Towards Optimal Multi-Dimensional Query Processing with Bitmap Indices

Doron Rotem, Kurt Stockinger, and Kesheng Wu

Computational Research Division
Lawrence Berkeley National Laboratory
University of California
1 Cyclotron Road, Berkeley, California 94720, USA
{D_Rotem, KStockinger, KWu}@lbl.gov

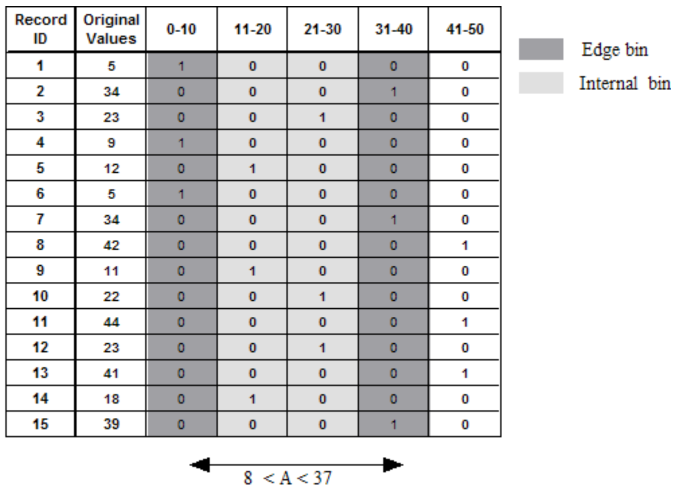
Abstract. Bitmap indices have been widely used in scientific applications and commercial systems for processing complex, multi-dimensional queries where traditional tree-based indices would not work efficiently. This paper studies strategies for minimizing the access costs for processing multi-dimensional queries using bitmap indices with binning. Innovative features of our algorithm include (a) optimally placing the bin boundaries and (b) dynamically reordering the evaluation of the query terms. In addition, we derive several analytical results concerning optimal bin allocation for a probabilistic query model. Our experimental evaluation with real life data shows an average I/O cost improvement of at least a factor of 10 for multi-dimensional queries on datasets from two different applications. Our experiments also indicate that the speedup increases with the number of query dimensions.

1 Introduction

Modern data warehouse and scientific applications produce large amounts of high-dimensional data. Due to the complexity and the size of these data sets, efficient query processing is vital for retrieving results in real time.

Bitmap indexing is a common technique for processing complex, multi- dimensional ad-hoc queries on read-only data. They have been introduced into several commercial DBMS products by database vendors including Red Brick Systems, Sybase, IBM and Oracle.

The basic bitmap index uses every distinct value of the indexed attribute as a key, and generates one bitmap containing as many bits as the number of records in the dataset for each key [12]. The sizes of these basic bitmap indices are relatively small for low-cardinality attributes, such as "gender", "types of houses sold in the San Francisco Bay Area", or "car models produced by Ferrari." However, for high-cardinality attributes such as "temperature values in a supernova explosion", the index sizes may be too large to be of any practical use. In this case, bitmap indices are often designed with bins [17]. This bitmap index strategy 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 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.


Edge bin
Internal bin
8 < A < 37

Record ID	Original Values	0-10	11-20	21-30	31-40	41-50
1	5	1	0	0	0	0
2	34	0	0	0	1	0
3	23	0	0	1	0	0
4	9	1	0	0	0	0
5	12	0	1	0	0	0
6	5	1	0	0	0	0
7	34	0	0	0	1	0
8	42	0	0	0	0	1
9	11	0	1	0	0	0
10	22	0	0	1	0	0
11	44	0	0	0	0	1
12	23	0	0	1	0	0
13	41	0	0	0	0	1
14	18	0	1	0	0	0
15	39	0	0	0	1	0

Fig. 1. Two-sided range query $8 < A < 37$ on a bitmap index with binning.

In this paper we are focusing on aggregation queries that are common in data warehousing and scientific applications. These types of queries do not return result records but rather statistical information on the result set, e.g. compute the size of the result set. Figure 1 shows a small example of evaluating such queries with binned bitmap indices. In this example we assume that an attribute A has values between 0 and 50. 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 (*bins*), namely [0,10], [11,20] etc. with a bin allocated to each sub-range. The values of the sub-ranges are called *bin boundaries*. In this example, the width of each bin is of the same size. 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 $8 < A < 37$". The correct result should be 9. We know that all records that fall into *internal bins* (highlighted in light gray) are sure hits (*qualifying records*). These records are indicated by a "1-bit" and are calculated by performing a *Boolean OR operation* on all internal bins. On the other hand, records that fall into so-called *edge bins* (highlighted in dark gray) contain both qualifying and non-qualifying values. In order to prune the *false positives*, the original data values need to be checked against the query constraint. In particular, all records of the edge bins with a bit set to "1", need to be checked. Such a check may involve additional accesses to disk pages depending on how the attribute values are stored.

Given the query $8 < A < 37$, let us look at the candidate check for the left edge bin. The candidate records are the records with IDs 1, 4 and 6. The values of these records are 5, 9 and 5, respectively. The only qualifying record is record 4 that represents the value 9. The other two records do not fulfill the query constraint and do not qualify. 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. The larger the number of candidates that need to be checked, the higher the total query processing costs.

