To appear in *Encyclopedia of GIS*, S. Shekhar, X. Hui, and X. Zhou, eds.
Springer, Berlin, second edition, 2016.

Foundations of Nearest Neighbor Queries in Euclidean Space[*]

Hanan Samet
Computer Science Department
Center for Automation Research
Institute for Advanced Computer Studies
University of Maryland
College Park, Maryland 20742
`hjs@cs.umd.edu` `www.cs.umd.edu/~hjs`

December 31, 2015

Abstract

A number of approaches to computing nearest neighbor queries in Euclidean space are presented. This includes the depth-first and best-first methods as well as a comparison. The best first method is shown to be capable of being extended to report the neighboring objects in increasing order from the query object so that the search can be incremental and there is no need to know the value of *k* in advance. The incremental algorithm is shown to be modifiable to also work for objects that have spatial extent instead of being restricted to be point objects. The best-first method is also shown to yield the *k* approximate nearest neighbors give an error tolerance value.

Keywords: nearest neighbor query, depth-first nearest neighbor query, best-first nearest neighbor query, incremental nearest neighbor query, approximate nearest neighbor query

1 Introduction

The nearest neighbor query is a key operation in geographic information systems (GIS), spatial databases. and location-based services, not to mention other disciplines like computer vision and machine learning where it is used for similarity search (e.g., [25]) and is usually discussed in the context of finding the *k* nearest neighbors as is also done here. It forms the heart of all queries (e.g., [10, 27]), where a location is given and one seeks to find the nearest object or objects (e.g., gas station, hotel, restaurant, etc.) that optionally satisfies another, usually nonspatial, condition (e.g., product type, price, opening hours, etc.). This process is facilitated by building an index on the data which is usually based on a hierarchical clustering. The idea is that the data objects are partitioned into clusters (termed *nonobjects*) which are aggregated to form other clusters, with the total aggregation being represented as a tree often referred to as a *search hierarchy*. The *k* nearest neighbors are found by applying either a depth-first or a best-first algorithm to the search hierarchy containing the data. The algorithms are generally applicable to any spatial index based on hierarchical clustering.

[*]This work was supported in part by the National Science Foundation under Grants IIS-12-19023 and IIS-13-20791. This chapter and all its figures copyright 2015 by Hanan Samet. All Rights Reserved.

The rest of this chapter is organized as follows. Sections 2 and 3) describe the depth-first and best-first k nearest neighbor methods, respectively, for arbitrary values of k, while Section 4 compares them. Next, Section 5 shows how to extend the best-first method to report objects in increasing order of distance from the query object thereby freeing us from having to know the value of k in advance, and, more importantly, there is no need to restart the k nearest neighbor search process when k increases as we can simply resume/continue the search for the $k+1^{st}$ and additional nearest neighboring objects from where the search last left off. The context of the discussion is in terms of spatial objects that are points in a Euclidean space. Section 6 shows how to modify the best-first incremental method to deal with spatial objects that have extent and whose representation decomposes them with respect to the space that they occupy so that there are multiple references to them, yet they are only reported once. Section 7 follows with a demonstration of how to modify the best-first method to yield approximate nearest neighbors given an error tolerance value. Section 8 concludes by mentioning other domains where similar algorithms have been applied.

2 Depth-First k Nearest Neighbor Method

The most common strategy for finding the k nearest neighbors is the depth-first method which explores the elements of the search hierarchy in a depth-first manner (e.g., [11]). The k nearest neighbors found so far are kept track of in a set L with the aid of a variable D_k that indicates the distance, using a suitably defined distance function d (Euclidean in this paper), of the current kth-nearest object from the query object q. The depth-first method visits every element of the search hierarchy. The *branch and bound* variant of the depth-first method yields better performance by not visiting every nonobject and its objects when it can be determined that it is impossible for the nonobject to contain any of the k nearest neighbors of q [11, 19]). For example, this is true if we know that for every nonobject element e of the search hierarchy, $d(q,e) \leq d(q,e_0)$ for every object e_0 in e (known as the *correctness criterion* [16]) and that the relation $d(q,e) > D_k$ is satisfied[1]. This can indeed be achieved if we define $d(q,e)$ as the minimum possible distance from q to any object e_0 in nonobject e (referred to as MINDIST in contrast to MAXDIST, the maximum possible distance, which unlike MINDIST cannot be used for pruning).

