Approximate k-Nearest Neighbor Search with the Area Code Tree

Wendy Osborn* and Fatema Rahman*
University of Lethbridge, Lethbridge, Alberta T1K 3M4, CANADA

Abstract

In this paper, we propose and evaluate a strategy for approximate *k*-nearest neighbor searching using the Area Code tree. The Area Code tree is a trie-type structure that manages area code representations of each point of interest (POI) in a data set. It provides a fast method for locating an approximate nearest neighbor for a query point. We first summarize the area code generation, insertion (used in overall construction) and searching approaches. Then, we evaluate the construction via repeated insertion, *k*-nearest neighbor searching, and accuracy of the Area Code tree, with some of the evaluation involving the comparison versus a basic benchmark brute force approach. We find that when the Area Code tree is used for locating approximate nearest neighbors, that low constant-time search is achieved. Also, in denser POI sets, higher accuracy is achieved for locating one-nearest neighbor. This ultimately makes the Area Code tree a strong candidate for approximate continuous nearest neighbor processing for location-based services.

Key Words: Nearest neighbor queries, spatial access methods, location-based services.

1 Introduction

A location-based service provides results to a user of a mobile device (e.g. smartphone, tablet) based on their location, interests and the type of query being performed [11]. One example of such a query is a *k*-nearest neighbor query [10, 12], which returns the nearest *k* points of interest (POI) to them. For example, a user may want to know the location of some of the nearest restaurants to them. The user may want to know the exactly closest restaurants to them. However, the user may also be happy with suggestions that, although not guaranteed to be the closest, may be close enough to satisfy them. This is an example of an approximate *k*-nearest neighbor, where a trade-off is being made between accuracy and efficiency.

Efficient nearest neighbor processing, exact or approximate, is important, but is especially important when it is initiated from a mobile device [11]. Many strategies have been proposed for nearest *k*-neighbor processing for location-based services. Several utilize spatial access methods [1], including [3, 4, 5, 6, 7, 13, 14]. Although all of these strategies return exact nearest neighbors, limitations of these approaches include repeated searching, the need to cache a significant amount of data on the mobile device, the requirement to know the query trajectory in advance, and maintaining a sparse index which leads to inefficient searches.

Repeated searching is not desirable, but may be the only option when storage on a mobile device is limited. A recently proposed data structure, the Area Code tree [8] manages POIs in a trie-type structure using an area code representation for each POI. Although it can be used to locate POIs efficiently, it cannot be used for exact nearest neighbor matching. However, given a preliminary evaluation on its efficiency, it is an excellent candidate for approximate *k*-nearest neighbor search for situations where a guaranteed exact answer is not required, for example, in the restaurant scenario given above.

Therefore, in this paper, we extend this approximate nearest neighbor strategy and propose one for locating *k*-nearest neighbors. We also evaluate the Area Code tree for *k*-nearest neighbor search accuracy, tree construction time, and also comparatively evaluate its search time against another strategy. We find that approximate *k*-nearest neighbor searching can be accomplished in very low and constant time, regardless of the number of POIs being indexed. With respect to accuracy, up to 60% accuracy is achieved when the Area Code tree is used for locating one-nearest neighbor in a dense POI set. This makes the Area Code tree a significant candidate for continuous approximate nearest neighbor search for location-based services. The remainder of the paper proceeds as follows. Section 2 summarizes related work in the area of continuous nearest neighbor processing for mobile devices. Section 3 summarizes the area code mapping, insertion and 1-nearest neighbor search algorithm for the Area Code tree. Given this, we propose a *k*-nearest neighbor search strategy for the Area Code tree. Section 4 presents the methodology and results of our performance evaluation. Finally, Section 5 concludes the paper and provides some directions of future research.

2 Related Work

In this section, we summarize related work in nearest neighbor searching for location-based services. Although nearest neighbor strategies have been proposed in other

*Email: {wendy.osborn,f.rahman}@uleth.ca. Article is an extension of [9], presented at SEDE 2016.

contexts, they are considered outside of the scope of this work. Many strategies have been proposed in the literature [3, 4, 5, 6, 7, 13, 14].

A continuous nearest neighbor strategy proposed by Song and Roussopoulos [13] obtains a superset *m* of nearest neighbors, which attempts to keep the result current while the query point moves around, and a new query call to the server is not necessary. Their strategy utilizes existing stationery nearest neighbor strategies. A limitation of their strategy is in choosing the value of *m* so that fewer query calls are needed but not too much data needs to be stored on the mobile device.

Tao *et al.* [14] utilizes the R-tree [2] to speed up the repeated searching needed for their continuous query strategy. Lee *et al.* [5] improves upon this strategy by fetching both the required and some additional objects in order to reduce the number of repeated searches that are needed. Park *et al.* [7] also improves upon this strategy by locating all nearest neighbors along a trajectory by using the R-tree. A limitation exists in that the trajectory needs to be known in advance.

