Efficient Binning for Bitmap Indices on High-Cardinality Attributes

Doron Rotem, Kurt Stockinger, and Kesheng Wu
Computational Research Division
Lawrence Berkeley National Laboratory
1 Cyclotron Road, Berkeley, CA 94720, USA

ABSTRACT

Bitmap indexing is a common technique for indexing high-dimensional data in data warehouses and scientific applications. Though 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. Binning is a common technique for reducing storage costs of bitmap indices. This technique partitions the attribute values into a number of ranges, called bins, and uses bitmap vectors to represent bins (attribute ranges) rather than distinct values. Although binning may reduce storage costs, 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.

In this paper we study the problem of finding optimal locations for the bin boundaries in order to minimize these access costs subject to storage constraints. We propose a dynamic programming algorithm for optimal partitioning of attribute values into bins that takes into account query access patterns as well as data distribution statistics. Mathematical analysis and experiments on real life data sets show that the optimal partitioning achieved by this algorithm can lead to a significant improvement in the access costs of bitmap indexing systems for high-cardinality attributes.

1. INTRODUCTION

Scientific applications such as high-energy physics, astrophysics or combustion studies generate large amounts of data in the order of terabytes and petabytes. Analyzing these large data sets is a major challenge and often requires new scalable algorithms for efficiently querying high-dimensional search spaces. Similar to typical data warehouse applications, scientific data produced from various scientific instruments is mostly read-only. Thus, indexing techniques can be streamlined for fast data access without the need for dynamic updates which is common in transactional database systems.

Bitmap indexing is a common technique for indexing high-dimensional data in data warehouses and scientific applications. This index data structure is well suited for complex, multi-dimensional ad-hoc queries against read-only data and is supported by the major commercial database systems. Though 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 is 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.

An example of the problem we are considering here is given in Figure 1. Assume that we want to evaluate the query $37 \leq x < 63$. Bins 1, 2 and 3 contain data values relevant to the query. However, Bins 1 and 3 are edge bins since they contain also irrelevant values. Answering this query involves checking the values on disk corresponding to the four "1-bits" in these two columns. In this example only one of the four qualifies, namely, 61.7. We call this additional step the *candidate check*. As we can see from this example, the cost of performing a *candidate check* on an edge bin is related to the number of "1-bits" in that bin.

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 each attribute. Such constraints are still applicable even when bitmap compression is effectively used. Effective binning strategies attempt to compute bin location 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

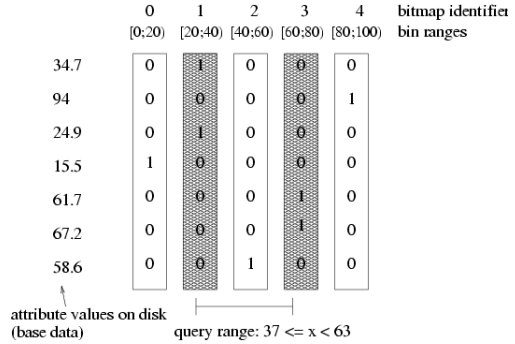

Figure 1: Range query $37 \leq x < 63$ on a bitmap index with binning.

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.

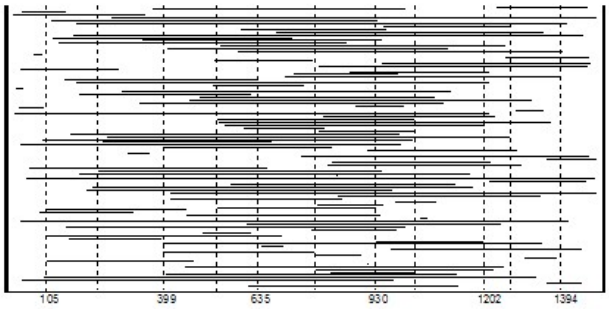

Figure 2: Uniformly distributed data with uniform range queries.

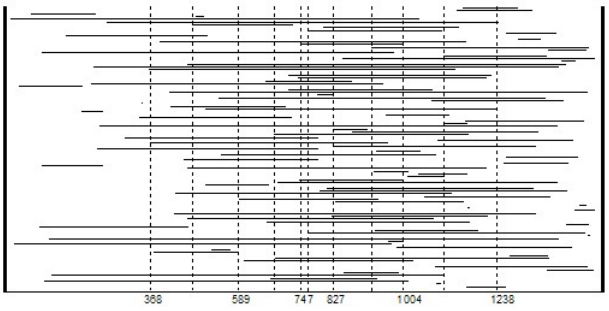

Figure 3: Normally distributed data with uniform range queries.

Figures 2, 3 and 4 illustrate by a small example the effect of data and query distribution on the optimal binning strategy. In all cases we show optimal binning into 12 bins (produced by our software) for an attribute using 100 simulated range queries.

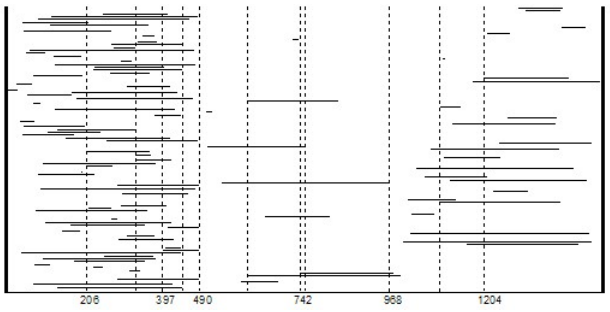

Figure 4: Normally distributed data with heavy left region queries.

