SIMP: Accurate and Efficient Near Neighbor Search in High Dimensional Spaces

Vishwakarma Singh
Department of Computer Science
University of California, Santa Barbara
vsingh@cs.ucsb.edu

Ambuj K. Singh
Department of Computer Science
University of California, Santa Barbara
ambuj@cs.ucsb.edu

ABSTRACT

Near neighbor search in high dimensional spaces is useful in many applications. Existing techniques solve this problem efficiently only for the approximate cases. These solutions are designed to solve r-near neighbor queries for a fixed query range or a set of query ranges with probabilistic guarantees, and then extended for nearest neighbor queries. Solutions supporting a set of query ranges suffer from prohibitive space cost. There are many applications which are quality sensitive and need to efficiently and accurately support near neighbor queries for all query ranges. In this paper, we propose a novel indexing and querying scheme called *Spatial Intersection and Metric Pruning* (SIMP). It efficiently supports r-near neighbor queries in very high dimensional spaces for all query ranges with 100% quality guarantee and with practical storage costs. Our empirical studies on three real datasets having dimensions between $[32\text{-}256]$ and sizes up to 10 million show a superior performance of SIMP over LSH, Multi-Probe LSH, LSB tree, and iDistance. Our scalability tests on real datasets having as many as 100 million points of dimensions up to 256 establish that SIMP scales linearly with the query range, the dataset dimension, and the dataset size.

1. INTRODUCTION AND MOTIVATION

Search for near neighbors of a given query object in a collection of objects is a fundamental operation in numerous applications, e.g., multimedia similarity search, data mining, information retrieval, and pattern recognition. The most common model for search is to represent objects in a high-dimensional attribute space, and then retrieve points near a given query point using a distance measure. Many variants of the near neighbor problem, e.g., nearest neighbor queries, top-k nearest neighbors, approximate nearest neighbors, and r-near neighbor (r-NN) queries, have been studied in the literature. In this paper, we study r-NN queries, also called r-ball cover queries.

Let $\mathcal{U}$ be a dataset of N points in a d-dimensional vector space $\mathcal{R}^d$. Let $d(.,.)$ be a distance measure over $\mathcal{R}^d$. An r-NN query is defined by a query point $q \in \mathcal{R}^d$ and a search range r from q. It is a range query whose result set contains all the points p satisfying $d(q, p) \leq r$. An r-NN query is useful for constructing near neighbor graphs, mining of collocation patterns [31], and mining density based clusters [14]. An r-NN query can also be repeatedly used with increasing query ranges, starting with an expected range, to solve the nearest neighbor family of problems [16, 19, 22].

In order to get a practical performance in query processing, data points are indexed and searched using an efficient data structure. A good index should be space and time efficient, should yield accurate results, and should scale with the dataset dimension and size. There are many well-known indexing schemes in the literature, mostly tree-based [4, 10, 17], for efficiently and exactly solving near neighbor search in low dimensions. It is known from the literature that the performances of these methods deteriorate and become worse than sequential search for sufficiently large number of dimensions due to the curse of dimensionality [34]. iDistance [19] is a state-of-the-art method for an exact r-NN search. It has been shown to work well for datasets of dimensions as high as 30. A major drawback of iDistance is that it works well only for clustered data. It incurs expensive query costs for other kinds of datasets and for very high dimensional datasets.

There are many applications which need efficient and accurate near neighbor search in very high dimensions. For example, content based multimedia similarity search uses state-of-the-art 128-dimensional SIFT [25] feature vectors. Another example is DNA sequence matching which requires longer seed length (typically 60-80 bases) to achieve higher specificity and efficiency while maintaining sensitivity to weak similarities [8]. A common practice is to use dimensionality reduction [30] techniques to reduce the dimensions of the dataset before using an index structure. These techniques being lossy transformations do not guarantee optimal quality. These techniques are also not useful for a number of datasets, e.g., strings [35] and multimedia [25, 26], which have intrinsically high dimensionality.

State-of-the-art techniques for r-NN search in very high dimensions trade-off quality for efficiency and scalability. Locality Sensitive Hashing (LSH) [18, 16] is a state-of-the-art method for approximately solving r-NN query in very high dimensions. LSH, named *Basic*-LSH here, solves (r_0, ϵ)-neighbor problem for a fixed query range r_0. It determines whether there exists a data point within a distance r_0 of query q, or whether all points in the dataset are at least a distance $(1+\epsilon)r_0$ away from q. In the first case, *Basic*-LSH returns a point within distance at most $(1+\epsilon)r_0$ from q. *Basic*-LSH constructs a set of L hashtables using a family of hash functions to fetch near neighbors efficiently. *Basic*-LSH is extended, hereby named *Extended*-LSH, to solve ϵ-approximate nearest neighbor search by building several data structures for different values of r. *Extended*-LSH builds a set of L hashtables for each value of r in $\{r_0, (1+\epsilon)r_0, (1+\epsilon)^2 r_0, \cdots, r_{max}\}$, where r_0 and

r_{max} are the smallest and the largest possible distance between the query and the data point respectively.

LSH based methods and their variants, though efficient and scalable, lack 100% quality guarantee because of their probabilistic nature. In addition, they are unable to support queries over flexible ranges. *Basic*-LSH is designed for a fixed query range r_0, and therefore yields poor result quality for query ranges $r > r_0$. *Extended*-LSH suffers from prohibitive space costs as it maintains a number of index structures for different values of r. The space usage of even *Basic*-LSH becomes prohibitive for some applications like biological sequence matching [8] where a large numbers of hashtables are required to get a satisfactory result quality. Recently, Lv et al. [26] improved *Basic*-LSH to address its space issue with a novel probing sequence, called Multi-Probe LSH. However, Multi-Probe LSH does not give any guarantee on quality or performance. Tao et al. [33] proposed a B-tree based index structure, LSB tree, to address the space issue of *Extended*-LSH and the quality issue of *Basic*-LSH for query ranges which are any power of 2. Nonetheless, LSB tree is an approximate technique with a high space cost.

We find that none of the existing methods simultaneously offer efficiency, 100% accuracy, and scalability for near neighbor queries over flexible query ranges in very high dimensional datasets. These properties are of general interest for any search system. In this paper, we propose a novel in-memory index structure and querying algorithm called **SIMP** (**S**patial **I**ntersection and **M**etric **P**runing). It efficiently answers r-NN queries for any query range in very high dimensions with 100% quality guarantee and has practical storage costs. SIMP adopts a two-step pruning method to generate a set of candidate near neighbors for a query. Then it performs a sequential search on these candidates to obtain the true near neighbors.

The first pruning step in SIMP is named *Spatial Intersection Pruning*(**SIP**). SIMP computes multiple 2-dimensional projections of the high dimensional data. Each projection is computed with respect to a different reference point. SIMP partitions each 2-dimensional projection into grids. It also computes the projection of the query answer space. SIMP generates a set of candidates for a r-NN query by an intersection of the 2-dimensional grids and the projection of the query answer space. It preserves all the true neighbors of a query by construction. A hash based technique is used in this step to gain space and query time efficiency. The second step of pruning is called *Metric Pruning* (**MP**). SIMP partitions the dataset into tight clusters. It uses triangle inequality between a candidate, candidate's nearest cluster center, and the query point to further filter out false candidates.

We also design a statistical cost model to measure the performance of SIMP. We show a superior performance of SIMP over state-of-the-art methods iDistance and p-stable LSH on three real datasets having dimensions between $[32\text{-}256]$ and sizes up to 10 million. We also compared SIMP with Multi-Probe LSH and LSB tree on two real datasets of dimensions 128 and 256 and sizes 1 million and 1.08 million respectively. We observed that SIMP comprehensively outperforms both these methods. Our scalability tests on real datasets of sizes up to 100 million and dimensions up to 256 show that SIMP scales linearly with the query range, the dataset dimension, and the dataset size.

Our main contributions are: (a) a novel algorithm that solves r-NN queries for any query range with 100% quality in a very high dimensional search space; (b) statistical cost modeling of SIMP; and (c) extensive empirical studies. We discuss related work in section 2. We develop our index structure and query algorithm in section 3. A statistical cost model of SIMP is described in section 4. We present experimental results in section 5. We describe schemes for selecting the parameters of SIMP in section 6.

2. LITERATURE SURVEY

Near neighbor search is well solved for low dimensional data (usually less than 10). Gaede et al. [15] and Samet et al. [30] present a survey of these multidimensional access methods. All the indexing schemes proposed in the literature fall into two major categories: space partitioning and data partitioning. Berchtold et al. [6] partition the space using a Voronoi diagram and answer a query by searching for the cell in which the query lies. Space partitioning trees like KD-Tree recursively partition the space on different dimensions. Data partitioning techniques like R-Tree [17], M-Tree [10] and their variants enclose relatively near points in Minimum Bounding Rectangles or Spheres and recursively build a tree. The performance of these techniques deteriorates rapidly with an increase in the number of data dimensions [34, 7].

For very high dimensions, space filling curves [23] and dimensionality reduction techniques are used to project the data into low dimensional space before using an index. Weber et al. [34] proposed VA-file to compress the dataset by dimension quantization and minimize the sequential search cost. Jagadish et al. [19] proposed **iDistance** to exactly solve r-NN queries in high dimensions. The space is split into a set of partitions and a reference point is identified for each partition. A data point p is assigned an index key based on its distance from the nearest reference point. All the points are indexed using their keys in a B+-Tree. iDistance performs well for clustered data of dimensions up to 30. The metric pruning of SIMP is inspired from this index structure.

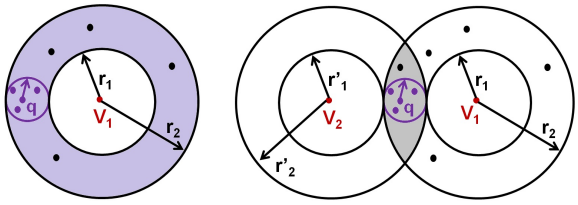
Near Neighbor search is efficiently but approximately solved in very high dimensions [16, 2]. Locality sensitive hashing (LSH) proposed by Indyk et al. [18] provides a sub-linear search time and a probabilistic bound on the result quality for approximate r-NN search. LSH uses a family of hash functions to create hashtables. It concatenates hash values from k hash functions to create a hash key for each point. It uses L hashtables to improve the quality of search. LSH hash functions put nearby objects in the same hashtable bucket with a higher probability than those which are far apart. One can determine near neighbors by hashing the query point and retrieving elements stored in the bucket containing the query. Many families of hash functions have been proposed [16, 9, 12, 1] for near neighbor search. p-**stable LSH** [12] uses vectors, whose components are drawn randomly from a p-stable distribution, as a family of hash functions for l_p norm. As discussed in section 1, LSH suffers from the quality and the space issues.