Hu *et al.* [3] proposes a proactive caching strategy, which caches previous results and the R-tree nodes required to obtain them on the mobile device. The cache is always searched first for a new query, with additional results fetched from the server. Limitations include the significant overhead of caching and local processing on the mobile device.

Jung *et al.* [4] utilize a grid index for continuous nearest neighbor searching. The grid index allows for quick elimination of regions of space from consideration if they do not overlap the query point. A limitation with this strategy is that the grid index can be sparse due to wasted space.

In summary, some general limitations of these approaches include caching a significant amount of data on the mobile device, repeated searching, using a sparse index which leads to inefficient searches, and requiring knowledge of the query path in advance. Repeated searching, however, may be the only option if storage is limited on a mobile device. The Area Code tree [8] attempts to provide the ability to perform repeated searching that is efficient, but at the cost of accuracy. It is also more compact that existing spatial access methods, given that only POIs are stored, and not many co-ordinates for many bounding rectangles. The Area Code tree is summarized in the next section.

3 Area Code Tree

The Area Code Tree [8] is a trie-type structure for approximate nearest neighbor searching. It stores points of interest (POIs) that are represented in an area code format. In this section, we summarize the mapping, construction and nearest neighbor search for the Area Code tree.

3.1 Mapping

An area code is a sequence of digits that indicate the relative location of a POI in space. It is obtained by recursively partitioning the space containing points into quadrants. For a particular POI, the space is partitioned until the POI is equal to the middle of a quadrant. At each level of partitioning, the quadrants are numbered as follows: SW (1) SE (2) NW (3) and NE (4). Beginning with the top-most partition, a POI obtains a digit at each level, depending on which quadrant it resides in. Figure 1 depicts an example of space partitioning and mapping. The POI maps to the area code 4132, since the POI resides in the top-most NE quadrant, followed at the next level by the SW quadrant, then the NW quadrant, and finally the SE quadrant.

3.2 Tree Construction

Once the area code for a POI is determined, is it inserted into the Area Code Tree, beginning with the most significant area code digit. To construct a tree, each area code is inserted one at

