Constrained Nearest Neighbor Queries

H. Ferhatosmanoglu, I. Stanoi, D. Agrawal, A. El Abbadi
Computer Science Department, University of California at Santa Barbara
{hakan,ioana,agrawal,amr@cs.ucsb.edu}

Abstract

In this paper we introduce the notion of *constrained nearest neighbor queries (CNN)* and propose a series of methods to answer them. This class of queries can be thought of as nearest neighbor queries with range constraints. Although both nearest neighbor and range queries have been analyzed extensively in previous literature, the implications of constrained nearest neighbor queries have not been discussed. Due to their versatility, CNN queries are suitable to a wide range of applications from GIS systems to reverse nearest neighbor queries and multimedia applications. We develop methods for answering CNN queries with different properties and advantages. We prove the optimality (with respect to I/O cost) of one of the techniques proposed in this paper. The superiority of the proposed technique is shown by a performance analysis.

1 Introduction

Two dimensional range queries are used frequently in various applications such as spatial databases [Sam89, GG98] and Geographic Information Systems (GIS) [CDN+97]. In such applications the data points are usually represented by two dimensional vectors corresponding to their locations. Since rectangular geometry is easy to handle, a typical approach in GIS is to handle complex shapes by simplifying them to their minimum bounding boxes. However, emerging applications are being required to offer more user flexibility in defining queries over various types of regions. In the context of geo-spatial digital libraries for example, recently there have been proposals for GIS applications to handle queries with more complex and accurate structures, such as polygons. Numerous index structures have been developed to facilitate range searching in two and higher dimensions including grid files [NHK84], quad-trees [Sam89], kdb-trees [Rob81], hB-trees [LS90], R-trees and variants [Gut84, BKSS90, BKK96].

Another very important class of queries in applications that involve spatial data is that of nearest neighbor (NN) queries [RKV95]. Similar to range queries, nearest neighbor queries are also commonly used in spatial applications. In general, the k-nearest neighbor, $k-$NN, problem is defined as finding the k nearest data points to the query point q. Traditionally NN queries span over the entire data set, and the similarity comparison is based on a distance function (e.g. Euclidean) between two points. In GIS systems for example, there are various applications of nearest neighbor queries, such as finding the closest city to the given city or the closest restaurant to the current location. In an image database a possible similarity query is to find the images most similar to a given image, where images are represented as d dimensional feature vectors.

In this paper, we highlight the importance of another type of query where the above independent classes of queries are combined. We define *constrained nearest neighbor* (CNN) queries as nearest neighbor queries that are constrained to a specified region. This type of query is targeted towards users who are particularly interested in nearest neighbor in a region bounded by certain spatial conditions, rather than in searching for nearest neighbors in the entire data space.

We develop techniques to process such constrained nearest neighbor queries, considering the following objectives. The amount of data becomes larger each day and the structure that supports indexing and query processing on large data sets should optimize the I/O cost during query processing [BBC+98]. Reducing the number of pages accessed during query processing is crucial, and a large amount of pruning of the search space during an NN search is therefore necessary. A desirable CNN technique should minimize the number of pages accessed and in general avoid the retrieval of unnecessary pages. Since both range and nearest neighbor queries are independently well-studied and efficient index structures are developed for them, the proposed technique should build upon of the current state-of-art indexing techniques that have been developed for such queries. In this paper, we focus on R-tree based structures that are widely and successfully used in spatial databases [SR87, Gut84, BKSS90, KF93, KF94, BKK96, PF99, EKK00]. We discuss techniques for constrained nearest neighbors by either merging conditions for range and nearest neighbor queries, or by modifying the current NN algorithms for R-tree like structures without changing the underlying index structure.

After presenting several examples of applications of CNN queries in Section 2, we propose methods for answering constrained nearest neighbor queries. We briefly mention two simple algorithms, i.e., the *Incremental NN Search* and *NN Search with Range Query*, which are straight forward approaches for solving the problem. They sequentially execute range and nearest neighbor queries and hence can be grouped under the common idea of *2-Phase Methods* (Section 3). We propose a single-phase technique, the *Integrated Method* in Section 4, that interleaves the range constraints with nearest neighbor conditions effectively. We show how to adapt the existing NN algorithms for processing CNN queries efficiently. In Section 5, we prove the optimality of one of the proposed techniques. In Section 6, we evaluate the proposed integrated CNN algorithm and compare it with the two-phase method. Conclusion is in Section 7.

2 Motivating Examples

There are several cases where a user may enforce spatial constraints on a nearest neighbor search. In a GIS application, an example of a CNN search is a query asking 'the nearest city, or cities, to the south of a given location'. The query point is defined in the same way as in a regular nearest neighbor query, i.e., a two dimensional point that represents the location. The query result is the closest data point to the query point that satisfies the given constraint, i.e. to the south of the query point. Note that, in a regular nearest neighbor query no such restriction can be directly specified for the query result. Figure 1(a) illustrates this query q on a simple spatial data set where the cities are stored on a map, i.e., a bounded two-dimensional data space, and represented as points with (x, y) coordinates of their locations. The query result of this particular query includes only the point(s) that have y coordinate(s) less than the query's y coordinate. The query result set r is defined as $\{r | r_y < q_y \wedge \forall p\ p_y < q_y \rightarrow d(p,q) \geq d(r,q)\}$ where r_y and q_y represent the y coordinates of the result and query point respectively, and d is the distance function between points. As can be seen in Figure 1(a) the regular nearest neighbor of the query point is point a, however the constrained nearest neighbor query returns r_1 as the query result. We note that, since the data space is bounded within a rectangle, this query is actually a nearest neighbor query restricted to the lower rectangle in Figure 1(a).