Many improvements and variants [3, 28, 29, 26, 13, 21, 33, 24] have been proposed for the LSH algorithm to solve the approximate near neighbor search. Lv et al. [26] proposed a heuristic called **Multi-Probe LSH** to address the space issue of *Basic*-LSH [16]. They designed a novel probing sequence to look up multiple buckets in hashtables of *Basic*-LSH. These buckets have a high probability of containing the near neighbors of a query. Multi-Probe LSH does not have any quality guarantee and may need a large number of probes to achieve a desired quality, thus making it inefficient.

A B-tree based index, called **LSB** tree (a set of LSB trees is called an LSB forest), was proposed by Tao et al. [33] for near neighbor search in relational databases to simultaneously address the space issue of *Extended*-LSH [18] and the quality issues of *Basic*-LSH for query ranges which are any power of 2. Each d-dimensional point is transformed into an m-dimensional point by taking its projection on m p-stable hash functions similar to p-stable LSH [12]. Points are indexed using a B-Tree on their z-order values which are obtained by partitioning the m-dimensional space with equi-width bins. Near neighbor search is carried by obtaining points based on the length of the longest common prefix of z-order. LSB tree is an approximate technique with weak quality guaran-

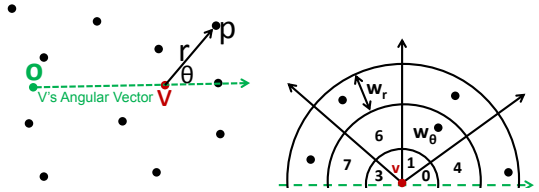
$q(r_q)$: A query with point q and search range r_q	v: A viewpoint	n_v: Number of viewpoints
$\mathcal{G}(v)$: Polar grid of viewpoint v	N: Number of data points in $\mathcal{U}$	$h(s)$: A hashtable of SIMP
p: A data point	$\mathbf{o}$: Origin of the data space	s: Signature of a hashtable
θ: Angle of a data point p relative to a viewpoint v and its angular vector ov	S: Percentage of the data points obtained by an algorithm as candidate for a query (selectivity)	C: Candidate set obtained by an algorithm for a query
r: Distance of a point p from a viewpoint v	b: Bin id of a point in a polar grid	P: Number of probes used by Multi-Probe LSH
L: Number of hashtables in the index structure	k : Size of a hash signature s	d: Dimension of the dataset
w_r: Radial width between rings of a polar grid	w_θ: Angle between radial vectors of a polar grid	n_z: Number of mballs used for Metric Pruning

Table 1: A descriptive list of notations used in the paper.



(a) One viewpoint (b) Two viewpoints

Figure 1: Spatial Intersection Pruning. (a) Answer space of query $q(r_q)$ is bounded within the distance range [r_1, r_2] relative to viewpoint v_1. Shadowed region contains all the candidates of query $q(r_q)$ relative to viewpoint v_1. (b) Shadowed region contains all the candidates of query $q(r_q)$ relative to two viewpoints v_1 and v_2. Intersection over two viewpoints gives better pruning of the false candidates.



(a) An example dataset (b) Polar grid relative to v

Figure 2: (a) A dataset with data space origin o and a viewpoint v. Values r and θ for point p are computed relative to v and its angular vector ov. (b) Partition of the data space using equi-width w_r rings and equi-angular w_θ radial vectors relative to viewpoint v and v's angular vector ov. Each bin is given a unique id that places a canonical order on the bins.

tees and can have prohibitive costs. It is noteworthy that all the LSH based techniques create index structures independent of the data distribution.

A cost model for near neighbor search using partitioning algorithms was provided by Berchtold et al. [5]. An M-Tree cost model was presented by Ciaccia et al. [11]. Weber et al. [34] and Böhm et al. [7] developed these ideas further.

3. ALGORITHM

We develop the idea of SIMP using a dataset $\mathcal{U}$ of N points in a d-dimensional vector data space $\mathcal{R}^d$. Each point p has a unique identifier. We use Euclidean metric to measure the distance $d(.,.)$ between a pair of points in $\mathcal{R}^d$. We take $\mathbf{o}$ as the origin of the data space. An r-NN query $q(r_q)$ is defined by the query point q and the search range r_q. The answer set of the query $q(r_q)$ contains all the points of the dataset $\mathcal{U}$ which lie within a hyper sphere of radius r_q and center at q. This hyper sphere is the answer space of the query $q(r_q)$. We describe all notations used in this paper in table 1.

3.1 Preliminaries

We first explain the idea of intersection for *Spatial Intersection Pruning (SIP)* that effectively prunes false candidates. Then we describe how intersection is performed using multiple projections of the data points, each relative to a different reference point, to find candidates. We show that the projection relative to a random viewpoint is locality sensitive: a property that helps SIMP effectively prune false candidate by intersection. Finally, we explain the idea of *Metric Pruning (MP)*.

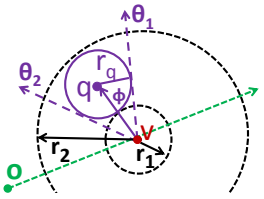


Figure 3: Bounding range B={$[r_1, r_2]$,$[\theta_1,\theta_2]$} of qball of a query $q(r_q)$ relative to a viewpoint v and angular vector $\mathbf{o}v$.

We explain SIP using figure 1. Let v_1 be a randomly chosen reference point, called a **viewpoint**, in the d-dimensional data space $\mathcal{R}^d$. We compute distance r=$d(v_1, p)$ for each point p in the dataset relative to v_1. Let the answer space of a query $q(r_q)$ be contained in the distance range $[r_1, r_2]$ from v_1 as shown in figure 1. All the points having their distances in the range $[r_1, r_2]$ from v_1 form the set of candidate near neighbors for the query $q(r_q)$. We show the region containing the candidates in shadow in figure 1(a). Let v_2 be another viewpoint. Let $[r'_1, r'_2]$ be the distance bounding range for the answer space of the query $q(r_q)$ relative to v_2. Now, the true near neighbors of the query $q(r_q)$ must lie in the intersection of the range $[r_1, r_2]$ and $[r'_1, r'_2]$ as shown with shadowed region in figure 1(b). We see that an intersection over two viewpoints bounds the answer space more tightly, and thus achieves better pruning of false candidates.

We describe how the intersection is performed using an example dataset shown in figure 2(a). Let v be a viewpoint. The vector $\mathbf{o}v$ joining origin $\mathbf{o}$ to the viewpoint v is called v's **angular vector**. We compute distance r=$d(v, p)$ and angle θ for each point p in the dataset relative to v and $\mathbf{o}v$. The range of the distance r is $[0, r_{max}]$, where r_{max} is the distance of the farthest point in the dataset from v. The angle θ lies in the range $[0°, 180°]$ and is computed as

$$\theta = cos^{-1}(\mathbf{o}v.vp/(d(\mathbf{o}, v) \times d(v, p))) \quad (1)$$

Thus, we get the projection of the d-dimensional dataset onto a 2-dimensional polar (r, θ) space relative to v. We partition this polar space into grids using equi-width w_r rings and equi-angular w_θ radial vectors relative to the viewpoint v as shown in figure 2(b). This partitioned data space is called v's **polar grid** $\mathcal{G}(v)$. For a given query $q(r_q)$, we compute the projection q' of the query point q relative to the viewpoint v and the angular vector ov. Then, the projection of the answer space of the query $q(r_q)$ relative to v and $\mathbf{o}v$, named **qball**, is a circle with radius r_q and center q'. All the true neighbors of $q(r_q)$ are contained in this qball. We make a choice to use polar coordinates because both of its dimensions, distance r and angle θ, reflect an aggregate value of all the original

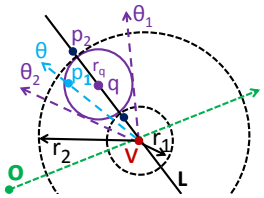
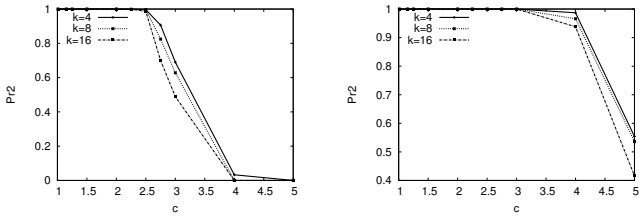


Figure 4: We see that point p_2 having $d(q, p_2) > r_q$ lies outside the bounding range of the qball of query $q(r_q)$ relative to viewpoint v lying on the line L. Point p_1 having $d(q, p_2) \leq r_q$ always lies within the bounding range of the qball of query $q(r_q)$ for any viewpoint v.



(a) 128 dimensional data (b) 256 dimensional data

Figure 5: Values of Pr_2 obtained for varying c, where $d(q, p)$=$c \times r_q$, and varying number n_v of viewpoints used for spatial intersection on 128 and 256 dimensional real datasets. The value of Pr_1 is always 1.

dimensions in $\mathcal{R}^d$. Any other coordinate system can be similarly used in place of polar coordinates.

Let the qball of the query $q(r_q)$ be enclosed within the bounding range B=$\{[r_1, r_2], [\theta_1, \theta_2]\}$ relative to a viewpoint v as shown in figure 3. We find a set of bins of polar grid $\mathcal{G}(v)$ that encloses the bounding range B. Points contained in these bins form the set of candidates for the query $q(r_q)$ relative to v. For a set of n_v viewpoints, a candidate set is obtained for each viewpoint independently. An intersection of the candidate sets obtained from the n_v viewpoints gives the final set of candidates.