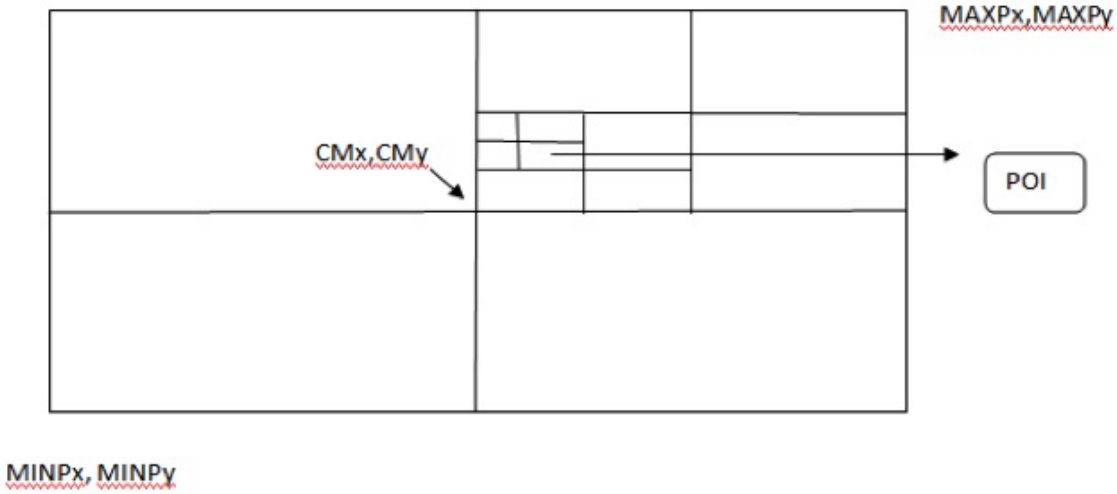


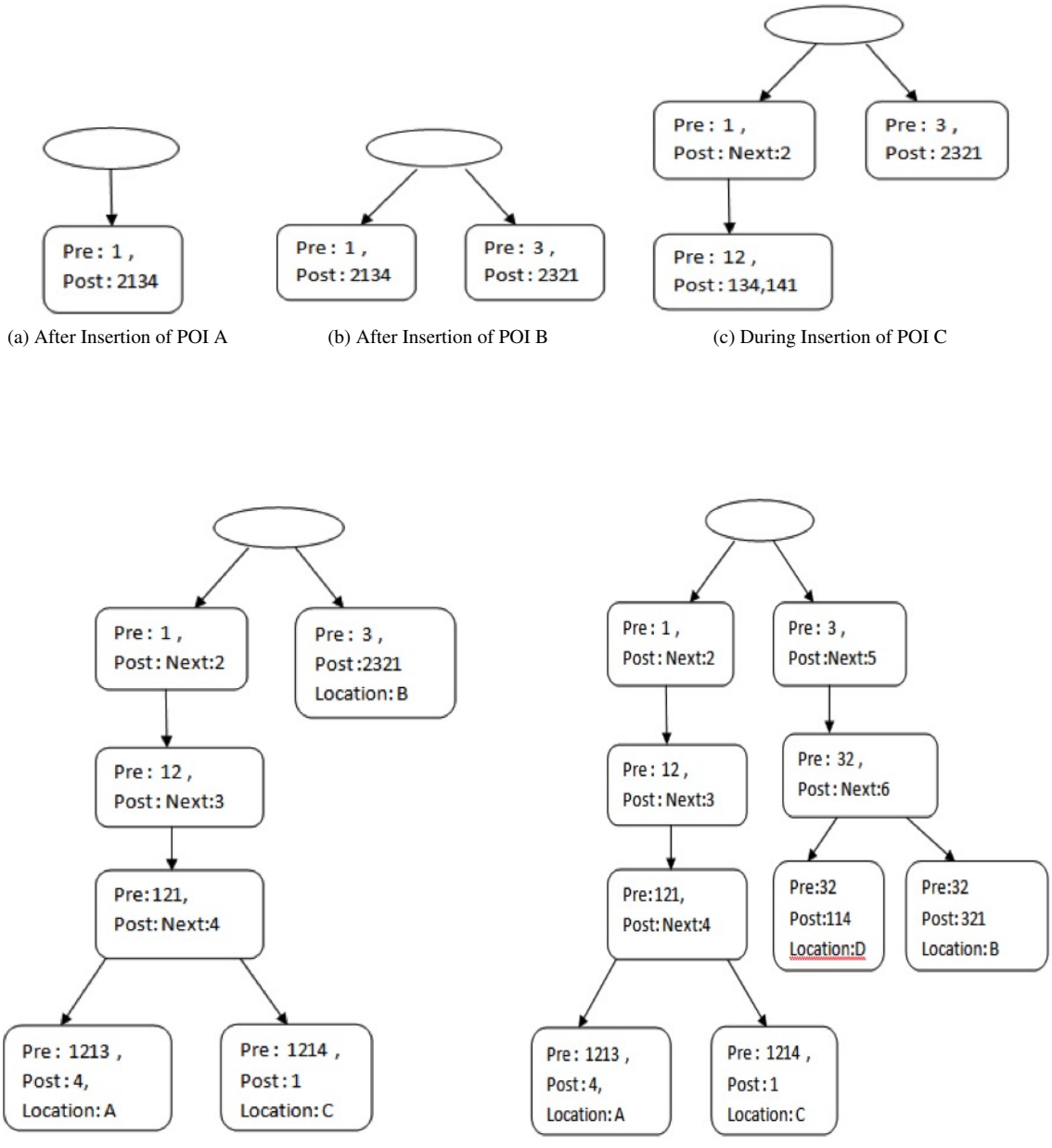
Figure 1: Area code mapping (from [8])

a time, using the existing strategy for any trie structure. Figures 2 and 3 depict the construction of a small Area Code Tree, containing the POIs A,B,C,D and E, and each with area codes 12134, 32321, 12141, 32114, and 21324 respectively. POI A is inserted first. When POI B is inserted, a new node is created for it, since the existing node contains area codes that start with 1, and the area code for B starts with 3. When POI C is inserted, additional nodes are created for the common prefix strings (12, 121) contained by both A and C. A similar case occurs when inserting POI D, with common prefix string 32. Finally, when POI E is inserted, a new path is started since its leading digit, 2, is not contained by any other existing path.

3.3 K-Nearest Neighbor Searching

Given that every POI is mapped to a string of digits, this provides a method for quickly identifying approximate k-nearest neighbors to the query point. We first describe the 1-nearest neighbor case, and then describe the extensions to k nearest neighbors.

