An Intelligent & Incremental Approach to kNN using R-trees

DJ Oneil      Esten Rye

Abstract
In order for Geographical Information Systems (GIS) to query spatial data quickly, an efficient index structure that can use information about the query type is needed. Current GIS systems use R-tree variants to index geographical data. However, existing R-trees don’t take into account common classifications that objects belonging to a minimum spanning box may share. In this paper we describe a method to apply area classifications to R-tree implantations to decrease the search space for k-Nearest-Neighbor (kNN) queries.

Introduction
In recent years, spatial applications, especially geographic ones, have gained significant importance as data management problems. The widespread use of the Internet and easy access to high-speed connections has spurred on this growth even faster. Applications, such as Google Maps, have taken advantage of the increased bandwidth and advances in web programming technologies to move state-of-the-art GIS technology to the mainstream.

GIS is a special application of spatial database systems used to store and analyze geographical data. Because of the spatial properties of geographical data, spatial database queries can be expensive. A common database technique for speeding up queries is to use indexes. Traditionally, database systems have used the B-tree data structure as the staple indexing method. However, B-trees were designed with queries for simple numerical and string data in mind. GIS information, on the other hand, represents objects using points, lines, and regions. Though these objects can be represented in a relational model, they cannot be effectively transferred to a B-tree index.

In order to mitigate the inherent inadequacies of B-trees for spatial data, R-trees were developed. R-trees use Minimum Bounding Boxes (MBB) to represent regions. The original motivation of R-trees was to locate specific query objects or perform regional joins quickly given a query region. This technique is very good for performing such range queries because the tree can be traversed until the result objects are found. However, with kNN queries, there is no definite query region. Depending on the density of the geographical region, the number of objects requested, and the classification of the object requested, it may take many incremental iterations for the R-tree search to find the result set. The kNN problem using R-trees has inherent deficiencies. However, because R-trees are the main data structure that dictates the efficiency of existing market leader implementations and kNN queries are quite common, it is important that methods for increasing the speed of these queries are investigated.

In this paper we propose extending the R-tree data structure to handle arbitrary classification heuristics and modify the search algorithm for general kNN queries. We have decided to focus exclusively on R-trees because they are the current industry
standard for spatial indexes and they have been the topic of much research. The motivation for classification heuristics is that in many spatial problem domains there is well known domain specific classifications that apply to spatial regions, such as land-use regulation zoning in cities. This approach stems from the observation that R-trees were designed to be a general abstraction for all spatial database problems and do not take advantage of geographic specific properties; or the properties of any spatial system for that matter. Our approach for kNN queries is an incremental approach that uses simple Euclidean distance and concentric circles to incrementally investigate candidate regions. In order to reduce the tree space that is re-traversed, regions that are wholly contained in the previously searched circles are ignored.

Our key contributions can be summarized as follows:

- We extend the R-tree node data structure to handle arbitrary classifier heuristics.
- We propose incrementally considering MBBs that intersect with concentric circles as an incremental method for finding kNN.

**Problem Definition**

The kNN problem can be stated as follows: given a set of \( n \) points and a query point, \( q \), the k-Nearest-Neighbor (kNN) problem is concerned with finding the \( k \leq n \) points closest to the query point. Figure 1 shows an example of the kNN problem. On the left side is a set of \( n = 10 \) points in a two-dimensional space with a query point, \( q \). The right shows the problem solution, \( s \), with \( k = 3 \).

![Figure 1. An example of a nearest-neighbor problem domain and solution.](image)

**Related Work**

The nearest neighbors problem has been researched extensively due to its wide applicability to a number of fields, including GIS, artificial intelligence and pattern recognition [2], clustering problems [3], and outlier detection. [4] Traditional methods used Voronoi Diagram variants and Delaunay triangulation to decrease the search space. [5] Other methods have focused on extending nearest neighbors to higher dimensions [6] and what it means to find nearest neighbors in high dimensional spaces. [7] Due to the complexity of high dimensional spaces, there has been a number of approximation techniques proposed. These techniques include using randomized projections and
statistical methods such as sampling and histograms. Many of these methods propose entirely new data structures that address specific aspects of Nearest Neighbors (NN). Because R-trees are the standard in spatial databases, a number of methods have been proposed for finding kNN. However, to our knowledge, no method has been formally proposed that extends R-trees to handle arbitrary heuristics. Also, though many kNN algorithms use Euclidean distance as an approximation, we do not know of any that incrementally apply the Euclidean distance metric using concentric circles.