We next show that the projection relative to a random viewpoint v is locality sensitive. Let v be a randomly chosen viewpoint from the data space and $\mathcal{G}(v)$ be its polar grid. Let the qball of a query $q(r_q)$ be enclosed within the bounding range B=$\{[r_1, r_2], [\theta_1, \theta_2]\}$ relative to v. The bounding range B is computed as shown in section 3.3. Let p be a point at distance r=$d(v, p)$ from v and at an angle θ from v's angular vector ov. Point p is chosen as a candidate if $r_1 \leq r \leq r_2$ and $\theta_1 \leq \theta \leq \theta_2$. If n_v randomly chosen viewpoints are used for intersection, then a point p is a candidate only if it lies in the intersection of the bounding ranges obtained from each of the viewpoints independently. Let Pr_1 be the probability that p is selected as a candidate when $r \leq r_q$. Let Pr_2 be the probability that that p is selected as a candidate when $r > r_q$. Probabilities are computed with respect to the random choices of viewpoints. We say that the projection with respect to a random viewpoint is locality sensitive if $Pr_2 < Pr_1$.

LEMMA 1. *The projection of points on a polar (r, θ) space relative to a random viewpoint is locality sensitive.*

PROOF. Let points p_1 and p_2 be such that $d(q, p_1) \leq r_q$ and $d(q, p_2) > r_q$. Let p_1 be at an angle θ_{p_1} and p_2 be at an angle θ_{p_2} relative to the angular vector ov of a random viewpoint v.

Point p_1 satisfies $r_1 \leq d(q, p_1) \leq r_2$ and $\theta_1 \leq \theta_{p_1} \leq \theta_2$ for any viewpoint v by construction as shown in figure 4. Therefore, Pr_1=1.

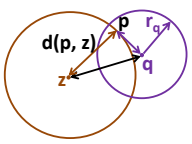


Figure 6: Metric Pruning. z is the nearest mcenter of point p. p is a candidate for $q(r_q)$ only if $r_q \geq d(q, p) \geq \mid d(p, z) - d(q, z) \mid$.

Next we show that $Pr_2 < 1$ by geometric considerations using figure 4. We draw a line L passing through q and p_2 as shown in figure 4. We draw the enclosing rings of the query $q(r_q)$ at radii r_1=$(d(v, q) - r_q)$ and r_2=$(d(v, q) + r_q)$ from a randomly chosen viewpoint v on the line L. We see that the point p_2 lies outside the enclosing ring, i.e., $d(v, p_2) < r_1$ or $d(v, p_2) > r_2$. This is true for any viewpoint lying on the line L. Therefore, $Pr_2 < 1$ □

We empirically observed that the probability Pr_2 rapidly decreases with an increase in the distance of a point p from the query. This also implies that the probability of a false near neighbor being selected as candidate decreases with its distance from the query point. To derive these results, we computed the values of Pr_2 for a query $q(r_q)$ for varying c, where $d(q, p)$=$c \times r_q$, using Monte Carlo methods. We also computed values of Pr_2 for different number of viewpoints n_v used for intersection. We used two real datasets of dimensions 128 (SIFT) and 256 (CHist) for the simulation. We describe these datasets in section 5. We performed simulation using 10 million random viewpoints. We observed that the value of Pr_2 decreases at a fast rate with increasing value of c, as shown in figure 5. For example, the value of Pr_2 is zero for 128-dimensional dataset for c=4 and n_v=4. The value of Pr_1 is always 1.

SIMP achieves a high rate of pruning of false near neighbors due to Lemma 1 and the intersection of the qball of a query with polar grids of multiple viewpoints. This makes SIMP a very efficient method for r-NN search with 100% quality guarantee. In this paper, we develop a suitable data structure and search algorithm using hashing to implement the idea of SIP.

To get extra performance, we augment SIP with metric pruning. This pruning is based on triangle inequality and is carried with respect to data clusters. To obtain a good pruning, these clusters need to be quite small. As a result, we cluster all the data points into n_z tight clusters where a cluster is named an **mball**. We name the center of an mball as an **mcenter**. Let z be the nearest mcenter to a point p. From triangle inequality, we know that a point p is a candidate only if $r_q \geq d(q, p) \geq \mid d(p, z) - d(q, z) \mid$, as shown in figure 6. The nearest mcenter z of each point p and the distance $d(p, z)$ are pre-computed for an efficient pruning using triangle inequality at runtime.

3.2 Index Structure

The index structure of SIMP consists of data structures for efficient processing of both SIP and MP. A set of hashtables and bit arrays constitute the index structure of SIP. The index structure of MP consists of three associative arrays.

We first discuss the construction of the index structure for SIP. We randomly choose n_v viewpoints V=$\{v_i\}_{i=1}^{n_v}$ from the d-dimensional dataset. We construct a polar grid for each of the n_v viewpoints. For each polar grid $\mathcal{G}(v)$, a data point p is assigned the id of the bin of the polar grid in which it lies. The bin id of a point p having distance r=$d(v, p)$ and angle θ relative to a viewpoint v and v's angular vector ov is

$$b = (\lfloor r/w_r \rfloor \times (\lfloor 180^\circ/w_\theta \rfloor + 1)) + \lfloor \theta/w_\theta \rfloor.$$

For example, if we take w_r=50 and w_θ=45° for creating a polar

grid and a point p at r=70 and θ=40° from the viewpoint, then the bin id of p is $1 \times 5 + 0$=5. The distance r of any point p from a viewpoint v lies between $[0, r_{max}]$, where r_{max} is the distance of the farthest point in the dataset from v. The angle θ of any point p relative to ov lies between $[0°, 180°]$.

Polar grids incur a high space cost if stored separately. A polar grid can be stored in an array whose entry at index i is the collection of the data points in the bin i of the polar grid. A point is stored only by its identifier in the array. This gives a space cost of $n_v \times N$ for n_v polar grids and a dataset of size N. This space cost may become unmanageable for large values of n_v and N. For example, if N is 100 million points and n_v=100, then the total space cost of the index is 40GB.

We develop a hash based index structure to reduce the index space. We assume without loss of generality that n_v=$k \times L$ for some integers k and L. We use a value of k=4 for SIMP. We split n_v viewpoints into L groups, each of size k. A set of k viewpoints is called a *signature* s=$\{v_1, \cdots, v_k\}$. Thus, we construct L signatures s_i such that $(s_i \cap s_j)$=$\emptyset$ for any two signatures s_i and s_j, and $\cup_{i=1}^{L} s_i$ =V. We create a hashtable $h(s)$ for each signature s. We generate a k-size $\{b_1 \cdots b_k\}$ hash key for each data point p for a signature s by concatenating the bin ids of p obtained from the polar grids of the viewpoints $v \in s$. All the data points are hashed into the hashtable $h(s)$ using their k-size keys by a standard hashing technique. A point is stored by its identifier in the hashtable. Each hashtable $h(s)$ defines an intersection over k polar grids. The space cost of L hashtables is $L \times N$, which is k times less than the space cost of storing n_v polar grids ($n_v \times N$).

We describe the creation of hashtables with an example. Let V=$\{v_1, v_2, v_3, v_4\}$ be a set of n_v=4 viewpoints. We create two signatures s_1=$\{v_1, v_2\}$ and s_2=$\{v_3, v_4\}$ for k=2. We create hashtables $h(s_1)$ and $h(s_2)$ for signatures s_1 and s_2 respectively. Let the bin ids of a point p in the polar grids of viewpoints v_1, v_2, v_3, and v_4 be b_1, b_2, b_3, and b_4 respectively. Then, point p is hashed into $h(s_1)$ and $h(s_2)$ using keys b_1b_2 and b_3b_4 respectively.

We also maintain a bit array called *isEmpty* for each polar grid $\mathcal{G}(v)$. The size of this bit array is equal to 1 if r_{max}=0 and is equal to $((\lfloor r_{max}/\ w_r \rfloor + 1) \times (\lfloor 180°/w_\theta \rfloor + 1))$ if $r_{max} > 0$. The actual memory footprint of a bit array is negligible. A bin's id in a polar grid $\mathcal{G}(v)$ is its index in the bit array of $\mathcal{G}(v)$. All the bits corresponding to empty bins of a polar grid $\mathcal{G}(v)$ are marked true.

The index structures of MP are created as follows. We split all the data points into n_z mballs. We assign a unique identifier to each mcenter. We store mcenters in an associative array using their identifiers as keys. We store the identifier and the distance of the nearest mcenter for each data point in associative arrays using the point's identifier as key.

3.3 Search Algorithm

In this section, we present the search algorithm for SIMP to retrieve candidates for a query $q(r_q)$ using the data structures of SIP and MP. SIMP performs a sequential scan on these candidates to obtain the near neighbors of the query.

In the SIP step, SIMP first finds the nearest viewpoint v_1 to the given query point q. Then it obtains hashtable $h(s)$ corresponding to v_1, i.e., $v_1 \in s$. SIMP computes the keys of the buckets of $h(s)$ containing the candidates of the query $q(r_q)$ as follows. For a query $q(r_q)$, SIMP finds a set of bins enclosing the qball of the query from the polar grid of each of the k viewpoints $v \in s$. SIMP takes a cartesian product of these k sets to get the keys of the buckets containing candidates. The union of the data points in these buckets gives the set of candidates from the hashtable $h(s)$. For example, let viewpoints $\{v_1, v_2\}$ be the signature s of hashtable $h(s)$ for k=2. Let $\{b_{11}, b_{12}\}$ be the set of bins of polar grid $\mathcal{G}(v_1)$ and $\{b_{21}, b_{22}\}$ be the set of bins of $\mathcal{G}(v_2)$ enclosing the qball of the query $q(r_q)$. The union of the data points contained in the buckets of $h(s)$ having keys $\{b_{11}b_{21}, b_{11}b_{22}, b_{12}b_{21}, b_{12}b_{22}\}$ gives the set of candidates from $h(s)$.