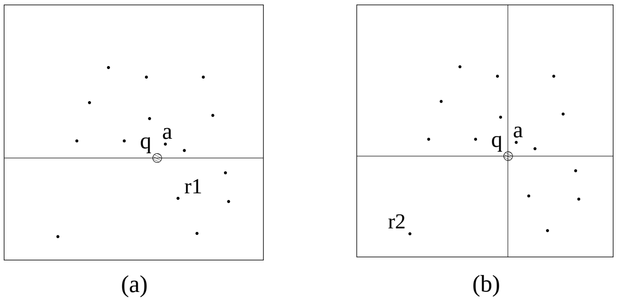
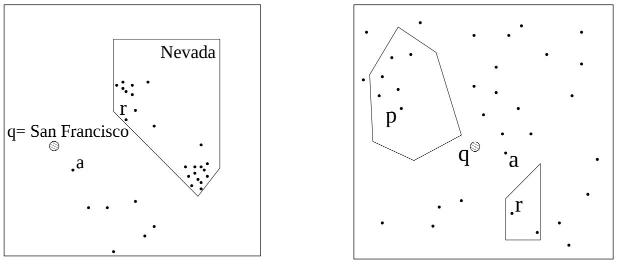


Figure 1: Constrained NN Query: Examples 1 and 2

Figure 1(b) illustrates the query 'find the nearest city to the south-west of the query point location q'. Now the query result becomes r_2 where again a is the regular nearest neighbor and r_1 was the query result of the same point q with the previous constraint. Similar to the previous example, the spatial constraint defined in this query is the restriction to the lower-left rectangle in the data space. Note that r_2 is actually the furthest point in the data set to the query point.

In the previous examples, the query point was spatially connected to the constraint. An example of a different type of constrained nearest neighbor query can be described as follows. A user who lives in the city of San Francisco may request to find 5 casinos in the state of Nevada that are the closest to San Francisco. The system defines the city of San Francisco as a two-dimensional query point q and defines a range for the state of Nevada (Figure 2(a)). The region for the state of Nevada can be defined by the user as a two-dimensional geometrical shape, such as a rectangle or an n-sided polygon. The query needs to find the closest points within this specific range. The state of Nevada is approximated by a convex pentagon and the city of San Francisco is denoted by q. The result of this query is the point r although point a would be the result of a regular NN query.

It is also desirable to be able to define multiple ranges for a single nearest neighbor query, such as asking for the nearest neighbor within multiple regions on the map, e.g. possible vacation places in several desired regions. Figure 2(b) shows a CNN query constrained to two spatial regions (a hexagon



(a) (b)

Figure 2: Constrained NN Query: Examples 3 and 4

and a quadrilateral). The regular nearest neighbor of the point q is point a, while the nearest negibhor restricted to the hexagonal constraint is point p, and the result of the NN query constrained to both regions is point r.

A final example is derived from the domain of information management for advanced transportation systems. A traveler may pose a query asking for the closest restaurants within a 2 miles neighborhood of his/her route on a particular region of a highway. The constraint is a region that covers 2 mile neighborhood, from left and right, of the highway. This region can be complicated depending on the shape of the highway. If the portion of the highway that is of interest is through a straight line, then this range is defined as a two-dimensional rectangle traversed in the middle by the highway. For more complicated highway routes, in general, the constrained range can be defined with a two-dimensional polygon.

In this paper, we present a general model for constrained nearest neighbor queries, where all the above motivational examples are subsets of this framework. We develop the algorithm for NN queries constrained to convex polygons which can be used as a primitive for several more complex queries, e.g. NN queries with non-convex polygon constraints.

3 2-Phase Algorithms for Answering CNN Queries

Constrained Nearest Neighbor Queries naturally involve both range and nearest neighbor queries. A simple solution for CNN queries can comprise two phases, incorporating in sequence these two types of queries. Different orders of these phases lead to specific advantages, that can be beneficial to different applications. We present these two alternatives to serve as a comparison with an integrated single phase approach.

Incremental NN Search. In the incremental NN search, the first step computes the non-constrained nearest neighbors by an incremental NN algorithm that outputs the k nearest points in the order of their distance to the query point. The second step checks whether the current output NN point satisfies the given constraint. At each step, the number of nearest neighbors retrieved increases by k, and the points that have already been tested are excluded from the filtering step. The algorithm continues retrieving all the pages that contain the regular nearest neighbors, until a point is retrieved that satisfies the NN constraints. There may be several regular neighbors that do not satisfy the given condition but need to be retrieved since they are closer than the query result point that satisfies the constraint. As seen in the example given in Figure 1(b), the query result of the constrained NN query is actually the furthest point to the query point. The *Incremental NN Search* approach first considers and then discards all other points in the data set before finding r_2. One special case that arises is that the queried region R might not actually contain any data points. To avoid querying the entire search space, we enforce a limit on the search: when a possible nearest neighbor is further from the query point than the constraining region R (measured by $maxdist(q, R)$), then there is no nearest neighbor in R(see Figure 3). The steps of the algorithm are shown in Figure 4.

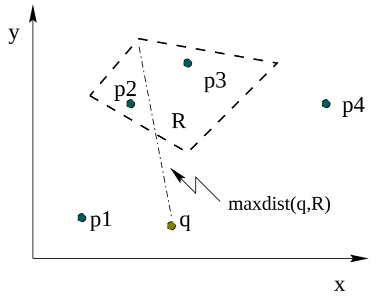


Figure 3: Illustrative Example

NN Search with Range Query. Another approach for answering constrained NN queries can reverse the two phases. For some cases, the method of first retrieving candidates for nearest neighbor data points and then testing for inclusion in the constrained region, can lead to unnecessary access of leaf nodes. Depending on the size and positioning of the constrained region R, it may be more efficient to first perform a range query that retrieves the data points in R and then testing for the nearest neighbor condition (Figure 4). The obvious tradeoff is that if the size of region R is comparable to the entire data space, then testing for nearest neighbor by comparing all distances of points in R to the query point can be computationally expensive. However, if the constrained region R includes only a relatively small number of data points / pages to be accessed, this approach can be considerably more efficient than the previous algorithm.