**Approach**

Our solution to the kNN problem revolves around extending the R-tree to take into account category data for searching out the k-Nearest-Neighbors of a query point. Before we describe our algorithm for our extended R-tree, we first must describe the algorithms and characteristics of the R-tree.

**R-tree**

The R-tree is a tree data structure that is similar to the B-tree. It is often used for spatial access methods and answering queries like “Find all the gas stations within five miles of my current location.” The R-tree partitions space into hierarchically nested, possibly overlapping, minimum bounding rectangles. Entries in non-leaf nodes store two pieces of data, a pointer to a child node and the minimum bounding box that surrounds all the data in that child node. The algorithms for insertion and deletion ensure that elements that are located near each other are placed into the same leaf-node by using the bounding boxes stored in the non-leaf nodes. Like the non-leaf nodes, leaf nodes also store two pieces of information, the location of the actual data item and the minimum bounding box that represents the item. Figure 2 shows an example R-tree.
R-trees have two algorithms that we will modify, insert and search. The insertion algorithm works by traversing the tree until it finds the leaf node M whose bounding box needs the least enlargement to accommodate the entry I(B, p), where B is the bounding Box of I and p is a pointer to the actual data item. If the leaf node is full, the algorithm splits the node M into two new nodes M1 and M2 to include all the old entries and I and propagates those changes up the tree. The insertion algorithm for the R-tree is displayed in Figure 3.
Procedure: Insert
Input: B /* bounding box. */
      P /* pointer to actual data item. */
Begin
  Set N = RootNode
  If N is a leaf:
    Return N
  Set S = ∞
  Set M = NULL
  While M is NOT a leaf
    For each F in N
      Set ENLARGE to amount of enlargement of F.B to accommodate B
      If ENLARGE < S
        Set M = F
      End For
    End While
  If M.Items is Full
    Split M into M1 and M2 to include all old entries and I
    Propagate the split upwards and grow the tree if root is forced to split
  Else
    Add I to M.Items
  End

Figure 3. R-tree Insertion Algorithm.

The search algorithm takes in a search rectangle S and an R-tree with Root T. It works by first searching the nodes of T to see if the search rectangle overlaps the rectangles of any of the nodes. It continues to do this recursively until it has traversed the tree to a leaf node. Once it has reached the leaf node it evaluates each item in the leaf node to see if the search rectangle overlaps the items bounding box B. If it does, it adds the item to the result set. After iterating through each item, it returns the result set. The search algorithm for the R-tree is shown in Figure 4.

Procedure: Search
Input: S /* search rectangle. */
      T /* root of R-tree. */
      RESULT /* globally visible (by function) sorted result set. */
Begin
  If T is NOT a leaf
    For each N in T
      If S overlaps N.B:
        Append Search(S, N) to RESULT
    Else
      For each E in T
        If S overlaps E.B
          Append E to RESULT
    End
  Return RESULT
End

Figure 4. R-tree Search Algorithm.
New kNN Algorithm

The major differences between the traditional approach presented above and our approach are that we add the arbitrary classifier heuristic to the insert and search and that we use a circle as the query region rather than the box. The following sections address these two additions.

Classifier Heuristic:
For our purposes, a classifier heuristic is a heuristic that uses a discrete set of $n$ possible values. An example of such a heuristic would be zoning. In the zoning example, the set of values might be {“Residential”, “Industrial”, “Commercial”}. For any spatial database, multiple heuristics can be defined.

Heuristics are added to the R-tree by augmenting the node structure with the extra heuristical data. To do this, each node would contain an associative array of all possible heuristics. For each heuristic, all classifier heuristic values that appear in the node’s subtree should be listed. The MBBs closer to the leaves of the R-tree should segregate heuristics more and more. The leaf nodes should contain only one value for each heuristic.

Concentric Circle Query Region:
Our method uses the Euclidean distance as an approximation for travel distance. We are assuming that the basic GIS information can be projected to a 2 or 3 dimensional space where Euclidean distance is geographically meaningful. In 2 or 3 space, Euclidean distance measures the hypotenuse of a triangle with legs that are at a right angle. For physical travel distances (driving, flights, etc…), Euclidean distance is a good approximation because it explicitly measures the straight line (shortest path) distance and implicitly estimates the distance traveled by city blocks. For kNN, the goal is to minimize the candidate set as much as possible. For this reason, circles with a radius that represents the Euclidean distance are more accurate than boxes.