Letting $A(e)$ denote the set of nonobject immediate descendants e_i of nonobject element e of the search hierarchy, using the above definition of distance for nonobject elements (i.e., MINDIST) makes it possible to obtain even better performance as a result of speeding up the convergence of D_k to its final value by processing elements e_i of $A(e)$ in increasing order of $d(q,e_i)$ (i.e., a MINDIST ordering). In this way, once an element e_i in $A(e)$ is found such that $d(q,e_i) > D_k$, then $d(q,e_j) > D_k$ for all remaining elements e_j of $A(e)$. This means that none of these remaining nonobject descendants of e need to be processed further, and the algorithm backtracks to the parent of e, or terminates if e is the root of the search hierarchy.

[1]This stopping condition ensures that all objects at the distance of the kth-nearest neighbor are examined. Note that if the size of L is limited to k and if two or more objects are at distance D_k, then some of them may not be reported in the set of q's k nearest neighbors.

3 Best-First k Nearest Neighbor Method

An alternative strategy to the depth-first method is the best-first method (e.g., [8, 13, 14, 15]) which explores the nonobject elements of the search hierarchy in increasing order of their distance from q (hence the characterization as "best-first") rather than in a predetermined order, as in the depth-first method. In other words, at each step of the algorithm, the next nonobject element to be visited is the closest one to q which has yet to be visited. This is achieved by storing the nonobject elements of the search hierarchy in a priority queue *Queue* according to this order. *Queue* is initialized to contain the root of the search hierarchy at a distance of 0 from q, and as nonobject elements are dequeued, their immediate descendants e that are nonobject elements are enqueued with their corresponding distances from q if $d(q,e) < D_k$, while immediate descendants o that are objects are inserted into L if $d(q,o) < D_k$, where D_k, the distance of the current kth-nearest neighbor of q, is initialized to ∞. The algorithm repeatedly removes nonobject elements from *Queue* until it is empty or until encountering a nonobject element that is farther from q than D_k, at which time it halts as it has found the k nearest neighbors, now in L which, as in the depth-first algorithm, keeps track of them with the aid of variable D_k. In order for the algorithm to be correct, the distance $d(q,e)$ of any nonobject element e from the query object q must be less than or equal to the distance from q to any object in e's descendants [16]. Again, as in the depth-first algorithm, this property is satisfied by letting $d(q,e)$ be MINDIST. The best-first method finds much use in computer vision where it serves as a key stage of the SIFT algorithm [21].

4 Comparison of the Depth-First and Best-First Algorithms

The drawback of the best-first method is that the priority queue may be rather large as can be seen in the example space decomposition represented by an R-tree [12] object hierarchy in Figure 1. Using a Euclidean distance metric, the necessary amount of storage may be as large as the total number of nonobjects (and hence on the order of the number of objects) if the distance of each of the nonobjects from the query object q ($\times$ in the figure) is approximately the same. In low dimensions, such an event is relatively rare as its occurrence requires two seemingly independent events — that is, that all objects lie in an approximate hypersphere centered at some point p and that the query object q be coincident with p. However, in high dimensions, where most of the data lies on the surface (e.g., [5]) and the curse of dimensionality [6] comes into play, and in metric spaces with concentrated distance histograms, this situation is less rare. In contrast, the amount of storage required by the depth-first method is bounded. In particular, it is proportional to the sum of k and the maximum depth of the search hierarchy, where, in the worst case, all of the sibling nonobject elements must be retained for each partially explored nonobject element in the search hierarchy while executing the depth-first search.

Nevertheless, the advantage of the best-first method over the depth-first method is that it has been shown to be I/O optimal for $k = 1$ [4]. This means that the algorithm does not visit more than the minimum number of nonobject elements—that is, it avoids visiting nonobject elements that will eventually be determined to be too far from q due to poor initial estimates of D_k. This is equivalent to stipulating that the algorithm is range-optimal [16], which means that the cost of finding the k nearest neighbors is the same as that of a range search with the search radius set to the distance from q to its kth-nearest neighbor.

