Indexing the Edges – A simple and yet efficient approach to high-dimensional indexing

Beng Chin Ooi Kian-Lee Tan Cui Yu Stephane Bressan
Department of Computer Science
National University of Singapore
3 Science Drive 2, Singapore 117543
{ooibc,tankl,yucui,steph}@comp.nus.edu.sg

ABSTRACT

In this paper, we propose a new tunable index scheme, called iMinMax(θ), that maps points in high dimensional spaces to single dimension values determined by their maximum or minimum values among all dimensions. By varying the tuning "knob" θ, we can obtain different family of iMinMax structures that are optimized for different distributions of data sets. For a d-dimensional space, a range query need to be transformed into d subqueries. However, some of these subqueries can be pruned away without evaluation, further enhancing the efficiency of the scheme. Experimental results show that iMinMax(θ) can outperform the more complex Pyramid technique by a wide margin.

1. INTRODUCTION

Many multi-dimensional indexing structures have been proposed in the literature (see [1] for a survey). In particular, it has been observed that the performance of hierarchical tree index structures such as R-trees [2] and R*-trees [3] deteriorates rapidly with the increase in the dimensionality of data. This phenomenon is caused by two factors. First, let us consider the fan-out of internal nodes in an R-tree. Suppose we conform to the classic definition of an R-tree where all nodes are of a fixed size B. The fan-out of an internal node is clearly bounded by

$$\left\lfloor \frac{B}{\sum_{i=1}^{d}(2 \cdot s_i)} \right\rfloor$$

where s_i is the size of data elements corresponding to dimension i. (The expression $\sum_{i=1}^{d}(2 \cdot s_i)$ constitutes the storage needed to define a minimal bounding region in a d-dimensional space.) Clearly, the fan-out of an R-tree is inversely proportional to the dimensionality of data objects (d). The smaller fan-out contributes not only to increased overlap between node entries but also the height of the corresponding R-tree. One obvious solution is to reduce the amount of overlap by increasing the fan-out of selected nodes. Notwithstanding, the size of a node cannot be enlarged indefinitely, since any increase in node size contributes ultimately to additional page accesses and CPU cost, causing the index to degenerate into a semi-sequential scan within an index. Second, as the number of dimensions increases, the area covered by the query increases tremendously. Consider a hyper-cube with a selectivity of 0.1% of the domain space ([0,1],[0,1],...,[0,1]). This is a relatively small query in two to three-dimensional databases. However, for a 40-dimensional space, the query width along each dimension works out to be 0.841, which causes the query to cover a large area of the domain space. Consequently, many leaf nodes of a hierarchical index have to be searched. The above two problems are so severe that the performance is worse off than a simple sequential scan of the index keys [4] [5]. However, sequential scanning is expensive as it requires the whole database to be searched for any range queries, irrespective of query sizes. Therefore, research efforts have been driven to develop techniques that can outperform sequential scanning. Some of the notable techniques include the VA-file [4] and the Pyramid scheme [5].

This paper adopts a slightly different approach that reduces high-dimensional data to a single dimensional value. It is motivated by two observations. First, data points in high dimensional space can be ordered based on the maximum value of all dimensions. (We have adopted the maximum value in our discussion. However, similar observations can be made with the minimum value.) Second, if an index key does not fit in any query range, the data point will not be in the answer set. The former implies that we can represent high-dimension data in single dimensional space, and reuse existing single dimensional indexes. The latter provides a mechanism to prune the search space.

In this paper, we propose a new tunable indexing scheme, called iMinMax(θ), that addresses the deficiency of the simple approach discussed above. iMinMax has several nice features. First, iMinMax(θ) adopts a simple transformation function to map high dimension points to a single dimension space. Let x_{min} and x_{max} be respectively the smallest and largest values among all the d dimensions of the data point $(x_1, x_2, \ldots, x_d)$ $0 \leq x_j \leq 1, 1 \leq j \leq d$. Let the corresponding dimension for x_{min} and x_{max} be d_{min} and d_{max} respectively. The data point is mapped to y over a single

POD 2000, Dallas, TX USA

dimensional space as follows:

$$y = \begin{cases} d_{min} + x_{min} & \text{if } x_{min} + \theta < 1 - x_{max} \\ d_{max} + x_{max} & \text{otherwise} \end{cases}$$

We note that the transformation actually partitions the data space into different partitions based on the dimension which has the largest value or smallest value, and provides an ordering within each partition. Second, B^+-tree is used to index the transformed values. Thus, iMinMax(θ) can be implemented on existing DBMSs without additional complexity, making it a practical approach.

Third, in iMinMax(θ), queries on the original space need to be transformed to queries on the transformed space. For a given range query, the range of each dimension is used to generate a range subqueries on the dimension. The union of the answers from all subqueries provides the candidate answer set from which the query answers can be obtained. iMinMax's query mapping function facilitates effective range query processing: (i) the search space on the transformed space contains all answers from the original query, and it cannot be further constrained without the risk of missing some answers; (ii) the number of points within a search space is reduced; and (iii) some of the subqueries can be pruned away without being evaluated.

Finally, by varying θ, we can obtain different families of iMinMax(θ) structures. At the extremes, iMinMax(θ) maps all high dimensional points to the maximum (minimum) value among all dimensions; alternatively, it can be tuned to map some points to the maximum values, while others to the minimum values. Thus, iMinMax(θ) can be optimized for data sets with different distributions. Unlike the Pyramid technique, no apriori knowledge or reconstruction is required.

