An Efficient Nearest Neighbor Algorithm for P2P Settings[1]

Egemen Tanin
Dept. of Computer Science & Software Engineering
NICTA Victoria Laboratory
University of Melbourne
Victoria, Australia
egemen@cs.mu.oz.au

Deepa Nayar
Dept. of Computer Science & Software Engineering
University of Melbourne
Victoria, Australia
dnayar@cs.mu.oz.au

Hanan Samet
Dept. of Computer Science
Center for Automation Research
Institute for Advanced Computer Studies
University of Maryland
Maryland, USA
hjs@cs.umd.edu

ABSTRACT
New Peer-to-Peer (P2P) applications like P2P job-employee seeker networks and P2P virtual cities, for application domains such as collaborative urban planning and forming virtual communities, are about to emerge. An important component in these applications is spatial data, i.e., data with locational components. Many requests initiated on spatial data involve finding the spatial objects that are *nearest* to a query location. In this paper, we propose an efficient algorithm that finds the spatial objects that are nearest to a given query location on a P2P network in the order of their minimum distance to the query point. The proposed algorithm makes use of a distributed spatial index that does not rely on the use of a central server. The algorithm is designed to be more efficient by utilizing the parallel nature of the P2P network. A demonstration of the proposed algorithm was implemented as a prototype P2P application that finds events and places of interest in a city.

Categories and Subject Descriptors
H.2.8 [**Database Management**]: Database Applications -- *Spatial databases*; C.2.4 [**Computer Communication Networks**]: Distributed Systems -- *Distributed applications*; E.1 [**Data Structures**]: Distributed Data Structures

General Terms
Algorithms, Management

Keywords
Geographic Information Systems, Spatial Data, Nearest Neighbor Query, Quadtree, Peer-to-Peer Networks

1. INTRODUCTION
Peer-to-peer (P2P) networks are becoming increasingly popular as a powerful means for data exchange. They are quite scalable and easy to deploy. Although P2P networks attract significant interest, methods for accessing complex data such as *spatial data* on P2P networks are still at their infancy. Several future P2P applications like P2P job-employee seeker networks and P2P virtual cities will need to answer queries involving spatial data, i.e., data with locational components. Frequently more than one location is associated with a spatial object. Hence, it commonly differs from conventional *point data* in that the objects may also have extent. For example, lakes can be represented as spatial objects whose extent is defined by the space that they occupy. There are many ways to describe such objects including their boundary, the locations that make up their interior, their minimum axis-aligned bounding box, etc. Thus a spatial description is more than just a longitude and a latitude value. Retrieving such complex data, given a query, on a P2P network is a non-obvious task. There is no central server or administration that the data is stored. Hence, classical indexing methods and querying algorithms cannot be easily applied. Yet, efficient solutions to querying and locating spatial data on P2P networks can enable many government as well as other public domain networked applications. For example, from a digital government point of view, government agencies can form ad hoc virtual environments where they present and exchange their data without dedicated servers or can help the general public to exchange this data among themselves using existing, mostly unused, computational resources available at users' machines.

Often, given some user data, we need to find its closest matches in a large data set. For instance, given a spatial data set and a query point, we frequently need to find the spatial objects closest to this query point. In our case of a P2P network with spatial objects, this *nearest neighbor* (NN) query involves finding the *closest spatial objects* on a large dynamic network. *Closeness* can be measured by any number of similarity measures. In our NN queries, we are dealing with finding the nearest spatial objects in the metric space using the Euclidean distance between the object and

[1] This work was supported in part by the US National Science Foundation under Grant EIA-00-91474 and Microsoft Research.

the query point. This can be considered as a specialization of the *similarity search* concept used in databases.

For example, in a future P2P virtual city, a user may point to a location on a spatial layout and request the system to find events (such as concerts), restaurants, parks, public facilities, or various places of interest that are closest to this location, i.e., for a metropolitan area. The efficient implementation of NN queries is a mater of interest in spatial databases for some time. In P2P networks, the issue of handling spatial data and queries has just started to surface. In this paper, we propose an efficient algorithm to find the spatial objects that are nearest to a query point in a P2P system by making use of an index that provides distributed hashing of spatial data. The algorithm is designed to be more efficient by utilizing the parallel nature of the P2P network. A demonstration of the proposed algorithm was implemented as a prototype P2P application that finds events and places of interest in a city.

The rest of this paper is organized as follows. Section 2 reviews the related work in spatial databases and in P2P systems. In Section 3 we present our early work in P2P spatial data management. Section 4 introduces our NN algorithm and describes the details of the algorithm using a distributed quadtree index. Section 5 presents the analysis of our algorithm. Section 6 describes our prototype P2P implementation. Section 7 states our concluding remarks and describes our future work.