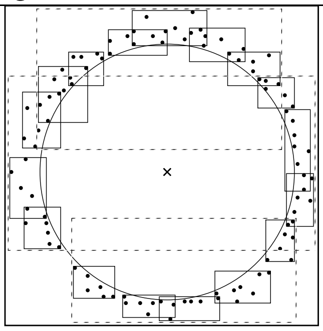

Figure 1: An example of an R-tree of points with a cluster size of 8, showing a worst case for the size of the priority queue for nearest neighbor search using the best-first algorithm around query point x with a Euclidean distance metric.

As we saw above, the implementations of both the depth-first and best-first methods make heavy use of a lower bound MINDIST corresponding to the minimum distance at which a nearest object can be found vis a vis the distance D_k to the current kth-nearest object from q. It has also been shown [26] that an upper bound MAXNEARESTDIST corresponding to the maximum possible distance at which a nearest neighbor is guaranteed to be found can be used to overcome the shortcomings that we pointed out earlier of both the depth-first (i.e., pruning) and best-first (i.e., size of the priority queue) k-nearest neighbor methods for arbitrary values of k (i.e., $k \geq 1$). MAXNEARESTDIST was first introduced in [20] for the case of $k = 1$ for the purpose of improving the initial estimate of D_k in the depth-first method. It was also proposed in [24] as an alternative to the MINDIST ordering for the processing of the nonobject immediate descendants of a nonobject element in the depth-first method but again limited to $k = 1$. However, for the purpose of ordering, MINDIST has been shown to be more useful [15, 24]. Thus MAXNEARESTDIST is not used for this purpose.

5 Finding the *k* Nearest Neighbors in Incremental Order

Some implementations of the best-first nearest neighbor method (e.g., [13, 14, 15, 22]) store both the objects and the nonobjects in the same priority queue thereby enabling the algorithms that employ this method to be incremental. This means that now both the objects and nonobjects are visited in increasing order of their distance from q, and the objects are also reported in increasing order of their distance from q. These implementations are designed for the case that k is not known in advance, thereby making them inappropriate for use with the MAXNEARESTDIST upper bound as no nonobject elements can be excluded from *Queue* since they may all be eventually needed should k get sufficiently large.

6 Finding *k* Nearest Neighbors for Non-Point Objects

The incremental implementation of the best-first nearest neighbor method in Section 5 can only handle spatial indexes where each object is represented just once (e.g., an R-tree). This is fine when the multidimensional data consists of points. However, in many applications, such as computer graphics, geographic information systems (GIS), computer vision, and so on, the multidimensional data also has extent (e.g., lines, rectangles, regions, surfaces, volumes). In this case, it may not be desirable, or even possible, to represent the objects as points or by one entity such as a minimum bounding box (as is done in the R-tree), as these entities are not disjoint thereby complicating certain search queries. Instead, the domain of the data (i.e., the space occupied by the objects) is decomposed into disjoint blocks containing the objects or disjoint segments of the objects, and the spatial index is used to facilitate the identification and access of the blocks that are occupied by the objects as well as to indicate the identity of the objects. Examples of such indexes include the region quadtree [25], members of the PM and PMR quadtree [17] family, the R$^+$-tree [34], and so on.

The disadvantage of spatial indexes that use disjoint blocks and that segment the objects is the presence of duplicates in the sense that some objects may be reported several times for queries (e.g., finding all objects in a rectangular query region w since the object could have been decomposed into several segments, each of which overlaps a part of w). Nevertheless, there has been some work on developing algorithms for spatial queries that report objects just once, even though they may be duplicated in several blocks or nodes (e.g., [1, 2]). Therefore, the extension of the incremental implementation of the best-first $k$nearest neighbor method to handle these spatial indexes means that we must be able to deal with the possible presence of duplicate instances of the objects in the search hierarchy. There are two issues here. The first is to avoid inserting into the priority queue duplicate instances of an object that has already been reported. The second is to make sure to detect all duplicate instances that are currently in the priority queue when reporting the object.