Our bitmap indexing software called *FastBit* [7] uses the binning strategy of this example. *FastBit* has been used in production for High-Energy Physics experiments [20] over the last several years. Recently we integrated our bitmap index technology directly into the ROOT analysis framework [16] 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. Moreover, by studying the query workloads of these experiments, we identified certain patterns that could be helpful for designing more efficient bitmap indices that take into account both data distributions and query distributions. Given a fixed number of bins[1], the goal is to find the optimal bin boundaries such that the number of candidates and thus the query processing costs is minimized.

Before we outline the main contributions of this paper, we provide a brief taxonomy of the major work on optimizing bitmap indices based on query access patterns. We discuss further related work in Section 2.

1.1 Brief Taxonomy of Optimizing Bitmap Indices Based on Query Access Patterns

In Table 1, most of the previous results in this area are classified according to the problem dimensionality and the type of queries considered. Earlier work in the area attempted to adjust bin sizes dynamically as the data is updated ([21]). Most of the later work assumes read-only data and uses dynamic programming algorithms to achieve optimal bin boundary placement ([10],[15],[14]). In the multi-dimensional case, additional strategies for speeding up the queries were studied. These include sophisticated strategies for pruning the potential subset of candidates using Boolean operations [17] and dynamically reordering the scanning of the bitmap indices based on estimated attribute selectivities [15]. Some more discussion about related work can be found in Section 2.

[1] For practical reasons the number of bins is often fixed. This guarantees that the size of the bitmap index is below a certain storage threshold.

	One- Dimensional	**Multi-dimensional**
Queries not considered		Dynamic bucket expansion and contraction [21]
Point Queries	Optimal binning: dynamic programming [10]	
One-sided range queries		Optimal evaluation strategies for fixed binning and fixed evaluation order [17]
Two-sided range queries	Optimal binning: dynamic programming using only query endpoints [14]	-General problem NP complete [15] -Optimal query evaluation reordering [15] -Dynamic programming for fixed bin allocation (**current paper**) -Bin allocation for probabilistic model (**current paper**)

Table 1. Taxonomy of results on optimal binning for bitmap indexes.

1.2 Main Contributions of this Work

– We present optimization strategies to reduce the cost of query processing using bitmap indices. The key features of our approach are: a) Optimal placement of bin boundaries for multi-dimensional data sets. b) Reordering the evaluation of multi-dimensional queries based on attribute selectivity and I/O costs.
– Finding optimal bin allocations for a common query model with independent probabilities of attributes being included in queries.
– We analyze the query processing costs of our *Opt-binning* strategy for queries on two different data sets from applications that are used in production. This guarantees that we evaluate our strategy on real use cases as opposed to simplified synthetic data. Our results show that our binning strategy yields significant improvements over traditional binning methods.

The rest of the paper is organized as follows. In Section 2 we discuss related work on bitmap indices and binning strategies. In Section 3 some of our previous results on single attribute optimal binning are presented. These results are used as a building block in the multi-dimensional binning algorithms of Section 4. In Section 5 we present results for the optimal choice of bin allocations for a probabilistic query model. Section 6 discusses an additional optimization strategy, namely attribute reordering. In Section 7 we evaluate our novel strategies for optimizing multi-dimensional queries based on data sets from two different applications. Finally in Section 8 we present our conclusions and discuss some future work.

2 Related Work

Bitmap indices are used for speeding up complex, multi-dimensional queries for On-Line Analytical Processing and data warehouse [6] as well as for scientific applications [17]. The first commercial product to use the name bitmap index is Model 204 [12]. Improvements on this approach called *bit sliced-index* are discussed in [13].

In [4, 5] three bitmap encoding strategies are introduced: *equality, range* and *interval* encoding. Equality-encoded bitmap indices show the best performance for processing *equality queries* such as $velocity = 108$. *Range encoding* and *interval encoding* are optimized for *one-sided* and *two-sided range queries*, respectively. An example of a one-sided range query is $density < 10^9$. A two-sided range query, for instance, is $10^8 < density < 10^9$.

The authors of [22] 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 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 [2, 9]. The authors demonstrated that the scheme named Byte-aligned Bitmap Code (BBC) [3] shows the best overall performance characteristics. More recently a new compression scheme called Word-Aligned Hybrid (WAH) [19] was introduced. This compression algorithm significantly reduces the overall query processing time compared to BBC. The main 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 (low number of distinct values). However, scientific applications are often based on data values with high attribute cardinalities. The work presented in [17] 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 [21]. 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 [10] focuses on one-dimensional point (equality) queries rather than range queries discussed in this paper. We extend the work of [10] by analyzing multi-dimensional range queries.

Binning strategies could also be used to provide histogram information. The optimal construction of histograms for range queries that uses binning algorithms and is discussed in [11, 8]. The main difference between binning for histograms and our work is that for bitmap indices precise answers are required. 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 original data on disk.