2. RELATED WORK

Hjaltason and Samet (2003) present a comprehensive analysis of various similarity search algorithms in metric spaces. Their main contribution to field is through a *priority queue* based *ranking* algorithm that can find the results of a ranking query in an incremental fashion. Formally, ranking is a more general case of the NN query where a user can ultimately retrieve all the objects of a database in the order of their distance from a query point. The algorithm works on many classical spatial indexing methods including R-trees and quadtrees (for these data structures Samet 1990 and 2005 contain very detailed presentations). The entries of the priority queue consist of spatial objects as well as the blocks of the space that the underlying spatial data structure uses. The first entry in the queue is the root of the data structure. This entry is retrieved in the first iteration of the algorithm and the children of the root are then inserted back to the priority queue using their distance from the query point. The next iteration of the algorithm retrieves the new highest priority child/block or object. Hence, in this fashion, at each iteration of the algorithm, the element with the smallest distance is taken out and visited and its children are inserted into the queue (it is important to note that eventually all the spatial objects will be retrieved as they are also inserted into the queue by the previous iterations). We use the same incremental approach in our P2P NN algorithm. Unfortunately, this sequential algorithm is not designed for the distributed nature of the P2P networks.

NN queries have become the topic of interest in distributed data processing in recent years. A NN algorithm, using R-trees, in sensor networks, is presented by Demirbas and Ferhatosmanoglu 2003. NN queries are performed in a distributed fashion using a self-stabilizing P2P indexing structure called the *peer-tree* in their work. They also mention the importance of parallelizing the NN search with the main limiting constraint being the power consumption on sensors for their case. Unfortunately, they do not present a detailed and general solution for this issue in their work.

Very recently, Batko et al. August 2004 study the problem of executing P2P NN queries on metric spaces using a distributed data structure. They use the data structure called *GHT* (Generalized Hyperplane Tree) introduced in their previous work (Batko et al. June 2004). This data structure is mainly for non-spatial data and hence cannot be directly applied to the spatial domain. Nevertheless, they also mention that exploiting the parallel nature of P2P networks is an issue and their NN algorithm proposes a parallel approach. They state that the priority queue approach that we described earlier cannot be easily adapted for parallelism for their case. Hence, they use a radius estimation-based approach to create an ever-increasing search circle. They parallelize this front. In our work, we parallelize the priority queue based NN work described in Hjaltason and Samet 2003.

Banaei-Kashani and Shahabi 2004 present a family of access methods for similarity search in P2P networks. They name this family *SWAM*. They use a Voronoi diagram based scheme for indexing the spatial data. Accessing the first NN is especially very trivial using the neighboring cells of a Voronoi diagram. Although they do not mention the parallelism of their algorithms, their work is already tuned for similarity search and hence they easily can maintain a high level of efficiency. We think that our parallel approach can be applied for ranking in their indices too. Yet, Voronoi diagrams are harder to maintain in high dimensional data than other classical indices such as quadtrees and R-trees. Also, as they do not have a hierarchical access structure: queries have to traverse a planar data structure, in comparison to a hierarchical one, starting from a query initiating peer. To address this issue, they propose using random graphs. These graphs are probabilistic in nature in comparison to classical hierarchical spatial data structures.

Finally, Sahin et al. 2004 introduces a generic content-based similarity search scheme for documents in P2P systems. Their work is also focused on mostly finding similar documents over a P2P network and cannot be easily applied to spatial data.

3. BACKGROUND WORK

We use a distributed quadtree index that we recently developed to demonstrate our algorithm although other indices can be utilized as well, e.g., emerging spatial indices such as P2P R-trees (Mondal et al. 2004). We now describe this indexing mechanism (Harwood and Tanin 2003; Tanin et al. 2004) that can facilitate spatial data on P2P networks.

Our P2P index uses distributed hash tables (DHTs) and specifically the *Chord* (Stoica et al. 2003) method at its base although any similar key-based lookup method can easily be utilized. In Chord, a hash function is used to map arbitrary data strings (or keys) onto a logical name space. The Chord method also maps, along with these keys (e.g., file names), the peer addresses (i.e., IP addresses) in the P2P network to this logical name space. It then partitions the space among the peers. It presents an efficient way to hop between the peers to route towards the desired peer, i.e., to the peer that contains the file that the user is looking for given a file name. It is shown that Chord can find this file in $O(\log n)$ time with high probability for n peers.