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 Figures 3 and 4 the values of the attributes are normally distributed (truncated in the range [0, 1500]) with mean 750 and standard deviation 230. In Figure 3, the queries are generated randomly from a uniform distribution. We note that the bin sizes vary showing wider bins on the edges of the range to reflect more sparsely populated subregions.

In Figure 4 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 can 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 life query logs that reveal very distinct patterns in the way scientists submit queries to an astrophysics database (see Figures 8 and 9). As we will show in this paper, knowledge of data and query distribution, even if only approximate, can be used to significantly 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 life data shows that the I/O costs involved in 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 Figures 16 and 19, the I/O costs for the *candidate check* can be several orders of magnitude higher than the costs for the index scan.

1.2 Outline and Contributions

Our research is inspired by a similar approach of [10] on finding optimal bin boundaries for bitmap indices. In [10], the author solved this problem for point queries and raised several interesting open research issues that we build upon in terms of optimal binning strategies for range queries.

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 proving that only query endpoints need to be considered as potential locations for bin boundaries rather than all possible values of the attribute as in [10].
- We tested our algorithm on real life 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 [17]. 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 real queries and real data from a large astrophysics applications. Section 5 concludes the paper and raises open issues for future work.

2. RELATED WORK

Bitmap indices are used for speeding up complex, multi-dimensional queries for On-Line Analytical Processing and data warehouse [7] as well as for scientific applications [14]. They were first implemented in a commercial DBMS called Model204 [11]. Improvements on this approach were discussed in [12].

In [5, 6] the following bitmap encoding strategies are introduced: *equality, range* and *interval* encoding. Equality encoding is optimized for so-called exact match queries of the form $a = v$ where a is an attribute and v the value to be searched for. Range encoding, on the other hand, is optimized for one-sided range queries of the from $a\ op\ v$ where $op \in \{<, \leq, >, \geq\}$. Finally, interval encoding shows the best performance characteristics for two sided-range queries of the form $v_1\ op\ a\ op\ v_2$.

The authors of [19] 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.

As already mentioned before, simple bitmap indices are efficient for low-cardinality attributes but they show a considerable storage overhead for high-cardinality attributes. One way of reducing the storage complexity is to use bitmap compression. Note, an efficient bitmap compression scheme not only has to reduce the size of bitmaps but also has to perform bitwise Boolean operations efficiently.

Various bitmap compression schemes were studied in [1, 9]. The authors demonstrated that the scheme named Byte-aligned Bitmap Code (BBC) [2, 3] shows the best overall performance characteristics. More recently a new compression scheme called Word-Aligned Hybrid (WAH) [16] 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 integer or floating point values with low attribute cardinalities. However, scientific data is often based on floating point values with high attribute cardinalities. The work presented in [14] demonstrated that bitmap indices with binning can significantly speed up multi-dimensional queries against high-cardinality attributes.

A further bitmap index with binning called *range-based* bitmap indexing was introduced in [18]. The idea is to evenly distribute skewed attribute values onto various bins in order to achieve uniform search times for different queries. The bin ranges for the bitmap vectors are chosen based on a dynamic bucket expansion and contraction approach. In the initial phase, the number of entries per bucket (bin) are counted. If the number of entries per bucket grows beyond a certain threshold, the bucket is dynamically expanded into buckets with smaller ranges. Finally, multiple buckets with adjacent ranges are combined. The authors demonstrated that the algorithm efficiently redistributes highly skewed data. However, performance results about query response times were not discussed.

In [10] a methodology for building space efficient bitmap indices is introduced for high-cardinality attributes based on binning. The work in [10] focuses on point (equality) queries rather than range queries discussed in this paper. Similar to our approach, an optimal dynamic programming algorithm is used for efficiently choosing bin ranges. The experiments were executed on synthetic data and workloads (point queries) following a Zipf distribution, which is common in real applications. The author raised several interesting open research issues that inspired our research. We extend the work of [10] by analyzing range queries. In addition, we evaluate our optimal binning algorithm on real data and real query workloads taken from the Sloan Digital Sky Survey [13, 15].

Notation	Explanation
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
$B =< b_1, b_2, ..., b_k >$	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
$w_i = \sum_{q \in E(b_i)} p_q$	Weight of bin b_i, the sum of probabilities of all queries in $E(b)$
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.

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 such value is an integer in the 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 associated 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 with one bitmap (consisting of N bits) associated with each bin as previously described. An integer constraint k, 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 $\begin{pmatrix} n-1 \\ k-1 \end{pmatrix}$ 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 defined as an edge bin for query q if the range defined by the query q overlaps some part of the range defined by 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.