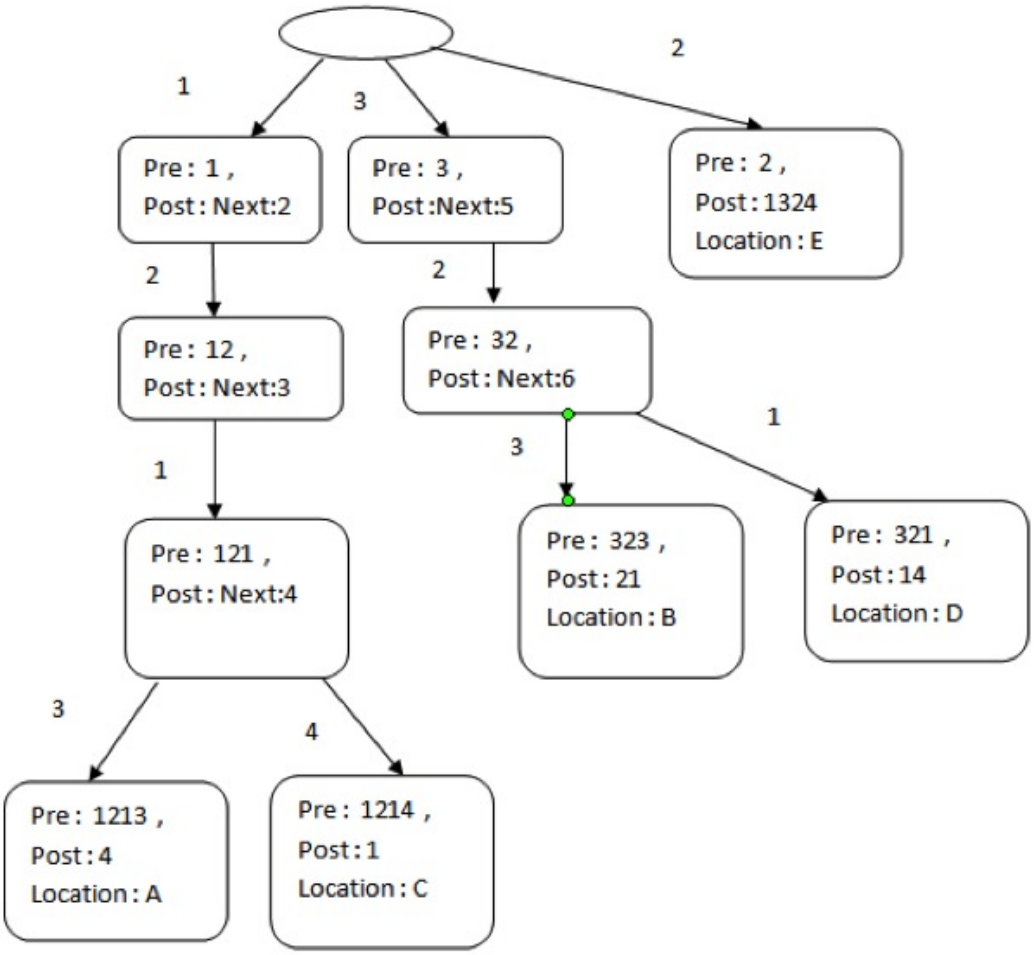
Any search will begin by first mapping the query point to an area code using the same strategy that is mentioned above for



(a) After Insertion of POI A (b) After Insertion of POI B (c) During Insertion of POI C

(d) After Insertion of POI C (e) After Insertion of POI D

Figure 2: Index construction - part 1



(a) After Insertion of POI E

Figure 3: Index construction - part 2

mapping POIs. Then, beginning with the most significant digit of the query point area code, a path is followed down the Area Code tree while digits in the query match digits in the tree nodes. If a match does not exist at a particular level, the closest match is taken. For example, suppose we have the query area code of 12144. Referring back to Figure 3, the closest match is the POI C with area code 12141, since a path exists that matches the first 4 digits of the query area code, with the closest node to the last digit in the query area code containing the value of 1.

Using this basis, the 1-nearest neighbor search can be extended to locate k nearest neighbors. Figure 4 depicts a sketch of our k-nearest neighbor strategy for the Area Code tree. It consists of three stages, which we describe below:

1. Finding nearest neighbors. This stage uses a slight modification of the 1-nearest neighbor approach. A search is initiated for the area code that is the closest to the query. However, if k nearest neighbors are sought, then area codes for other POIs that exist along the immediate search path are also obtained and added to the result set, until k is reached, or until the leaf node for the original search is reached.
2. Finding nearby neighbors. It is possible that the number of POIs obtained in the first step is not enough to satisfy the query. Therefore, using the area code for each POI in the initial result set, a search for other POIs is performed, with any added to the updated result set. This continues until either k is reached, or all POIs in the initial result set have been exhausted.
3. Finding further neighbors. This step is similar to the second step, except that the POIs obtained in the second step are the basis for locating additional nearby POIs.