For two-dimensional spatial data, the data space can be recursively subdivided into four regions, i.e., using a quadtree (Samet 1990, 2005). Our index makes use of a specific version of the quadtree concept to assign the responsibilities for regions of space to the peers of a P2P network. We use MX-CIF quadtrees (Kedem 1982; Sevcik and Koudas 1996) although other quadtree types can easily be used within our framework. In the MX-CIF quadtree, each object is associated with the smallest block of space that contains the entire object that is indexed. In our index, each quadtree block is uniquely identified by its centroid, named as a *control point.* The control points are hashed using their coordinate values (as a string key) with the Chord method to associate each of these keys with a peer. With a good hash function one can achieve a good level of load-balance among the peers of the network. Each control point generates a unique key. Then, objects associated with a quadtree block can be stored in the owner peer of that quadtree block. Hence, we use control points like buckets to store data. Queries can run from the root to the leaves of the quadtree using the Chord routing algorithms that facilitate efficient jumps from one peer to another peer on the P2P network. The traversal of the quadtree can be optimized by using caching of the peer addresses rather than solely relying on the Chord routing methods.

In order to avoid the single point of failure had all tree operations begun at the peer that stores the root control point, we introduced the concept of a fundamental minimum level, f_{min} , and also further balanced the load in the system. The concept of f_{min} forces objects to be stored and query processing to start at a level $l \geq f_{min}$. If the value of f_{min} is 0, then this is equivalent to using the standard MX-CIF quadtree. A fundamental maximum level f_{max} is also used to avoid objects being stored at a level greater than f_{max}. If the value of f_{min} and f_{max} are the same, then the index will look like a grid instead of a tree.

4. A P2P NEAREST NEIGHBOR ALGORITHM

Using our P2P quadtree index (Harwood and Tanin 2003; Tanin et al. 2004), we now present our priority queue based NN algorithm for P2P networks. We use the base concepts from Hjaltason and Samet 2003. Yet, a direct implementation of it has disadvantages on a P2P platform. Primarily, we do not need to run the algorithm in a sequential manner as we do not have to have a single thread of control in a P2P setting. For a networked system, the delays per contact to a peer and in a sequential manner can add up to a significant amount.

For P2P settings, in theory, we can send a message to all of the peers in the P2P network to find the NNs simultaneously. But realistically, if other peers take the same approach for their queries then we will be creating an all-to-all communication mechanism in our P2P network. The bandwidth required to send millions of messages from a single peer for a large system will also be problematic. Hence, for a system with millions of peers, sending messages to all the peers in the network is not a practical approach. So, the question is what is a reasonable amount of parallelism that we can harvest from the independent peers of a P2P network?

Our heuristic that aims at this objective is to maintain a query processing front of all those control points, hence blocks of the spatial data structure that are in the priority queue, that still have the *possibility* of returning a closer object than the current block that is at the top of the priority queue. So rather than a priority queue with a single point of entry, we maintain a front of multiple entries that are being processed in parallel. This heuristic attempts to maximize the parallelism that we can harvest on a P2P network from a single peer's point of view, while avoiding a single peer to send messages to many peers that would be redundant for a NN computation.

A NN query is first initiated on a single peer in the P2P network. This peer maintains the priority queue of quadtree blocks (mapping to a control point each) that are being processed for the query. To process a block, we have to contact from this query initiating peer to the peer that owns that block, i.e., the control point. Hence, in our parallel algorithm, we contact, rather than just the top entry of the priority queue, a multiple number of these peers. Assuming that $f_{min} = 0$, the query starts at the root and initially there is only the root block in the priority queue. Hence, we contact the peer that has the root block (control point) and wait for a response. In the case of our MX-CIF quadtree, this block may contain objects (the ones that lie on the subdivision lines for the blocks of the next level) and hence the closest one to the query point can be the first nearest object. So objects can be returned back to the query initiating peer. Also, the peer that is responsible for the root block knows how many objects the child blocks do have (see Tanin et al. 2004). This information is used to return the children that has objects and hence can contribute to the NN query. Next, the query initiating peer inserts these blocks into the priority queue along with any objects that are returned and proceeds with the next iteration of the algorithm.

In the next iteration, rather than contacting only one peer, all those blocks and hence the peers that maintain them that still have the chance of returning a closer neighbor have to be contacted. The decision for selecting which ones should be contacted is important and guides the behavior of the algorithm. Hence, in comparison to the original priority queue based algorithm by Hjaltason and Samet 2003, there is an additional criterion that controls the flow, namely the *Worst Case*. Abbreviated as WC, this criterion is used to ensure that the relevant peers that can still help finding a closer neighbor for the next nearest neighbor are contacted. This algorithm works, without any alterations, for even f_{min}

> 0, where there are many blocks in the queue as soon as the algorithm starts.