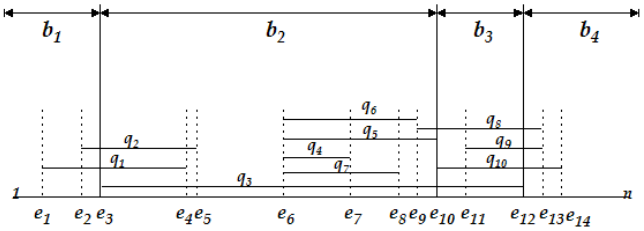

Figure 5: Query endpoints and bin boundaries. Horizontal lines represent query ranges. Dotted vertical lines mark query endpoints.

In Figure 5 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} have 1 edge bin and each of the queries q_1, q_2, q_8, q_9 have 2 edge bins. As explained earlier, when query q is specified, a significant fraction of the I/O costs it incurs are related to the number of data pages we need to read in order to perform *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 5 $E(b_1) = \{q_1, q_2\}$; $E(b_2) = \{q_1, q_2, q_4, q_5, q_6, q_7, q_8\}$; $E(b_3) = \{q_9\}$; $E(b_4) = \{q_8, q_9, q_{10}\}$.

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.

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 [12]:

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

The expected *candidate check* I/O cost of answering the

queries in Q when an attribute range is partitioned by the set of bins B is denoted by $Cost(Q, B)$ defined as

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

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, *OptBin* is defined as follows:

Given a set of range queries Q and the values of an attribute in the range [1,n], find the location of the k-1 bin boundary points that partition the attribute values into k bins such that the I/O cost for 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

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

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 in Figure 5, the distinct endpoints of 10 range queries are shown.

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 $EP(Q)$.*

PROOF. Assume $B \in \Pi(k, n)$ has a boundary point $x_i \notin EP(Q)$. For a bin $b_i \in B$, let n_i denote the number of its data points and let w_i (bin weight) denote the sum of probabilities of queries with endpoints in it, i.e., $w_i = \sum_{q \in E(b_i)} p_q$. We first assume that both bins $b_i = [x_{i-1}, x_i)$ and $b_{i+1} = [x_i, x_{i+1})$ have non-zero weights (i.e., $w_i > 0$ and $w_{i+1} > 0$) other cases are much simpler and will be dealt with later. Let $x, y \in EP(Q)$ be the closest members of $EP(Q)$ (query endpoints) to x_i in bins b_i and b_{i+1} respectively such that x is an endpoint of query $q_x \in E(b_i)$ and y an endpoint of query $q_y \in E(b_{i+1})$ (see Figure 6).

The non-zero weight assumption above guarantees the existence of such queries. Based on Equation 1 the total cost contributed by bins b_i and b_{i+1} to $Cost(Q, B)$ is

$$P(1 - e^{-n_i/P})w_i + P(1 - e^{-n_{i+1}/P})w_{i+1} \quad (4)$$

Let $B_x \in \Pi(k, n)$ be a binning obtained from B by replacing x_i with x. This replacement only affects the bins b_i and b_{i+1}

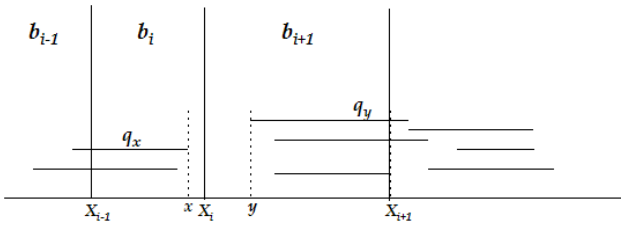

Figure 6: Replacing a bin boundary with a close query endpoint.

as it may cause m_x data points to move from bin b_i to bin b_{i+1} for some integer $m_x \geq 0$. After this replacement, the contribution of bin b_i and bin b_{i+1} to $Cost(Q, B_x)$ will be at most

$$P(1 - e^{-(n_i - m_x)/P})w_i + P(1 - e^{-(n_{i+1} + m_x)/P})w_{i+1} \quad (5)$$

as q_x may have no remaining endpoints in bin b_i thus reducing w_i. If $m_x = 0$ then $Cost(Q, B_x) \leq Cost(Q, B)$ and the lemma is proved so we assume $m_x > 0$.

The difference D_x between the two costs is

$$\begin{aligned} D_x &= Cost(Q, B_x) - Cost(Q, B) \\ &\leq P[(1 - e^{-(n_i - m_x)/P}) - (1 - e^{-n_i/P})]w_i + \\ &\quad P[(1 - e^{-(n_{i+1} + m_x)/P}) - (1 - e^{-n_{i+1}/P})]w_{i+1} \\ &= P(1 - e^{m_x/P})(e^{-n_i/P}w_i + e^{-n_{i+1}/P}w_{i+1}) \quad (6) \end{aligned}$$

If $D_x \leq 0$ the lemma is proved, otherwise we have $D_x > 0$, this in turn implies

$$\frac{w_i e^{-n_i/P}}{w_{i+1} e^{-n_{i+1}/P}} \leq e^{-m_x/p} \quad (7)$$

Similar arguments show that replacing x_i with y results in a binning $B_y \in \Pi(k, n)$ where m_y data points move from bin b_{i+1} to bin b_i for some integer $m_y \geq 0$. The difference in cost denoted by D_y is defined as

$$D_y = Cost(Q, B_y) - Cost(Q, B) \quad (8)$$

Again if $D_y \leq 0$ the lemma is proved, otherwise we have $D_y > 0$ which in turn implies

$$\frac{w_i e^{-n_i/P}}{w_{i+1} e^{-n_{i+1}/P}} \geq e^{m_y/p} \quad (9)$$

Since $e^{-m_x/p} < 1$ and $e^{m_y/p} > 1$ (assuming $m_x > 0$ and $m_y > 0$), Equations 7 and 9 can not both hold, so at least one of D_x or D_y are negative, thus proving the lemma.