In the algorithm presented below, an initial search circle is generated using some predefined constant. The constant could be a single global constant, or it could depend on the regional characteristics (e.g. smaller distance for population dense areas). When the R-tree is traversed, the algorithm checks whether the current MBB intersects the query circle. If any part of it does, then the subtree of the MBB is checked. If all MBBs have been checked and there are still not $k$ objects found, a function is applied to the current circle to incrementally increase it. The algorithm is restarted, except that it keeps track of the previous query circle. Now when the algorithm checks an MBB, if the MBB intersects the current query circle and it is not completely contained in the previous query circle, then its subtree is searched. By eliminating boxes completely contained in the query circle, the algorithm can significantly reduce the nodes that it has already searched.

kNN Algorithm for R-tree:
For the extended R-tree data structure, the insert function needs to be altered slightly. First, the inserted query region or point needs to be classified. If it is not classified, then the algorithm should assume that the user has some special knowledge about the regions in the R-tree such that the insertion can take on the classifications of the parent MBB. As the insertion object traverses the tree, the classifications at the nodes that are expanded
should be updated to include any new classifications the object may add. Because of the
constraint that leaf regions have only one classification for each heuristic, the leaf regions
may have to be finer grained.

Similarly, for updates, the tree must be traversed from the bottom up and the nodes
updated appropriately. Of course, if the parent node of the current subtree already
contains the new classification, then the update is over.

The following is the extended search algorithm for kNN on an R-tree. The new
search algorithm takes as parameters the QUERY_POINT, which will search as the
center of the search circles. The CLASSIFIERS is an array of classifiers that represent
the query. The search array does not have to include values for all classifier heuristics. If
a particular heuristic is missing, it is simply skipped at each node. The CIRCLE
parameter represents the current query circle. The PREV_CIRCLE represents the
previous query circle. For the initiating call of the function, this should default to zero so
that the first search doesn't exclude any data in a smaller circle from the search space.
The RESULT is a functionally global array that holds the sorted array of unique matches
found by the algorithm. At the end of the search, the first $k$ values of the array become
the actual search result.

---

**Figure 5.** k-Nearest-Neighbor search algorithm for R-tree

Figure 6 depicts a simple example of the new k-Nearest-Neighbors search. The
space is made up of objects that are either of the “circle” or “square” type. A user
queries for the $k=3$ nearest neighbors to the query point, $q$, of type “square”. In this
example, it is easy to see that, though regions $R_3$ and $R_6$ contain nearest neighbors,
they do not contain neighbors of the requested query types. Because of the
constraint that leaf nodes (R3, R4, R5, R6, R7, R8) must contain only objects that are of on type, we can see that boxes R3 and R6 will be skipped.

![Figure 6](image.png)

**Figure 6.** A kNN problem and solution with classified data and query.

**Validation**

We validate our algorithm with two directions. First we show the relevance of our approach by examining the example heuristic of zoning being used to reduce the query space of the example query “What are the k-nearest gas stations?” Second, we perform an analysis of the theoretical algorithmic complexity of our algorithm.