We implemented the iMinMax(θ) and evaluated its performance against the more complex Pyramid technique. Our experimental results on both uniform and skewed data sets show that the proposed scheme can be more efficient than the Pyramid technique by more than 50% of retrieval costs.

The rest of our paper is organized as follows: In Section 2, we describe existing work. In Section 3, we shall present iMinMax(θ) in detail, and in Section 4, the search algorithms. Section 5 presents the experimental study and reports our findings. We conclude in Section 6.

2. RELATED WORK

There is a considerable amount of work on high-dimensional indexing. In this section, we shall review two of the recently proposed indexing structures, namely the Pyramid technique [5] and the VA file [4]. While the Pyramid technique is used for performance comparison in this paper, the VA-file is a potential candidate for further performance study.

The basic idea of the Pyramid Technique is to transform d-dimensional data points into 1-dimensional values, and then store and access the values using a conventional index such as the B^+-tree.

There are mainly two steps in its partitioning method. First, it splits the data space into $2d$ pyramids, which share the center point of the data space as their top and have $(d-1)$-dimensional surface of the data space as their base.

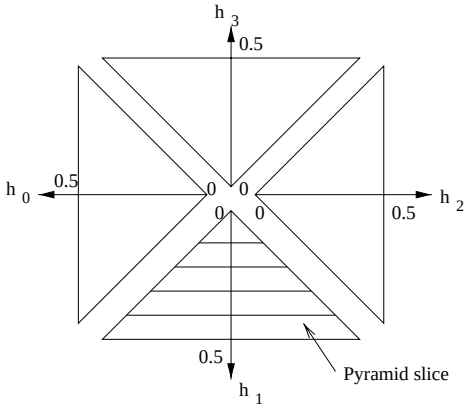


Figure 1: Index key assignment in the Pyramid technique

All points located on the i-th $(d-1)$-dimensional surface of cube (the base of the pyramid) have the common property: either their i-th coordinate is 0 or their $(i-d)$-th coordinate is 1. On the other hand, all points ν located in the i-th pyramid p_i have the furthest distance from the top on the i-th dimension. The dimension in which the point has the longest distance from the top determines which pyramid the point lies.

Another property of Pyramid Technique is that the location of a point ν within its pyramid is indicated by a single value, which is the distance from the point to the center point according to dimension j_{max} (see Figure 1). The data on the same slice in a pyramid have the same pyramid value. That is, any objects fall on the slice will be represented by the same pyramid value. As a result, many points will be indexed by the same key in a skewed distribution. It has been suggested that the center point can be shifted to handle data skewness. However, this incurs recalculation of all index values, i.e. redistribution of the points among the pyramids, and reconstruction of the B^+-tree.

To retrieve a point q, the pyramid value P_ν of q is computed, and used to serach the B^+-tree. All points with P_ν will be checked and retrieved. To perform a range query, the pyramids that intersect the search region are determined, and for each pyramid, individual subquery range is determined. Each subquery is used to search the B^+-tree. For each range query, $2d$ subqueries may be required, one against each pyramid.

The VA-file (vector approximation file) [4] is based on the idea of object approximation by mapping a coordinate to some value that reduces storage requirement. The basic idea is to divide the data space into 2^b hyper-rectangular cell where b is the tunable number of bits used for representation. For each dimension i, b_i bits are used, and 2^{b_i} slices are generated in such a way that all slices are equally full. The data space consists of 2^b hyper-rectangular cell, each of which can be represented by a unique bit string of length b. A data point is then approximated by the bit string of the cell it falls into.

To perform a point or range query, the entire approximation file must be sequentially scanned. Objects whose bit string satisfies the query must be retrieved and checked. Typically, the VA-file is much smaller than the vector file and hence is far more efficient than direct sequential scan of data file and the variants of R-tree. However, the performance of the VA-file is likely to be affected by data distributions and hence the false drop rate; the number of dimensions and the volume of data.

3. INDEXING ON THE EDGES

In a multi-dimensional range search, all values of all dimensions must satisfy the query range along each dimension. If any of them fails, the data point will not be in the answer set. Based on this observation, a straightforward approach is to index on a small subset of the dimensions. However, the effectiveness of such an approach depends on the data distribution of the selected dimensions. Our preliminary study on indexing one single dimension showed that the approach can perform worse than sequential scanning. This led us to examine novel techniques that index on the "edges". An "edge" of a data point refers to the maximum or minimum value among all the dimensions of the point. The proposed technique, iMinMax(θ) uses either the values of the *Max edge* (the dimension with the maximum value) or the values of the *Min edge* (the dimension with the minimum value) as the representative index keys for the points. Because the transformed values can be ordered and range queries can be performed on the transformed (single dimensional) space, we can employ single dimensional indexes to index the transformed values. In this paper, we employ the B$^+$-tree structure since it is supported by all commercial DBMSs. Thus, iMinMax(θ) can be readily adopted for use.

In the following discussion, we consider a unit d-dimensional space, i.e., points are in the space ([0,1],[0,1],...,[0,1]). We denote an arbitrary data point in the space as $x = (x_1, x_2, \ldots, x_d)$. Let $x_{max} = \max_{i=1}^{d} x_i$ and $x_{min} = \min_{i=1}^{d} x_i$ be the maximum value and minimum value among the dimensions of the point. Moreover, let d_{max} and d_{min} denote the dimensions at which the maximum and minimum values occur. Let the range query be $q = ([x_{11}, x_{12}], [x_{21}, x_{22}], \ldots, [x_{d1}, x_{d2}])$. Let $ans(q)$ denote the answers produced by evaluating a query q. In the following discussion, we shall present the transformation function that maps a point in a d dimensional space to a single dimensional space, and discuss how range queries can be evaluated.