Our algorithm, assuming the current nearest neighbor is at distance m (i.e., the first spatial object in the priority queue), instead of removing elements from the priority queue one at a time for processing, computes the maximum distance MaxDist(q, t) at which an object can be found in the top element t of the priority queue and then processes all elements e in the priority queue whose distance Dist(q, e) from query point q is less than Min(MaxDist(q, t), m). This is the WC criterion. Hence, with this criterion we look at two pieces of information: i) the first spatial object in the priority queue that, in the worst case, can be the next NN (this object can be held in a separate buffer or a pointer to this object can be utilized for efficiency) ii) the maximum possible distance from the query point to an object in the top element of the priority queue.

Alternatives for the subcondition (ii) are stated for sequential algorithms in the literature for various spatial data structures and they can be easily used with our algorithm to parallelize the NN search. For example, an alternative (ii) can be, which might yield a tighter criterion when the elements in the priority queue are minimum bounding boxes, the maximum distance MaxNearestDist(q, t) at which a nearest neighbor of q can be found in the bounding box t of the top element in the priority queue. This metric is called the MinMaxDist by Roussopoulos et al. 1995 and MaxNearestDist by Samet 2003. In general, the difference between the methods of Roussopoulos et al. and Samet is that the former is only useful for depth-first nearest neighbor finding and for just one neighbor, whereas the latter is useful for both depth-first and best-first nearest neighbor finding for arbitrary values of k (i.e., k-th nearest neighbor).

When a NN query is initiated at a peer, the peer calls NNQuery(). This method, shown below, inserts all the blocks (control points) at the f_{min} level into the priority queue. In our prototype, the queue is implemented as a sorted linked list that also enables parallel access to the rest of the queue, rather than just to the top element of the queue. Other, more efficient, implementations of this in-memory data structure on the query initiating peer are also possible but probably unnecessary as the time spent on sending messages among the peers is several orders of magnitude more than the time for the in-memory operations. The order in the list is based on the distance of the quadtree block associated with each control point from the query point.

As we do not want to contact all the peers with these control points at once, the peer that the query point q lies in is contacted along with all the others within the initial WC distance to start the algorithm. In the beginning, this is equal to the maximum distance between the query point and the borders of the block that contains the control point that q lies in.

When a peer receives a NN query operation that was initiated on another peer, it calls DoNNQuery() which determines whether it has any objects and also checks whether any of the child control points have any objects. Finally a message is sent back to the query initiating peer containing the objects and the valid children, if any.

At the query initiating peer, ReceiveMessage() is used to handle messages that are returned by other peers in a mutually exclusive manner. If a peer returns objects or its child control points then, these are inserted into the sorted list corresponding to the priority queue and they are accessed in the next iteration. It is important to note that the messages from different peers can return in a sequence that is different than the original sequence and the algorithm would still work in the initially intended manner.

SendMessagesWithin() is the method used to contact all control points from whom we are not already waiting for a message and that fall within the new WC distance. Obviously, elements of the priority queue that are not control points/blocks but just spatial objects are returned to the user when they become the top element. We can wait for a user command to continue with the next iteration when a nearest object is found. This makes the algorithm a truly incremental ranking algorithm from the user's point of view.

UpdateWCDist() is the method used to update the distance of WC when a message is received. This is done by examining the current top element in the priority queue (and the first spatial object available in the queue if one exists).

```
NNQuery(Query_Point q) {
        PQueue = GetControlPoints(q, $f_{min}$)
        c = FindControlPoint(q, $f_{min}$)
        WCDist = MaxDist(q, c)
        SendMessagesWithin(WCDist)
}
DoNNQuery(Control_Point u)  {
        m = CreateReplyMessage()
        m.Put(u.GetLocalObjects())
        for each Child v of u do
                if (v.HasObjects()) then m.Put(v)
        SendMessageBack(m)
}
Synchronized ReceiveMessage(Message m) {
        for each Object o in m do
                PQueue.add(o)
        for each Control_Point u in m do
                PQueue.add(u)
        PQueue.remove(SenderOf(m))
        WCDist = UpdateWCDist()
        SendMessagesWithin(WCDist)
}
```

Figure 1 shows how a NN query proceeds using our algorithm. The dark dots in the figure represent the 16 f_{min} level control points for $f_{min} = 2$. Three objects X, Y, and Z are represented by shaded rectangles. According to the rules defining the MX-CIF quadtree and with $f_{min} = 2$, object X is stored at level 2 with control points – BA, BB, BC, and BD. Object Y is stored at level 2 with control point AD and object Z is stored at level 3 with control point DDA. The f_{min} level control point containing the query point q is CC. We show two WC distances that are computed by the algorithm. This is because the first control point cannot locate any objects and reports that its children also do not have any objects. In general, the WC criterion should present a shrinking behavior when new, smaller and closer blocks are investigated until an object is retrieved from the top of the queue. But it can also expand without requiring a change from our algorithm. This facilitates the continuation of the algorithm for ranking. But expansions for this criterion has an additional benefit for P2P systems. As we cannot lock the whole P2P quadtree for a single NN query, it is possible that while we traverse the quadtree, a delete request can remove the spatial objects for a block that has been previously inserted into the priority queue, creating a need for expansions in the WC distance.