In the case that exactly one of the bins b_i or b_{i+1} have zero weight we can show that $D_y < 0$ if $w_i = 0$ (endpoint x does not exist) and $D_x < 0$ if $w_{i+1} = 0$ (endpoint y does not exist). In both cases this shows that replacing $x_i \notin EP(Q)$ with a point in $EP(Q)$ results in a binning with equal or lesser cost then $Cost(Q, B)$. Finally, in the case that both bins b_i and b_{i+1} have zero weight, x_i can be eliminated altogether and the two bins merged without any cost increase. □

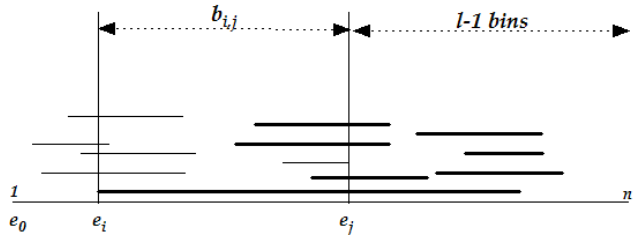

Figure 7: Dynamic programming, queries in $R(Q, j)$ are emphasized using thick lines.

THEOREM 2. *There exists an optimal binning $B^*_{opt} \in \Pi(k, n)$ with all its bin boundary points in $EP(Q)$.*

PROOF. Assume there exists an optimal binning $B_{opt} \in \Pi(k, n)$ with one or more boundary points not in $EP(Q)$. Using the above lemma, we can replace these boundary points one by one with a point in $EP(Q)$ where each such replacement does not increase the cost. Let us denote the resulting binning $B^*_{opt} \in \Pi(k, n)$, then B^*_{opt} is also optimal and has all its boundary points in $EP(Q)$. □

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 [4], 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 7). 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 range 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 7). Using the above notation, our goal is to find $B_{opt}(e_0, k)$.

THEOREM 3. *Given a set of queries Q with an ordered set of distinct endpoints $EP(Q) =< e_1, e_2, .., e_r >$consisting of r distinct endpoints then*

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

PROOF. Assume an optimal binning, $B_{opt}(e_i, l)$ is given with all its boundary points in $EP(Q)$ (this is always possible based on Theorem 2). 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 7). Therefore e_j is a candidate only if j satisfies $i < j \le r-(l-2)$. 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 3 allows us to devise an efficient dynamic programming algorithm as it expresses the optimal cost of binning into l bins in terms of smaller binning problems and contributions of single bins. The algorithm uses three basic tables

- `BinCost` is an $r \times r$ array used for tabulating single bin costs, i.e. `BinCost`$[\, i, j] = Cost(b_{i,j})$ for $1 \le i < j \le r$
- `OptCost` is a $k \times r$ array used for tabulating optimal costs for binning subproblems for up to k bins, i.e., `OptCost`$[\, i, j] = Cost(B_{opt}(e_j, i))$ for $1 \le i \le k$ and $1 \le j \le r - i$.
- `OptBin` is a $k \times r$ array used to store the location of the bins. After the execution of the algorithm in Listing 1, the smallest value of OptCost in row k is the minimal cost of having k bin boundaries (k+1 bins). Let OptCost[k, j_1] be the first element with the minimal cost, the first bin boundary of optimal bins is e_{j_1}. Let j_2 be OptBin[k, j_1], the second bin boundary is e_{j_2}. Later bin boundaries e_{j_i} can be defined recursively as

$$j_i = \texttt{OptBin}[k - i + 2, j_{i-1}], \qquad j = 2, \ldots, k.$$

The computation of `BinCost` utilizes a number of different arrays to accumulate various partial sums of p_q. They are important to reduce the computation complexity from $O(r^3)$ to $O(r^2)$. The formula used in Listing 1 is based on the observation that all queries contributing to the bin weight either have the left endpoint or the right endpoint in the bin. The temporary array `dpq` is a cumulation of total query probabilities of those with left endpoints at a given value. The temporary array `bpq` stores the total query probabilities of those with left endpoint less than or equal e_i and right endpoints in the range (e_i, e_j). The recursion for `bpq` is

$$\texttt{bpq}[i, j] = \texttt{bpq}[i, j-1] + \texttt{cpq}[i, j-1],$$

with $\texttt{bpq}[i, i+1] = 0$. Note that array `bpq` is computed progressively and only the last element of each row is stored. The content of array `cpq` can be derived with a small amount of computation which we omit here.