3.1 Mapping High Dimensional Data to Single Dimension Space

iMinMax(θ) adopts a simple mapping function that is computationally inexpensive. The data point x is mapped to a point y over a single dimensional space as follows:

$$y = \begin{cases} d_{min} + x_{min} & \text{if } x_{min} + \theta < 1 - x_{max} \\ d_{max} + x_{max} & \text{otherwise} \end{cases}$$

where θ is a real number.

First, we note that θ plays an important role in influencing the number of points falling on each index hyperplane. In fact, it is the tuning knob that affects the hyperplane an index point should reside. Take a data point (0.2, 0.75) in 2-dimensional space for example, with $\theta = 0.0$, the index point will reside on the Min edge. By setting θ to 0.1 will push the index point to reside on the Max edge. The higher the value of $\theta \geq 0$, the biasness the function is expressing towards the Max edge. When $\theta = 0.1$, the Max edge has the preference of about 10% more. Similarly, we can "favor" the transformation to the Min edge with $\theta < 0$. In fact, at one extreme, when $\theta \geq 1.0$, the transformation maps all points to their Max edge. and by setting $\theta \leq -1.0$, we always pick the value at the Min edge as the index key. For simplicity, we shall denote the former extreme as iMax, the latter extreme as iMin, and any other variation as iMinMax (dropping θ unless its value is critical).

Second, we note that the transformation actually splits the (single dimensional) data space into different partitions based on the dimension which has the largest value or smallest value, and provides an ordering within each partition. This is effected by including the dimension at which the maximum value occurs, i.e., the first component of the mapping function.

Finally, the unique tunable feature facilitates the adaptation of iMinMax(θ) to data sets of different distributions (uniform or skewed). In cases where data points are skewed toward certain edges, we may "scatter" these points to other edges to evenly distribute them by making a choice between d_{min} and d_{max}. Statistical information such as the number of index points can be used for such purpose. Alternatively, one can either use the information regarding data distribution or information collected to categorically adjust the partitioning.

3.2 Mapping Range Queries

Range queries on the original d-dimensional space have to be transformed to the single dimensional space for evaluation. In iMinMax(θ), the original query on the d-dimensional space is mapped into d subqueries — one for each dimension. Let us denote the subqueries as $q_1, q_2, \ldots, q_d$, where $q_i = [l_i, h_i]$ $1 \leq i \leq d$. For the jth query subrange in q, $[x_{j1}, x_{j2}]$, we have q_j as given by the expression in Equation 1.

The union of the answers from all subqueries provides the candidate answer set from which the query answers can be obtained, i.e., $ans(q) \subseteq \cup_{i=1}^{d} ans(q_i)$. We shall now prove some interesting results.

Theorem 1. Under iMinMax(θ) scheme, $ans(q) \subseteq \cup_{i=1}^{d} ans(q_i)$. Moreover, there does not exist $q_i' = [l_i', h_i']$, where $l_i' > l_i$ or $h' < h_i$ for which $ans(q) \subseteq \cup_{i=1}^{d} ans(q_i')$ always holds. In other words, q_i is "optimal" and narrowing its range may miss some of q's answers.

Proof: For the first part, we need to show that any point x that satisfies q will be retrieved by some $q_i, 1 \leq i \leq d$. For the second part, we only need to show that some points that satisfy q may be missed. The proof comprises three parts, corresponding to the three cases in the range query mapping function.

Case 1: $\min_{i=1}^{d} x_{i1} + \theta \geq 1 - \max_{i=1}^{d} x_{i1}$
In this case, all the answer points that satisfy the query q have been mapped to the Max edge, i.e., a point x that

$$q_j = \begin{cases} [j + \max_{i=1}^{d} x_{i1},\ j + x_{j2}] & \text{if } \min_{i=1}^{d} x_{i1} + \theta \geq 1 - \max_{i=1}^{d} x_{i1} \\ [j + x_{j1},\ j + \min_{i=1}^{d} x_{i2}] & \text{if } \min_{i=1}^{d} x_{i2} + \theta < 1 - \max_{i=1}^{d} x_{i2} \\ [j + x_{j1},\ j + x_{j2}] & \text{otherwise} \end{cases}$$

Equation 1: Transformation function for queries.

satisfies q is mapped to x_{max}, and would have been mapped to the d_{max}th dimension, and has index key of $d_{max} + x_{max}$. The subquery range for the d_{max}th dimension is $[d_{max} + \max_{i=1}^{d} x_{i1}, d_{max} + x_{d_{max}2}]$. Since x satisfies q, we have $x_i \in [x_{i1}, x_{i2}]$, $\forall i, 1 \leq i \leq d$. Moreover, we have $x_{max} \geq x_{i1}$ $\forall i, 1 \leq i \leq d$. This implies that $x_{max} \geq \max_{i=1}^{d} x_{i1}$ $\forall i, 1 \leq i \leq d$. We also have $x_{max} \leq x_{d_{max}2}$. Therefore, we have $x_{max} \in [\max_{i=1}^{d} x_{i1},\ x_{d_{max}2}]$, i.e., x can be retrieved using the d_{max}th subquery. Thus, $ans(q) \subseteq \cup_{i=1}^{d} ans(q_i)$.