4 Integrated Single Phase Approach

We now propose a more efficient approach that merges the conditions of nearest neighbor and regional constraints in one phase. Our goal is to obtain the benefits of pruning the search space early on, according to both the range and nearest neighbor conditions. This requires the modification of previous nearest neighbor algorithms that search over the entire data space. We start by briefly describing prior NN algorithms and then develop an integrated CNN method.

Incremental NN Search Algorithm

let n = number of points in the data set

let P_{tested}= set of points already tested = $\emptyset$

let $Search(q,k)$ = finds k-nearest neighbors to query point q

1. for $i = 1$ *to* (n/k) do
   (a) retrieve $k \times i$ NN $\{n_1, \cdots n_{ki}\}$ with $SearchNN(q,(k \times i))$
   (b) $P_{to-test} = \{n_1, \cdots n_{ki}\} - P_{tested}$
   (c) if $\exists n \in \{n_1, \cdots n_k\}$ such that $n \in R$, then done. Return n.
   (d) else $P_{tested} = Ptested \cup P_{to-test}$
   (e) if $\exists n_i$ s.t. $dist(n_i, q) > maxdist(q, R)$, then return null.

NN Search with Range Query

1. find the set of points P retrieved by a range query over the MBR(R).
2. exclude from P the points $p_j = \{p_j \in MBR(R)$ $p_j \notin R\}$. $P = P - p_j$.
3. find $p_n = \{p_n \in P$ s.t. $dist(p_n, q) = min(dist(p_i, q))$, $p_i \in P\}$.

Figure 4: Incremental NN Search vs. NN Search with Range Query

4.1 Overview and Limitations of the NN approach

Recently, several indexing structures have been developed based on R-trees under various assumptions. Points in a data space can be grouped in clusters limited by their minimum bounding rectangles (MBR), which in turn are grouped to form larger clusters. An MBR is defined as the smallest rectangle parallel with the axis that completely encloses a given set of points or sub-rectangles. Each of its faces must therefore contain at least one of the enclosed data points. Different levels in the indexing tree correspond to different levels of granularity, where each internal node stores pointers to the rectangles contained and the coordinates to position the corresponding MBRs. Note that for 2-dimensional data an MBR can be defined by the x and y coordinates of two diagonally opposite corners.

R-trees are not only very effective as underlying indexing structures in a database, but their properties facilitate knowledge discovery algorithms. Roussopoulos et al. [RKV95] proposed to take advantage of the mapping of the data space into R-trees, and developed an efficient technique for finding a nearest neighbor $NN(q)$ to a query point q. We believe that this technique is a valid and an effective starting point for developing a method that accommodates a larger class of queries, constrained nearest neighbor searches. In [RKV95], a branch-and-bound approach was used to search for nearest neighbors to a query point, based on the depth-first traversal and pruning of the R-tree. In order to reduce the I/O overhead, a crucial part of the proposed algorithm is the exclusion from the search space of regions that cannot be part of the query answer. At each level, the pruning comparison involves distances to the children nodes as well as the distance to the previously found nearest neighbor candidate. Only nodes that can lead to a possible nearest neighbor are further considered, while the remaining leaves and corresponding subtrees are pruned out of the search space.

The proximity comparisons in [RKV95], based on the Euclidean distance metric, use the notion of $mindist(q, M)$ *(the shortest distance from the query point to a given MBR M)* and $minmaxdist(q, M)$ *(minimum, over all dimensions, of the distance from the query point to the furthest point on the closest face of the MBR)*. These metrics ensure that for a minimum bounding rectangle M, there is at least one data point within the range $[mindist(q, M), minmaxdist(q, M)]$. An MBR M' whose $mindist(q, M') > minmaxdist(q, M)$ will not lead to a nearest neighbor of query point q and therefore can be safely discarded. Similarly, a data point p whose distance to the query point satisfies the condition $dist(p, q) > minmaxdist(q, M)$, cannot be a candidate to the set of nearest neighbors. Furthermore, if there exists a point p such that $dist(q, p) < mindist(q, M)$, then the subtree rooted at MBR M should be discarded from the search space.

Since we are interested in supporting NN queries constrained to a given region, we modify the pruning criterion to satisfy the necessary constraints. Our goal is to use the existing index structure and NN algorithm, and extend the notion of $mindist(q, M)$ and $minmaxdist(q, M)$ so that the pruning rules described above still hold. To illustrate the necessity of new conditions, Figure 5 presents the different cases for containment of an MBR in a region R.

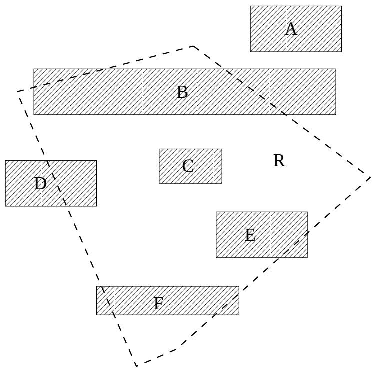


Figure 5: Different cases for positioning of MBRs with respect to a region R

As shown, an MBR can be outside of the given region, or either fully or partially in R. Consider the different cases, and the implication of the various positioning of MBRs:

1. MBR A: lies outside R, and should not be considered in the constrained $NN(q)$ search.

2. MBR B and F: although there is an overlap with the constraining region R, no edge in these rectangles belongs entirely to R. Hence there is no guarantee that there exist any data points in the region of intersection with region R and both B and F should not be used to discard other MBRs from the NN search.

3. MBR C: all edges in C belong to R. In this case the pruning conditions can be used as in the case of searching the whole data space and (no modifications to $mindist(q, C)$ and $minmaxdist(q, C)$) are necessary.

4. MBR D: one edge of D is contained entirely in the queried region R. Following the definition of MBRs, we can make the assumption that there is at least one data point on the overlapping edge and the pruning rules can be used by considering $minmaxdist(q, D)$ only with respect to this edge.

5. MBR E: two of the edges of rectangle E are also contained in region R. Then there must be at least one data point on each of these edges, and the $minmaxdist(q, E)$ should be computed solely over the two overlapping edges.