The first issue is resolved by noting that it could occur if we encounter an object o as a descendant of an ancestor element A where the distance of o from query object q is smaller than that of A from q. This situation, which could arise when using a spatial index based on a disjoint decomposition of the underlying space, is actually a violation of the correctness criterion (defined in Section 2), which stipulates that, for any object e_0 in the subtree represented by e_t, we have $d_t(q,e_t) \leq d_0(q,e_0)$. Therefore, we modify the correctness criterion to be "for all objects $o \in S$ (note that there could be multiple occurrences of particular objects), there exists at least one element e_0 in the search hierarchy that represents o and that satisfies $d_t(q,e_t) \leq d_0(q,e_0)$ for all ancestors e_t of e_0." So, once o has been reported, subsequent encounters of it with a smaller distance from q than a distance to one of its nonobject ancestor element are ignored by not inserting it into the priority queue. However, duplicate instances of an object can still be inserted into the priority queue prior to the object being reported.

The second issue arises when it is possible that there are multiple instances of an object o in the priority queue at the same time when reporting the object. This situation is detected by stipulating that the objects be ordered in the priority queue by their identity so that when the priority queue contains duplicate instances of two or more objects with the same distance, then all duplicate instances of one of the objects will appear before all duplicate instances of the remaining objects. This ensures that duplicate instances of two different objects at the same distance are not obtained from the queue in an interleaved manner. We also stipulate that nodes must be retrieved from the queue before spatial objects at the same distance. Otherwise, an object o may be retrieved from the queue and reported before a node A that contains o (i.e., A is an ancestor node, where we note that the presence of duplicate instances of objects means that an object may have more than one ancestor of a given type) that is at the same distance from the query object as o. This means that o was contained in another node B that has already been dequeued by virtue of being an ancestor of o. Without such a stipulation, we would have the situation where when o was encountered again in node A, there would be no way of knowing that o had already been reported on account of being a descendant of B.

7 Incremental Approximate k Nearest Neighbor Method

In many applications, obtaining exact results is not critical. Therefore, users are willing to trade accuracy for improved performance by using an approximation error tolerance ε. The approximation criterion is that the distance between the query object q and the resulting candidate nearest neighbor o' is within a factor of $1+\varepsilon$ of the distance to the actual nearest neighbor o—that is, $d(q,o') \leq (1+\varepsilon)\cdot d(q,o)$. This is the basis of the bulk of the results on approximate nearest neighbor search (e.g., [3]).

Both the depth-first and best-first k-nearest neighbor methods can be adapted to return the approximate nearest neighbors although it is more common to adapt the best-first methods and this is our focus here. The reason is that the motivation for loosening the criterion as to what constitutes a nearest neighbor is to speed up the process, which is more in the spirit of the best-first methods that are inherently designed to cease processing as soon as the desired number of neighbors is found rather than possibly exploring the entire search hierarchy, as is the case for the depth-first method. Regardless of whether the approximate k-nearest neighbor algorithm is an adaptation of the best-first or depth-first k-nearest neighbor method, it is important to observe that, as a result, some of q's exact k nearest neighbors that are within a factor of $1+\varepsilon$ of the distance D_k of q's approximate kth-nearest neighbor (including the exact kth-nearest neighbor o_k) may be missed (i.e., those neighbors o where $D_k/(1+\varepsilon) \leq d(q,o) < D_k$).

A number of approximate k-nearest neighbor algorithms use of a priority queue *Queue* to support a "best-first" traversal of the search hierarchy. Most of these algorithms (e.g., [3]) only store the nonobject elements of the spatial index in *Queue* (of course, sorted in the order of their distance from q). In particular, they do not store the objects (usually points) in *Queue*. Thus, these algorithms have the same control structure that we described for the best-first algorithm in Section 3.

These algorithms work by shrinking the distance D_1 (D_k) from q to the nearest (kth-nearest, in which case the k objects are maintained in L) object o by a factor of $1+\varepsilon$ and by only inserting the nonobject element e_p into *Queue* if the minimum distance from q to an object in e_p (i.e., $\text{MinDist}(q,e_p)$) is less than this shrunken distance. D_1 (D_k) and o (L) are updated as nonobject leaf elements are removed from *Queue*.