Now, let $q_i' = [l_i' + \epsilon_l, h_i' - \epsilon_h]$, for some $\epsilon_l > 0$ and $\epsilon_h > 0$. Consider a point $z = (z_1,\ z_2,\ \ldots, z_d)$ that satisfies q. We note that if $l_i < z_{max} < l_i + \epsilon_l$, then, we will miss z if q_i' has been used. Similarly, if $h_i - \epsilon_h < z_{max} < \max_{i=1}^{d} x_{i2}$, then, we will also miss z if q_i' has been used. Therefore, no q_i' provides the tightest bound that guarantees that no points will be missed.

Case 2: $\min_{i=1}^{d} x_{i2} + \theta < 1 - \max_{i=1}^{d} x_{i2}$
This case is the inverse of Case 1, i.e., all points in the query range belongs to the Min edge. As such, we can apply similar logic.

Case 3
In case 3, the answers of q may be found in both the Min edge and the Max edge. Given a point x that satisfies q, we have $x_i \in [x_{i1}, x_{i2}]$, $\forall i, 1 \leq i \leq d$. We have two cases to consider. In the first case, x is mapped to the Min edge, its index key is $d_{min} + x_{min}$, and it is index on the d_{min}th dimension. To retrieve x, we need to examine the d_{min}th subquery, $[d_{min} + x_{d_{min}1},\ d_{min} + x_{d_{min}2}]$. Now, we have $x_{min} \in [x_{d_{min}1}, x_{d_{min}2}]$ (since x is in the answer) and hence the d_{min}th subquery will be able to retrieve x. The second case, which is mapping x onto the Max edge and can be similarly derived. Thus, $ans(q) \subseteq \cup_{i=1}^{d} ans(q_i)$.

Now, let $q_i' = [l_i' + \epsilon_l, h_i' - \epsilon_h]$, for some $\epsilon_l > 0$ and $\epsilon_h > 0$. Consider a point $z = (z_1,\ z_2,\ \ldots, z_d)$ that satisfies q. We note that if $l_i < z_{max} < l_i + \epsilon_l$, then, we will miss z if q_i' has been used. Similarly, if $h_i - \epsilon_h < z_{max} < h_i'$, then, we will also miss z if q_i' has been used. Therefore, no q_i' provides the tightest bound that guarantees that no points will be missed.

□

We would like to point out that in an actual implementation, the leaf nodes of the B$^+$-tree will contain the high-dimensional point, i.e., even though the index key on the B$^+$-tree is only single dimension, the leaf node entries contain the triple (x_{key}, x, ptr) where x_{key} is the single dimensional index key of point x and ptr is the pointer to the data page containing other information that may be related to the high-dimensional point. Therefore, the false drop of Theorem 1 affects only the vectors used as index keys, rather than the actual data itself.

Theorem 2. Given a query q, and the subqueries $q_1,\ q_2,\ \ldots,\ q_d$, q_i need not be evaluated if any of the followings holds:

$$(i)\ \min_{j=1}^{d} x_{j1} + \theta \geq 1 - \max_{j=1}^{d} x_{j1} \text{ and } h_i < \max_{j=1}^{d} x_{j1}$$

$$(ii)\ \min_{j=1}^{d} x_{j2} + \theta < 1 - \max_{j=1}^{d} x_{j2} \text{ and } l_i > \min_{j=1}^{d} x_{j2}$$

Proof: Consider the first case: $\min_{j=1}^{d} x_{j1} + \theta \geq 1 - \max_{j=1}^{d} x_{j1}$ and $h_i < \max_{j=1}^{d} x_{j1}$. The first expression implies that all the answers for q can only be found in the Max edge. We note that the point with the smallest maximum value that satisfies q is $\max_{j=1}^{d} x_{j1}$. This implies that if $h_i < \max_{j=1}^{d} x_{j1}$, then the answer set for q_i will be an empty set. Thus, q_i need not be evaluated.

The second expression means that all the answers for q are located in the Min edge. The point with the largest minimum value that satisfies q is $\min_{j=1}^{d} x_{j2}$. This implies that if $l_i > \min_{j=1}^{d} x_{j2}$, then the answer set for q_i will be empty. Thus, q_i need not be evaluated.

□

Example 1. Let $\theta = 0.5$. Consider the range query ([0.2,0.3], [0.4,0.6]) in 2-dimensional space. Since $0.2 + 0.5 > 1 - 0.4 = 0.6$, we know that all points that satisfy the query falls on the Max edge. This means that the lower bound for the subqueries should be 0.4, i.e., the two subqueries are respectively [0.4,0.3] and [0.4,0.6]. Since the first subquery's upper bound (i.e, 0.3) is smaller than 0.4, it need not be evaluated because no points will satisfy the query.

Theorem 3. Given a query q, and the subqueries $q_1, q_2, \ldots, q_d$, at most d subqueries need to be evaluated.

Proof: The proof is straightforward, and follows from Theorem 2.

□

From theorems 2 and 3 we have a glimpse of the effectiveness of iMinMax(θ). In fact, for very high dimension spaces, we can expect significant savings from the pruning of subqueries.

Example 2. In this example, we illustrate how iMinMax can keep out points from the search space. Figure 2 shows the example. Here, we have two points A(0.2,0.5) and B(0.87,0.25) in 2-dimensional space. If we employ either iMax or iMin, at least one false drop will occur. On the other hand, using iMinMax(0.5) effectively keeps both points out of the search space.