3 Choosing Optimal Bin Boundaries for Single-Attribute Case

The *OptBin* problem for the single attribute case is defined as follows:

Assume a dataset D with one attribute in the range of [1,n], a set of range queries Q and a constraint k on the number of bins. Find a set of $k-1$ optimal bin boundaries such that the query processing costs, i.e. the costs of the candidate check, are minimized.

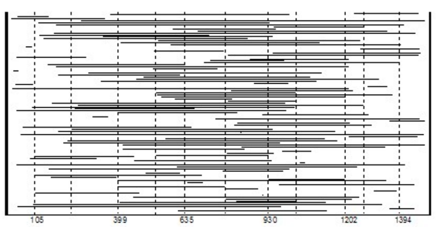
This problem is solved with a dynamic programming algorithm introduced in [14]. Its time complexity is $O(kr^2)$ where r is the number of distinct query endpoints of queries in Q and k is the constraint on the number of bins. As expected, the amount of cost savings (in terms of reduced I/O for candidate check) achieved by the algorithm increases with the degree of accuracy of our estimations of data and query distributions. Fortunately, as bitmaps are mainly used for read-only data, histograms representing data distribution information can be collected at a minimal cost during bitmap index construction. Our experience with scientific data also shows that query distribution can be collected effectively by analyzing workload traces and understanding the kind of phenomena the scientists are studying.

We will use an example to show that the optimal binning strategy depends both on the *data distribution* and the *query distribution*. The *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. The *query distribution* characterizes the location of query endpoints and the popularity of queries. The goal is to align bin boundaries with the query endpoints and thus reduce the number of candidates in the edge bins. In addition, more bins can be allocated to data regions that are heavily hit by queries.

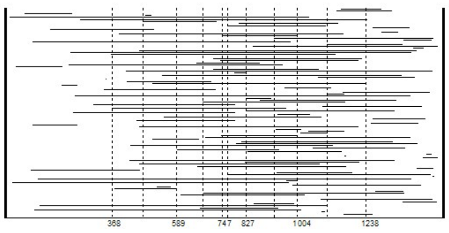
In the example we assume a bitmap index with 12 bins. Using the dynamic programming algorithm introduced in [14], we calculated the optimal bin boundaries for a one-dimensional attribute using 100 simulated range queries.

In the first example both the data and the query distribution generated from a uniform distribution. In Figure 2 (a) the values of the data and the query endpoints fall in the range [0, 1500]. Next we calculated the optimal bin boundaries. The result is a set of bins where the width of each bin is approximately the same size.

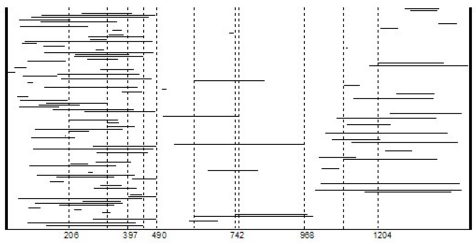
In our second example we fixed the query distribution but changed the data distribution. In Figure 2 (b) the values of the attribute are Gaussian distributed (truncated in the range [0, 1500]) with mean 750 and standard deviation 230. Again we calculated the optimal bin boundaries. We note that the bin sizes vary showing wider bins on the edges of the range to reflect more sparsely popu-



(a) Uniform range queries on *uniform* data



(b) Uniform range queries on *Gaussian* data



(c) Left-heavy range queries on *Gaussian* data

Fig. 2. Optimal bin boundaries for various data and query distributions. The horizontal lines represent range queries, e.g. $350 \le A < 1201$. The vertical lines indicate the optimal bin boundaries that are calculated using our dynamic programming algorithm taking into account both the data and the query distribution.

lated subregions. In short, the width of the bins changes depending on the data distribution.

In our third example (see Figure 2 (c)) 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. The optimal bin boundaries are characterized as follows. 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).

4 The Multi-Dimensional Candidate Check Problem

The general *MultiOptBin* problem is defined as follows:

Given a multi-dimensional dataset D, a set of range queries Q and a constraint k on the total number of bins, find t integers $k_1, k_2, ..., k_t$ *where* $k =$

$\sum_{i=1}^{t} k_i$ *and locations for bin boundaries such that* k_i *bins are allocated for the bitmap index for attribute* A_i *and the total expected I/O cost of candidate check is minimized.*

The Multi-Dimensional candidate check problem is much more complex then the single attribute case as several new factors are introduced. First, before we can deal with bin boundary placement, we need to decide how many bins must be allocated for each attribute. This in turn is dependent on several factors such as the likelihood of an attribute to appear in a query as well as its selectivity.

In general, the total cost of multi-dimensional candidate check is a weighted sum of candidate checks costs, $cost(A_i)$, for each attribute A_i appearing in the query. The weights depend on attribute selectivities as each candidate check results in pruning of the potential candidate subset. We only need to check candidates that survived all previous prunings.

The candidate check cost $cost(A_i)$ for each attribute A_i is a non-increasing function of k_i, the number of bins allocated to A_i. Unfortunately, in the general case the function is not known and depends on the data and query distribution. It is therefore not surprising that the exact optimal solution to *MultiOptBin* is NP-hard problem as shown in the next theorem.