Note that the guarantees brought by the measure $minmaxdist(q, M)$ do not hold if a minimum bounding rectangle is only partially in the constrained search space. In the case where the boundaries of the search region coincide with those of the data space, the constrained $NN(q)$ problem is reduced to the previously defined $NN(q)$ search. However, since we are interested in a more general case, we modify the method proposed by [RKV95] to support this extended class of NN queries. In the next section, we present a method for answering $NN(q)$ queries constrained to a convex polygon region, with minimal changes to the traditional unconditional NN approach.

It has been shown in [BBKK97] that, as opposed to the method for answering NNqueries in [RKV95], Hjaltason and Samet's solution [HS95] accesses only the pages necessary for the query answer. The algorithm proposed by [HS95] differs in two ways. First and most importantly, visiting order of the branches is based on the comparison of the nodes in a Partition List, which includes the children of the current branch as well as the last nodes visited in other branches. By comparison, the method of [RKV95] at each step only compares the position of the children, and consequently follows a branch entirely. It may be the case that, during the visit of one branch, some of the nodes can be pruned out because of the comparison with other branches. The visiting order proposed by [HS95] is therefore optimal [BBKK97], we and hence, apply it to the CNN Algorithm. Sorting the nodes in the Partition List is straight forward, based on the $mindist$ from the current MBR to the query point. Another difference between the two NN methods is that $minmaxdist$ is ignored in [HS95], which reduces its pruning power and increases the computation cost due to useless comparisons. The basic outline of an NN search algorithm follows a hybrid of the method proposed by [RKV95] and [HS95] and is depicted in Figure 6.

In order to extend the method to support the class of CNN queries, we need to modify the pruning conditions and use the constraint region to improve pruning power. For NN search, pruning is based on guarantees of the computed $mindist(q, M)$ and $minmaxdist(q, M)$. $Mindist(q, M)$ to a minimum bounding rectangle M is used to ensure that all data points included in the corresponding rectangle are as far or further from the query point. During pruning, it is used to exclude from the search an MBR M' that cannot be closer to the query point than at least one data point in M. $Minmaxdist(q, M)$ guarantees that at least one data point in M is at this distance or closer to the query point q. In a non-conditional NN algorithm, they represent tight bounds. The main problem, as discussed above, is that if an MBR is only partially intersecting the queried region, some of the

Single-Phase CNN Search Algorithm

1. initialize Partition List with children of the root
2. sort Partition List by *mindist*
3. while (Partition List not empty)
   (a) if node is leaf, compute distance *dist* from query point.
      i. if *dist* less than current distance to nearest neighbor, node becomes nearest neighbor
   (b) else, if non-leaf node, continue traversal and prune tree:
      i. add children of current node to Partition List
      ii. Prune Partition List by minmaxdist. Sort Partition List by *mindist*

Figure 6: Single-Phase Algorithm for Answering CNN Queries

guarantees do not always hold.

4.2 Modifications

In this section we extend the existing solutions to nearest neighbor queries in order to integrate CNN query processing over R-trees. We concentrate on the case of a convex polygon region, because we believe it represents a large variety of queries. Besides constraint regions that are directly defined as convex polygons, our model also characterizes regions that implicitly fall in the class of convex polygons. As described in Section 2, an application of the CNN method in GIS systems is answering queries of the type "find the nearest neighbor limited to the north-east area of a certain region". This is a special case of the convex polygon constraint, since the query conditions together with the boundary of the data space form a rectangle. Another case worth mentioning is the application of CNN methods to answering reverse nearest neighbor (RNN) queries [KM00]. One solution for RNN queries is the division of a 2-D data space into six equal regions by lines through the query point [SAA00]. In this case, the boundaries of the data space are restricted by two lines which again form a convex polygon. Furthermore, for a more general case, note that a non-convex polygon can also be obtained by subtracting a convex polygon from another convex polygon.

We showed that applying the pruning conditions of the non-constrained nearest neighbor approach can lead to incorrect results. The guarantees of the $mindist(q, M)$ and $minmaxdist(q, M)$ measures do not necessarily hold in the case of constrained nearest neighbor searches. In the following, we discuss the following modifications:

1. In order to correctly and efficiently prune the search space, we redefine containment of a minimum bounding rectangle into the constraining region.
2. For optimality, we also redefine the notion of $mindist(q, M)$ to assure minimum access of the

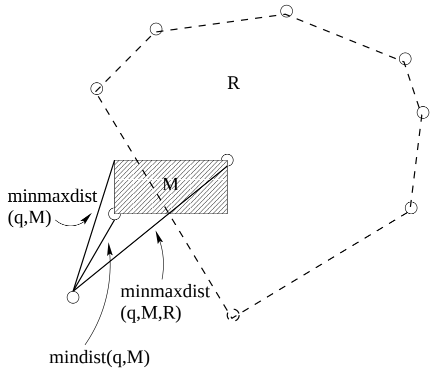


Figure 7: Example of $minmaxdist(q, M)$ and $minmaxdist(q, M, R)$

$minmaxdist(q, M, R)$	
minmaxdist(q,M,R)	$\begin{cases} = \text{minimum over all dimensions of the distance to furthest} \\ \text{vertex on closest face IN R} \\ = \infty \text{ if no face IN R} \end{cases}$

Figure 8: Modifications to minmaxdist

pages.

4.2.1 Containment

In Figure 7 we illustrate a case where $minmaxdist(q, M)$ is calculated over the entire minimum bounding rectangle M and leads to a false dismissal. Since $minmaxdist(q, M)$ happens to fall on a face that is outside of the constraining region R, there is actually no guarantee that there exists a data point contained both in M and in R. To extend the meaning of $minmaxdist(q, M)$ and retain its pruning properties, it should be computed only over the faces of M that are completely contained in region R (Figure 8). We refer to it as $minmaxdist(q, M, R)$. In general, if all faces of an MBR are either partially included or not included in R, then there is no guarantee that there exists a data point in the constrained area.