5. ANALYSIS

In this section, we compare our approach with the sequential algorithm presented by Hjaltason and Samet 2003 for a P2P setting. To simplify the analysis, we assume that:

a) We have a perfect MX-CIF quadtree with height h, i.e., all leaves are at the same level,
b) The caching algorithm presented in Tanin et al. 2004 does not encounter any misses and hence, we do not need to use the Chord methods for control point look-ups, i.e., during the tree traversals,
c) The user runs the ranking process until completion,
d) Message passing is our main concern and in-memory operations are negligible,
e) Messages are small and hence latency, rather than message bandwidth is the main issue,
f) The latency is a constant to all the peers from any given query initiating peer,
g) $f_{min} = 0$ (and $f_{max} = h$),
h) There is only one spatial object per leaf and there are no spatial objects in the internal nodes,
i) The quadtree is formed over a square region, i.e., the bounding box of the data is a square.

We believe that this analysis provides an insight as to how the original and the new algorithms (for P2P settings) differ. Yet, for a real-life comparison and for observing the full benefits of the new algorithm, we need to run realistic experiments with our work (which is stated as future work on which we are currently focusing our efforts).

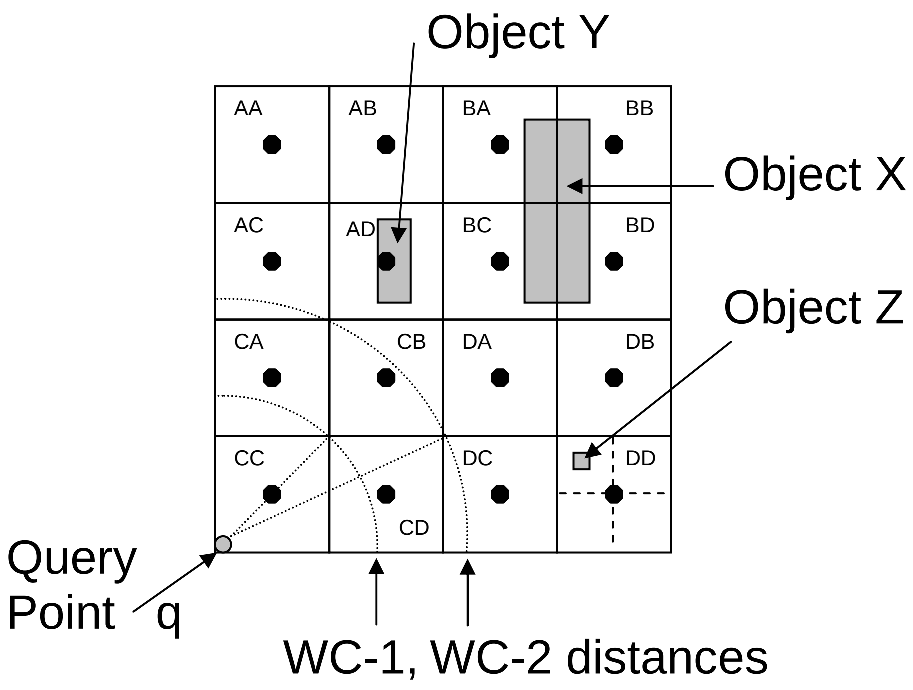


Figure 1. Spatial objects, control points, query point, and two WC fronts in a quadtree with $f_{min} = 2$.

Given our assumptions, the sequential version of the algorithm (if implemented directly on the P2P quadtree index presented in Tanin et al. 2004) will have to visit every node of the tree in a sequential manner to rank all the objects in the P2P system. Therefore this process has a complexity of $O(4^h)$.