Algorithm 1 getBins

In: $q(r_q)$: query, $d(q, v)$: distance of query from the viewpoint v
In: θ: angle of the query relative to ov
In: A: *isEmpty* bit array of $\mathcal{G}(v)$
1: $BC \leftarrow \phi$ /* ids of enclosing bins */
2: $[r_1, r_2], [\theta_1, \theta_2] \leftarrow$ bounding range of the qball of $q(r_q)$
3: startBinR $\leftarrow \lfloor r_1/w_r \rfloor$, endBinR $\leftarrow \lfloor r_2/w_r \rfloor$
4: startBin$\theta \leftarrow \lfloor \theta_1/w_\theta \rfloor$, endBin$\theta \leftarrow \lfloor \theta_2/w_\theta \rfloor$
5: **for all** $r_i \in$ [startBinR, endBinR] **do**
6: **for all** $\theta_i \in$ [startBinθ, endBinθ] **do**
7: $BC \leftarrow BC \cup (r_i \times (\lfloor 180°/w_\theta \rfloor + 1) + \theta_i)$
8: **end for**
9: **end for**
10: **for all** $b \in BC$ **do**
11: **if** $A[b]$ is True **then**
12: $BC \leftarrow BC \setminus b$
13: **end if**
14: **end for**
15: **return** BC

We explain here the method to determine a set of bins of a polar grid $\mathcal{G}(v)$ that encloses the qball of a query $q(r_q)$ relative to a viewpoint v. We first compute the bounding range B=$\{[r_1, r_2], [\theta_1, \theta_2]\}$ of the qball relative to the viewpoint v as shown in figure 3. Let $d(q, v)$ be the distance of the query point q from the viewpoint v. The values of r_1 and r_2 are obtained as follows:

$$r_1 = \begin{cases} d(q,v) - r_q & \text{if } r_q < d(q,v) \\ 0 & \text{if } r_q \geq d(q,v) \end{cases} \quad (2)$$

$$r_2 = d(q,v) + r_q \quad (3)$$

We find the aperture ϕ of the qball of $q(r_q)$ relative to v as follows:

$$\phi = 2 \times sin^{-1}(r_q/d(q,v)). \quad (4)$$

We determine angle θ of the query point q relative to viewpoint v and its angular vector $\mathbf{o}v$ using Equation 1. The values of θ_1 and θ_2 are given by

$$\theta_1 = \begin{cases} \theta - \phi/2 & \text{if } \phi/2 \leq \theta \\ 0° & \text{if } \phi/2 > \theta \\ 0° & \text{if } r_q \geq d(q,v) \end{cases} \quad (5)$$

$$\theta_2 = \begin{cases} \theta + \phi/2 & \text{if } (\phi/2 + \theta) < 180° \\ 180° & \text{if } (\phi/2 + \theta) \geq 180° \\ 180° & \text{if } r_q \geq d(q,v) \end{cases} \quad (6)$$

We compute the bins from a polar grid $\mathcal{G}(v)$ using the bounding range B=$\{[r_1, r_2], [\theta_1, \theta_2]\}$ as described in Algorithm 1. We first obtain the set of bins enclosing the qball of the query for radial partitioning and angular partitioning independently in steps 3 and 4 respectively. We iterate through these bins to compute a set of bins of $\mathcal{G}(v)$ that encloses the qball in steps [5-9]. We remove empty bins from this set using *isEmpty* bit array of $\mathcal{G}(v)$ in steps [10-14].

In the MP step, SIMP uses triangle inequality between a candidate obtained from SIP step, candidate's nearest mcenter, and the query point q. It retrieves the nearest mcenter z of a candidate and the distance to the mcenter $d(p, z)$ from the index. SIMP computes the distance $d(q, z)$ between the mcenter z and the query point q. SIMP discards a candidate if $r_q <| \ d(p, z) - d(q, z) \ |$.

We describe the execution of SIMP using Algorithm 2. Spatial intersection pruning is performed in steps [1-11]. SIMP finds

Algorithm 2 SIMP

In: $q(r_q)$: query, $\mathcal{H}$: set of hashtables
In: Z: array containing the nearest mcenter of each data point
In: pz: array of distances of the points from their nearest mcenter
1: $C \leftarrow \emptyset$ /*candidate set */
2: $v_1 \leftarrow$ nearest viewpoint to q
3: $h(s) \leftarrow \mathcal{H}(v_1 \in s)$
4: $Y \leftarrow [1]$
5: **for all** $v \in s$ **do**
6: $BC \leftarrow$ getBins($q(r_q)$, $d(q,v)$, θ, $isEmpty$)
7: $Y \leftarrow Y \times BC$
8: **end for**
9: **for all** $key \in Y$ **do**
10: $C \leftarrow C \cup$ points $\in h[key]$
11: **end for**
12: /* Metric Pruning */
13: $qz \leftarrow [\,]$ /* list of distances of mcenters z from q*/
14: **for all** $c \in C$ **do**
15: $z \leftarrow$ c's nearest mcenter from Z
16: **if** $qz[z] \neq$ Null **then**
17: $d' \leftarrow qz[z]$
18: **else**
19: $d' \leftarrow qz[z] \leftarrow d(q,z)$
20: **end if**
21: **if** $\mid pz[c] - d' \mid > r_q$ **then**
22: $C \leftarrow C \setminus c$
23: **end if**
24: **end for**
25: /*Sequential search */
26: **for all** $c \in C$ **do**
27: **if** $d(q,c) > r_q$ **then**
28: $C \leftarrow C \setminus c$
29: **end if**
30: **end for**
31: **return** C

hashtable $h(s)$ whose signature s contains the nearest viewpoint v_1 to the query point q in steps [2-3]. For each viewpoint v in signature s, SIMP obtains a set of bins BC enclosing the qball of the query $q(r_q)$ in step 6. SIMP computes a set Y of hash keys by a cartesian product of the set of bins BC in step 7. SIMP takes a union of the points in the buckets of hashtable $h(s)$ corresponding to the hash keys Y in steps [9-11]. Next, SIMP applies metric pruning in steps [13-24]. For each candidate, SIMP gets the identifier of its nearest mcenter z from a pre-computed array Z in step 15. SIMP computes the distance of the query q from z if it is not previously computed; otherwise it retrieves the distance from an array qz in steps [16-20]. A candidate is tested using the triangle inequality in step 21. Finally, all the true neighbors are obtained by computing the actual distance of each of the candidates from the query point in steps [26-29].

Extension to nearest neighbor search: The SIMP algorithm for r-NN search can be extended for top-k nearest neighbor search using the approach proposed by Andoni et al. [16]. For a dataset, an expected distance $E(r)$ of top-k nearest neighbors from query points is estimated under the assumption that the query distribution follows the data distribution. We start the r-NN search with r=$E(r)$. If no nearest neighbor is obtained, then we repeat SIMP with range $((1+c) \times r)$ until at least k points are retrieved.

4. STATISTICAL COST MODELING AND ANALYSIS

We develop statistical models to measure the query costs of SIMP. For a query $q(r_q)$, the number of buckets of a hashtable probed in steps [5-11] and the number of candidates C to which distance is computed in steps [26-30] of Algorithm 2 define the cost of SIMP. We develop models to find the expected number of buckets of a hashtable $h(s)$ probed and the expected number of candidates obtained for a query $q(r_q)$.

Data distribution: For a viewpoint v and a point p, let r=$d(p,v)$ be the distance of p from v and θ be the angle of p relative to v's angular vector ov. Let $\mathcal{P}_v(r,\,\theta)$ be the spatial probability mass function of the data space relative to the viewpoint v. A hashtable $h(s)$ defines an intersection over k viewpoints. Let $[r,\,\theta]$ represent the list $[r_i, \theta_i]_{i=1}^{k}$. We represent a point p relative to k viewpoints of a hashtable $h(s)$ as $p([r,\,\theta])$, where $[r,\,\theta]$ is the list of distances and angles of p relative to all the viewpoints $v \in s$.

$$p([r,\,\theta]) = [\cup_{i=1}^{k}(r_{v_i},\,\theta_{v_i}) \text{ for all } v_i \in s] \tag{7}$$

Let $\mathcal{P}([r,\,\theta])$ be the joint spatial probability mass function of the data space over k viewpoints. Let Q be the query space. The joint spatial probability mass function $\mathcal{Q}([r,\,\theta])$ of the query space is taken to be similar to the data space mass function $\mathcal{P}([r,\,\theta])$. All the expectations are computed with respect to query mass function $\mathcal{Q}([r,\,\theta])$. A query $q(r_q)$ is represented as $q(r_q, [r,\,\theta])$ relative to k viewpoints.

Expected number of hash buckets probed: We compute the set of bins BC of a polar grid $\mathcal{G}(v)$ enclosing the qball of the query $q(r_q)$ using Algorithm 1. All the bits of *isEmpty* bit array are taken to be false. The total number of buckets $Y([r,\theta])$ of a hashtable $h(s)$, whose signature s has k viewpoints, probed for a query $q(r_q,$ $[r,\,\theta])$ is given by

$$Y([r,\,\theta]) = \prod_{i=1}^{k} \mid BC_{v_i} \mid$$

The expected number of buckets probed in a hashtable $h(s)$ is obtained by taking a sum over the query space Q with respect to the query mass function $\mathcal{Q}([r,\theta])$.

$$E(Y) = \sum_{x=1}^{|Q|} Y([r,\theta]_x) \times \mathcal{Q}([r,\theta]_x) \tag{8}$$

Expected number of candidates: To obtain the expected number of candidates $E(C)$, we first derive the probability of a random point p being chosen as a candidate by spatial intersection pruning. The bounding range B=$\{[r_1,\,r_2]$,$[\theta_1],\,\theta_2]\}$ of the qball of a query $q(r_q)$ relative to a viewpoint v is obtained as discussed in section 3.3. For a viewpoint v, the probability that p is selected as a candidate is

$$\begin{aligned} Pr_v(p \text{ is candidate}) &= Pr(p \in B) \\ &= \sum_{\theta_1}^{\theta_2} \sum_{r_1}^{r_2} \mathcal{P}_v(r,\,\theta). \end{aligned}$$

