ones in all observed queries. For instance, each of the variables *ra* and *dec* appeared in 30.3% of all range conditions of the queries. *petromag_z* was

[1]Zipf's law may be stated mathematically as: $p_i(z,n) = \frac{1/i^z}{\sum_{j=1}^{n} 1/j^z}$ where n is the number of elements, i is their rank, and z is the exponent characterizing the distribution.

used in 28.5% of the range conditions. The variables *ra* and *dec* describe the position of celestial objects in the sky in terms of *right ascension* and *declination*, and *petromag_z* defines the *Petrosian flux* [13].

4.2.2 Building Bitmap Indices

The next step of our study was to build the bitmap indices. For each of the three attributes we built bitmap indices based on the three binning strategies we discussed in the beginning of this section, namely *equi-width*, *equi-depth* and *opt-binning*. One of the key parameters for building bitmap indices is to decide on the optimal number of bins which is usually a trade-off between query speed and index size [4]. For our experiments we have chosen 1,000 bins which has shown to be a good trade-off in previous experiments.

We first built bitmap indices with 1,000 bins based on *equi-width* and *equi-depth* binning strategies. For all our experiments we used equality encoded bitmap indices and WAH compression [17].

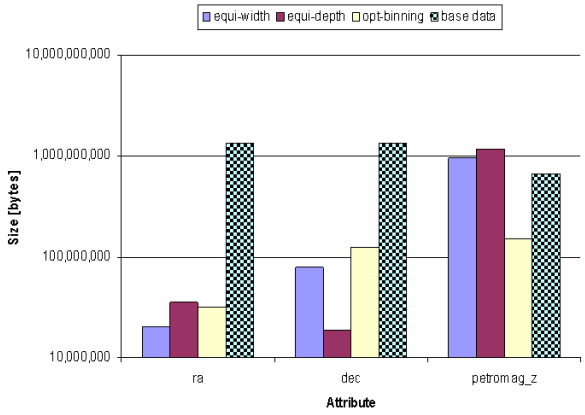

Figure 8: Size of the base data and of the compressed bitmap indices with various binning strategies.

Next we built bitmap indices based on our novel strategy *opt-binning* that we introduced in Section 3. The optimal bin boundaries were calculated from the real query workloads on the SDSS data set. We have chosen the 5,000 most frequently used conditions for each attribute and ran the dynamic programming algorithm for calculating the optimal bin boundaries for each of the three attributes. The sizes of the three selected attributes and the respective sizes of the compressed bitmap indices based on three different binning strategies are shown in Figure 8. *Base data* refers to the original data values consisting of 168 million records. The attributes *ra* and *dec* are of type *double* with a total size of 1.4 GB each. The attribute *petromag_z* is of type *float* with a total size of 0.7 GB. We observe that with *equi-width* and *equi-depth* binning, the attributes *ra* and *dec* compress very well and are only a small fraction of the original data size. However, the index for the attribute *petromag_z* is about twice the size of the base data as expected for attributes with random distribution [17]. With our novel binning strategy *opt-binning* the attributes *ra* and *dec* compress slightly worse but the total size of all three attributes combined is significantly lower than with the other two binning strategies.

4.2.3 Optimized Query Performance

After building the bitmap indices, we measured the query performance for the three different binning strategies. Given a set of 5,000 query workloads from real SDSS data analysis, we randomly sampled 1,000 queries for each attribute to test our strategies. Again, we flushed the disk cache before performing each query. This ensures that the query response time includes the full costs of disk I/O.

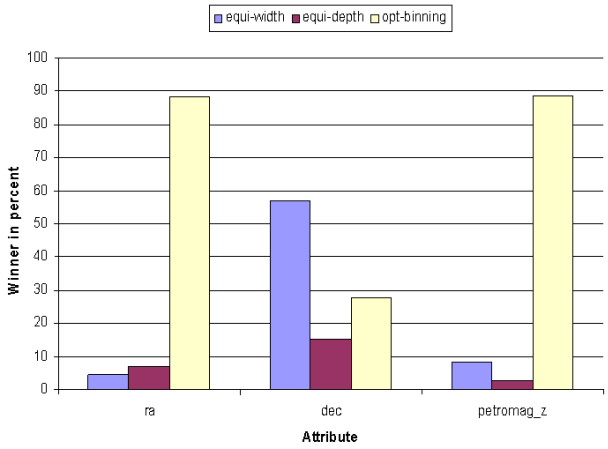

Figure 9: Winning binning strategies for 1,000 randomly sampled queries on three different attributes.