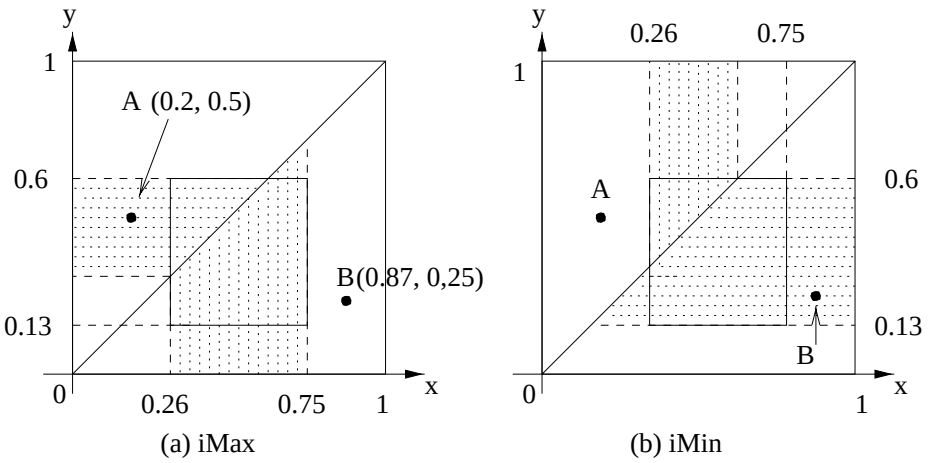


Figure 2: Sample search space for 2-dimensional space.

4. IMINMAX(θ) SEARCH ALGORITHMS

In our implementation of iMinMax(θ), we have adopted the B$^+$-tree [6] as the underlying single dimensional index structure. However, for greater efficiency, leaf nodes also store the high-dimensional key, i.e., leaf node entries are of the form (key, v, ptr) where key is the single dimensional key, v is the high-dimensional vector whose transformed value is key, and ptr is the pointer to the data page containing information related to v. Keeping v at the leaf nodes can minimize page accesses to non-matching points. We note that multiple high-dimensional keys may be mapped to a single key value.

The search, insert and delete algorithms are similar to the B$^+$-tree algorithms. The additional complexity arises as we have to deal with the additional high-dimensional key (besides the single dimensional key value). In this paper, we shall present the search algorithms (for both point and range queries). Insert and delete algorithms are similar to those of B$^+$-tree, and so we omit them.

4.1 Point Search Algorithm

In point search, a point p is issued and all matching tuples are to be retrieved. Suppose θ has been tuned for performance purposes, the maximum and minimum values of p have to be used to search the tree. In this case, a θ that will cause the search to be done on the other edge can be used to call the algorithm.

The algorithm is summarized in Figure 3. Based on θ, the search algorithm first maps p to the single-dimensional key, x_p, using the function **transform(point,** θ**)** (line 1). For each query, the B$^+$-tree is traversed (line 2) to the leaf node where x_p may be stored. If the point does not exist, then a NULL value is returned (lines 3-4). Otherwise, for every matching x_p value, the high-dimensional key of the data is compared with p for a match. Those that match are accessed using the pointer value (lines 7-13); otherwise, they are ignored. We note that it is possible for a sequence of leaf nodes to contain matching key values and hence they all have to be examined. The final answers are then returned (line 14).

4.2 Range Search Algorithm

Range queries are slightly more complicated than point search. Figure 4 shows the algorithm. Unlike point queries, a d-dimensional range query r is transformed into d subqueries (line 2,3). The ith subquery is denoted as $r_i = [l_i, h_i]$. Next, routine **pruneSubquery** is invoked to check if r_i can be pruned (line 4). If it can be, then it is ignored. Otherwise, the subquery is evaluated as follows (lines 5-12). The B$^+$-tree is traversed to the appropriate leaf node. If there are no points in the range of r_i, then the subquery stops. Otherwise, for every $x \in [l_i, h_i]$, the high-dimensional key of the data is compared with p for a match. Those that match are accessed using the pointer value. As in point search, multiple leaf pages may have to be examined. Once all subqueries have been evaluated, the final answers are then returned (line 13).

5. PERFORMANCE STUDY

We implemented iMinMax(θ) and the Pyramid technique [5] in C, and use the B$^+$-tree as the single dimensional index structure. Each index page is 4 KB pages. We did not buffer any data pages in this study. Therefore, every page touched incurs an I/O. However, it should be noted that the traversal paths of the d subqueries generated by iMinMax(θ) do not overlap and hence share very few common internal nodes. This is also true for the subqueries generated by the Pyramid technique. For the performance study reported, we did not use Theorem 2 to prune any subqueries.

We conducted many experiments. Here, we report some of the more interesting results on range queries. A total of 500 range queries are used. Each query is a hyper-cube and has a default selectivity of 0.1% of the domain space ([0,1],[0,1],...,[0,1]). The query width is the d-th root of the selectivity: $\sqrt[d]{0.001}$. As an indication on how large the width of a fairly low selectivity can be, the query width for 40-dimensional space is 0.841, which is much larger than half of the extension of the data space along each dimension. Different query size will be used for non-uniform distributions. The default number of dimensions used is 30. Each I/O corresponds to the retrieval of a 4 KB page. The average I/O cost of the queries is used as the performance metrics.