For k viewpoints, a random point p is a candidate only if p lies in the intersection of the bounding ranges of all the k viewpoints. Let $\{B_1, \cdots, B_k\}$ be the bounding ranges with respect to k viewpoints. Then, the probability that p is a candidate is

$$Pr_v(p \text{ is candidate}) = Pr(p \in B \cap \cdots \cap B_k).$$

The intersection of the bounding ranges of k viewpoints is not independent. Therefore, to obtain the probability $Pr_v(p$ is candidate$)$, we compute the bounding volume of the qball of query $q(r_q)$ for each viewpoint v independently. The bounding volume for a viewpoint v is obtained by a cartesian product of the bounding distance and angular ranges of the qball relative to v. The joint bounding

d=128			d=256		
r_q	n_z=5,000	n_z=15,000	r_q	n_z=5,000	n_z=15,000
50	0.947	0.947	300	0.703	0.693
100	0.953	0.952	400	0.698	0.687
150	0.954	0.954	500	0.739	0.733
200	0.954	0.954	600	0.743	0.734

Table 2: Error ratio $\xi(\%)$ for expected number of candidates $E(C)$ for varying query ranges r_q and varying number of mballs n_z.

volume of k viewpoints is obtained by

$$\text{Vol} = \prod_{i=1}^{k} [r_1^i, r_2^i] \times [\theta_1^i, \theta_2^i].$$

The probability that a random point p is a candidate, if k polar grids are used, is obtained by taking a sum of the joint spatial probability mass function $\mathcal{P}([r, \theta])$ of the data space over the joint volume

$$Pr^{SIP}(p \text{ is candidate}) = \sum_{Vol} \mathcal{P}([r, \theta]). \quad (9)$$

Next, we derive the probability that a random point p is chosen as a candidate by metric pruning. The distance probability distribution of points of a data space relative to a point p_1 is given by

$$\mathcal{F}_{p_1}(r) = Pr(d(p_1, p) \leq r).$$

The probability that a random point p, having the nearest mcenter z, is a candidate for a query $q(r_q)$ is

$$\begin{aligned} Pr^{MP}(p \text{ is candidate }) &= Pr(\mid d(p,z) - d(q,z) \mid \leq r_q) \\ &= \mathcal{F}_q(r_q) \end{aligned} \quad (10)$$

where $\mathcal{F}_q$ is the distance distribution of $\mid d(p,z) - d(q,z) \mid$ relative to q. It is not feasible to compute $\mathcal{F}_q$ at runtime or store it for all possible queries. Therefore, we approximate it with $\mathcal{F}_{z_q}$, which is the distance distribution of $\mid d(p,z) - d(z_q,z) \mid$ relative to the nearest mcenter z_q of the query point q.

The probability of a random point p being chosen as a candidate for query $q(r_q, [r, \theta])$ using both SIP and MP is:

$$\begin{aligned} Pr(p_{candidate}) = &\ Pr^{MP}(p \text{ is candidate}) \\ &\times Pr^{SIP}(p \text{ is candidate}) \end{aligned} \quad (11)$$

The total number of candidates C for a query $q(r_q, [r, \theta])$ is given by C=$Pr(p \text{ is cand}) \times N$, where N is the dataset size. The expected number of candidates $E(C)$ is obtained by taking a sum over the query space Q with respect to query mass function $\mathcal{Q}([r, \theta])$.

$$E(C) = \sum_{x=1}^{|Q|} Pr(p \text{ is candidate }) \times \mathcal{Q}([r, \theta]_x) \times N \quad (12)$$

It is worth noting that Equation 12 gives the expected number of candidates $E(C)$ for Algorithm SIMP. If the distance probability distribution $\mathcal{F'}_q$ of the data points relative to the query point q is known, then the actual number of candidates is $\mathcal{F'}_q(r_q) \times N$.

We empirically verified the robustness of our model for $E(C)$ using error ratio ξ. If $E(C_a)$ is the average number of candidates obtained empirically, then

$$\xi = \mid E(C_a) - E(C) \mid / E(C_a) \quad (13)$$

We computed $\xi(\%)$ for multiple query ranges r_q and various number of mballs n_z on two real datasets of dimensions 128(SIFT) and 256 (CHist) which are described in section 5. The values of $\xi(\%)$ are shown in table 2. We see that $\xi(\%)$ on both the datasets is less than 1% for all query ranges r_q and the number of balls n_z.

Space complexity: SIMP has a linear space complexity in the dataset size N. We compute the memory footprint of SIMP for a d-dimensional dataset having N points. Let the space usage of a word be W bytes. Let each dimension of a point take one word. Then, the space cost of the dataset is $(N \times d \times W)$ bytes. Let a point identifier take one word. A point is stored by its identifier in a hashtable. Therefore, the space required for L hashtables is $(L \times N \times W)$ bytes. The space cost of n_z mcenters is $(n_z \times d \times W)$ bytes. $(N \times log_2(d_{max}) + N \times log_2(n_z))$ bytes are required to store the distance and the identifier of the nearest mcenter for each point. d_{max} is the maximum distance of a point from its nearest mcenter. We fairly assume that $log_2(d_{max}) \leq W$ and $log_2(n_z) < W/2$. Therefore, the total memory footprint is $N \times W \times (d + L + (n_z/N)d + 1 + 1/2)$ bytes = $a \times N \times W \propto N$. Here, a is a constant proportional to d and W is a constant. Thus, we see that the space complexity of SIMP is $O(N)$.

5. EMPIRICAL EVALUATIONS

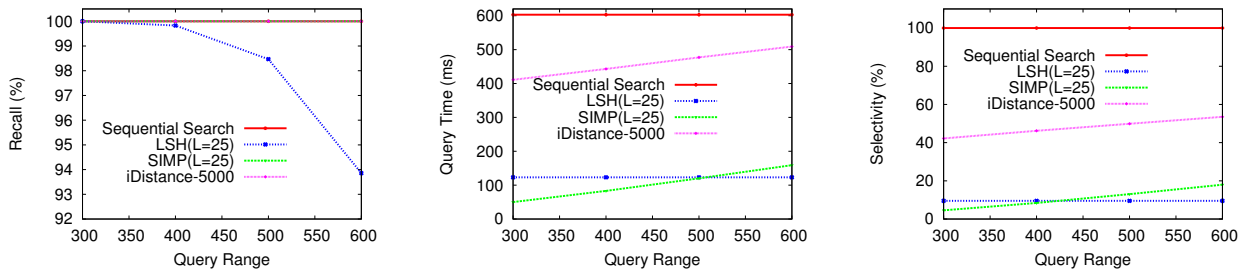
We empirically evaluated the performance of SIMP on five real datasets. We compared SIMP with four alternative methods: (1) p-stable LSH [12], (2) Multi-Probe LSH [26], (3) LSB tree [33], and (4) iDistance [19]. All these methods are briefly described in section 2. p-stable LSH and iDistance are state-of-the-art methods for an approximate and an exact search respectively, while Multi-Probe LSH and LSB tree have been recently proposed. We first introduce the datasets and the metrics used for measuring the performance of the algorithms. Then we describe the query workload for our experiments and the construction of an specific instance of SIMP index. Next we describe the performance comparison of SIMP with the alternative methods. Finally, we show scalability results of SIMP on datasets having as many as 100 million points.

Dataset description: We used 5 real datasets of various dimensions and sizes for our experiments. The first real dataset, called *SIFT*, contains 128-dimensional 1 million SIFT [25] feature vectors extracted from real images [20]. SIFT is a state-of-the-art feature used for content based image retrieval and object recognition. The second dataset, called *SIFT10M*, and the third dataset, called *SIFT100M*, has 10 million and 100 million 128-dimensional SIFT feature vectors of real images respectively. We obtained these three datasets from INRIA Holiday dataset[1].

The fourth real dataset, called *CHist*, has 256-dimensional $1,082$-$,476$ color histograms of images. For this, we downloaded random images from Flickr[2]. We transformed each image into gray-scale. Then we extracted a 256-dimensional histogram from each image by counting the number of occurrences of each color in the image. The fifth dataset, called *Aerial* [32], has 10 million points of 32 dimensions. We obtained $82,282$ gray-scale aerial images from the Alexandria Digital Library[3]. These aerial images are satellite images and air photos of different regions of California. The size of these images varies from 320×160 pixels to 640×480 pixels. We split each of the aerial images into non-overlapping tiles of size 32×32 pixels. The total number of tiles obtained are $10,625,200$. We computed a 32-dimensional histogram of the pixel values of each tile in a manner similar to Color Structure Descriptor [27].

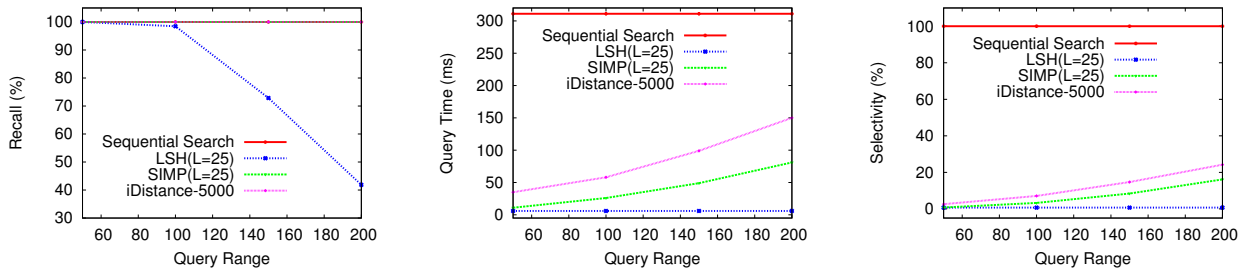
Performance metrics: We measured the performance of the algorithms using following metrics: (1) *recall*, (2) *selectivity*, (3) *query time*, and (4) *space usage*. These metrics validate the quality of results, the efficiency, and the scalability of the algorithms. *Recall* measures the result quality of an algorithm. It is the ratio of the

[1]http://lear.inrialpes.fr/jegou/data.php
[2]http://www.flickr.com/
[3]http://www.alexandria.ucsb.edu/



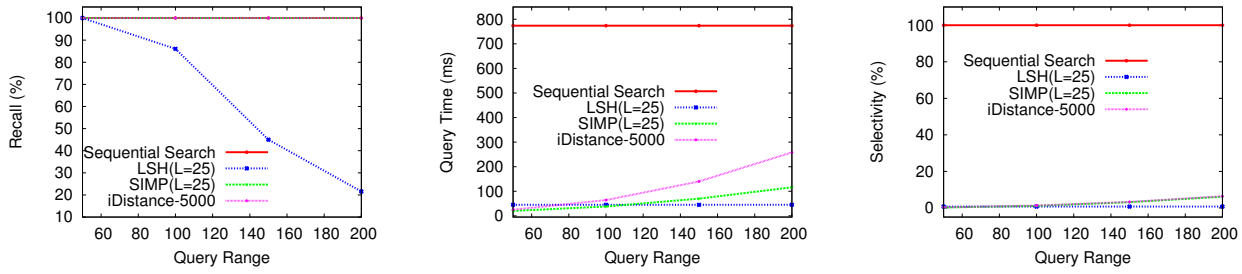
(a) Recall (b) Query Time (c) Selectivity

Figure 7: Comparative study of performance of SIMP with p-Stable LSH and iDistance on 256-dimensional CHist dataset.