The algorithms process the elements in *Queue* in increasing order of their distance from q and halt when the element t corresponding to nonobject element e with distance $\text{D}(t)$ (i.e., $\text{MinDist}(q,e)$) is removed from *Queue*, where $\text{D}(t)$ is greater than the shrunken distance as all remaining nonobject elements in *Queue* are at the same or greater distance, and thus the objects that they contain are too far away from q. Of course, letting $\varepsilon = 0$ means that this method can also be used to find the exact k nearest neighbors, in which case it is equivalent to the best-first k nearest neighbor algorithm. In [3] it is shown that the use of this algorithm with a BBD-tree [25] enables finding the ε-nearest neighbor in time inversely proportional to ε.

The drawback of implementations that make use of the shrinking approach (e.g., [3]) is that they can only be used to find the approximate nearest neighbor or the k approximate nearest neighbors. In particular, such implementations cannot obtain the neighbors incrementally (regardless of whether or not they are approximate) because, once a nearer object than the object o' currently deemed to be the nearest has been found or once it has been decided not to enqueue a nonobject element a, we cannot go back and process o' or a later when more neighbors are needed.

Hjaltason and Samet [16] observe that instead of shrinking distances (e.g,, D_k) by a factor of $1+\varepsilon$, the distances of the nonobject elements e_p (i.e., $\mathrm{MinDist}(q,e_p)$) could be expanded by a factor of $1+\varepsilon$ and used in all comparisons with D_k rather than the shrunken distances. In other words, they suggest multiplying the key values for all nonobject elements in the priority queue *Queue* by $1+\varepsilon$; that is, for a nonobject element e_p, use $(1+\varepsilon)\cdot\mathrm{MinDist}(q,e_p)$ as a key. The advantage of using this simple and equivalent approximation mechanism is that it enables them to transform an incremental exact nearest neighbor algorithm that uses it into an incremental approximate nearest neighbor algorithm, where k need not be known in advance, by simply always also enqueueing the objects with their true distance, as well as always enqueueing a nonobject element with its expanded distance.

From a practical point of view, in the incremental approximate nearest neighbor algorithm, enqueueing a nonobject element e_t with a larger distance value (i.e., by a factor of $1+\varepsilon$ in this case) means that we delay its processing, thereby allowing objects to be reported "before their time." In particular, once e_t is finally processed, all of the objects o that we have reported satisfy $d(q,o) \leq (1+\varepsilon)\cdot d_t(q,e_t)$ (which is greater than $d_t(q,e_t)$ if $\varepsilon > 0$). Therefore, it could be the case that there exists an object c in e_t with a distance $d(q,c) \leq d(q,o)$, yet o is reported before c. It should be clear that in this case the algorithm does not necessarily report the resulting objects in strictly increasing order of their distance from q. With this modification, for the object o'_k returned as the kth-nearest neighbor by the algorithm and the actual kth-nearest neighbor o_k, we have $d(q,o'_k) \leq (1+\varepsilon)\cdot d(q,o_k)$. The same technique of expanding all distances by a factor of $1+\varepsilon$ can also be applied to both the nonincremental best-first and the depth-first k-nearest neighbor algorithms to yield a corresponding approximate k-nearest neighbor algorithm.

8 Concluding Remarks

We have discussed methods for finding nearest neighbors in a Euclidean vector space. They are also applicable to non-vector data (e.g., [18]) such as data lying in a metric space many of which are surveyed in [7, 9, 18, 25, 35] but are beyond the scope of this chapter. Similar ideas have also been applied to find nearest neighbors in spatial networks (e.g., [23, 28, 29, 30, 31, 32, 33]).

Cross-References

Spatial data structures; Indexing; Spatial databases

Further Reading:

1. See [25] for a comprehensive treatment of multidimensional as well as metric data structures and how to perform nearest neighbor queries.

2. See `http://donar.umiacs.umd.edu/quadtree/index.html` for JAVA applets that illustrate the incremental best-first nearest neighbor method for a variety of spatial data structures.