For our parallel algorithm, the closest NN will be found in $O(h)$ while each level will be visited in parallel. The algorithm starts at the root and gradually focuses on one part of the tree while restricting the WC incrementally. Eventually we will reach to the leaves. The difference from the sequential algorithm is that at every step of the descent in the traversal of the tree, from one level to another, many nodes will be visited in parallel, and many intermediate nodes will have already been inserted into the priority queue. These nodes will be used later on for ranking, i.e., finding the remaining objects in the order of their distance to the query point. In fact, the blocks that contain the next few NNs, given our assumptions, may have already been visited while processing the current NN and the spatial objects that they contain may have already been inserted into the priority queue. When we process the objects and blocks that have not been visited, all other blocks within a WC will also be automatically visited, again in parallel. This, given our initial assumptions for a perfect quadtree, will lead to a wave of parallel expansions of the quadtree blocks from the query point. This wave moves discretely and there will be $O(4^{h/2})$ increments. This is because there are as many increments as there are number of leaves on one dimension of the quadtree. At each increment, we will need to process only a single level of the quadtree to reach to the neighboring leaves. Hence, the overall complexity of our algorithm will be $O(2^h)$. This is $O(2^h)$ faster than the sequential algorithm if it was directly implemented on the P2P network.

6. AN APPLICATION PROTOTYPE

We have implemented a two-dimensional prototype application to demonstrate our algorithm. The application facilitates insertion, deletion, range (or window queries as they are commonly called in spatial databases or in GIS), and NN queries for spatial data. It functions as a P2P lookup service for a virtual city. Users can find events and places of interest on this P2P application. The range queries are implemented in a similar manner as they are described in Harwood and Tanin 2003 where the P2P quadtree index was introduced. NN queries, the focus of this paper, are implemented using the NN algorithm presented above. The user interface of the application consists of a map for a city that comes with the application itself. We also have two text boxes and some control functionality, e.g., query buttons. The map is used to define rectangular objects (i.e., marking places in the city) to insert data into the P2P network (e.g., events and locations of interest). For each inserted object, a description text can also be entered into a text box. For each query, feedback is given through another text box. Only the original owner of an object can delete the object while everyone in the P2P network can query or insert objects. Screenshots of this P2P application are shown in Figures 2 and 3. Figure 2 shows all the inserted spatial objects as a result of a range query for an entire district in a city after zooming in (as it is seen by one peer in the network). To perform a NN query, the graphical user interface allows the user to select a query point by selecting a position on the map. Pressing the *Neighbor Query* button retrieves the first spatial object that is nearest to the query point. A screenshot of a NN query and a result is shown in Figure 3. Many other features of this application are not shown in this paper such as object insertions, retrieval of details for a resultant object, etc.

7. CONCLUSIONS AND FUTURE WORK

Performing NN queries on spatial data is an important operation. In many future P2P applications these queries will require efficient algorithms to run on distributed data structures. We have developed an efficient algorithm that facilitates NN queries on spatial data in P2P networks. The algorithm uses a P2P index that we developed in our earlier work to find the spatial objects that is nearest to a query point. Mainly by organizing a communication scheme for the peers that can or cannot aid in executing the current NN query, we restrict the number of messages passed between the peers of the network while still continuing to utilize the parallel processing power of the P2P network. At any iteration of the algorithm, we basically maintain a parallel front of viable nodes that can still contribute to the NN query and hence we adapt the priority queue based solution presented in Hjaltason and Samet 2003 for distributed environments. Our algorithm has been implemented in a prototype P2P application that provides a lookup service for a P2P virtual city. Although the implementation has been in a two-dimensional setting, it can be easily extended to higher dimensions. In addition to this, although it is implemented on a P2P quadtree index, the algorithm can easily accommodate a P2P R-tree index. We are currently in the process of experimenting with our work. A comparison of different versions of the algorithm using various different parallel front metrics and indices will be a fundamental contribution to the area of distributed computing and spatial data management.