Since $minmaxdist(q, M, R)$ depends only on M's faces entirely included in region R, we need a simple way to test their containment:

1. construct infinite lines parallel with the x-axis, through the upper and lower corners of the MBR.

2. we calculate the x-coordinate of the intersection of these lines with the edges of polygon R.

3. considering the positioning of the x-coordinate of MBR's corners with respect to these intersection points, test if each of these corners is included in R.

4. an edge of the MBR is entirely contained in R iff its vertices are both contained in R.
5. finally, the value of $minmaxdist(q, M, R)$ is calculated over the MBR's edges that are entirely in R.

To explain these steps in more detail, assume that a polygon is described by its vertices as pairs according to the edges of the region. Note that for a convex polygon this is enough information to figure out the line equations and fully describes the region.

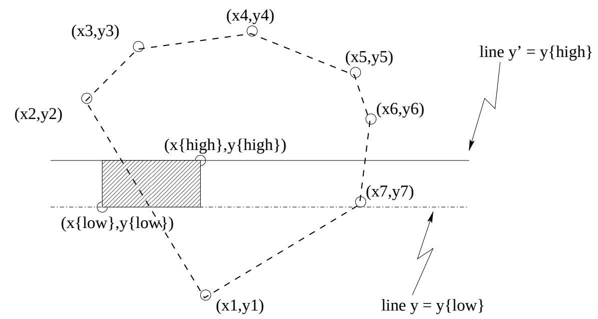


Figure 9: construction of lines y and y'

Note that, for an MBR defined as $[(x_{low}, y_{low}), (x_{high}, y_{high})]$, by constructing the lines $y = y_{low}$ and $y' = y_{high}$, (Figure 9), we obtain y and y', where each is parallel with the x-axes and intersects either two or none of the edges of the constraining polygon. Then, by construction, vertices $[x_{low}, y_{low}]$ $[x_{high}, y_{low}]$ are on line y and the upper corners of the MBR, $[x_{low}, y_{high}]$ $[x_{high}, y_{high}]$, are on line y'. If, for example, line y does not intersect any edge of the convex polygon, then both of the MBR vertices on y will be outside of region R. Otherwise the positioning of the MBR's vertices with respect to the intersection of the corresponding y or y' with the edges of R should be tested. If an intersection exists, the polygon edges that intersect y are the only two edges, say e_1 and e_2, that have one end below and one above the y-coordinate y_{low}. Let the segment e_1 be bounded by vertices $[x_{e_1i}, y_{e_1i}]$ and $[x_{e_1j}, y_{e_1j}]$, and the segment e_2 have vertices $[x_{e_2i}, y_{e_2i}]$ and $[x_{e_2j}, y_{e_2j}]$. Then e_1 and e_2 must have the property that $y_{e_1i} \geq y_{low}$, $y_{e_2i} \geq y_{low}$ and respectively $y_{e_1j} \leq y_{low}$, $y_{e_2j} \leq y_{low}$.

Let y intersect the edges e_1 and e_2 of R, and y' intersect edges e'_1 and e'_2 of polygon R. Also, let the points of intersection be $[x_1, y_{low}]$,$[x_2, y_{low}]$ for y and $[x'_1, y_{high}]$,$[x'_2, y_{high}]$ for y'. The x-coordinates of the points of intersection will therefore be :

1. $x_1 = y_{low} \times \frac{y_{e_1i} - y_{e_1j}}{x_{e_1i} - x_{e_1j}} - 1$, $x_2 = y_{low} \times \frac{y_{e_2i} - y_{e_2j}}{x_{e_2i} - x_{e_2j}} - 1$ for y
2. $x'_1 = y_{high} \times \frac{y_{e_1i} - y_{e_1j}}{x_{e_1i} - x_{e_1j}} - 1$, $x'_2 = y_{high} \times \frac{y_{e_2i} - y_{e_2j}}{x_{e_2i} - x_{e_2j}} - 1$ for y'

Knowing the intersection, the positioning of MBR's corners is checked for inclusion in R as follows:

1. $[x_{low}, y_{low}] \in R$ iff $x_1 \leq x_{low} \leq x_2$.
2. $[x_{high}, y_{low}] \in R$ iff $x_1 \leq x_{high} \leq x_2$.
3. $[x_{low}, y_{high}] \in R$ iff $x'_1 \leq x_{low} \leq x'_2$.
4. $[x_{high}, y_{high}] \in R$ iff $x'_1 \leq x_{high} \leq x'_2$.

4.2.2 Modifying $mindist(q, M)$

Modifying the notion of $minmaxdist(q, M)$ to $minmaxdist(q, M, R)$ leads to a simple and correct algorithm. This method can be optimized by also restricting *mindist* as necessary. We developed the technique to update the $minmaxdist(q, M)$ according to the position of the MBR M with respect to the constraint. Besides $minmaxdist(q, M)$, $mindist(q, M)$ is used as a minimum bound for the distance of a point in M to the query. This bound is used in the algorithm to eliminate the unnecessary MBRs, therefore also the pages in the subtrees rooted at these MBRs. In a typical database application, the data set does not fit into memory, hence the goal during the computation of constrained nearest neighbors is to prune the search space as early as possible. We now discuss how to modify the notion of *mindist* to improve the pruning of the search space.

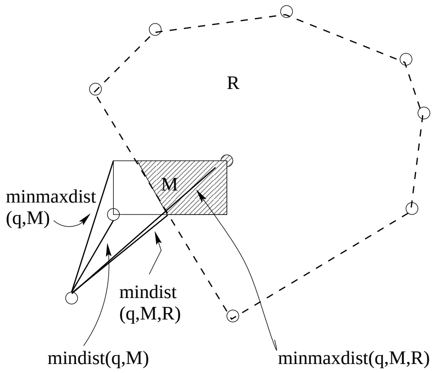


Figure 10: modifying both $mindist(q, M)$ and $minmaxdist(q, M)$

Let the intersection of an MBR with the polygon R be I_R. From the definition of $mindist(q, M)$, the distance between any data point in an MBR M must be greater than the *mindist* of the query point to M. Since the region I_R is a subset of the MBR M, there is a tighter bound $mindist(q, I_R)$, such that $mindist(q, I_R) \geq mindist(q, M)$, which offers the same guarantees as using the $mindist(q, M)$ value for non-conditional nearest neighbor search. During pruning, $mindist(q, I_R)$ can be used to exclude the corresponding MBR M from the search, if it is larger than the $minmaxdist(q, M', R)$ of any other MBR M'.