Referring back to our original example, suppose we have the query area code of 12144. In Figure 3, the closest match is the POI with area code C (12141). However, in the first step of our k-nearest neighbor strategy, the POI A is also retrieved for our result. If k=2, then the search is completed. Otherwise, the result set containing (C,A) is processed using the second step of our strategy to obtain any more required points from the

```
knn = n; //provided by User
query = xxxxx; //converted from (x,y)
              //provided by User
result = list();

tree_ptr = AC_root;

//first, find "matching" area code for query,
//and all others along same path
result
   = find_nearest_neighbors(tree_ptr,
      query,knn);

//if we need more POIs to make inn
if(size(result) < knn)
     result
        = find_nearby_neighbors(tree_ptr,
           query, knn, result)

//if we still need more POIs,
//we look even further from query
if(size(result) < knn)
     result
        = find_further_neighbors(tree_ptr,
           query, knn, result)

return result;
```

Figure 4: *k*-nearest neighbor searching sketch

remaining part of the tree.

4 Evaluation

In this section, we present the methodology and results of our performance evaluation of the Area Code tree. We compare its search performance with the Brute Force method, which consists of searching the entire set of POIs to find the *k*-nearest neighbors. We chose this comparison for our preliminary evaluation due to the Brute Force method being a basic benchmark for processing nearest neighbor queries. In addition, we evaluate the construction time and the accuracy of our proposed structure. Construction is performed by inserting one POI area code at a time, while accuracy is measured by determining the percentage of times overall that accurate *k*-nearest neighbors are found when the Area Code tree is used to locate them.

4.1 Methodology

For our evaluation, we use twenty-one synthetically generated data sets that represent collections of different POIs across New Zealand. Ten data sets consist of POIs drawn from across the North Island of New Zealand, with each set containing 1000, 2000, 3000,..., up to 10,000 POIs respectively. An additional ten data sets contain POIs from across the Waikato Region of New Zealand (part of the North Island), again with each file containing 1000, 2000, 3000,..., up to 10,000 POIs respectively. The Waikato data sets are denser, which allows us to evaluate the Area Code Tree in denser POI data. The remaining data set contains 10 User locations along their trajectory, which will serve as the *k*-nearest neighbor queries for our evaluation.

First, for each POI set mentioned above, an Area Code Tree is created, to give a total of 20 area code trees. Then, using these trees and the User location set, the following tests are performed:

- For all 20 area code trees, we perform a 1-nearest neighbor search for each of the 10 points in the User location set.
- For the North Island 10,000 and Waikato 10,000 POI sets only, we also perform a *k*-nearest neighbor search for each point in the User location set. We perform the search for *k*=2,3,4,...10.

Every search is also performed on the same data sets using the Brute Force method. Therefore, a total of 400 *k*-nearest neighbor comparisons are performed.

The performance criteria that are measured are as follows:

- For each tree construction, the overall construction time (in seconds), where the construction time includes the time required to calculate the area codes and to insert each area code into the tree.
- For every *k*-nearest neighbor search, the average search time (in milliseconds).
- The accuracy of the Area Code tree, which is measured by recording for each search, the percentage of POIs found by Area Code Tree that matched those found by the Brute Force search.

4.2 Results

We first present the results of the search and accuracy experiments, followed by the tree construction results. First, Figures 5 and 6 depict the results of the 1-nearest neighbor comparison using all of the the Waikato and North Island POI sets, respectively. For both figures, the x-axis contains values that represent 1000s of POIs (i.e. 1 is 1000 POIs, up to 10 for 10,000 POIs), while the y-axis contains the average search time in seconds.

We find that for both the Waikato and North Island POI sets, searching using the Area Code Tree achieves significantly better search times over the Brute Force method. The average search time for the Area Code tree is less than 0.005 seconds (i.e. 5ms), and is regardless of both the density of the dataset and the number of POI area codes in the area code tree. For the Brute Force method, the average search time increases linearly,