Figure 9 shows the winning binning strategies for 1,000 randomly sampled queries on three different attributes. For the attributes *ra* and *petromag_z* the binning strategy *opt-binning* performs better than the other two stragies in almost 90% of the queries. For the attribute *dec* the binning strategy *equi-depth* performs best in 57% of the queries, *opt-binning* wins in 28%. *opt-binning* does not perform best for the attribute *dec* since the optimization algorithm is based on the simplified assumption that the data is uniformly distributed across the disk pages (see Equation 1). All our experiments, however, are performed on real data where the physical clustering of data onto pages is often difficult to approximate by simple functions. Both *ra* and *dec* are coordinates of the dataset. They are clustered physically in the sense that the entries that are close together have nearly the same values. This is because the data collected in a data file are from a small patch of the sky, i.e., with *ra* and *dec* values that are very close together. Therefore, the values of *ra* and *dec* are not randomly distributed across the pages. The data values of *petromag_z*, on the other hand, are not physically clustered on disk. This fact can be derived from the size of the compressed bitmap index of this attribute which is about twice the size of base data [17].

Due to space limitation we only analyze the query response time for the attribute *petromag_z*. The average query response times for *equi-width*, *equi-depth* and *opt-binning* are 21.7, 21.7 and 13.1 seconds respectively. On average, the strategy *opt-binning* outperforms the two other binning

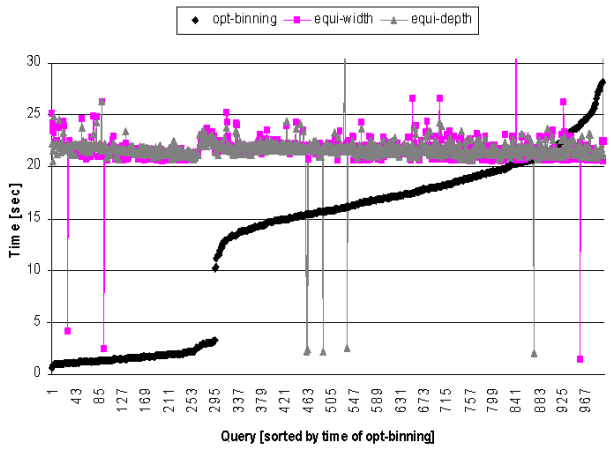

Figure 10: Response time for 1,000 queries on attribute "petromag_z" for three different binning strategies.

strategies by nearly a factor of 2 (see Figure 10). We also see in this figure that for more than a quarter of the queries, *opt-binning* is better by a factor of 4 than the other two strategies, and is a winner for about 90% of the queries.

5. CONCLUSIONS

In this paper we have introduced a novel algorithm for improving the query response time of bitmap indices by computing optimal bin boundaries. We presented an analytical model of our strategy and evaluated the performance of synthetic and real query workloads for a large data set from the Sloan Digital Sky Survey. We showed that our algorithm outperforms traditional methods by a factor of 2 for nearly 90% of the analyzed user queries. The algorithm can be used as a tool for initially placing bin boundaries for constructing a bitmap index. Subsequently our algorithm can be used for periodically reorganizing the bitmap index based on observed query workloads, data distribution and physical clustering of values on data pages. Due to its dynamic programming approach, the algorithm tabulates optimal bin placement and the corresponding candidate check costs for each value of k (number of bins) up to a required maximum. In a multi-attribute environment this can be used to evaluate different combinations of bin allocations to the various attributes subject to a global constraint for the total size of the bitmap index.