LISTING 1. Algorithm to Compute OptBin and OptCost arrays.

```
Input arrays:
   cc[r]:  cumulative count of records with values.
      less than e_i
   pq[r,r]:  pq[i,j] is the probability of
      query e_i ≤ A < e_j, i.e., p_q.
Internal arrays:
   BinCost[r,r]:  the cost of a single bin.
   cpq[r,r]:  a cumulation of array pq.
   bpq[r]:  a cumulation of array cpq.
   dpq[r]:  a cumulation of array cpq.
Output arrays:
   OptBin[k,r]:  location of the bin boundaries.
   OptCost[k,r]:  total cost of the bins.

// fill the arrays cpq and dpq
For i = 1 to r;
   cpq[i,i] = 0;
   For j = i+1 to r;
      cpq[i,j] = pq[i, j]
For i = 1 to r;
   For j = i+1 to r;
      cpq[i,j] = cpq[i,j] + cpq[i-1,j];
For i = 1 to r;
   dpq[i] = 0;
   For j = i+1 to r;
      dpq[i] = dpq[i] + cpq[i,j]

// Compute BinCost and bpq
For i = 1 to r;
   bpq[i] = 0;
   BinCost[i,i+1] = 0;
   For j = i+2 to r;
      bpq[i] = bpq[i] + cpq[i,j-1];
      BinCost[i,j] = P(1-exp((cc[i]-cc[j])/P)) *
      (dpq[j-1] - dpq[i] + bpq[i]);
   bpq[i] = bpq[i] + cpq[i,j];

// Initialize OptCost and OptBin
For j = 1 to r
   OptBin[1,j] = j;
   // compute the cost of bin [ e_j, n]
   OptCost[1,j] = P(1-exp((cc[j]-N)/P)) *
   (dpq[r] - dpq[j] + bpq[j]);

// Iterate on OptCost
For i = 2 to k;
   For j = 1 to r-i;
      loc = j + 1;
      cst = OptCost[i-1, loc];
      For g = j+2 to r-i+1;
         tmp = BinCost[j,g] + OptCost[i-1,g];
         If (tmp < cst) then
              cst = tmp;
              loc = g;
      OptCost[i,j] = cst;
      OptBin[i,j] = loc;

// Finalize the total cost by adding the left most bin
For i = 1 to k;
   For j = 0 to r-i;
      OptCost[i,j] = OptCost[i,j] + dpq[j] *
      P(1-exp(-cc[j]/P)).
```

It is easy to see that the `BinCost` array can be computed in $O(r^2)$ time. Based on Theorem 3, each entry in `OptCost` is computed by finding the minimum over $O(r)$ values so that `OptCost` can be computed in $O(kr^2)$. The total computational complexity of the algorithm is therefore $O(kr^2)$. The following lemma provides bounds on r.

LEMMA 4. *Let r denote the number of distinct query endpoints in Q (elements in $EP(Q)$), i.e, $r = |EP(Q)|$ then*

$$\left\lfloor \frac{1}{2}\left(\sqrt{1+8\,|Q|}-1\right)\right\rfloor \leq r \leq \min(2\,|Q|\,, n) \qquad (10)$$

PROOF. The first inequality in Equation 10 follows by observing that x endpoints can define at most $x(x+1)/2$ distinct queries, the result then follows by solving the inequality $|Q| \geq r^2 + r/2$. The second inequality in Equation 10 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 against real data and 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*.

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.

4.1 Query Workloads and Data Distribution

Our experiments are 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 first had to transform this data set from FITS-format [8] to binary in order to incorporate the data into our indexing software. The next step was to select 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 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].

A representative subset of the workload patterns of these three attributes is shown in Figures 8, 9 and 10. For the attributes *ra* and *dec* we can observe similar patterns that are characterized by various range queries that probe small ranges of the sky (the *right endpoint* and the *left endpoint* of the query are very close to each other with respect to the whole domain space of the attribute). For *petromag_z* we can observe range queries where one query range (left endpoint) is fixed whereas the other one (right endpoint) is systematically increased by a small percentage.

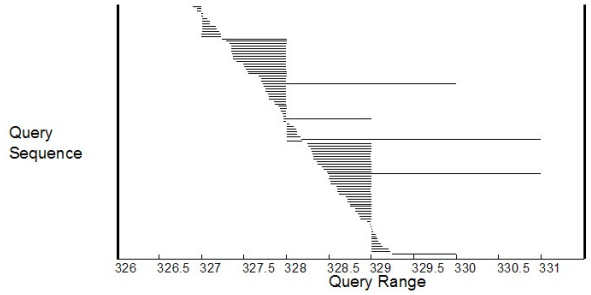

Figure 8: Subset of the query history of attribute "ra" with endpoints in the range of [326.91, 330.81]. Note that the domain space of the attribute "ra" is in the range of [0, 360].

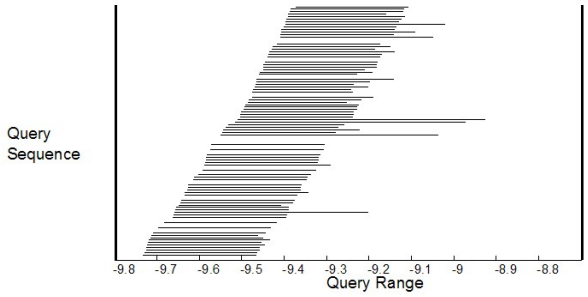

Figure 9: Subset of the query history of attribute "dec" with endpoints in the range of [-9.73, -8.92]. Note that the domain space of the attribute "dec" is in the range of [-11.27, 68.77].