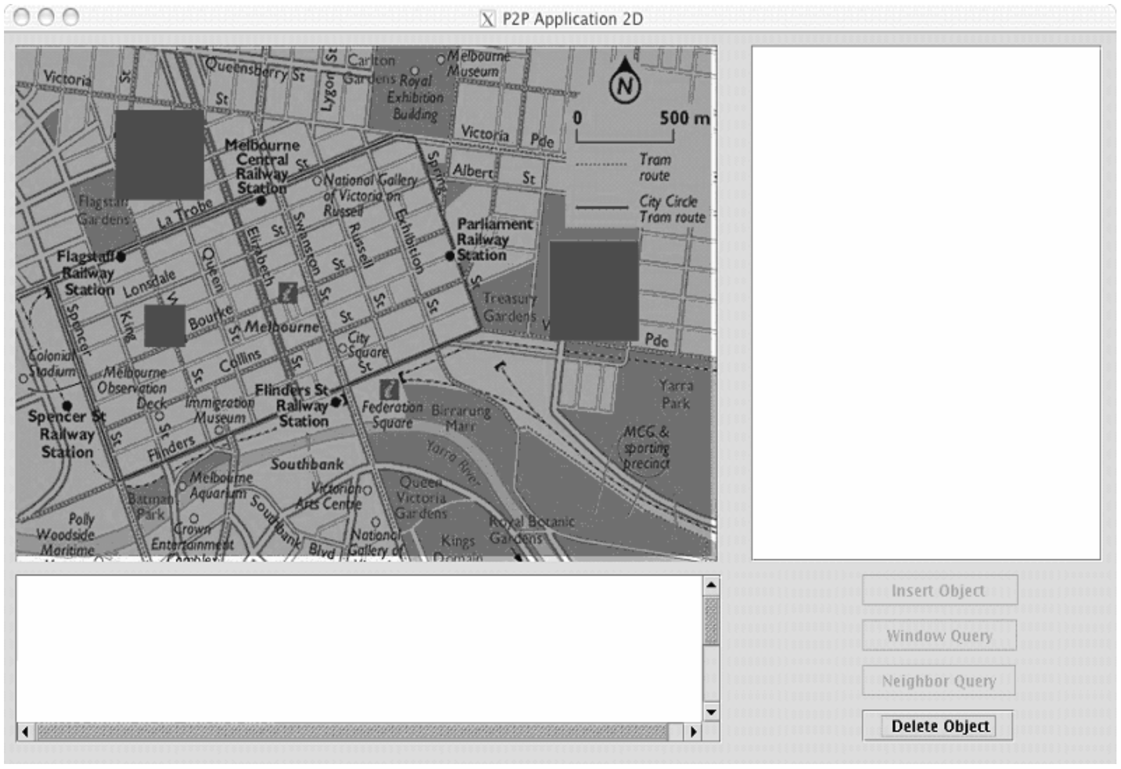
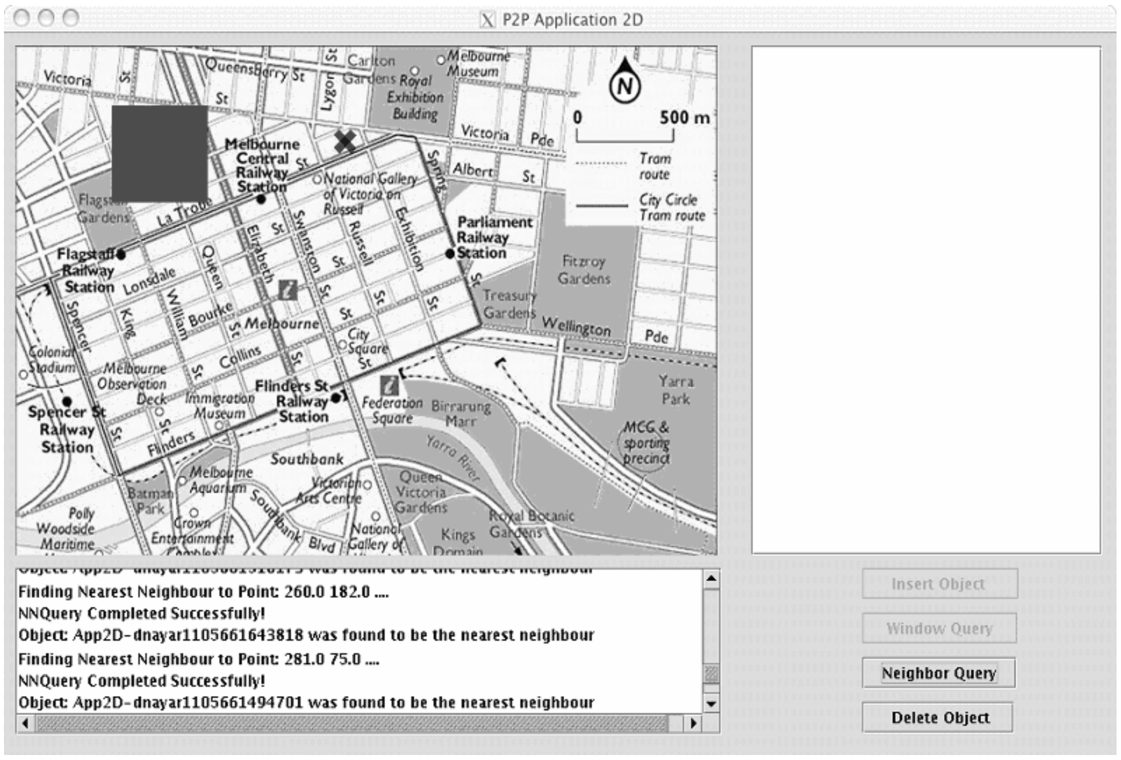


Figure 2. Screenshot of a range query for a region after zooming in to a certain district in a city; showing all existing spatial objects in the P2P network for that district (the three dark rectangles).



```
Finding Nearest Neighbour to Point: 260.0 182.0 ....
NNQuery Completed Successfully!
Object: App2D-dnayar1105661643818 was found to be the nearest neighbour
Finding Nearest Neighbour to Point: 281.0 75.0 ....
NNQuery Completed Successfully!
Object: App2D-dnayar1105661494701 was found to be the nearest neighbour
```

Figure 3. Screenshot of a NN query (the cross represents the query point and the rectangle represents the nearest spatial object; further clicks on the "Neighbor Query" button can be used to continue to rank other hits to a query).