Algorithm PointSearch

Input: point p, θ, root of the B$^+$-tree R
Output: tuples matching p

```
1.    x_p ← transform(p, θ)
2.    l ← traverse(x_p, R)
3.    if x_p is not found in l
4.        return (NULL)
5.    else
6.        S ← ∅
7.        for every entry in l with key x_p, (x_p, v, ptr)
8.            if v == p
9.                tuple ← access(ptr)
10.               S ← S∪ tuple
11.       if l's last entry contains key x_p
12.           l ← l's right sibling
13.           goto 7
14.       return (S)
```

Figure 3: Point search algorithm.

Algorithm RangeSearch

Input: range query $r = ([x_{11}, x_{12}], [x_{21}, x_{22}], \ldots)$, root of the B$^+$-tree R
Output: answer tuples to the range query

```
1.    S ← ∅
2.    for (i = 1 to d)
3.        r_i ← transform(r, i)
4.        if NOT(pruneSubquery(r_i, r))
5.            l ← traverse(l_i, R)
6.            for every entry in l with key x ∈ [l_i, h_i], (x, v, ptr)
7.                if v == P
8.                    tuple ← access(ptr)
9.                    S ← S∪ tuple
10.           if l's last entry contains key x < h_i
11.               l ← l's right sibling
12.               goto 6
13.   return (S)
```

Figure 4: Range search algorithm.

5.1 Effect of Dimensions

In the first set of experiments, we vary the number of dimensions from 8 to 50. The data set is uniformly distributed over the domain space. There are a total of 100K points.

In the first experiment, besides the Pyramid scheme, we also compare against the MAX scheme and the sequential scan (seq-scan) technique. The MAX scheme is the simple scheme that maps each point to its maximum value. However, the transformed space is not partitioned. Moreover, two variations of iMinMax(θ) are used, namely iMax (i.e., $\theta = 1$) and iMinMax($\theta = 0.0$) (denoted as iMinMax). Figure 5 shows the results. First, we note that both the MAX and seq-scan techniques perform poorly, and their I/O cost increases with the higher number of dimensions. MAX performs slightly worse because of the additional internal nodes to be accessed and the high number of false drops.

Second, while the number of I/Os for iMinMax, iMax and Pyramid also increases with increasing number of dimensions, it is growing at a much slower rate. Third, we see that iMinMax performs the best, with Pyramid following closely, and iMax performing worse than Pyramid. iMinMax outperforms iMax and Pyramid since its search space touches fewer points.

In a typical application, apart from the index attributes, there are many more large attributes which make sequential scan of an entire file not cost effective. Instead, a feature file which consists of vectors of index attribute values is used to filter out objects (records) that do not match the search condition. However, we note that for queries which entail retrieval of a large proportion of objects, direct sequential scan may still be cost effective. The data file of the iMinMax technique can be clustered based on the leaf nodes of its B$^+$-tree to reduce random reads. The clusters can be formed in

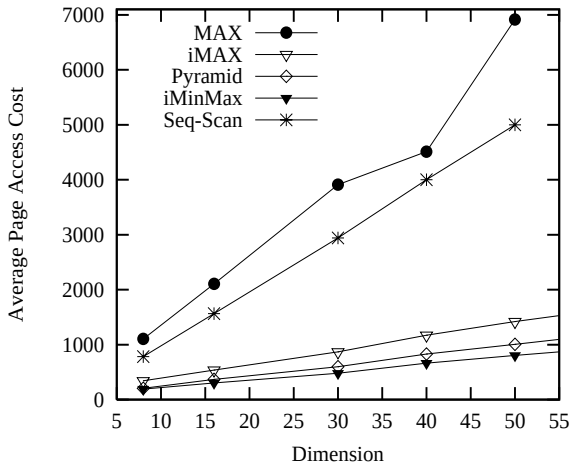


Figure 5: Effect of dimensions on uniformly distributed data set.

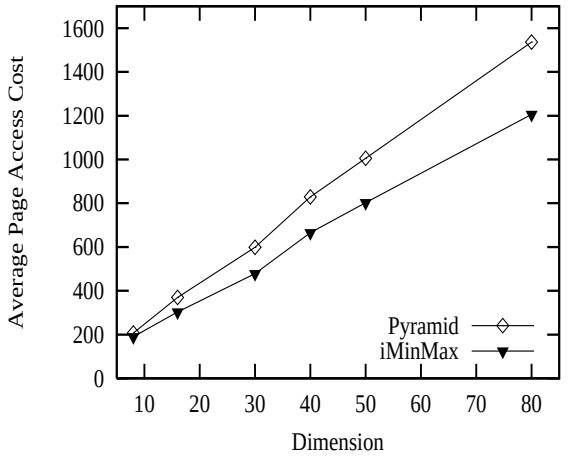


Figure 6: Comparing iMinMax and Pyramid schemes.

such a way they allow easy insertion of objects and expansion of their extent. Other optimizations such as those at the physical level are possible to make the B^+-tree behave like an index sequential file. Based on above argument and experimental results, for all subsequent experiments, we shall restrict our study to iMinMax and Pyramid techniques.

We further evaluated Pyramid and iMinMax and the results are shown in Figure 6. We observe that iMinMax remains superior, and can outperform Pyramid by up to 25%.

5.2 Effect of Data Set Sizes and Query Sizes

In this set of experiments, we study several different factors - the data set sizes, the query selectivities. For both studies, we fixed the number of dimensions at 30. Figure 7 shows the results when we vary the data set sizes from 100K to 500K points. Figure 8 shows the results when we vary the query selectivities from 0.01% to 10%.