The data distributions of these attributes are given in Figures 11, 12 and 13. For the attribute *ra* the distribution is fairly uniform whereas for the attributes *dec* and *petromag_z* the distributions are very skewed.

4.2 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 [6].

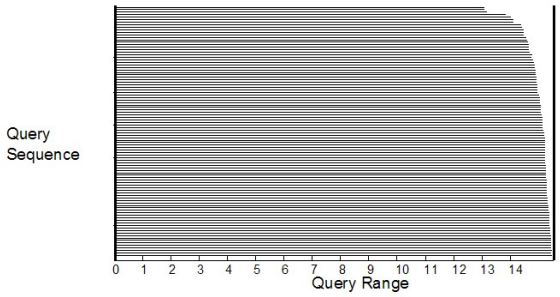

Figure 10: Subset of the query history of attribute "petromag_z" with endpoints in the range of [0, 15.47]. Note that the domain space of the attribute "petromag_z" is in the range of [4.11, 53.62].

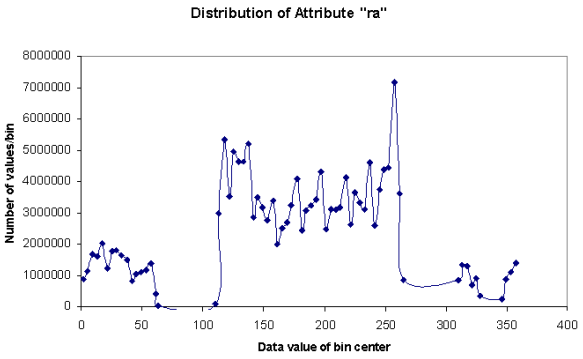

Figure 11: Data distribution of attribute "ra".

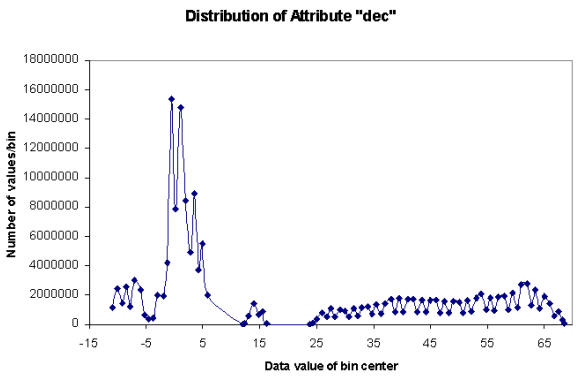

Figure 12: Data distribution of attribute "dec".

For our experiments we have chosen 1,000 bins which has shown to be a good trade-off in previous experiments [16].

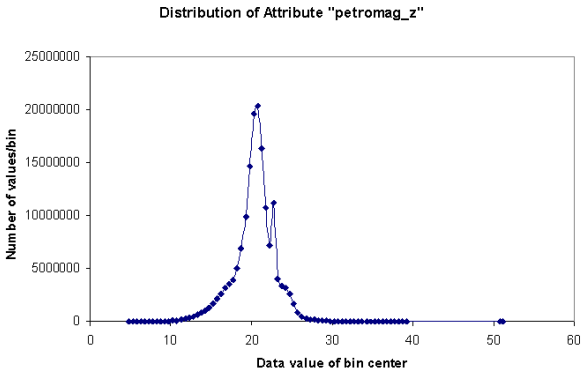

Figure 13: Data distribution of attribute "petromag_z".

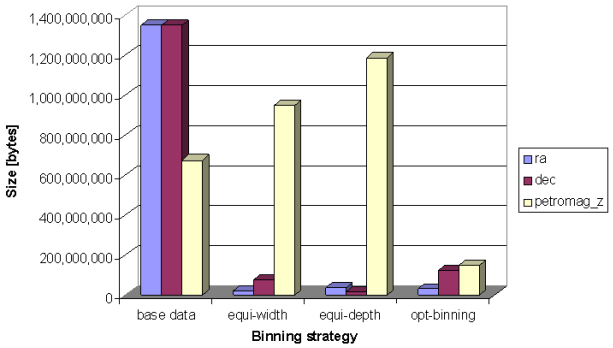

Figure 14: Size of the base data and of the compressed bitmap indices with various binning strategies.

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 [16]. 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 two SCSI disks.

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 against 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 14. *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 can 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 [16]. 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. This suggests that *opt-binning* rearranges the bin boundaries in such a way, that the size of the compressed bitmap index gets significantly reduced.

4.3 Optimized Query Performance

After having built the bitmap indices, we will now measure 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. 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.