Theorem 1. *The MultiOptBin is NP-hard even if all queries in* Q *include a range for only one attribute and all* s_i*'s (attribute selectivities) are equal.*

The proof of this theorem can be found in [15].

In our experiments we achieved a significant speed-up over naive bin allocations by using a sub-optimal strategy. The strategy consists of allocating a fixed number of bins per attribute and then computing optimal bin boundaries by applying the dynamic-programming algorithm separately on each attribute.

A closed-form solution to the multi-dimensional bin allocation problem can be computed if all $cost(A_i)$'s are differentiable functions under a probabilistic model of query distributions. In Section 5 we show how this solution is computed.

5 Probabilistic Query Model

A different query model that appears in many works on multi-dimensional range queries (see for example [1]) assumes that the query set Q is described by a probabilistic model. For a query q, we denote by $A_i \in q$ the fact that a range corresponding to attribute A_i appears in q. This model assumes that the probability of an attribute appearing in a query, denoted by p_i, depends on the attribute itself but is independent of the other attributes appearing in the query, i.e, the probability p_q that a query q is submitted to the system satisfies $p_q = \left[\prod_{A_i \in q} p_i\right]\left[\prod_{A_i \notin q} (1-p_i)\right]$.

Let us renumber the attributes such that $p_1 \leq p_2 \leq ... \leq p_t$. In this case it is intuitive not to allocate the same number of bins to each attribute but rather to favor attributes with larger probabilities by allocating more bins to them. The exact allocation also depends on the values of the s_i's (*CC-selectivities*) of the attributes. The following theorem quantifies this observation for the case that the s_i's are unknown and assumed equal, i.e., $s_i = s$ for $1 \leq i \leq t$. We also assume that $cost(A_i)$, the cost of candidate check on attribute A_i, is inversely proportional to k_i the number of bins allocated to it and proportional to the number of candidate values, N_i, that still need to be checked, i.e., $\cos t(A_i) = O(\frac{N_i}{k_i})$. This assumption is intuitively reasonable for attributes with close to uniform data distribution. Increasing the number of bins by a factor f makes each bin (including the edge bins) "narrower" by the same factor. Therefore the I/O cost for the candidate check is reduced approximately by a factor of $\frac{1}{f}$. In fact our next closed form result can be applied to any expression for the function $cost(A_i)$ as long as it is differentiable in terms of k_i.

Theorem 2. *Given a query set on t attributes each with CC-selectivity s and probability $p_1 \leq p_2 \leq ... \leq p_t$.*

The optimal bin allocation satisfies:

$$\frac{k_{i+1}}{k_i} = \sqrt{\frac{p_{i+1}}{p_i(1 - p_{i+1}(1 - s))}}, for 1 \leq i < t \tag{1}$$

Proof (Outline): Let $C(j, m)$ denote the expected cost of candidate check on attributes $A_1, A_2, .., A_j$ with m available bins on a database of N records. (For simplicity we omit N from the notation). Then

$$C(t, k) = p_t(\frac{1}{k_t} + C(t - 1, k - k_t)s) + (1 - p_t)C(t - 1, k - k_t) \tag{2}$$

The first term in the above equation corresponds to the probability of attribute A_t appearing in a random query. In this case we will search its bitmap at a cost proportional to $\frac{1}{k_t}$. We then have $k - k_t$ bins for the rest of the index on attributes, and therefore we will pay the cost of $C(t-1, k-k_t)$ on the rest of the index on attributes $A_1, A_2, .., A_{t-1}$ with a candidate pool that is a fraction s of the original one. The second term denotes the cost in case attribute A_t is not mentioned in the query.

This leads recursively to the expression

$$C(t, k) = \sum_{i=1}^{t} \frac{p_i}{k_i} \prod\nolimits_{j=i+1}^{t} (1 - p_j(1 - s)) \tag{3}$$

The result of the theorem follows by finding the minimum of $C(t, k)$ subject to the constraint $k = \sum_{i=1}^{t} k_i$ using Lagrange multiplier techniques. This involves solving the set of equations

$$\frac{\partial}{\partial k_i} \left[\sum_{i=1}^{t} \frac{p_i}{k_i} \prod\nolimits_{j=i+1}^{t} (1 - p_j(1 - s)) + \lambda(\sum_{i=1} k_i - k) \right] = 0 \tag{4}$$

for i=1,2,..,t.

$$k = \sum_{i=1}^{t} k_i \tag{5}$$

□

To illustrate the above theorem, we show bin allocations of 1000 bins among four attributes with probabilities 0.1, 0.2, 0.7 and 0.8 (see Figure 3). For all attributes we assume the same selectivity s. We show the optimal bin allocation for s ranging between 0 and 1. For example, for selectivity $s = 0.4$, the attributes A_1 to A_4 get the following number of bins allocated 85, 129, 317 and 469 (see dashed line). Also note that with smaller values of s, the ratio between the number of bins allocated to A_4 and A_3 is larger. Further experimental results are given in Section 7.