References

[1] W. G. Aref and H. Samet. Uniquely reporting spatial objects: yet another operation for comparing spatial data structures. In *Proceedings of the 5th International Symposium on Spatial Data Handling*, pages 178–189, Charleston, SC, August 1992.

[2] W. G. Aref and H. Samet. Hashing by proximity to process duplicates in spatial databases. In *Proceedings of the 3rd International Conference on Information and Knowledge Management (CIKM)*, pages 347–354, Gaithersburg, MD, December 1994.

[3] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu. An optimal algorithm for approximate nearest neighbor searching in fixed dimensions. *Journal of the ACM*, 45(6):891–923, November 1998. Also see *Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 573–582, Arlington, VA, January 1994.

[4] S. Berchtold, C. Böhm, D. A. Keim, and H.-P. Kriegel. A cost model for nearest neighbor search in high-dimensional data space. In *Proceedings of the 16th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, pages 78–86, Tucson, AZ, May 1997.

[5] S. Berchtold, C. Böhm, and H.-P. Kriegel. Improving the query performance of high-dimensional index structures by bulk-load operations. In *Advances in Database Technology—EDBT'98, Proceedings of the 6th International Conference on Extending Database Technology*, H.-J Schek, F. Saltor, I. Ramos, and G. Alonso, eds., vol. 1377 of Springer-Verlag Lecture Notes in Computer Science, pages 216–230, Valencia, Spain, March 1998.

[6] K. S. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. When is "nearest neighbor" meaningful? In *Proceedings of the 7th International Conference on Database Theory (ICDT'99)*, C. Beeri and P. Buneman, eds., vol. 1540 of Springer-Verlag Lecture Notes in Computer Science, pages 217–235, Berlin, Germany, January 1999.

[7] C. Böhm, S. Berchtold, and D. A. Keim. Searching in high-dimensional spaces: index structures for improving the performance of multimedia databases. *ACM Computing Surveys*, 33(3):322–373, September 2001.

[8] A. J. Broder. Strategies for efficient incremental nearest neighbor search. *Pattern Recognition*, 23(1–2):171–178, January 1990.

[9] E. Chávez, G. Navarro, R. Baeza-Yates, and J. Marroquín. Searching in metric spaces. *ACM Computing Surveys*, 33(3):273–322, September 2001.

[10] C. Esperança and H. Samet. Experience with SAND/Tcl: a scripting tool for spatial databases. *Journal of Visual Languages and Computing*, 13(2):229–255, April 2002.

[11] K. Fukunaga and P. M. Narendra. A branch and bound algorithm for computing *k*-nearest neighbors. *IEEE Transactions on Computers*, 24(7):750–753, July 1975.

[12] A. Guttman. R-trees: a dynamic index structure for spatial searching. In *Proceedings of the ACM SIGMOD Conference*, pages 47–57, Boston, June 1984.

[13] A. Henrich. A distance-scan algorithm for spatial access structures. In *Proceedings of the 2nd ACM Workshop on Geographic Information Systems*, N. Pissinou and K. Makki, eds., pages 136–143, Gaithersburg, MD, December 1994.

[14] G. R. Hjaltason and H. Samet. Ranking in spatial databases. In *Advances in Spatial Databases—4th International Symposium, SSD'95*, M. J. Egenhofer and J. R. Herring, eds., vol. 951 of Springer-Verlag Lecture Notes in Computer Science, pages 83–95, Portland, ME, August 1995.

[15] G. R. Hjaltason and H. Samet. Distance browsing in spatial databases. *ACM Transactions on Database Systems*, 24(2):265–318, June 1999. Also University of Maryland Computer Science Technical Report TR–3919, July 1998.

[16] G. R. Hjaltason and H. Samet. Incremental similarity search in multimedia databases. Computer Science Technical Report TR–4199, University of Maryland, College Park, MD, November 2000.

[17] G. R. Hjaltason and H. Samet. Speeding up construction of PMR quadtree-based spatial indexes. *VLDB Journal*, 11(2):109–137, October 2002. Also University of Maryland Computer Science Technical Report TR–4033, July 1999.