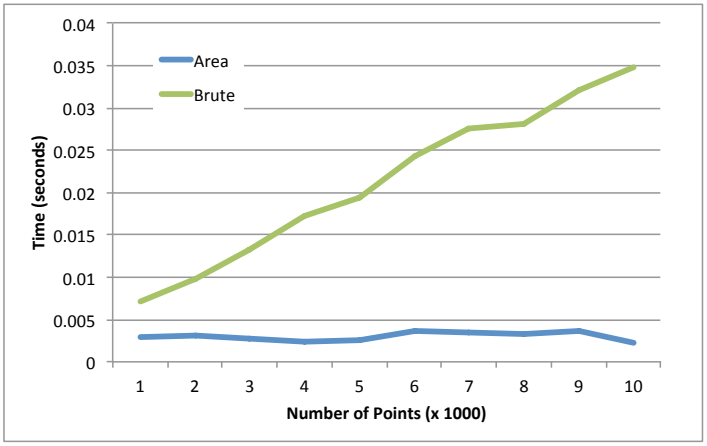


Figure 5: 1-nearest neighbor - Waikato POIs

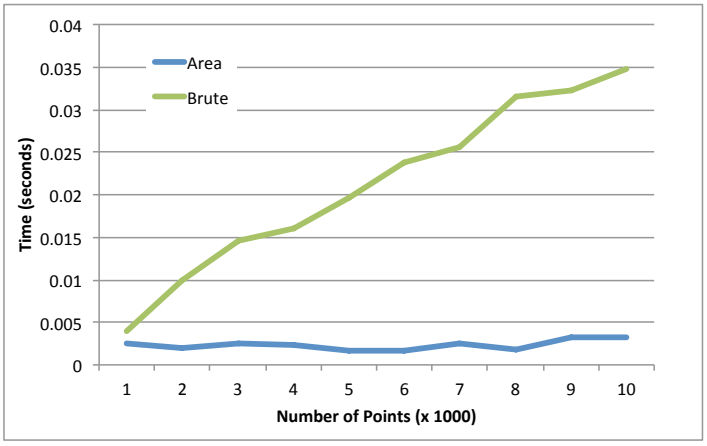


Figure 6: 1-nearest neighbor - North Island POIs

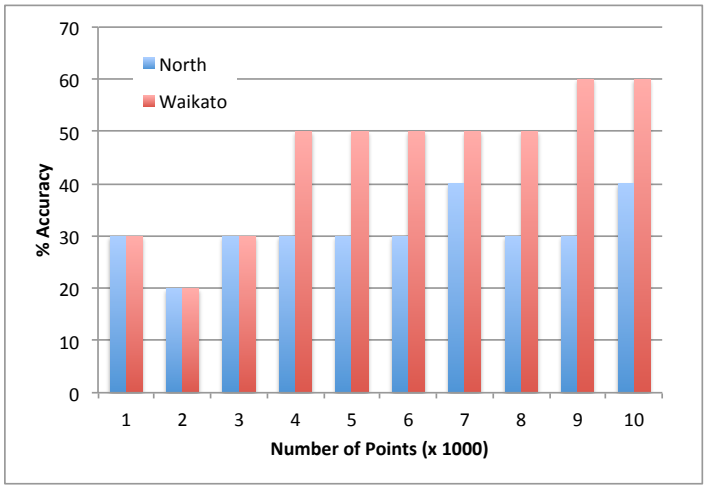


Figure 7: 1-nearest neighbor - accuracy

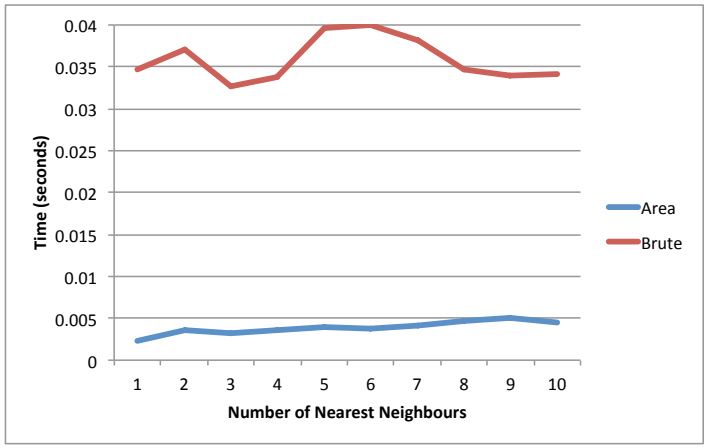


Figure 8: k-nearest neighbor - Waikato POIs

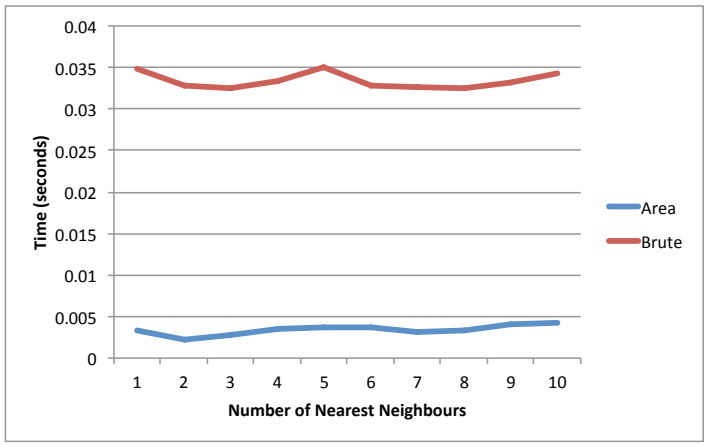


Figure 9: k-nearest neighbor - North Island POIs

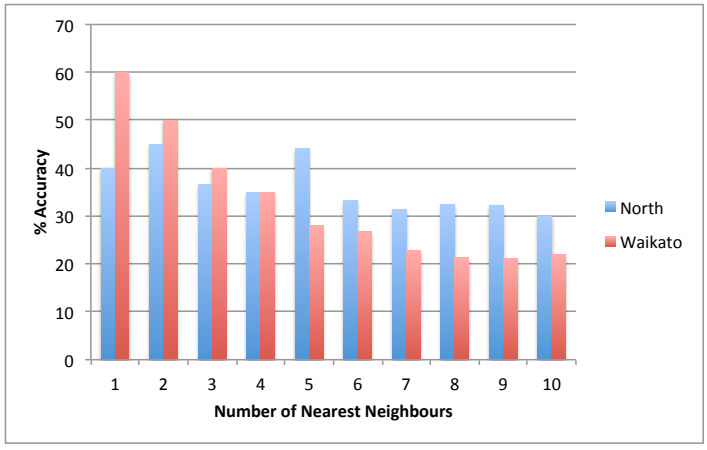


Figure 10: k-nearest neighbor - accuracy

between approximately 0.007 seconds (i.e. 7ms) up to almost 0.035 seconds (i.e. 35ms) for searching in 10,000 area codes.

Figures 8 and 9 depict the results of the *k*-nearest neighbor comparison for the 10,000 Waikato and 10,000 North Island POI sets, respectively. For both figures, that x-axis contains the *k* values used, while the y-axis contains the average search time in seconds. We again find in both cases a significant improvement in running time when the Area Code tree is used for *k*-nearest neighbor searching. We also find that for both Area Code tree searching and the Brute Force approach, similar results are achieved, regardless of the number of nearest neighbors being sought or the density of the POI set. The average search time using the Area Code tree is under 0.005 seconds (i.e. 5ms), while for the Brute Force approach it is between 0.03 and 0.04 seconds (i.e. 30 and 40ms).

Figures 7 and 10 depict the accuracy of the Area Code tree in locating 1-nearest neighbor and *k*-nearest neighbors, respectively. Here, we can see that the North Island and Waikato POI sets differ significantly in their performance.

For 1-nearest neighbor search, we find that for the dense POI sets (i.e Waikato), the Area Code tree can achieve between 40% and 60% accuracy . For the less dense POI sets (i.e. North Island) however, the accuracy was lower, around 30% in most cases. Therefore, for the 1-nearest neighbor case we conclude that the Area Code Tree provides fast nearest neighbor searching, that is expected to improve in accuracy as the data set size increases in density.