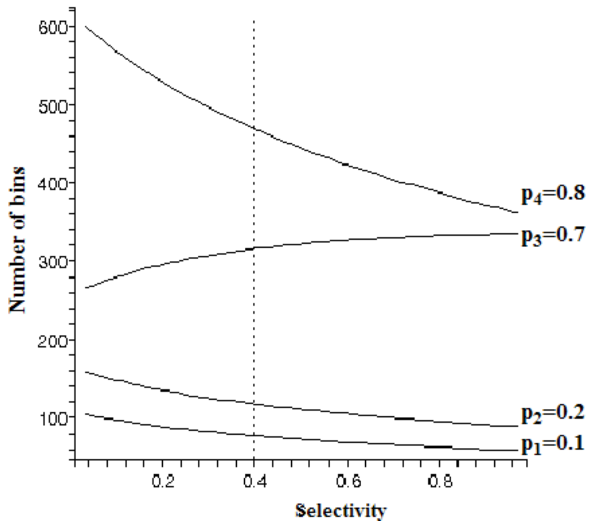


Fig. 3. Optimal bin allocation as a function of candidate selectivity.

6 Query Evaluation With Attribute Reordering

Assuming that bitmap indices are already provided for each attribute, we proceed to show that additional speed-up can be achieved by reordering the scanning of the bitmap indices for attributes appearing in the query.

For a given multi-dimensional query q that involves t attributes there are $t!$ different ways of ordering the attributes for the candidate elimination step of the strategy presented in [17]. In this section we show that an optimal attribute order can be obtained if we have some estimation of the ratio of surviving candidates after performing candidate check on each attribute A_i. This ratio is denoted by s_i (*CC-selectivity*), it can be estimated by the degree of overlap of the query q with its edge bins of A_i and some knowledge of the data distribution. We also need some estimation of the I/O costs incurred by performing the candidate check on attribute A_i which we denote by $cost(A_i)$.

Theorem 3. *Given a query $q = \bigcap_{i=1}^{t} r_i$ assume the I/O cost involved in candidate checking for range r_i is $cost(A_i)$ and the fraction of records surviving the candidate check on this range (CC-selectivity) is s_i.*

We assume that for all i , $0 < s_i < 1$ thus omitting the trivial cases where for some range r_i either $s_i = 0$ or $s_i = 1$. Using the notation $g_i = \frac{cost(A_i)}{1-s_i}$, the optimal ordering of attributes for candidate check evaluation is according to the sorted non-decreasing order of g_i 's.

Our bitmap index software *FastBit* provides two mechanisms for estimating the query selectivity. One approach is to estimate the selectivity based on the cost model presented in [17] Another approach is retrieve the selectivity information directly from the bitmap indices by counting the number of hits during the candidate check phase.

7 Experimental Results

In this section we evaluate the performance of multi-dimensional queries that are optimized with our novel bitmap index strategies. The performance measurements are based on two different data sets from applications that are used in production. This guarantees that we evaluate our strategy on large real data sets as opposed to simplified synthetic data sets.

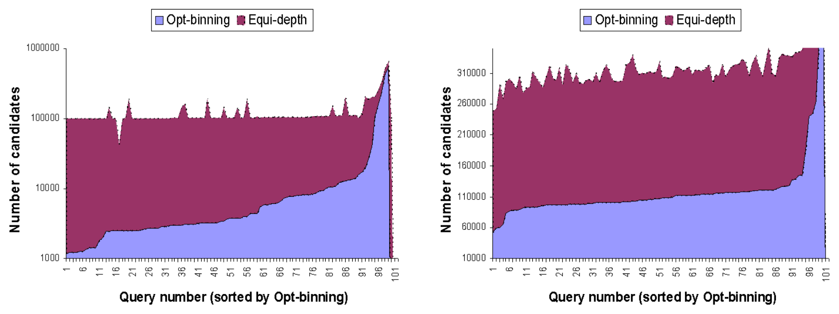
The first data set contains network traffic data that was collected at Berkeley Lab in May 2005. The second data set is based on a supernova explosion from the Tera Scale Initiative [18]. The goal of our experiments is to compare our binning approach with one of the most commonly used binning strategies in production environments that is called *Equi-depth*. This binning strategy places the bin boundaries in such a way that each bin has approximately the same number of entries. The advantage of equi-depth binning is that it reduces the worst-case query processing costs.

As we have pointed out in the introductory section, the bottleneck of bitmap indices with bins is the *candidate check*. In other words, the higher the number of candidates that need to be checked against the query constraint, the higher the query processing costs. Thus, in this section we use the term *query processing costs* as a synonym for *candidate check costs*.

7.1 Network Traffic Data

Data Characteristics The network traffic data set we used for our experiments contains incoming and outgoing network traffic to and from Berkeley Lab. The data set includes attributes such as *DestinationIP*, *SourceIP*, *DestinationPort*, *SourcePort*, *SourceBytesPerPacket*, *DestinationBytesPerPacket*, *StartTime*, etc. The data set contains 10.2 million records and 8 attributes.