Since the intersection region, I_R can have a complex geometrical shape, we use the fact that finding the intersection of a rectangle with a polygon is a well-known problem in computer graphics. There are several techniques that efficiently identify the intersection polygon. One such an algorithm is Sutherland and Hodgman's polygon-clipping algorithm [FvFH96]. Once the edges of the intersection polygon are calculated, $mindist(q, M, R)$ is simply the minimum of all distances from the query point q to these edges.

If there is enough memory available in the system, or for small data sets, modifying only $minmaxdist(q, M)$ offers a very simple and efficient method for answering CNN queries. However, for large data sets modifying *mindist* is needed to reduce the I/O time.

5 Proof of Optimality of the Proposed Algorithm

In this section, we will prove that the proposed CNN technique is optimal with respect to the number of I/O accesses. Our development follows the methodology given in [BBKK97]. Let Q be a query point and CNN be the constrained nearest neighbor of Q. Then CNN-dist = $||Q - CNN||$ is the distance of the constrained nearest neighbor and the query point. The *CNN-sphere* of a query point Q is defined as the sphere with center Q and radius r=CNN-dist. In previous sections, we defined the intersection of CNN-sphere and the constraint R as the *CNN-region* and denoted it as I_R.

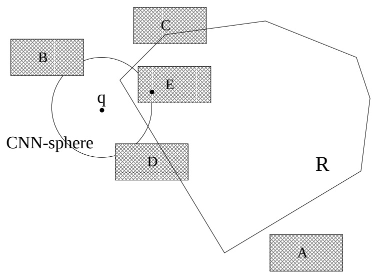


Figure 11: Different cases of pages with respect to R and CNN-sphere

Consider the different cases and the implications of the various positioning of the pages (Figure 11):

1. Page A intersects neither the constraint R nor the CNN-sphere.
2. Page B intersects the CNN-sphere but not R.
3. Page C intersects R but not CNN-sphere.
4. Page D intersects both R and CNN-sphere but not CNN-region I_R.
5. Page E intersects the CNN-region I_R.

We define the optimality of a CNN algorithm as follows.

Definition 1 Optimality
An algorithm for constrained nearest neighbor search is optimal iff it retrieves only the pages that intersect the CNN-region I_R.

From the definitions, an algorithm is optimal iff it retrieves only the pages in case 5.

Lemma 1 *The proposed single phase integrated algorithm is an optimal CNN algorithm.*

Proof

It is easy to see that any partition intersecting the CNN-region I_R is accessed during the search since they are not pruned in any phase of the algorithm. We need to prove the minimality of the accessed partition set. The algorithm prunes Page A and B by checking the intersection of the constraint with the bounding boxes that contain them. Page C is pruned by the mindist elimination (the same reasoning in [BBKK97] applies). We need to show that a page in case 4 (page D) is pruned by the algorithm.

Assume CNN-opt accesses a page D, i.e., a page that intersects both R and CNN-sphere but not I_R. During the algorithm, the *constrained mindist* ($mindist(q, D, R)$) is computed with respect to the CNN-Region I_R, i.e., intersection of the page D and the constrained R. The constrained mindist cannot be less than r, the radius of the CNN-sphere, i.e., $mindist(q, D, R) \geq r$. (Otherwise the intersection of the page and the constraint would have intersected the CNN-sphere and hence intersected the I_R, which is a contradiction).

Let CNP_0 be the partition (data page) that contains the constraint nearest neighbor, CNP_1 be the partition that contains CNP_0, ..., and CNP_k be the root partition that contains CNP_0, ..., CNP_{k-1}. Thus,

$$r \geq mindist(q, CNP_0, R) \geq \ldots \geq mindist(q, CNP_k, R).$$

Consequently,

$$mindist(q, D, R) > r \geq mindist(q, CNP_0, R) \geq \ldots \geq mindist(q, CNP_k, R).$$

Since CNP_k is in the root-page, CNP_k is replaced during the search process by CNP_{k-1} and so on, until CNP_0 is loaded. If as assumed, the algorithm accesses D, D has to be on top of the partition list at some point during the search. Since $mindist(q, D, R)$ is smaller than the *constrained mindist* of any partition containing the nearest neighbor, D can not be loaded until CNP_0 has been loaded. If CNP_0 is loaded, however, the algorithm prunes all partitions which have a *constrained mindist* smaller than r. Therefore, D is pruned and not accessed which is in contradiction to the assumption.

□

6 Performance Evaluation

In this section, we analyze the performance of the proposed CNN algorithms and compare it with the straight forward way of doing it. We have performed several experiments using real data sets.

The first data set, *Color Histogram*, is a 64-dimensional color image histogram data set of size 10,000. The vectors represent color histograms computed from a commercial CD-ROM. The second data set, *Satellite Image Texture (Landsat)*, is a 10,000 60-dimensional vectors representing texture feature vectors of Landsat images [MM96]. Texture information of blocks of large aerial photographs are computed using Gabor filters. The Gabor filter bank consists of 5 scales and 6 orientations of filters, therefore the total number of filters is $5 \times 6 = 30$. The mean and standard deviation of each filtered output are used to create the feature vector. Therefore the dimensionality of each vector becomes $30 \times 2 = 60$. This data set poses challenging problems in multi-dimensional indexing and is widely used for performance evaluation of index structures and similarity search algorithms [Man00, GIM99].