We first analyze the performance of queries against attribute *ra*. The total run time of 1,000 queries based on the three different binning strategies is is about 6,000 seconds (more than 1.5 hours). As we can see in Figure 15, the binning strategy *opt-binning* shows the best performance (average query response time 1.6 seconds) for 88.4% of all queries, followed by *equi-depth* (average query response time 2.0 seconds) and *equi-width* binning (average query response time 2.2 seconds). In particular, the total run time of all 1000 queries based on *opt-binning* is about 20% lower than for all 1000 queries based on *equi-width* binning. For *opt-binning* we can observe three different trends in the query performance:

- For queries, where both ranges fall on optimal bin boundaries, the response time per query is on average 0.1 seconds. When compared with the average query response time of *equi-width* and *equi-depth* binning, this is a performance improvement of a factor of 20. The reason for this large performance improvement is that for queries that fall on the bin boundaries, there is no expensive *candidate check*. This effect can be seen in Figure 16 where we show the time for the index scan and the *candidate check* for the same queries.

- For queries where only one query range falls on a bin boundary, the query response time is on average 0.9 seconds. When compared with the average query response time of *equi-width* and *equi-depth* binning, this is a performance improvement of more than a factor of 2.

- For all other queries with ranges that do not fall on bin boundaries, the response time is between 1 and 2.4 seconds.

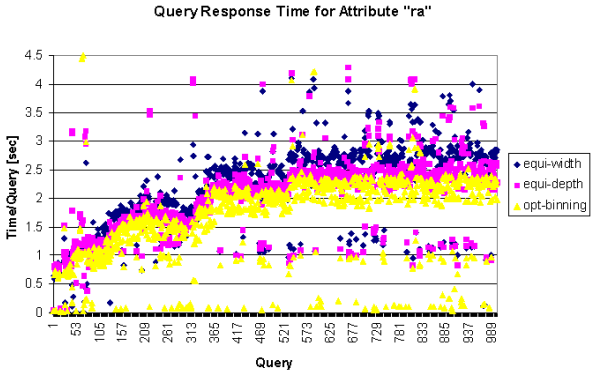

Figure 15: Query response time against attribute "ra" for three different binning strategies.

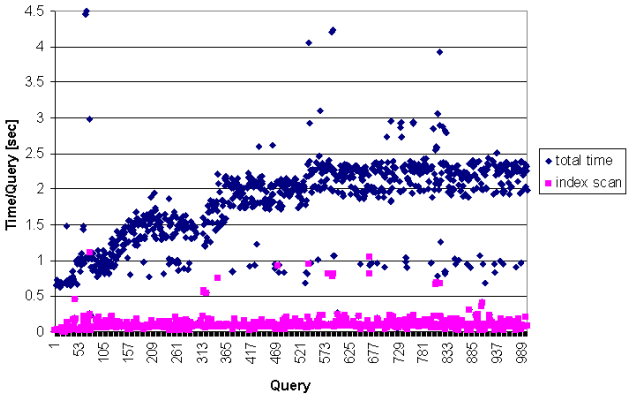

Figure 16: Query response time and index scan for attribute "ra" with opt-binning. The time for the *candidate check* is the difference between the total time and the time for the index scan. Note that the costs for the *candidate check* are up to a factor of 20 higher than the costs for the index scan.

Next we analyzed the performance of queries against attribute *dec* (see Figure 17). The total response time for all 1,000 queries is some 8,000 seconds (more than 2 hours). The average query response times for *equi-width*, *equi-depth* and *opt-binning* are 2.56, 2.73 and 2.53 seconds respectively. Note that the differences between the various strategies is less significant for attribute *dec* than for attribute *ra*. However, *opt-binning* performs slightly better than the other two strategies.

Finally we analyze the performance of queries against attribute *petromag_z*. Note that both the data distribution and the query workload are considerably different from the two previously analyzed attributes. The total response time

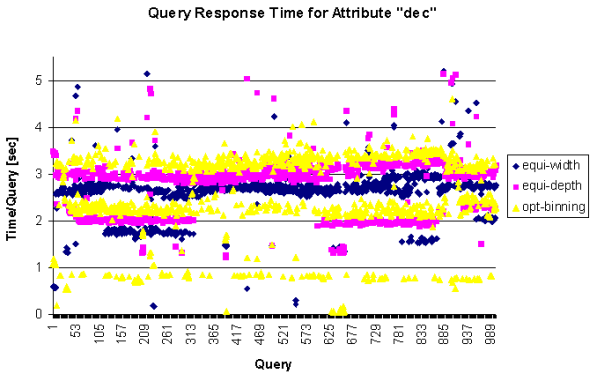

Figure 17: Query response time against attribute "dec" for three different binning strategies.

for all 1,000 queries is some 16 hours. The average query response times for *equi-width*, *equi-depth* and *opt-binning* are 21.7, 21.7 and 13.1 seconds respectively (see Figure 18). The strategy *opt-binning* performs better than the two other binning strategies in 88.6% of all the 1,000 queries. Unlike in our previous observations, we can only see two trend lines in the query performance for *opt-binning* rather than three. The reason is that in all queries, at least one range falls on a bin boundary.

