Trajectory Indexing Using Movement Constraints[*]

Dieter Pfoser

Research Academic Computer Technology Institute
Akteou 11 & Poulopoulou Str.
11851 Athens, Hellas
pfoser@cti.gr

Christian S. Jensen

Department of Computer Science
Aalborg University
9220 Aalborg Øst, Denmark
csj@cs.aau.dk

Abstract

With the proliferation of mobile computing, the ability to index efficiently the movements of mobile objects becomes important. Objects are typically seen as moving in two-dimensional *(x, y)* space, which means that their movements across time may be embedded in the three-dimensional *(x, y, t)* space. Further, the movements are typically represented as trajectories, sequences of connected line segments. In certain cases, movement is restricted; specifically, in this paper, we aim at exploiting that movements occur in transportation networks to reduce the dimensionality of the data. Briefly, the idea is to reduce movements to occur in one spatial dimension. As a consequence, the movement occurs in two-dimensional *(x,t)* space. The advantages of considering such lower-dimensional trajectories are that the overall size of the data is reduced and that lower-dimensional data is to be indexed. Since off-the-shelf database management systems typically do not offer higher-dimensional indexing, this reduction in dimensionality allows us to use existing DBMSes to store and index trajectories. Moreover, we argue that, given the right circumstances, indexing these dimensionality-reduced trajectories can be more efficient than using a three-dimensional index. A decisive factor here is the fractal dimension of the network—the lower, the more efficient is the proposed approach. This hypothesis is verified by an experimental study that incorporates trajectories stemming from real and synthetic road networks.

1 Introduction

We are currently experiencing rapid technological developments that promise widespread use of on-line mobile personal information appliances [25]. Industry analysts uniformly predict that mobile Internet terminals will significantly outnumber the desktop computers on the Internet.

This proliferation of devices offers companies the opportunity to provide a diverse range of e-services. It is essential to many services, termed location-enabled services, that they be sensitive to the users' changing locations. Location awareness is made possible by a combination of political developments, e.g., the de-scrambling of the GPS signals and the US E911 mandate, and the continued advances in positioning technologies. Prominent examples of location-based

[*] A short version of this paper appeared in the proceedings of the 11th ACM GIS Symposium, 2003.

services concern vehicle navigation, tracking, and monitoring, where the positions of air, sea, or land-based equipment such as airplanes, fishing boats and freighters, and cars and trucks are of interest.

This paper is concerned with the indexing of the movements of mobile objects for post-processing (e.g., data mining) purposes. The size and shape of an object is often fixed and of little importance – only its position matters. Thus, the problem becomes one of recording the position of a moving object across time. The movement of an object may then be represented by a trajectory, or polyline, in the three dimensional *(x, y, t)* space composed from two spatial dimensions and one time dimension [24].

Depending on the particular objects and applications under consideration, the movements of the objects may be subject to constraints. Specifically, we may distinguish among three *movement scenarios*, namely *unconstrained movement* (vessels at sea), *constrained movement* (pedestrians), and *movement in transportation networks* (trains and, typically, cars) [22]. The latter scenario, which we will assume, occurs when the applications at hand are interested in the positions of the objects with respect to the transportation network. For example, we may expect that many applications will be interested only in the positions of cars with respect to the road network, rather than in their absolute coordinates [14]. The movement effectively occurs in a different space than for the first two scenarios.

When faced with a new type of data such as the movements of network-constrained objects, how to efficiently process queries against these data is an important challenge. It is instructive to consider three complementary approaches. First, we may attempt to use existing access methods, but this may either not be possible or may in itself not be efficient. Second, we may invent new access methods. These allow for more efficient indexing, but their integration into a database management system may be complex. Third, we may attempt to "model" or transform the data such that existing methods can be reused. In the context of trajectories, the first approach implies the use of, e.g., a three-dimensional R-tree to index the data. Adopting the second approach, we would develop a new trajectory index, e.g., [19] [24]. With the third approach, we might apply a transformation to the data, such as the duality transformation [15]. The second approach might be desirable in the long term, but may not be attractive in a short or medium term. For example, it took a dozen years before the R-tree found its way into some commercial database products [20]. Thus, depending on the type of data, it can prove beneficial to attempt to reuse existing access methods by applying transformations to the data. This allows for easy integration of the new type of data into commercial database systems.