(a) Recall (b) Query Time (c) Selectivity

Figure 8: Comparative study of performance of SIMP with p-Stable LSH and iDistance on 128-dimensional SIFT dataset.



(a) Recall (b) Query Time (c) Selectivity

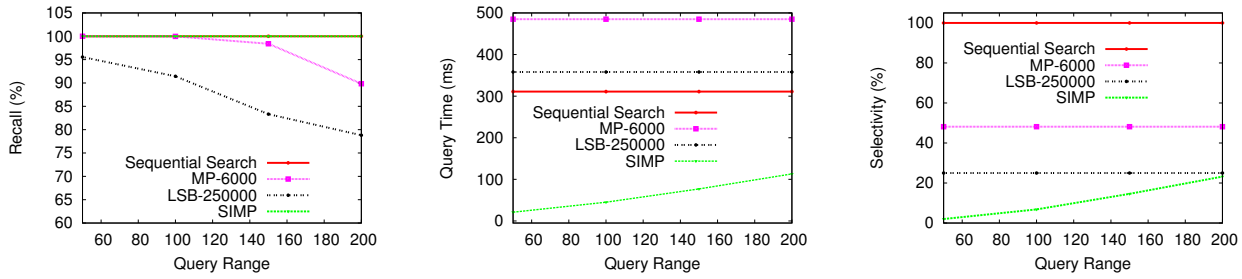
Figure 9: Comparative study of performance of SIMP with p-Stable LSH and iDistance on 32-dimensional Aerial dataset.

number of the true neighbors retrieved by an algorithm to the total number of the true neighbors in the dataset for a query. The true near neighbors of a query in a dataset are obtained using sequential search. iDistance and SIMP have 100% recall. The efficiency of the algorithms are measured by their selectivity, query time, and space usage. *Selectivity* of an indexing scheme is the percentage of the data points in a dataset for which the actual distance from the query is computed. *Query time* is the elapsed CPU time between the start and the completion of a query. We verify the space efficiency of the algorithms by computing the memory footprints of their index structures. The main factors governing the query cost of SIMP are the dataset size N, the dataset dimension d, and the query range r_q. We verify the scalability of SIMP by computing its query time for varying values of N, d, and r_q. We observed a sequential search time of 311ms for SIFT, 603ms for CHist, 774ms for Aerial, $3,209$ms for SIFT10M, and $28,000$ms for SIFT100M.

Query workload: We randomly picked $1,000$ query points from each of the SIFT, CHist, and Aerial datasets. We used the queries of SIFT also for both SIFT10M and SIFT100M. Each result on a dataset is reported as an average over all its query points. We performed experiments for multiple query ranges r_q for each query point q. We used query ranges r_q= $\{50, 100, 150, 200\}$ for SIFT, Aerial, SIFT10M, and SIFT100M and query ranges r_q=$\{300, 400, 500, 600\}$ for CHist. For a dataset, query ranges are chosen such that at least 90% of its data points have their top-1 nearest neighbors within the largest query range. We computed the cumulative mass function of distances of top-1 nearest neighbors of a large set of random query points from each dataset. We found that more than 90% of the queries of SIFT, Aerial, SIFT10M, and SIFT100M have their top-1 nearest neighbor within a query range of 200. The same was true for the query range of 600 for CHist.

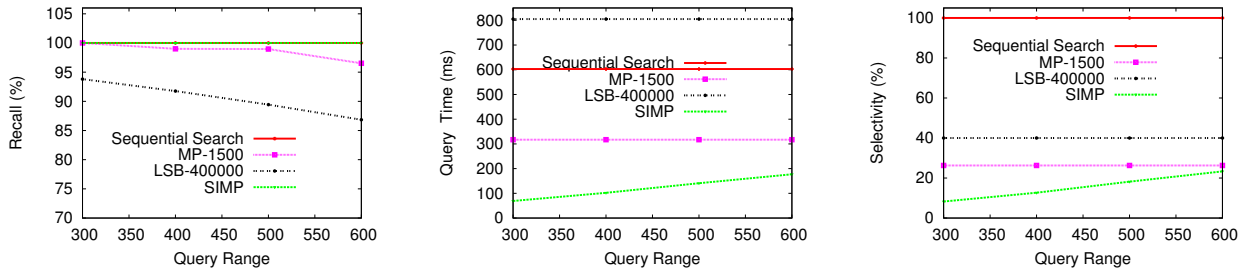
SIMP index: Here we describe a specific construct of SIMP index. The viewpoints for SIMP are picked randomly from the dataset. SIMP uses a fixed signature size of k=4 for its hashtables. The number of hashtables L is decided based on the memory constraints. We used two values of L=$\{1, 25\}$ for our experiments. SIMP requires values of w_r and w_θ to create polar grids. A fixed value of w_θ=45° is used for creating the polar grids. The value of w_r is learned for each dataset by training. For our experiments, we chose a training query range r_0 and also randomly picked a set of query points from each dataset. Then we measured the performance of SIMP for a set of values of w_r using r_0 and the query points. We chose the value of w_r which produced the best result for the dataset. We used k-means clustering to find mballs and mcenters for metric pruning (MP). For our experiments, we used n_z=$5,000$ mballs for metric pruning.

All the experiments were performed on Debian GNU/Linux 5.0 and quad-core Intel(R) Xeon(R) CPU $5,140$@2.33GHz with 4MB



(a) Recall (b) Query Time (c) Selectivity

Figure 10: Comparative study of performance of SIMP with Multi-Probe LSH (MP) and LSB Tree on 128-dimensional SIFT dataset.



(a) Recall (b) Query Time (c) Selectivity

Figure 11: Comparative study of performance of SIMP with Multi-Probe LSH (MP) and LSB Tree on 256-dimensional CHist dataset.

cache. All the programs were implemented in Java. We used Java Hotspot 64-bit (16.3 build) Server VM.

5.1 Performance comparison with p-stable LSH and iDistance

We present the performance comparison of SIMP with p-stable LSH [12] and iDistance [19] on SIFT, CHist, and Aerial datasets. We first describe the settings of the algorithms used for comparative studies. Then we give empirical evidences to show that SIMP is much superior than LSH on the result quality. Next we empirically show that SIMP is much more efficient than iDistance for all datasets and query ranges. We also show that SIMP scales linearly with the dataset dimension d and the query range r_q. The scalability of SIMP with the dataset size is shown in section 5.3. Finally, we compare the space efficiency of these algorithms.

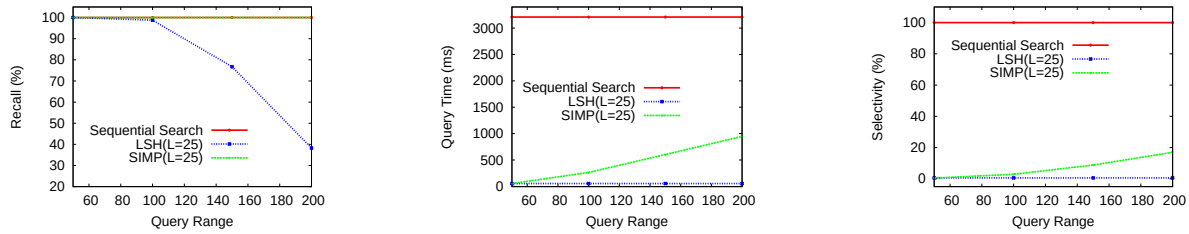
We used following settings of the algorithms for comparison. We implemented p-stable LSH similar to Datar et al. [12] for Euclidean norm. The parameters of LSH are the number of hashtables L, the number of hash values k' concatenated to generate a hash key, and the bin-width w used to bucket the projected values. We used the same number of hashtables L=25 for both LSH and SIMP. We learned the values of k' and w for LSH. We chose the query range r_0=50 for SIFT and Aerial and the query range r_0=300 for CHist for learning the parameters k' and w of LSH and the parameter w_r for SIMP. We measured the performance of LSH for a set of values of k' and w on each dataset using the training query range r_0 and a set of randomly picked query points. We chose the values of k' and w for which LSH had 100% recall with the least query time. For LSH, we learned the values w=1, 700 and k'=8 on CHist, w=350 and k'=8 on SIFT, and w=250 and k'=8 on Aerial. For SIMP, we learned the values w_r=300 on CHist, w_r=30 on SIFT, and w_r=100 on Aerial. We used n_z=5, 000 mballs for SIMP. We used the mcenters of these mballs as reference points for iDistance.

We observed that SIMP always guarantees 100% quality compared to LSH whose quality falls below 50% for larger query ranges. We show the performance of the algorithms on CHist, SIFT, and Aerial datasets in figures 7, 8, and 9 respectively. We see that SIMP and iDistance have 100% recall on datasets of all sizes and dimensions and for all query ranges. Unlike SIMP, the recall of LSH falls rapidly with an increase in the query range. LSH had a recall of only 21.6% for r_q=200 on Aerial dataset.

Our empirical results show that SIMP has a superior performance than iDistance on datasets of all sizes and dimensions and for all query ranges. Both the methods always yield 100% result quality but SIMP significantly outperforms iDistance in efficiency. We see from figures 7, 8, and 9 that iDistance has larger query time and selectivity than SIMP on all the datasets. This difference in performance widens with an increase in the dimension of the datasets and the query range. SIMP had a selectivity of 5% compared to 42% selectivity of iDistance on CHist dataset for the query range r_q=300 as seen in figure 7.