[18] G. R. Hjaltason and H. Samet. Index-driven similarity search in metric spaces. *ACM Transactions on Database Systems*, 28(4):517–580, December 2003.

[19] B. Kamgar-Parsi and L. N. Kanal. An improved branch and bound algorithm for computing *k*-nearest neighbors. *Pattern Recognition Letters*, 3(1):7–12, January 1985.

[20] S. Larsen and L. N. Kanal. Analysis of k-nearest neighbor branch and bound rules. *Pattern Recognition Letters*, 4(2):71–77, April 1986.

[21] D. Lowe. Object recognition from local scale-invariant features. In *Proceedings of 7th International Conference on Computer Vision*, vol. 2, pages 1150–1157, Corfu, Greece, September 1999.

[22] G. Navarro. Searching in metric spaces by spatial approximation. *VLDB Journal*, 11(1):28–46, August 2002.

[23] D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao. Query processing in spatial network databases. In *Proceedings of the 29th International Conference on Very Large Data Bases (VLDB)*, J. C. Freytag, P. C. Lockemann, S. Abiteboul, M. J. Carey, P. G. Selinger, and A. Heuer, eds., pages 802–813, Berlin, Germany, September 2003.

[24] N. Roussopoulos, S. Kelley, and F. Vincent. Nearest neighbor queries. In *Proceedings of the ACM SIGMOD Conference*, pages 71–79, San Jose, CA, May 1995.

[25] H. Samet. *Foundations of Multidimensional and Metric Data Structures*. Morgan-Kaufmann, San Francisco, 2006. (Translated to Chinese ISBN 978-7-302-22784-7).

[26] H. Samet. K-nearest neighbor finding using MaxNearestDist. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(2):243–252, February 2008.

[27] H. Samet, H. Alborzi, F. Brabec, C. Esperança, G. R. Hjaltason, F. Morgan, and E. Tanin. Use of the SAND spatial browser for digital government applications. *Communications of the ACM*, 46(1):63–66, January 2003.

[28] H. Samet, J. Sankaranarayanan, and H. Alborzi. Scalable network distance browsing in spatial databases. In *Proceedings of the ACM SIGMOD Conference*, pages 43–54, Vancouver, Canada, June 2008. Also see University of Maryland Computer Science Technical Report TR–4865, April 2007.

[29] J. Sankaranarayanan, H. Alborzi, and H. Samet. Efficient query processing on spatial networks. In *Proceedings of the 13th ACM International Symposium on Advances in Geographic Information Systems*, pages 200–209, Bremen, Germany, November 2005.

[30] J. Sankaranarayanan, H. Alborzi, and H. Samet. Distance join queries on spatial networks. In *Proceedings of the 14th ACM International Symposium on Advances in Geographic Information Systems*, pages 211–218, Arlington, VA, November 2006.

[31] J. Sankaranarayanan and H. Samet. Distance oracles for spatial networks. In *Proceedings of the 25th IEEE International Conference on Data Engineering*, pages 652–663, Shanghai, China, April 2009.

[32] J. Sankaranarayanan and H. Samet. Query processing using distance oracles for spatial networks. *IEEE Transactions on Knowledge and Data Engineering*, 22(8):1158–1175, August 2010. Best Papers of ICDE 2009 Special Issue.

[33] J. Sankaranarayanan, H. Samet, and H. Alborzi. Path oracles for spatial networks. *PVLDB*, 2(1):1210–1221, August 2009. Also *Proceedings of the 35th International Conference on Very Large Data Bases (VLDB)*.

[34] T. Sellis, N. Roussopoulos, and C. Faloutsos. The R^+-tree: a dynamic index for multi-dimensional objects. In *Proceedings of the 13th International Conference on Very Large Databases (VLDB)*, P. M. Stocker and W. Kent, eds., pages 71–79, Brighton, United Kingdom, September 1987. Also University of Maryland Computer Science Technical Report TR–1795, 1987.

[35] P. Zezula, Amato G, V. Dohnal, and M. Batko. *Similarity Search: The Metric Space Approach*. Springer-Verlag, Berlin, Germany, 2006.