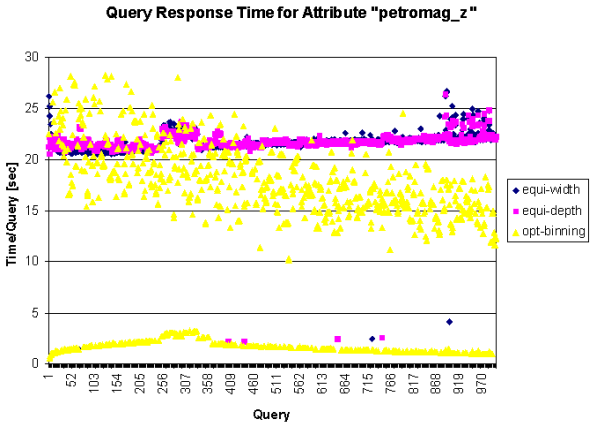

Figure 18: Query response time against attribute "petromag_z" for three different binning strategies.

Also note that there is a much larger difference in the response time between optimized and non-optimized queries. For instance, for optimized queries, the average response time is 1.5 seconds whereas for non-optimized queries the average response time is at least an order of magnitude higher. By looking at Figure 19 we can see that there is a big difference between the time spent on the index scan and the time spent on the *candidate check*. The reason for

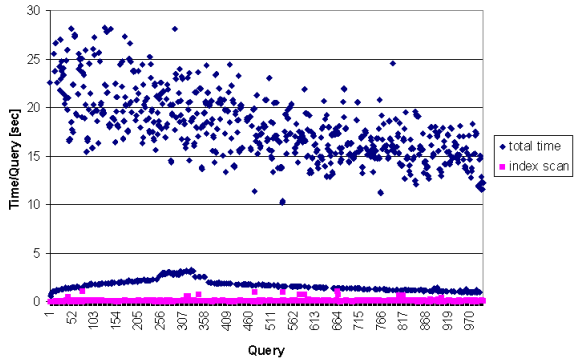

Figure 19: Query response time and index scan for attribute "petromag_z" with opt-binning. The time for the *candidate check* is the difference between the total time and the time for the index scan. Note that the costs for the *candidate check* are up to a factor of 200 higher than the costs for the index scan.

the different *candidate check* time is that the data values of the previous attributes *ra* and *dec* show a larger degree of physical clustering on disk than the attribute *petromag_z*. This is due the way the data is acquired by the astrophysics instruments. We can measure the degree of physical clustering of the attributes by analyzing the compression ratio of the bitmap index with *equi-width binning*. As we can see in Figure 14, the bitmap indices for the attributes *ra* and *dec* compress very well, whereas the bitmap index for the attribute *petromag_z* does not compress at all. In short, the more the values of an attribute are clustered, the lower is the probability of hitting many pages in the *candidate check*.

5. CONCLUSIONS AND FUTURE WORK

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 a detailed analytical model of our strategy and evaluated the performance of real query workloads against a large data set from the Sloan Digital Sky Survey. We have shown that our optimized binning strategy has the following two advantages: 1) The query response time gets significantly improved by optimally placing bin boundaries and thus minimizing the costs for the *candidate check*. 2) The total size of the bitmap indices gets reduced by re-arranging the bin boundaries and thus re-arranging attribute values between bins. Our algorithm can be used as a tool for periodically reorganizing the bitmap index based on observed query workloads.

As part of our future work we plan to extend our optimal binning strategy for multi-dimensional queries. This raises an interesting unsolved challenge such as computing the optimal number of bins for each attribute subject to a global storage constraint. Initial studies show that this requires taking into account the likelihood of attributes being included in queries, attribute selectivities [14] as well as attribute cardinalities.

6. REFERENCES

[1] S. Amer-Yahia and T. Johnson. Optimizing Queries on Compressed Bitmaps. In *International Conference on Very Large Data Bases*, Cairo, Egypt, September 2000. Morgan Kaufmann.

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

[3] G. Antoshenkov and M. Ziauddin. Query Processing and Optimization in ORACLE RDB. *VLDB Journal*, 5:229–237, 1996.

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

[5] C.-Y. Chan and Y. E. Ioannidis. Bitmap Index Design and Evaluation. In *SIGMOD*, Seattle, Washington, USA, June 1998. ACM Press.

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

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

[8] FITS - Flexible Image Transport System. http://heasarc.gsfc.nasa.gov/docs/heasarc/fits.html.

[9] T. Johnson. Performance Measurements of Compressed Bitmap Indices. In *International Conference on Very Large Data Bases*, Edinburgh, Scotland, September 1999. Morgan Kaufmann.

[10] N. Koudas. Space Efficient Bitmap Indexing. In *International Conference on Information and Knowledge Management*, McLean, Virginia, USA, November 2000. ACM Press.

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

[12] P. O'Neil and D. Quass. Improved Query Performance with Variant Indexes. In *Proceedings ACM SIGMOD International Conference on Management of Data*, Tucson, Arizona, USA, May 1997. ACM Press.

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

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

[15] 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*, Dallas, Texas, USA, May 2000. ACM Press.

[16] K. Wu, E. J. Otoo, and A. Shoshani. On the Performance of Bitmap Indices for High Cardinality Attributes. In *International Conference on Very Large Data Bases*, Toronto, Canada, September 2004. Morgan Kaufmann.

[17] 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, September 2004.

[18] 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.

[19] M.-C. Wu and A. P. Buchmann. Encoded Bitmap Indexing for Data Warehouses. In *International Conference on Data Engineering*, Orlando, Florida, USA, February 1998. IEEE Computer Society Press.