We observed that the selectivity and the query time of SIMP grows linearly with the dataset dimension d and the query range r_q. We see from figures 9 and 8 that SIMP has a selectivity of 0.2% and 0.7% on 32-dimensional Aerial and 128-dimensional SIFT datasets respectively for the query range 50. This shows that the selectivity of SIMP grows linearly with d. From figure 7, we see that the selectivity of SIMP grows from 5% for r_q=300 to 17% for r_q=600 on CHist dataset. This validates that the selectivity of SIMP grows linearly with r_q. The small values of the selectivity of SIMP on all the datasets verify that SIMP effectively prunes false candidates using its index structure. The linear behavior of SIMP with d and r_q is further confirmed by its query time on all the three datasets. We see from figure 7 that the query time of SIMP grows linearly from 50ms for r_q=300 to 150ms for r_q=600 on CHist. The query time of SIMP also increases only linearly with d.

It is evident from the empirical results that SIMP is a superior alternative for an accurate and efficient r-NN search. p-stable LSH and SIMP have similar performance for the training query range r_0. With an increase in the query range $r_q > r_0$, the search quality of p-stable LSH falls sharply, whereas SIMP gives 100% result quality with only a linear increase in the search cost. Further, p-stable LSH is inefficient for queries with $r_q < r_0$. Thus, we see that LSH index structure created for a fixed query range r_0 can



(a) Recall (b) Query Time (c) Selectivity

Figure 12: Comparative study of performance of SIMP with p-Stable LSH on 128-dimensional 10 million SIFT points.

not handle queries with varying query ranges accurately and efficiently. SIMP performs much better than iDistance across all the datasets of various dimensions. The performance difference between SIMP and iDistance grows with an increase in the dimension of the dataset. The poor performance of iDistance is because of its multiple searches on the B+ Tree and high selectivity. iDistance searches B+ tree for each mball intersected by an r-NN query.

Space cost comparison: We now discuss the space costs of SIMP, p-stable LSH, and iDistance. We compute memory footprints of the algorithms for CHist dataset using parameters W=4, N=1, 082, 476, L=25, d_{max}=22, 500, and n_z=5, 000. d_{max} is the maximum distance of any point from its nearest mcenter. The space cost of the dataset for each algorithm is $(N \times d \times W)$=1, 109MB. The memory footprint of LSH index is $(N \times W \times L)$=108MB and SIMP index is $(N \times W \times L) + (n_z \times d \times W + N \times log_2(d_{max}) + N \times log_2(n_z))$=117MB. The extra space usage of SIMP over p-stable LSH is from the data structures of the metric pruning. This overhead remains constant for any value of L. The index structure of iDistance needs a total of 14MB space. Each entry of iDistance needs 32 bits for storing its distance from nearest mcenter as key and 32 bits to store a pointer to a child node or a data object. Each leaf node also needs 8 bytes for storing pointers of its neighboring nodes. iDistance needs 5.12MB for storing 5, 000 mcenters. We take 512 entries per node for B+ tree of iDistance. For L=1, the memory usage of SIMP and iDistance is the same.

5.2 Performance comparison with Multi-Probe LSH and LSB Tree

Here we describe the performance comparison of SIMP with Multi-Probe LSH [26] and LSB tree [33] on 128-dimensional SIFT and 256-dimensional CHist datasets. We first describe the settings of the algorithms used for the comparison. Then we present results to show that SIMP significantly outperforms both Multi-Probe LSH and LSB tree on all the datasets for all the query ranges. Finally, we compare the space efficiency of the algorithms.

We used a similar index structure for Multi-Probe LSH as p-stable LSH. We implemented the Query-Directed probing sequence algorithm for Multi-Probe LSH as it was shown to perform better than the step-wise probing sequence [26]. Multi-Probe search algorithm does not have a terminating condition, whereas LSB search algorithm has a terminating condition for top-k search. Therefore, we made the following choices for the termination of Multi-Probe and LSB Tree in order to have a meaningful comparison, based on both quality and efficiency, with SIMP for r-NN queries. We terminated Multi-Probe after a fixed number of probes P. LSB search was terminated for a fixed selectivity S that is the percentage of the dataset explored by LSB search as candidates. We did not compare against LSB forest because of its high space overhead, which can be observed from its parameters in table 3 and has also been noted by the authors. We used L=1 hashtable for both Multi-Probe and

Dataset	t	f	p_2	m	L	H_{max}	u	B	w
SIFT	217	15	0.61	24	354	478019	18	4096	4
CHist	22500	23	0.61	26	521	1.3327E8	27	4096	4

Table 3: Parameters of LSB Tree and LSB Forest for two real datasets.

SIMP. We used n_z=5, 000 mballs for SIMP. We learned the values w_r=300 and w_r=50 on CHist and SIFT datasets respectively for SIMP. We learned the values w=1, 700 and k'=8 on CHist and w=350 and k'=8 on SIFT for Multi-Probe.

A performance comparison of the algorithms on SIFT and CHist datasets are shown in figures 10 and 11 respectively. We measured the performance of Multi-Probe for P=6, 000 probes and LSB tree for the selectivity S=25% on SIFT dataset. We used P=1, 500 probes for Multi-Probe LSH and a selectivity S=40% for LSB tree on CHist dataset. Figures 10 and 11 show that the recall of both Multi-Probe LSH and LSB tree decreases with an increase in the query range, while SIMP has 100% recall for all query ranges. Multi-Probe had a recall of 90% and LSB had a recall of 79% for the query range 200 on SIFT dataset as seen from figure 10. These results verify that SIMP always yields superior result quality than Multi-Probe and LSB.

We empirically found that SIMP is much more efficient than Multi-Probe and LSB tree. We see from figures 10 and 11 that both Multi-Probe and LSB have larger selectivity and query time than SIMP for all query ranges r_q on both the datasets. For r_q=50 on SIFT dataset, Multi-Probe was 24 times slower and had 24 times more selectivity than SIMP. For r_q=50 on SIFT dataset, LSB was 17 times slower and had 12 times more selectivity than SIMP. SIMP was also significantly better than LSB Tree and Multi-Probe on CHist dataset.

Our empirical evidences show that SIMP, that guarantees 100% quality, is a superior alternative for an accurate and efficient r-NN search over Multi-Probe LSH and LSB tree. Multi-Probe is a heuristic with no performance and quality guarantees. A large number of probes improves the result quality of Multi-Probe but worsens its efficiency. Multi-Probe yields a high query time for a large number of probes because of the high cost of the computation of the probing sequence and a high selectivity. The poor performance of LSB can be mainly attributed to two reasons. First, m-dimensional points obtained after projections are indexed into a B-Tree based on their z-order values. The value of m increases with an increase in the database size and dimension for constant values of the other parameters. It is well known from the literature that the performance of tree-based indices deteriorate for large dimensions, and so does B-Tree based LSB tree. The value of m is 24 for SIFT and 26 for CHist as seen in table 3, which are sufficiently large to make the LSB tree inefficient. Second, its query time increases with an increase in candidate size because of a large number of bit operations required for computing LLCP between

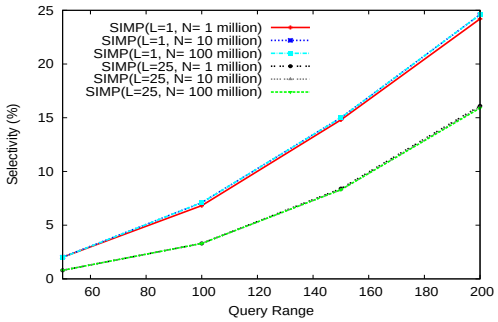


Figure 13: Selectivity of SIMP on 128-dimensional real datasets of varying sizes for varying number of hashtables L.

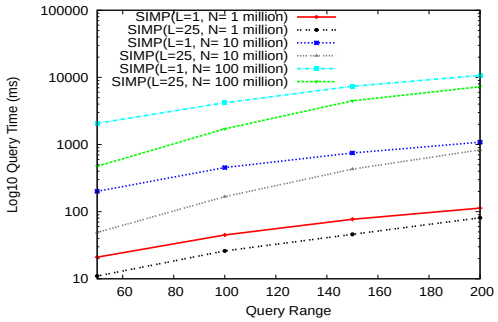


Figure 14: Query time of SIMP on 128-dimensional real datasets of varying sizes for varying number of hashtables L.

two z-orders. The size of a z-order value is $m \times u$=24 $\times$ 18=432 bits for SIFT dataset and 26 $\times$ 27=702 bits for CHist dataset.

Space cost comparison: Here, we discuss the space costs of each of the algorithms using parameters d=256, N=1, 082, 476, W=4, L=1, and n_z=5, 000. The space cost of the dataset for each algorithm is 1, 109MB. The memory footprint of Multi-Probe index is 4.5MB and SIMP is 13MB. The hashtables of Multi-Probe LSH and SIMP store only the identifier of a point, which takes one word (W bytes). The extra space usage of SIMP over Multi-Probe is again from the data structures of metric pruning. We compute the memory required by LSB tree using the parameters of CHist shown in table 3. We use 512 entries per node for LSB. Each entry of LSB tree stores a z-order value (to compute LLCP) as key and a pointer to a child node or a data object. A z-order value needs $m \times u$=702 bits for storage and a pointer needs 32 bits of storage. LSB tree also needs to store forward and backward pointers of the immediate neighbors of each leaf node. Thus, the total space required for LSB is 100MB. For L=1, we find that the memory footprint of LSB is at least 7 times worse than SIMP and 22 times worse than Multi-Probe. For L=2, LSB takes 200MB whereas the space cost of SIMP and Multi-Probe increase only by 4.5MB (for storing an extra hashtable) to 17.5MB and 9MB respectively.