8. REFERENCES

[1] F. Banaei-Kashani and C. Shahabi. *SWAM: A Family of Access Methods for Similarity-Search in Peer-to-Peer Data Networks*, Proceedings of the Conference on Information and Knowledge Management - ACM CIKM, Washington, DC, November 2004, pages 304-313.

[2] M. Batko, C. Gennaro, P. Savino, and P. Zezula, *Scalable Similarity Search in Metric Spaces*, Proceedings of the DELOS Workshop on Digital Library Architectures: Peer-to-Peer, Grid, and Service-Orientation, Edizioni Libreria Progetto, Padova, Italy, June 2004, pages 213-224.

[3] M. Batko, C. Gennaro, and P. Zezula, *A Scalable Nearest Neighbor Search in P2P Systems*, Proceedings of the 2nd International Workshop on Databases, Information Systems, and Peer-to-Peer Computing (held in conjuction with VLDB), Toronto, Canada, August 2004, pages 64-77.

[4] M. Demirbas and H. Ferhatosmanoglu, *Peer-to-Peer Spatial Queries in Sensor Networks*, Proceedings of the 3rd IEEE International Conference on Peer-to-Peer Computing, Linkoping, Sweden, September 2003, pages 32-39.

[5] Harwood and E. Tanin, *Hashing Spatial Content over Peer-to-Peer Networks*, Proceedings of the Australian Telecommunications, Networks, and Applications Conference - ATNAC, Melbourne, 2003.

[6] G. R. Hjaltason and H. Samet, *Index-Driven Similarity Search in Metric Spaces*, ACM Transactions on Database Systems, December 2003, Vol. 28, No. 4, pages 517–580.

[7] G. Kedem. *The Quad-CIF Tree: A Data Structure for Hierarchical Online Algorithms*, Proceedings of the 19th Design Automation Conference, Las Vegas, NV, June 1982, pages 352-357.

[8] Mondal, Yilifu, and M. Kitsuregawa, *P2PR-tree: An R-tree-based Spatial Index for Peer-to-Peer Environments,* Proceedings of the International Workshop on Peer-to-Peer Computing and Databases (held in conjunction with EDBT), Heraklion-Crete, Greece, March 2004, pages 516-525.

[9] N. Roussopoulos, S. Kelley, F. Vincent, *Nearest Neighbor Queries*, Proceedings of the ACM SIGMOD Conference, San Jose, CA, May 1995, pages 71-79.

[10] O. D. Sahin, F. Emekci, D. Agrawal, and A. E. Abbadi, *Content-Based Similarity Search over Peer-to-Peer Systems*, Proceedings of the International Workshop on Databases, Information Systems, and Peer-to-Peer Computing (held in conjuction with VLDB), Toronto, Canada, August 2004, pages 46-63.

[11] H. Samet. *Foundations of Multidimensional and Metric Data Structures*, Morgan-Kaufmann, San Francisco, CA, 2005.

[12] H. Samet, *Depth-First K-Nearest Neighbor Finding Using the MaxNearestDist Estimator*, Proceedings of the International Conference on Image Analysis and Processing, Mantova, Italy, September 2003, pages 486-491.

[13] H. Samet. *The Design and Analysis of Spatial Data Structures*, Addison-Wesley, Reading, MA, 1990.

[14] K. Sevcik and N. Koudas. *Filter Trees for Managing Spatial Data over a Range of Size Granularities*, Proceedings of the International Conference on Very Large Data Bases - VLDB, Mumbai, India, September 1996, pages 16-27.

[15] Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek and H. Balakrishnan, *Chord: A Scalable Peer-To-Peer Lookup Protocol for Internet Applications,* IEEE/ACM Transactions on Networking, February 2003, Vol. 11, No. 1, pages 17-32.

[16] E. Tanin, A. Harwood, H. Samet, S. Nutanong, and M. Truong, *A Serverless 3D World*, Proceedings of the Symposium on Advances in Geographic Information Systems - ACM GIS, November 2004, Arlington, VA, pages 157-165.