6. REFERENCES

[1] S. Amer-Yahia and T. Johnson. Optimizing Queries on Compressed Bitmaps. In *VLDB 2000*, Cairo, Egypt, Sept. 2000. Morgan Kaufmann.

[2] G. Antoshenkov. Byte-aligned Bitmap Compression. Technical report, Oracle Corp., 1994. U.S. Patent number 5,363,098.

[3] R. E. Bellman. *Dynamic Programming*. Dover Publications, Incorporated, 2003.

[4] C. Y. Chan and Y. E. Ioannidis. An Efficient Bitmap Encoding Scheme for Selection Queries. In *SIGMOD 1999*, Philadelphia, Pennsylvania, USA, June 1999. ACM Press.

[5] S. Chaudhuri and U. Dayal. An Overview of Data Warehousing and OLAP Technology. *ACM SIGMOD Record*, 26(1):65–74, March 1997.

[6] S. Guha, N. Koudas, and D. Srivastava. Fast Algorithms For Hierarchical Range Histogram Construction. In *PODS 2002*, Madison, Wisconsin, USA, June 2002. ACM Press.

[7] T. Johnson. Performance Measurements of Compressed Bitmap Indices. In *VLDB 1999*, Edinburgh, Scotland, Sept. 1999. Morgan Kaufmann.

[8] N. Koudas. Space Efficient Bitmap Indexing. In *CIKM 2000*, McLean, Virginia, USA, Nov. 2000. ACM Press.

[9] N. Koudas, S. Muthukrishnan, and D. Srivastava. Optimal Histograms for Hierarchical Range Queries. In *PODS 2000*, Dallas, Texas, USA, 2000. ACM Press.

[10] P. O'Neil. Model 204 Architecture and Performance. In *2nd International Workshop in High Performance Transaction Systems*, Asilomar, California, USA, 1987. Springer-Verlag.

[11] P. O'Neil and D. Quass. Improved Query Performance with Variant Indexes. In *SIGMOD 1997*, Tucson, Arizona, USA, May 1997. ACM Press.

[12] D. Rotem, K. Stockinger, and K. Wu. Efficient Binning for Bitmap Indices on High-Cardinality Attributes. Technical Report LBNL-56936, Berkeley Lab, Berkeley, California, USA, Nov. 2004.

[13] Sloan Digital Sky Survey. http://www.sdss.org/dr1/.

[14] K. Stockinger, K. Wu, R. Brun, and P. Canal. Bitmap Indices for Fast End-User Physics Analysis in ROOT. In *Nuclear Instruments and Methods in Physics Research*. Elsevier. to appear.

[15] K. Stockinger, K. Wu, and A. Shoshani. Evaluation Strategies for Bitmap Indices with Binning. In *International Conference on Database and Expert Systems Applications (DEXA 2004)*, Zaragoza, Spain, Sept. 2004. Springer-Verlag.

[16] A. Szalay, P. Kunszt, A. Thakar, J. Gray, and D. Slutz. Designing and Mining Multi-Terabyte Astronomy Archives: The Sloan Digital Sky Survey. In *SIGMOD 2000*, Dallas, Texas, USA, May 2000. ACM Press.

[17] K. Wu, E. J. Otoo, and A. Shoshani. On the Performance of Bitmap Indices for High Cardinality Attributes. In *VLDB 2004*, Toronto, Canada, Sept. 2004. Morgan Kaufmann.

[18] K. Wu, W.-M. Zhang, V. Perevoztchikov, J. Lauret, and A. Shoshani. The Grid Collector: Using an Event Catalog to Speedup User Analysis in Distributed Environment. In *Computing in High Energy and Nuclear Physics, (CHEP 2004)*, Interlaken, Switzerland, Sept. 2004.

[19] K.-L. Wu and P.S. Yu. Range-Based Bitmap Indexing for High-Cardinality Attributes with Skew. Technical report, IBM Watson Research Center, May 1996.

[20] M.-C. Wu and A. P. Buchmann. Encoded Bitmap Indexing for Data Warehouses. In *ICDE 1998*, Orlando, Florida, USA, Feb. 1998. IEEE Computer Society Press.