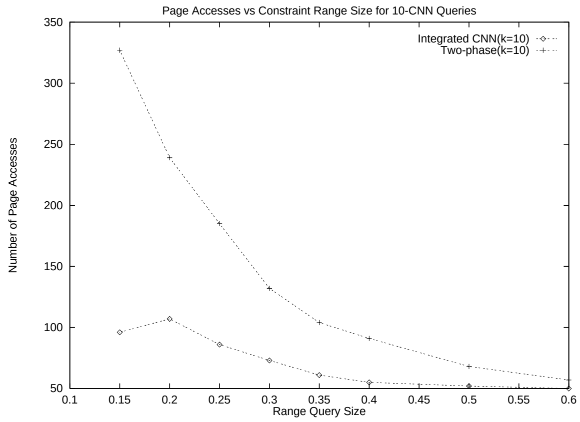
We built an R*-tree for both data sets. The page size is 8K, and the leaves are stored in disks. We implemented CNN algorithms (both integrated and two-phase algorithms) on these structures.

We compared the performance of the integrated single-phase technique with the two-phase technique (based on incremental NN and checking the constraint while finding the NN of the given point). We ran 10, 30, and 50 CNN queries and the cost metric is the number of page accesses during the CNN queries.

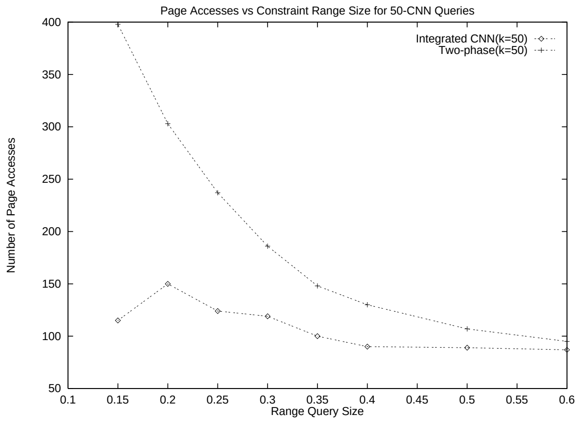
We picked the query points randomly from the data set. The constraint regions R was chosen again randomly with varying selectivities. The constraints are created as hyper-cubic range queries. We performed experiments on constraints from a spectrum of high-selectivity to low-selectivity. The average number of points covered by constraints with smaller edges is less than the average number of points covered by constraints with larger edges. Even the smallest constraints in the experiments had a reasonable selectivity (covers more than k number of points which is asked by the query). We varied the selectivity of the constraint and analyze the effects of the constraint range on the performance of the techniques.

Figure 12 shows the performance of the integrated technique and two-phase technique for Color Histogram data set. The x-axis shows the side length of the range constraint and the y-axis shows the number of page accesses as the result of the k-CNN queries. Figure 12(a) illustrates the results for 10-CNN and Figure 12(b) illustrates the results for 50-CNN queries. The results are very similar for other values of k, e.g. 30-CNN. The color histogram data set is very skewed and clustered. The dimensions are normalized within [0..1]. Even a small size range query returns a reasonable amount of data. The data set is clustered around the origin, therefore we picked our constraint regions by fixing one corner of the constraint to the origin and varied the region. As the constraint size increases, both techniques naturally merges to the same number of page accesses. For example, with a constraint which covers the whole data space, the CNN query is simply an NN query over the entire data space. As the constraint size decreases, and hence the selectivity of the range constraint increases, the speedup achieved by the integrated CNN approach increases. For example, for a 50-CNN query with an hyper-cubic constraint of side length 0.15 integrated approach returns 50 neighbors by accessing 115 pages, where the two-phase approach accesses 398 pages. The integrated approach is approximately 3.46 times faster than the two-phase approach. The speedup for 10-CNN query is 3.41.

Figure 13 illustrates the results of similar experiments for Landsat data set. Landsat data is also clustered, but data points are distributed more in the data space. They are not normalized as the color histogram data, therefore the range constraints were bigger than the Color Histogram data case to cover same number data points. Similar to the results in the previous data set, as constraints become very large both techniques access similar numbers of pages. For example, for an hyper-cubic range constraint of side length 4, the integrated approach accesses 169 pages while the two-phase approach accesses 227 pages for 10-CNN queries. For smaller constraint sizes (but still with reasonable selectivities), the integrated approach clearly outperforms the two-phase technique. For an hyper-cubic range constraint of side length 1.2, the integrated approach achieves a speedup of 3.9 over the two-phase approach. The speedup even becomes more as the number of neighbors is increased. For 50-CNN queries of the same constraint size, the integrated technique achieves 5.63



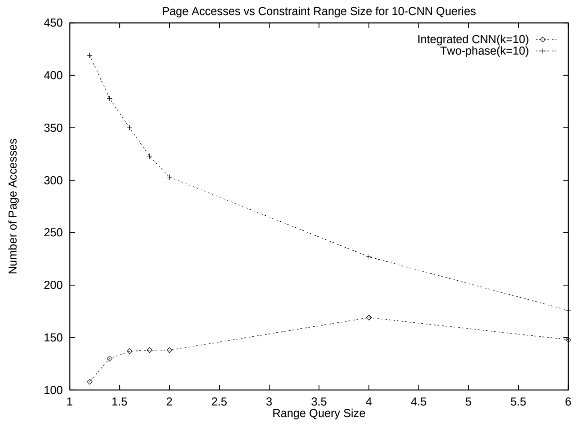
(a) 10-CNN



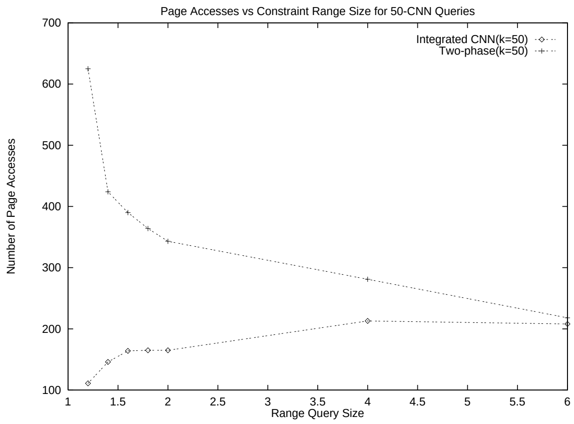
(b) 50-CNN

Figure 12: CNN queries on Color Histogram

speedup over the two-phase technique. The single phase integrated technique accesses 111 pages where the two-phase technique accesses 625 pages.