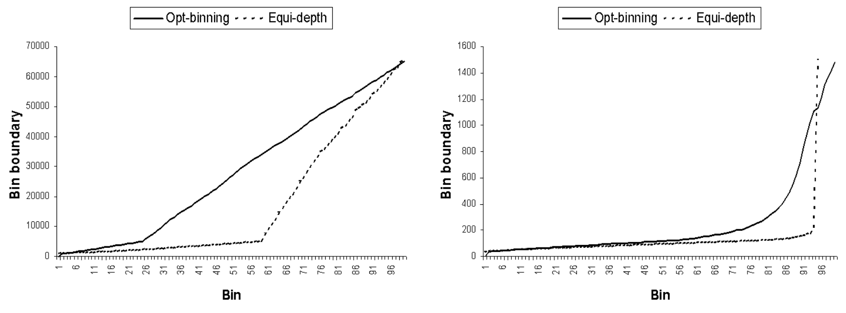
Query Processing Costs For our experiments we generated 1,000 random uniformly distributed queries that cover the whole domain space for each attribute. For these 1000 queries we ran our optimization algorithm and calculated the optimal bin boundaries for 100 bins per attribute. Next we built the bitmap indices accordingly. Figure 4 shows the query processing costs for two attributes for the binning strategies *Opt-binning* and *Equi-depth* binning. The costs are expressed in terms of the number of candidates that need to be accessed in the candidate check. The graphs show the performance of 100 randomly sampled queries that are sorted according to the costs of *Opt-binning*. For all measured attributes, the processing costs of *Opt-binning* are, on average, a factor of 3 smaller compared with *Equi-depth*.



(a) *DestinationPort* (b) *SourceBytesPerPacket*

Fig. 4. Query processing costs for attribute *DestinationPort* and *SourceBytesPerPacket*. Note: The query processing costs are proportional to the area of the graph. On average, the processing costs of *Opt-binning* are a factor of 3 smaller than for *equi-depth binning*.

Location of Bin Boundaries To show the effect of *Opt-binning* on the location of the bin boundaries, we plotted the boundaries of *Opt-binning* and compared them with *Equi-depth* binning (see Figure 5). For instance, for attribute *DestinationPort* in Figure 5 (a), the bin boundaries change significantly starting from bin 25. *Equi-depth* binning keeps the bin boundaries equally sized up to bin 60, whereas *Opt-binning* changes the bin boundaries already with bin 25 to reflect the query distributions.



(a) *DestinationPort* (b) *SourceBytesPerPacket*

Fig. 5. Bin boundaries for attribute (a)*DestinationPort* and (b) *SourceBytesPerPacket*.

Multi-Dimensional Queries Next, we measured the query processing costs for multi-dimensional queries. Figure 6 shows the cost improvement factor of *Opt-binning* over *Equi-depth*. For 4-dimensional queries, the cost improvement of *Opt-binning* over *equi-depth binning* is about a factor of 9. We can also see that as the number of query dimensions increases, the cost improvement increases as well. This is due the fact that our strategy *Opt-binning* does a dynamic *cost-based* reordering of the attributes for multi-dimensional queries as shown in Theorem 3.

Optimal Number of Bins In the next experiment we evaluated our novel optimization strategy for further reducing the query processing costs. In our previous experiments we used the same number of bins for each attribute. Now we calculate the optimal number of bins for each attribute depending on the probability of being contained in a multi-dimensional query expression. This optimization strategy is motivated by the observations we made during the analysis of real query workloads [14]. In multi-dimensional queries the number of attributes per query often changes. Thus, different attributes often show different probabilities of being contained in a certain query expression. The key idea of calculating the optimal number of bins is to increase the number of bins for those attributes that have a higher probability of being contained in a set of queries. For attributes that have a lower probability, the number of bins is reduced.

In order to study this strategy we generated 5000 queries with up to four dimensions. For each attribute we assume a different probability of being contained in the query expression. Given these probabilities, the optimal number of bins is calculated according to Equation 1. Table 2 shows the probabilities and the respective optimal number of bins for the four attributes of our query workload.

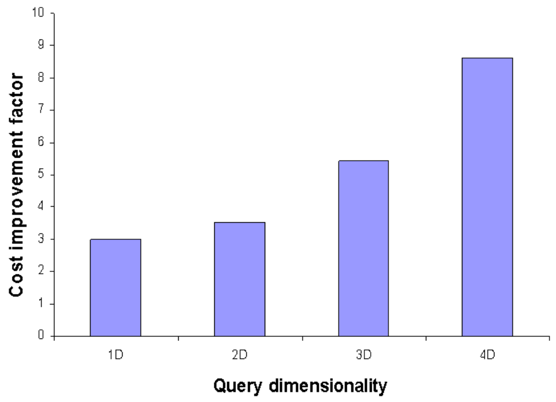


Fig. 6. Cost improvement factor of *Opt-binning* over *equi-depth binning* for multi-dimensional queries. For 4-dimensional queries, the cost improvement of *Opt-binning* over *equi-depth binning* is about a factor of 9.