5.3 Large Scale Performance Evaluation

We validated the scalability of SIMP on 128-dimensional real datasets of sizes up to 100 million. Our stress tests reveal that SIMP scales linearly with the dataset size and the query range at very high dimensions. We already showed in section 5.1 that SIMP scales linearly with the dataset dimension. We also further compared SIMP with p-stable LSH for a large workload of 14, 700 queries on 128-dimensional SIFT10M dataset having 10 million points. This test again confirmed that SIMP efficiently and accurately queries near neighbors for any query range, while p-stable LSH has a very poor

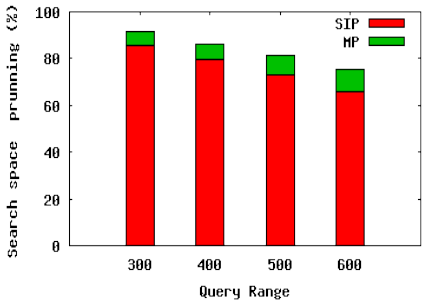


Figure 15: Data pruning (%) obtained by SIP and MP pruning steps of SIMP using n_z=5, 000 mballs for varying query ranges on CHist dataset.

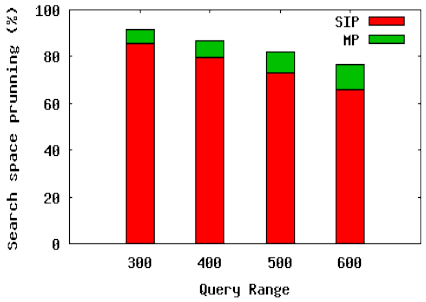


Figure 16: Data pruning (%) obtained by SIP and MP pruning steps of SIMP using n_z=15, 000 mballs for varying query ranges on CHist dataset.

recall for query ranges larger than the training query range r_0. We used a value of w_r=30 and n_z=5, 000 for SIMP for these studies.

We computed the selectivity and the query time of SIMP on SIFT, SIFT10M, and SIFT100M datasets to verify its scalability. We computed these values for varying number of hashtables L={1, 25} and varying query ranges r_q. We show the selectivity and the query time of SIMP in figures 13 and 14 respectively. Each result is an average over 1, 000 random queries. These results reveal that SIMP has similar selectivity for the dataset of any size for a given r_q and L. This property implies that SIMP scales linearly with the dataset size. This is further confirmed by the query time of SIMP. We see from figure 14 that the query time of SIMP increases approximately 10 times with 10 times increase in the dataset for a given r_q and L. We also observed that the query time of SIMP has a linear behavior with the query range.

SIMP had a query time of 0.4 seconds on 100 million points for r_q=50 and L=25 compared to 28 seconds of sequential search. For SIFT100M dataset, we observed by random sampling that every point has at least one near neighbor within the query range 50. This shows that SIMP can be used to efficiently and accurately find the nearest neighbors in very large datasets of very high dimensions.

The comparative results of SIMP with p-stable LSH on SIFT10M dataset for 14, 700 random queries is shown in figure 12. We used a value of L=25 for both SIMP and LSH. We learned the values of w=350 and k'=8 for LSH using r_0=50. We observed that SIMP is 60 times faster than sequential search for r_q=50. For r_q=50, LSH had a query time of 54ms and a selectivity of 0.59% compared to a query time of 53ms and a selectivity of 0.48% for SIMP. We found that recall of LSH fell to 38.10% for r_q=200 unlike SIMP which had 100% recall for all query ranges.

5.4 Effectiveness of Pruning Criteria

We performed experiments on CHist dataset to study the prun-

ing effectiveness of spatial intersection pruning (SIP) and metric pruning (MP) for varying query ranges. We show the results in figure 15 for n_z=5,000 and figure 16 for n_z=15,000. We observed that the total pruning achieved by SIMP decreases with an increase in the query range for a given number of mballs n_z. SIP being the first step contributes the most to the pruning. MP provides an additional pruning over SIP. We also observed that the contribution of MP increases with an increase in the query range.

6. PARAMETER SELECTION FOR SIMP

The tunable parameters of SIMP are the number of hashtables L, the radial bin-width w_r of polar grids, and the number of mballs n_z used for metric pruning. The parameters L and w_r play a similar role for SIMP as the number of hashtables and the bin-width of p-stable LSH. The parameter w_r is learned by training SIMP on a dataset using a training query range r_0. Though SIMP outperforms existing techniques even for L=1 hashtable, a better performance is achieved by using a larger number of hashtables. The value of L can be determined based on the available memory. The mcenters of SIMP play a similar role as the reference points of iDistance. The number of mballs n_z should be determined based on the data distribution. It can be computed by fixing a value of Root Mean Square Error for each cluster. It can also be learned by the methods proposed by Jagadish et al. [19] for iDistance. We suggest to use a value of n_z=5,000.

7. CONCLUSION

In this paper, we proposed SIMP for answering r-NN queries in a high dimensional space. SIMP offers both 100% accuracy and efficiency for any query range unlike state-of-the-art methods. SIMP uses projection, spatial intersection, and triangle inequality to achieve a high rate of pruning, and thus gains high performance. We efficiently implemented the spatial intersection approach by hashing. We also developed statistical cost models to measure SIMP's performance. SIMP captures data distribution through its viewpoints and mcenters. We empirically showed a better performance of SIMP over p-Stable LSH and iDistance on three real datasets of dimensions 32, 128, and 256 and sizes 10 million, 1 million, and 1.08 million respectively. We also showed a much superior performance of SIMP over Multi-Probe LSH and LSB tree on two real datasets of dimensions 128 and 256 and sizes 1 million and 1.08 million respectively. We empirically validated on the datasets of sizes up to 100 million and dimensions up to 256 that SIMP scales linearly with the query range, the dataset size, and the dataset dimension.

8. REFERENCES

[1] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Commun. ACM*, 51(1):117–122, 2008.

[2] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Wu. An optimal algorithm for approximate nearest neighbor searching. In *SODA*, pages 573–582, 1994.

[3] M. Bawa, T. Condie, and P. Ganesan. Lsh forest: self-tuning indexes for similarity search. In *WWW*, pages 651–660, 2005.

[4] J. L. Bentley. Multidimensional binary search trees used for associative searching. 18(9):509–517, 1975.

[5] S. Berchtold, C. Böhm, D. A. Keim, and H.-P. Kriegel. A cost model for nearest neighbor search in high-dimensional data space. In *PODS*, pages 78–86, 1997.

[6] S. Berchtold, D. A. Keim, H.-P. Kriegel, and T. Seidl. Indexing the solution space: A new technique for nearest neighbor search in high-dimensional space. *IEEE TKDE*, 12(1):45–57, 2000.

[7] C. Böhm. A cost model for query processing in high-dimensional data. *ACM TDS*, 25:129–178, 2000.

[8] J. Buhler. Efficient large-scale sequence comparison by locality-sensitive hashing. *Bioinformatics*, 17:419–428, 2001.

[9] M. S. Charikar. Similarity estimation techniques from rounding algorithms. In *ACM STOC*, pages 380–388, 2002.

[10] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *VLDB*, pages 426–435, 1997.

[11] P. Ciaccia, M. Patella, and P. Zezula. A cost model for similarity queries in metric spaces. In *PODS*, pages 59–68, 1998.

[12] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Symposium on Computational Geometry*, pages 253–262, 2004.

[13] W. Dong, Z. Wang, W. Josephson, M. Charikar, and K. Li. Modeling lsh for performance tuning. In *CIKM*, pages 669–678, 2008.

[14] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD*, pages 226–231. AAAI Press, 1996.

[15] V. Gaede and O. Günther. Multidimensional access methods. *ACM Comput. Surv.*, 30(2):170–231, 1998.

[16] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *VLDB*, pages 518–529, 1999.

[17] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *SIGMOD*, page 47âĂŞ57, 1984.

[18] P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *STOC*, 1998.

[19] H. V. Jagadish, B. C. Ooi, K.-L. Tan, C. Yu, and R. Zhang. idistance: An adaptive b+-tree based indexing method for nearest neighbor search. *ACM TDS*, 30(2):364–397, 2005.

[20] H. Jégou, M. Douze, and C. Schmid. Product quantization for nearest neighbor search. *IEEE TPAMI*, 2010.

[21] A. Joly and O. Buisson. A posteriori multi-probe locality sensitive hashing. In *ACM MM*, pages 209–218, 2008.

[22] C. A. Lang and A. K. Singh. Faster similarity search for multimedia data via query transformations. *Int. J. Image Graphics*, pages 3–30, 2003.

[23] J. K. Lawder and P. J. H. King. Using space-filling curves for multi-dimensional indexing. In *BNCOD*, pages 20–35, 2000.

[24] H. Lejsek, F. H. Ásmundsson, B. P. Jónsson, and L. Amsaleg. Nv-tree: An efficient disk-based index for approximate search in very large high-dimensional collections. *IEEE TPAMI*, 31:869–883, 2009.

[25] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 60(2):91–110, 2004.

[26] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li. Multi-probe lsh: efficient indexing for high-dimensional similarity search. In *VLDB*, pages 950–961, 2007.

[27] B. S. Manjunath, P. Salembier, and T. Sikora. *Introduction to MPEG-7: Multimedia Content Description Interface*. Wiley, 2002.

[28] R. Motwani, A. Naor, and R. Panigrahi. Lower bounds on locality sensitive hashing. In *SCG '06*, pages 154–157, 2006.

[29] R. Panigrahy. Entropy based nearest neighbor search in high dimensions. In *SODA*, pages 1186–1195, 2006.

[30] H. Samet. *Foundations of Multidimensional and Metric Data Structures*. Morgan Kaufmann Publishers Inc., 2005.

[31] S. Shekhar and Y. Huang. Discovering spatial co-location patterns: A summary of results. In *Proceedings of the 7th International Symposium on Advances in Spatial and Temporal Databases*, pages 236–256, 2001.

[32] V. Singh, A. Bhattacharya, and A. K. Singh. Querying spatial patterns. In *EDBT*, pages 418–429, 2010.

[33] Y. Tao, K. Yi, C. Sheng, and P. Kalnis. Quality and efficiency in high dimensional nearest neighbor search. In *SIGMOD*, pages 563–576, 2009.

[34] R. Weber, H.-J. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *VLDB*, pages 194–205, 1998.

[35] Z. Zhang, M. Hadjieleftheriou, B. C. Ooi, and D. Srivastava. Bed-tree: an all-purpose index structure for string similarity search based on edit distance. In *SIGMOD*, pages 915–926, 2010.