(a) 10-CNN



(b) 50-CNN

Figure 13: CNN queries on Landsat Texture Features

7 Conclusion

In this paper we introduced the notion of *constrained nearest neighbor queries (CNN)* and propose a series of methods to answer them. CNN queries are suitable for a wide range of applications. We presented various solutions, that either process the conditions of the NN search in sequential separate steps or interleave them into one phase. These approaches have inherent properties that lead to specific advantages for different constraints on the NN search. Since both range and nearest neighbor queries are independently well-studied and efficient index structures are developed for them, the proposed technique should build upon of the current state-of-art techniques that have been developed for these queries. We showed how to adapt the well-known NN query algorithms to support CNN queries without changing the underlying structure. We proved that the single-phase integrated approach is optimal with respect to the I/O cost. Experiments on real-life data sets show that the integrated approach is very effective for answering CNN queries.

References

[BBC+98] P. Bernstein, M. Brodie, S. Ceri, D. DeWitt, M. Franklin, H. Garcia-Molina, J. Gray, J. Held, J. Hellerstein, H. Jagadish, M. Lesk, D. Maier, J. Naughton, H. Pirahesh, M. Stonebraker, and J. Ullman. The Asilomar report on database research. *ACM Sigmod Record*, 27(4), December 1998.

[BBKK97] S. Berchtold, C. Bohm, D. Keim, and H. Kriegel. A cost model for nearest neighbor search in high-dimensional data space. In *Proc. ACM Symp. on Principles of Database Systems*, pages 78–86, Tuscon, Arizona, June 1997.

[BKK96] S. Berchtold, D. A. Keim, and H. P. Kriegel. The x-tree: An index structure for high-dimensional data. In *Proceedings of the 22nd International Conference on Very Large Databases (VLDB)*, pages 28–36, 1996.

[BKSS90] N. Beckmann, H. P. Kriegel, R. Schneider, and B. Seeger. The r*-tree: An efficient and robust access method for points and reactangles. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 322–331, May 1990.

[CDN+97] X. Cheng, R. Dolin, M. Neary, S. Prabhakar, K. Ravikanth, D. Wu, D. Agrawal, A. El Abbadi, M. Freeston, A. Singh, T. Smith, and J. Su. Scalable access within the context of digital libraries. In *IEEE Proceedings of the International Conference on Advances in Digital Libraries, ADL*, pages 70–81, Washington, D.C., 1997.

[EKK00] M. Ester, J. Kohlhammer, and H. P. Kriegel. The dc-tree: a fully dynamic index structure for data warehouses. In *Proceedings of the 16th International Conference on Data Engineering (ICDE)*, March 2000.

[FvFH96] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes. *Computer Graphics: Principles and Practice.* Addison Wesley, 1996.

[GG98] V. Gaede and O. Gunther. Multidimensional access methods. *ACM Computing Surveys*, 30:170–231, 1998.

[GIM99] A. Gionis, P. Indyk, and R. Motwani. Similarity searching in high dimensions via hashing. In *Proceedings of the Int. Conf. on Very Large Data Bases*, pages 518–529, Edinburgh, Scotland, UK, September 1999.

[Gut84] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 47–57, June 1984.

[HS95] G. R. Hjaltason and H. Samet. Ranking in spatial databases. In *Proc. of 4th Int. Symp. on Large Spatial Databases*, pages 83–95, Portland,ME, 1995.

[KF93] I. Kamel and C. Faloutsos. On packing r-trees. In *Proceedings of the 2nd International Conference on Information and Knowledge Management (CIKM)*, pages 490–499, 1993.

[KF94] I. Kamel and C. Faloutsos. Hilbert r-tree: An improved r-tree using fractals. In *Proceedings of the International Conference on Very Large Databases*, September 1994.

[KM00] F. Korn and S. Muthukrishnan. Influence sets based on reverse nearest neighbor queries. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, Dallas, USA, May 2000.

[LS90] D. B. Lomet and B. Salzberg. The hb-tree: A multi-attribute indexing method with good guaranteed performance. *ACM Transactions on Database Systems*, 15(4):625–658, December 1990.

[Man00] B. S. Manjunath. Airphoto dataset. http://vivaldi.ece.ucsb.edu/Manjunath/research.htm, May 2000.

[MM96] B. S. Manjunath and W. Y. Ma. Texture features for browsing and retrieval of image data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(8):837–42, August 1996.

[NHK84] J. Nievergelt, H. Hinterberger, and K.C.Sevcik. The grid file: an adaptable, symmetric multikey file structure. *ACM Transactions on Database Systems 9*, 1:38–71, March 1984.

[PF99] G. Proietti and C. Faloutsos. I/o complexity for range queries on region data stored using an r-tree. In *Proceedings of the International Conference on Data Engineering (ICDE)*, March 1999.

[RKV95] N. Roussopoulos, S. Kelly, and F. Vincent. Nearest neighbor queries. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 71–79, May 1995.

[Rob81] J. T. Robinson. The kdb-tree: A search structure for large multi-dimensional dynamic indexes. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 10–18, 1981.

[SAA00] I. Stanoi, D. Agrawal, and A. El Abbadi. Reverse nearest neighbor queries for dynamic databases. In *Proceedings of the ACM SIGMOD Workshop on Data Mining and Knowledge Discovery (DMKD)*, 2000.

[Sam89] H. Samet. *The Design and Analysis of Spatial Structures.* Addison Wesley Publishing Company, Inc., Massachusetts, 1989.

[SR87] T. Sellis and N. Roussopoulos. The r+-tree: A dynamic index for multidimensional objects. In *Proceedings of the 13th International Conference on Very Large Databases (VLDB)*, pages 507–518, May 1987.