Attribute	**Probability**	**Opt. #bins**
SourcePort	0.8	199
DestinationPort	0.7	123
StartTime	0.2	47
SrcBytesPerPacket	0.1	31

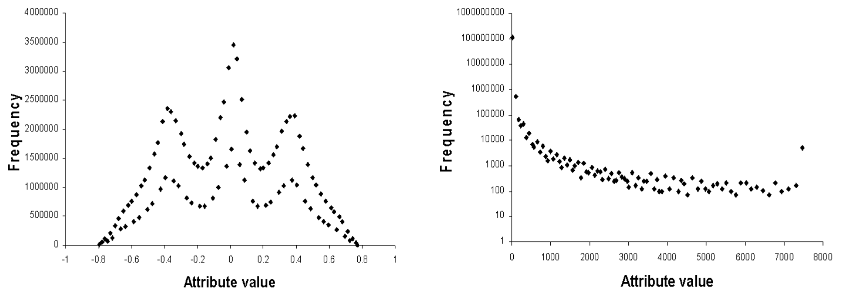
Table 2. Probabilities for attributes to be contained in a query expression along with the respective optimal number of bins.

Next we computed the optimal bin boundaries where each attribute has a different number of bins (as shown in Table 2). In addition, we computed the optimal bin boundaries where each attribute has 100 bins (as we did in our previous experiments). The experiments show that calculating the optimal number of bins reduces the I/O costs by another 30% compared with the optimal strategy where each attribute has the same number of bins.

7.2 Astrophysics Data

The astrophysics data set is one order of magnitude larger than the network traffic data set. The challenge was to measure how *Opt-binning* performs for this large data set of 110 million records and 6 attributes such as *x-velocity*, *y-velocity*, *z-velocity*, *density*, *pressure* and *entropy*. The data distributions of two of the attributes are shown in Figure 7.

Since the number of records of this data set is much larger than the previous one, we increased the number of bins to 1000. We also increased the number of



(a) *x-velocity*　　(b) *density*

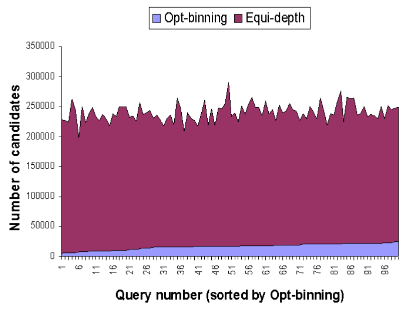
Fig. 7. Distribution of attributes (a) *x-velocity* and (b)*density*.

queries to 5000. In addition, we made an important change to the query distribution according to the following observation. In our previous work we studied the work load of real queries from the Sloan Digital Sky Survey [14] and High-Energy Physics [20]. The main observation is that the query distributions are often not uniform but centered around either the peaks of the data distribution or around the tails. Thus, we experimented with different query distributions that follow these basic rules.

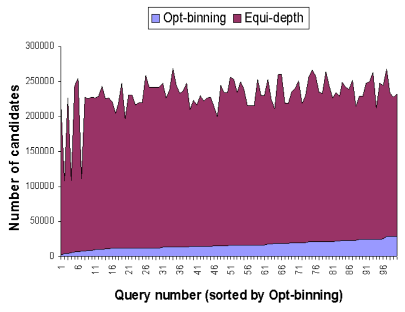
The distribution of attribute *x-velocity* is centered around 0 (see Figure 7 (a)). For this attribute we generated 5000 queries that are centered around 0 and calculated the optimal bin boundaries for 1000 bins accordingly. Figure 8 (a) shows that the query processing costs for *Opt-binning* are about a factor of 13 lower than the costs for *Equi-depth*.

Next, we generated 5000 queries for attribute *y-velocity* that has a similar distribution as attribute *x-velocity*. However, rather than centering the query distribution around 0, we produced a right-skewed distribution where most of the queries hit the right side (tail) of the data. Also for this kind of query workload, *Opt-binning* reduces the query processing costs by more than a factor of 10 compared with *Equi-depth* binning (see Figure 8 (b)).

Multi-Dimensional Queries Finally, we measured the query processing costs for multi-dimensional queries on two and three attributes. Figure 9 shows the cost improvement factor of *Opt-binning* over *Equi-depth* which is between 11.5 and 17.5. Similar to the network traffic queries, we can see that as the number of query dimensions increases, the cost improvement of *Opt-binning* increases even more significantly.



(a) *x-velocity*



(b) *y-velocity*

Fig. 8. Query processing costs for attribute *x-velocity* and *y-velocity*. Note: The query processing costs are proportional to the area of the graph. On average, the processing costs of *Opt-binning* are a factor of 10 to 13 smaller than for *equi-depth binning*.

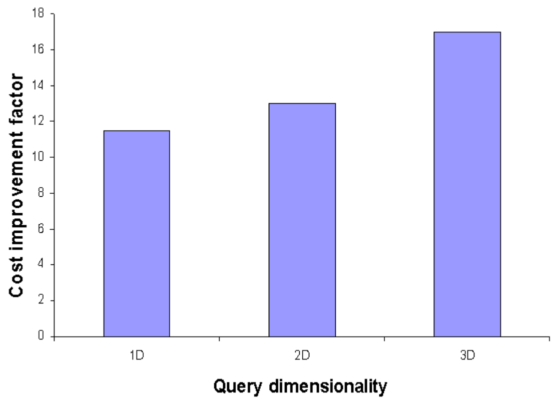


Fig. 9. Cost improvement factor of *Opt-binning* over *Equi-depth* for multi-dimensional queries. For 3-dimensional queries, the cost improvement of *Opt-binning* over *equi-depth binning* is about a factor of 17.5.