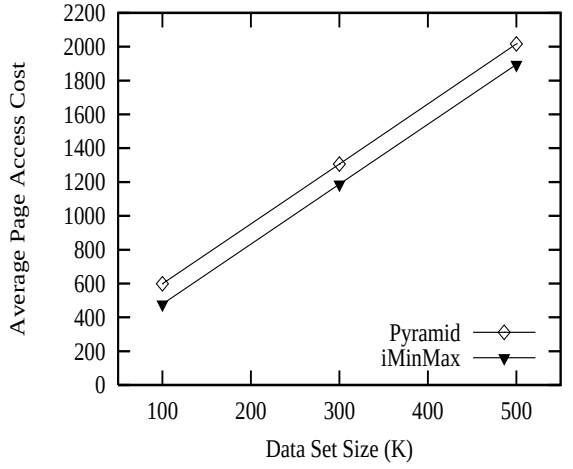


Figure 7: Effect of varying data set sizes.

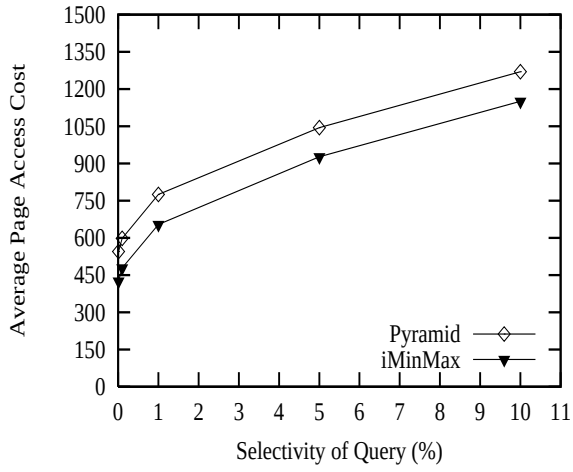


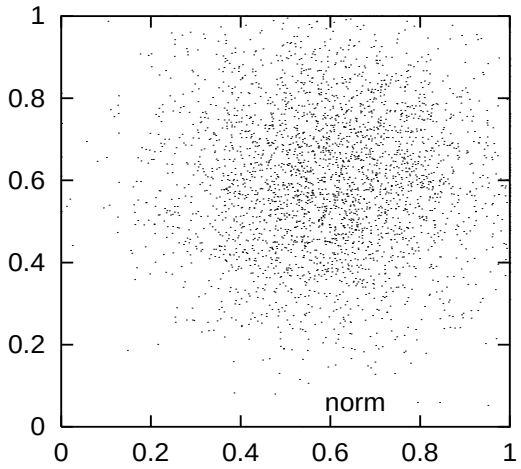
Figure 8: Effect of varying query selectivity (100K dataset).

As expected, both iMinMax and Pyramid incurred higher I/O cost with increasing data set sizes as well as the query selectivities. As before iMinMax remains superior over the Pyramid scheme. It is interesting to note that the relative difference between the two schemes seems to be unaffected by the data set sizes and query selectivities. Upon investigation, we found that both iMinMax and Pyramid return the same candidate answer set. The improvement of iMinMax stems from its reduced number of subqueries compared to the Pyramid scheme.

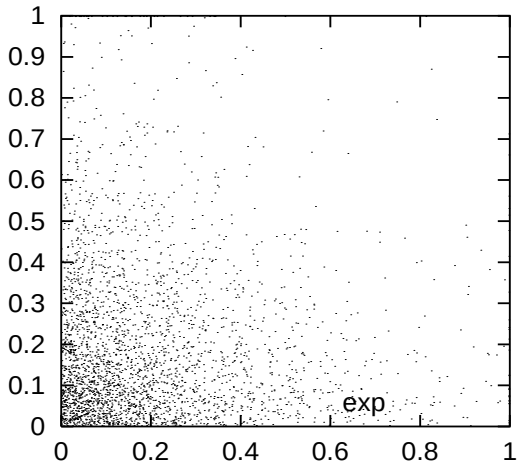
5.3 Effect of Data Distributions

In this experiment, we study the relative performance of iMinMax and Pyramid on skewed data distributions. Here, we show the results on two distributions, namely skewed normal and skewed exponential. Figure 9 illustrates two skewed data distributions in a 2-dimensional space.

The first set of experiments studies the effect of θ on skewed



(a) Normal distribution



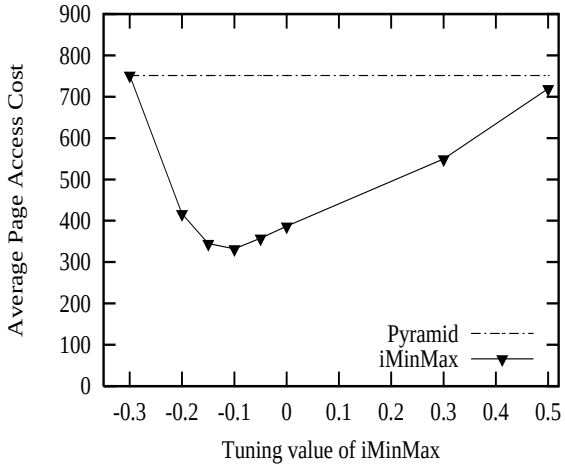
(b) Exponential distribution

Figure 9: Skewed data distribution.