One possible application of our algorithm is to use zoning as a heuristic to reduce the amount of possible results. For example, if you were searching for the k-nearest gas stations it would be illogical to consider data points that are in regions zoned for parks, recreation and preserves. Likewise, it would also be illogical to search data points zoned as Agriculture and Undeveloped when searching for the k-nearest shopping malls. The following table shows the 2000 and 2005 Land Use and Zoning Statistics for the city of Minneapolis in acres. If we are searching for the k-nearest gas stations, we first apply the zoning heuristic to the query area. Gas stations are typically built on land zoned for commercial use, so this reduces the area of eligible data points to 31,940 acres or roughly 1.7% of the total searchable area of Minneapolis. After eliminating the illogical boxes, we search for data points in the remaining boxes.
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Absolute (in acres)</td>
<td>Relative (percentage)</td>
<td></td>
</tr>
<tr>
<td>Residential Total</td>
<td>368,495</td>
<td>395,630</td>
<td>27,135</td>
</tr>
<tr>
<td>Single Family Residential</td>
<td>313,944</td>
<td>337,886</td>
<td>23,942</td>
</tr>
<tr>
<td>Farmsteads</td>
<td>19,759</td>
<td>17,159</td>
<td>-2,600</td>
</tr>
<tr>
<td>Multi-family Residential</td>
<td>34,791</td>
<td>40,584</td>
<td>5,793</td>
</tr>
<tr>
<td>Mixed Use</td>
<td>3,429</td>
<td>4,848</td>
<td>1,419</td>
</tr>
<tr>
<td>Commercial</td>
<td>31,940</td>
<td>35,309</td>
<td>3,369</td>
</tr>
<tr>
<td>Industrial Total</td>
<td>46,496</td>
<td>48,580</td>
<td>2,084</td>
</tr>
<tr>
<td>Industrial &amp; Utility</td>
<td>37,295</td>
<td>38,922</td>
<td>1,627</td>
</tr>
<tr>
<td>Extractive</td>
<td>6,439</td>
<td>6,962</td>
<td>523</td>
</tr>
<tr>
<td>Railway</td>
<td>2,762</td>
<td>2,697</td>
<td>-65</td>
</tr>
<tr>
<td>Institutional</td>
<td>32,548</td>
<td>34,165</td>
<td>1,617</td>
</tr>
<tr>
<td>Parks, Recreation &amp; Preserves</td>
<td>163,286</td>
<td>181,691</td>
<td>18,405</td>
</tr>
<tr>
<td>Major Vehicular Rights-of-Way</td>
<td>25,458</td>
<td>28,349</td>
<td>2,891</td>
</tr>
<tr>
<td>Airports</td>
<td>6,766</td>
<td>5,967</td>
<td>-799</td>
</tr>
<tr>
<td>Agriculture &amp; Undeveloped Total</td>
<td>1,101,752</td>
<td>1,045,347</td>
<td>-56,406</td>
</tr>
<tr>
<td>Agriculture</td>
<td>607,837</td>
<td>585,840</td>
<td>-21,996</td>
</tr>
<tr>
<td>Undeveloped Land</td>
<td>494,611</td>
<td>459,507</td>
<td>-35,104</td>
</tr>
<tr>
<td>Agricultural &amp; Vacant</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Industrial Parks not Developed</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Public &amp; Semi-Public Vacant</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Open Water</td>
<td>123,971</td>
<td>124,128</td>
<td>157</td>
</tr>
<tr>
<td>Total</td>
<td>1,904,141</td>
<td>1,904,014</td>
<td>-128</td>
</tr>
</tbody>
</table>

Notes:
- The bold rows in the table show major categories of land use.
- Percentages and acres are rounded to nearest whole number.

Figure 7. Land Use Statistics for Minneapolis, MN. [15]

The algorithmic complexity of our algorithm is in linear time. Our heuristic approach adds a constant amount of work h to the k-nearest neighbor search to the evaluation of every leaf node, non-leaf node and data item in the R-Tree, where h is the number of items in the classifiers array. More formally:

\[ nh + h \lg \frac{n}{\alpha} + \alpha h \]

Figure 8. Algorithmic Complexity Equation

Where n is the number of data items, \( \alpha \) is the number of elements per node, and h is the number of heuristics being used.

Conclusions and Future Work

In this paper we have presented a new method for expanding R-trees to better handle kNN queries. Our method uses user-defined heuristics to eliminate nodes that don’t contain objects of the query object classifiers. We argue that Euclidean distance is a
good approximation for GIS travel distance and incrementally use concentric circles to only consider Euclidean distance and reduce retraversing the same subtrees.

In the validation section we cited examples that indicate that there are many real applications that would have a significant boost in performance if heuristics were correctly applied. We derived the complexity of the algorithm. Also, intuitively, our method would meet or exceed speeds of existing R-tree implementations as long as the heuristics are thoughtfully implemented.

For future work, we would like to address some of the known limitations of our approach. These limitations can be summarized as follows:

- Our Euclidean distance assumption does not guarantee the exact solution will be found for kNN. It would be interesting to investigate metrics that use spatial information to make better estimates about length of travel routes. The Euclidean distance also doesn’t take into account practical factors such as the speed limit of the road traveled and construction.

- Intuitively, our approach is faster than current implementations. However, the validation section would be admittedly stronger if we validated our method against real R-tree variants on real data.

- Our method assumes that all spatial data can be perfectly classified immediately (when it is inserted). However, this may be an unreasonable assumption. Our method would be more useful if it could incrementally add in the classifier heuristics.

- It’s intuitive that at some point there are too many heuristics for queries to execute efficiently and quickly. In future work, it would be interesting to determine the threshold of heuristics and heuristics classifications for good performance.

- The algorithm currently demands that leaf nodes have exactly one classification for each heuristic. This might be too strict. If there are many heuristics that are not highly correlated, the leave regions may become too small and the complexity of searching the tree may increase significantly.

References