However, we find the opposite scenario when the search is increased to *k*-nearest neighbors. For the less dense POI sets, although some modest improvements are found in accuracy, for most cases, and in particular for higher values of *k*, we find that just over 30% accuracy is still being achieved. For the denser POI sets, we observe a steady decline in the accuracy of the search results, from 60% for 1-nearest neighbor to just over 20% accuracy for 10-nearest neighbors. In addition, it should be noted that, although not presented here, the only situations where 100% accuracy is achieved are for the 1- and 2-nearest neighbor searches. Therefore, we conclude that in less dense POI sets, searching in the Area Code tree seems to achieve consistent accuracy, regardless of the *k* value, while for the dense point sets, the best results are for lower values of *k* only, with a decline in accuracy as the value of *k* increases.

Finally, Table 1 depicts the overall construction times (in seconds) of the Area Code Tree for the various sets of POIs. We observe that the time it takes to construct an Area code tree via repeated insertion increases significantly with the size of the POI set. However, the absolute worst case is still under 3 minutes. For the North Island datasets, the time ranges from just a few seconds for 1000 POIs, up to 2.5 minutes for 10,000 POIs. For the Waikato Region datasets, the times range from a few seconds to over 3 minutes. Although these overall construction times are high when compared to the search times, two things must be noted. First, for static data sets, the Area Code tree only needs to be constructed once in order to be searched many times. Second, the occasional insertion can still be performed without having to re-construct the entire Area Code tree. Therefore, given the search performance results above, the construction time is a small price that must be paid.

Table 1: Tree construction times (in seconds)

Data Sets	#POIs	O/A Time
North Island	1000	4.04
	2000	10.98
	3000	19.60
	4000	31.46
	5000	44.31
	6000	61.20
	7000	81.80
	8000	103.61
	9000	127.37
	10000	158.24
Waikato	1000	5.02
	2000	12.63
	3000	23.38
	4000	37.45
	5000	55.19
	6000	75.97
	7000	99.50
	8000	127.75
	9000	159.02
	10000	192.71