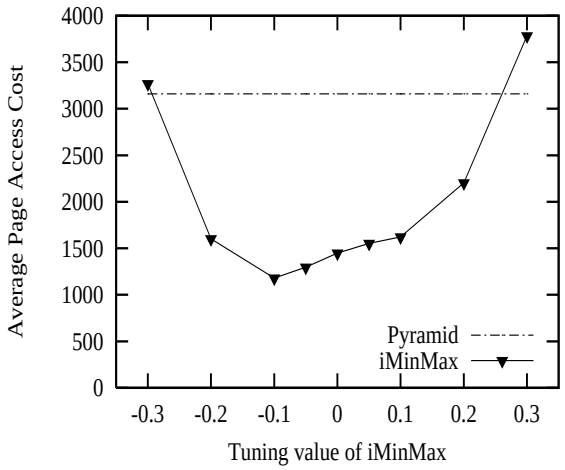
normal distribution. For normal distribution, the closer the data center is to the cluster center, the more we can keep points evenly assigned to each edge. For queries that follow the same distribution, the data points will have the same probability of being kept far from the query cube. In these experiments, we fix each dimension of the query to have a width 0f 0.4.

Figure 10(a) shows the results for 100K 30-dimensional points. First, we observe that for iMinMax, there exists a certain optimal θ value that leads to the best performance. Essentially, θ "looks out" for the center of the cluster. Second, iMinMax can outperform the Pyramid technique by a wide margin (more than 50%!). Third, we note that iMinMax can perform worse than the Pyramid scheme. This occurs when the distribution of points to the edges become skewed, and a larger number of points have to be searched. Because of the above points, we note that it is important to fine tune θ for different data distributions in order to obtain optimal performance. The nice property is that this tuning can be easily performed by varying θ.

In Figure 10(b), we have the results for 500K 30-dimensional points. As in the earlier experiment, iMinMax's effectiveness depends on the θ value set. We observe that iMinMax performs better than Pyramid over a wider range of tuning factors, and over a wider margin (more than 66%).



(a) 100K points



(b) 500K points

Figure 10: Skewed normal data set.

The second set of experiments looks at the relative performance of the schemes for skewed exponential data sets. As above, we fix each dimension of the query to have a width of 0.4. For exponential distribution (we choose to be exponential to small value), many dimensions will have small values, and a small number of them will have large values. Thus, lots of data points will have at least one big value. Because many of the dimensions are with small values, the data points tend to lie close along the edges of data space. We note that exponential data distribution can be far different from each other. They are more likely to be closed along the edges, or closed to the different corners depending

on the number of dimensions that are skewed to be large, or small. A range query, if it is with exponentially distribution characteristic, its subqueries will mostly be closed to the low corner. Therefore, tuning the keys to choose large values is likely to keep away more points from the query.

Figure 11 shows the results for 500K 30-dimensional points on skewed exponential distribution. The results are similar to that of the normal distribution experiments — iMinMax is optimal at certain θ values.

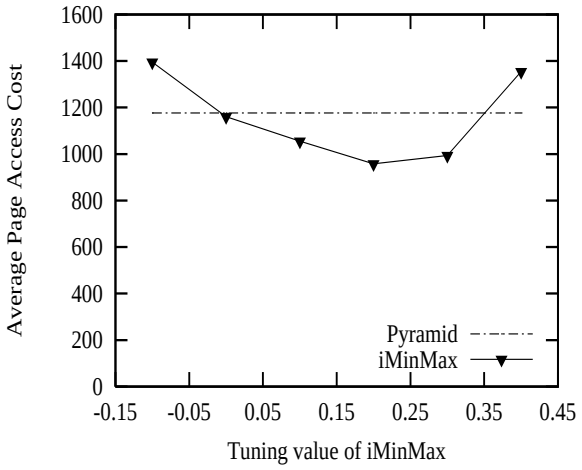


Figure 11: Skewed exponential data sets (500K points).

6. CONCLUSION

In this paper, we have proposed a simple and yet very efficient method for indexing very high dimensional data based on edges. We have shown by experiments that the method is significantly more efficient and dynamic than the Pyramid technique. Performance difference is expected to increase as the data volume and dimensionality increase, and for skewed data distributions. We are currently comparing iMinMax against the VA-file [4] and looking at how to generalize iMinMax(θ) for nearest neighbor search. We are also looking at the possibility and the effect of maintaining θ_i for each dimension. Finally, we are exploring how to determine θ adaptively.

7. ACKNOWLEDGEMENT

This work is partially supported by the Global-Atlas project funded by the National University of Singapore.

8. REFERENCES

[1] E. Bertino, *et. al. Indexing Techniques for Advanced Database Systems.* Chapters 2, 3. 39-75, Kluwer Academic Publishers, Boston, 1997.

[2] A. Guttman. *R-tree: A dynamic Index Structure for Spatial Searching.* SIGMOD'84, 47-54, 1984.

[3] N. Beckmann, H-P. Kriegel R. Schneider, B. Seeger. *The R*-tree: An Efficient and Robust Access Method for Points and Rectangles.* SIGMOD'90, 322-331, 1990.

[4] R. Weber, H. Schek, S. Blott. *A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces.* VLDB'98, 194-205, 1998.

[5] S. Berchtold, C. Bőhm, H-P. Kriegel. *The Pyramid-Technique: Towards Breaking the Curse of Dimensionality.* SIGMOD'98, 142-153, 1998.

[6] D. Comer. *The Ubiquitous B-tree.* ACM Computing Surveys. 11(2), 121–137, 1979.