8 Conclusions

In this paper we presented an algorithm for optimizing the costs of multi-dimensional queries with bitmap indices. Our approach is based on the following two steps: a) Given a set of data distributions and a set of query distributions, find an optimal placement for the bin boundaries such that the number of candidates that need to be checked against the query constraints is minimized. b)

Reorder the evaluation of the attributes in multi-dimensional queries according to the estimated attribute selectivity.

We performed both analytical and experimental studies to evaluate the efficiency of our strategy. Our experiments were based on two data sets from applications that are used in production. For the analyzed data sets we achieved a performance improvement in the range of 3 to 17. The results show that as the number of query dimensions increases, the efficiency of our algorithm increases as well. The increased performance improvement is due to the cost-based reordering of query attributes that significantly reduces the number of candidate records and thus the total cost for query processing.

Future work involves testing our optimization techniques against other binning strategies. A further direction of future work is to design bin allocation algorithms for probabilistic queries with attribute dependencies.

References

1. A. V. Aho and J. D. Ullman. Optimal Partial-Match Retrieval When Fields Are Independently Specified. *ACM Trans. Database Syst.*, 4(2):168–179, 1979.
2. S. Amer-Yahia and T. Johnson. Optimizing Queries on Compressed Bitmaps. In *International Conference on Very Large Data Bases (VLDB)*, Cairo, Egypt, September 2000. Morgan Kaufmann.
3. G. Antoshenkov. Byte-aligned Bitmap Compression. Technical report, Oracle Corp., 1994. U.S. Patent number 5,363,098.
4. C.-Y. Chan and Y. E. Ioannidis. Bitmap Index Design and Evaluation. In *SIGMOD*, Seattle, Washington, USA, June 1998. ACM Press.
5. C. Y. Chan and Y. E. Ioannidis. An Efficient Bitmap Encoding Scheme for Selection Queries. In *SIGMOD*, Philadelphia, Pennsylvania, USA, June 1999. ACM Press.
6. S. Chaudhuri and U. Dayal. An Overview of Data wharehousing and OLAP Technology. *ACM SIGMOD Record*, 26(1):65–74, March 1997.
7. FastBit - An Efficient Compressed Bitmap Index Technology. http://sdm.lbl.gov/fastbit/.
8. S. Guha, N. Koudas, and D. Srivastava. Fast Algorithms For Hierarchical Range Histogram Construction. In *PODS 2002*, Madison, Wisconsin, USA, June 2002. ACM Press.
9. T. Johnson. Performance Measurements of Compressed Bitmap Indices. In *International Conference on Very Large Data Bases (VLDB)*, Edinburgh, Scotland, September 1999. Morgan Kaufmann.
10. N. Koudas. Space Efficient Bitmap Indexing. In *International Conference on Information and Knowledge Management (CIKM)*, McLean, Virginia, USA, November 2000. ACM Press.
11. N. Koudas, S. Muthukrishnan, and D. Srivastava. Optimal Histograms for Hierarchical Range Queries. In *PODS*, Dallas, Texas, USA, 2000. ACM Press.
12. P. O'Neil. Model 204 Architecture and Performance. In *2nd International Workshop in High Performance Transaction Systems*, Asilomar, California, USA, 1987. Springer-Verlag.
13. P. O'Neil and D. Quass. Improved Query Performance with Variant Indexes. In *Proceedings International Conference on Management of Data (SIGMOD)*, Tucson, Arizona, USA, May 1997. ACM Press.

14. D. Rotem, K. Stockinger, and K. Wu. Optimizing Candidate Check Costs for Bitmap Indices. In *International Conference on Information and Knowledge Management (CIKM)*, Bremen, Germany, November 2005. ACM Press.
15. D. Rotem, K. Stockinger, and K. Wu. Optimizing I/O Costs of Multi-Dimensional Queries using Bitmap Indices. In *International Conference on Database and Expert Systems Applications (DEXA)*, Kopenhagen, Denmark, August 2005. Springer-Verlag.
16. 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.
17. 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.
18. TeraScale Supernova Initiative. http://www.phy.ornl.gov/tsi/.
19. 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 (VLDB)*, Toronto, Canada, September 2004. Morgan Kaufmann.
20. 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.
21. K.-L. Wu and P.S. Yu. Range-Based Bitmap Indexing for High Cardinality Attributes with Skew. In *COMPSAC*, pages 61–67, 1998.
22. M.-C. Wu and A. P. Buchmann. Encoded Bitmap Indexing for Data Warehouses. In *International Conference on Data Engineering (ICDE)*, Orlando, Florida, USA, February 1998. IEEE Computer Society Press.