5 Conclusion

In this paper, we present a *k*-nearest neighbor search strategy for the Area Code tree. We also evaluate the accuracy, tree construction time, and also comparatively evaluate the search time against a Brute Force strategy. We find that approximate *k*-nearest neighbor searching can be accomplished in very low and constant time, regardless of the number of POIs being indexed. With respect to accuracy, up to 60% accuracy is achieved when the Area Code tree is used for locating 1-nearest neighbor in dense POI sets. This makes the Area Code tree a significant candidate for continuous approximate nearest neighbor search for location-based services. The only significant factor is in the tree construction time. However, for fairly static data sets, this is a small price to pay for the savings in search time and increased accuracy (in some cases) that are achieved. Some future research directions include the following. First, the Area Code tree must have its *k*-nearest neighbor search performance evaluated further by comparing it with other spatial access methods. Second, further examination of the Area Code tree for continuous *k*-nearest neighbor searching must also take place. Finally, the Area Code tree needs to be expanded to utilize other types of "area codes", such as telephone area codes, and postal/zip codes. However, the results presented here show that the Area Code tree has promise for different applications in continuous query processing.

Acknowledgments

The authors would like to thank the reviewers of this manuscript and the previous conference paper for their thoughtful and constructive comments.

References

[1] V. Gaede and O. Günther, "Multidimensional Access Methods," *ACM Computing Surveys*, 30:170–231, 1998.

[2] A. Guttman, "R-trees: A Dynamic Index Structure for Spatial Searching," *Proc. ACM SIGMOD Int'l Conf. Management of Data*, pp. 47–57, 1984.

[3] H. Hu, J. Xu, W.S. Wong, B. Zheng, D.L. Lee and W.-C. Lee, "Proactive Caching for Spatial Queries in Mobile Environments," *Proc. 21st Int'l Conf. Data Engineering*, pp. 403–414, 2005.

[4] H.R. Jung, S.-W. Kang, M.B. Song, S.J. Im, J. Kim and C.-S. Hwang, "Towards Real-Time Processing of Monitoring Continuous k-Nearest Neighbor Queries," *Proc. 2006 Int'l Conf. Frontiers of High Performance Computing*, pp. 11-20, 2006.

[5] K.C.K. Lee, W.-C. Lee, H.V. Leong, B. Unger and B. Zhang,"Efficient Valid Scope Computation for Location-Dependent Spatial Queries in Mobile and Wireless Environments," *Proc. 3rd Int'l Conf. Ubiquitous Information Management and Communication*, pp. 131-140, 2009.

[6] W. Osborn and A. Hinze, "TIP-tree: A Spatial Index for Traversing Locations in Context-Aware Mobile Access to Digital Libraries," *Pervasive and Mobile Computing*, 15:26–47, December 2014.

[7] Y. Park, K. Bok and J. Yoo, "An Efficient Path Nearest Neighbor Query Processing Scheme for Location-Based Services," *Proc. 17th Int'l Conf. Database Systems for Advanced Applcations*, pp. 123–129, 2012.

[8] F. Rahman and W. Osborn, "The Area Code Tree for Nearest Neighbor Searching," *Proc. 2015 IEEE Pacific Rim Conf. Communications, Computers and Signal Processing*, pp. 153–158, 2015.

[9] F. Rahman and W. Osborn, "The Area Code Tree for Approximate Nearest Neighbor Searching in Dense Point Sets," *ISCA 25th Int'l Conf. Software Engineering and Data Engineering*, pp. 103–108, 2016.

[10] N. Roussopoulos, S. Kelley and F. Vincent, "Nearest Neighbor Queries,", *SIGMOD Rec.*, 24(2):71–79, May 1995.

[11] J.H. Schiller and A. Voisard, Editors, *Location-Based Services*, Morgan Kauffman, 2004.

[12] S. Shekhar and S. Chawla, *Spatial Databases: A Tour*, Prentice Hall, 2003.

[13] Z. Song and N. Roussopoulos, "K-Nearest Neighbor Search for Moving Query Point," *Proc. 7th Int'l Symp. Advances in Spatial and Temporal Databases*, pp. 79–96, 2001.

[14] Y. Tao, D. Papadias, and Q. Shen, "Continuous Nearest Neighbor Search," *Proc. 28th Int'l Conf. Very Large Data Bases*, pp. 287–292, 2002.

Wendy Osborn is an Associate Professsor of Computer Science at the University of Lethbridge. She obtained her B.C.S. (Hons) and M.Sc. degrees at the University of Windsor (Ontairo, Canada) in 1996 and 1998 respectively, and her Ph.D. at the University of Calgary (Alberta, Canada) in 2005. Her research interests include location-based services, spatial access methods, and distributed query processing for standard and non-standard data.

Fatema Rahman obtained her B.Sc. degree in Computer Science and Engineering at Jahangirnagar University (Dhaka, Bangladesh) in 2009. Currently, she is a Master of Science (Computer Science) candidate at the University of Lethbridge, and expects to complete her degree in the Summer of 2017. Her fields of interest include data management, mining and warehousing, pattern recognition and image processing.