In this work, we lower the dimensionality of the trajectories by exploiting that the objects are constrained to a transportation network. This allows for a simplified approach to the indexing of trajectories. Our approach thus combines the first and third approaches from above. Often, dimensionality reduction is used to discover redundant dimensions in general-purpose data to avoid indexing some dimensions. In contrast, we translate the data from 3D space to a lower-dimensional space consisting of one space and one time dimension. The data representation changes. The efficiency of this translation is measured in terms of query performance by comparing the number of I/O operations required by queries when using the two representations. Using the notion of fractal dimension to measure the intrinsic dimensionality of the movement space, we hypothesize that the lower the fractal dimensionality is, the more efficient is query processing using the translation.

Dimensionality reduction is a well-known technique used to obtain lower dimensionality for indexing purposes. As in spatial databases, a query is processed using, first, an index on a "reduced" dataset, which yields a result with perfect recall. False drops are then eliminated in a subsequent refinement step, thus obtaining perfect precision. Dimensionality can be reduced in several ways. One is to eliminate dependent attributes, which may be identified by measuring the various fractal dimensions of the dataset [33]. Another technique, termed Global/Local Dimensionality Reduction [5] [8], tries to map the data to points in a lower-dimensional space while preserving distances and the overall structure of the data set. These approaches are based on pattern recognition techniques, e.g., Multidimensional Scaling. In this paper, we use fractal dimensions to predict the query performance of our proposal. Previously, fractals and fractal dimensions were used in predicting the query performance for range queries [7], spatial join queries [2] [9], and nearest-neighbor queries [21].

Past work in indexing of spatiotemporal data concerns either past data or present and future data [6]. The latter direction includes works on indexing the present positions of moving points, e.g., [1] [15] [22] [27] [28] [30] [32]. Within the former direction, to which the present work also belongs, most approaches deal with spatial data changing discretely over time, e.g., overlapping Quad-trees [34] and R-tree variants for spatial data [19]. A method that takes continuous changes into account and especially aims at processing spatiotemporal queries is the TB-tree [24]. The MV3R-tree [31] focuses on indexing the past locations of moving shapes. The method combines multi-version B-trees and 3D R-trees. The work presented in [13] [15] aims at reducing the dead space introduced by approximating the trajectory data in the index. Further, [16] [26] compare the indexing of trajectories in native space vs. parametric space. The SETI index [4] proposes the indexing of trajectories based on a grid-based subdivision of the occupied space and assigning a temporal index to each grid cell. Although the method can be readily implemented using simple spatial indexes the grid spacing needs to be adjusted to a specific data set as well as the maintenance cost of the set of temporal indexes is considerable. A method similar to the SETI approach, by at the same time exploiting the network aspect of the movement is proposed in [10]. A temporal index is assigned to every network edge. Although the method shows competitive performance to other trajectory indexes, the costly maintenance of the numerous index structures makes it impractical.

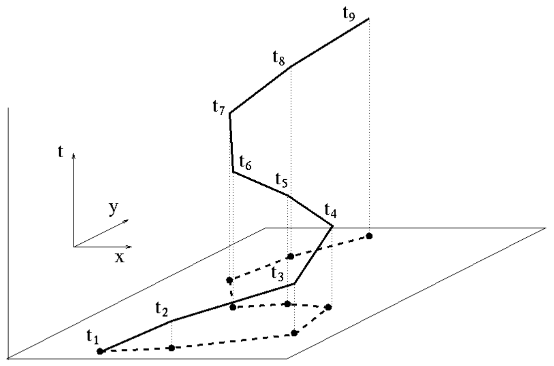
The outline of the paper is as follows. Section 2 introduces trajectory data and the related queries and indexing problems. Section 3 offers algorithms for dimensionality reduction. Section 4 discusses the applicability of the approach by relating the intrinsic dimensionality of the movement space to the efficiency of query performance. Section 5 gives an empirical evaluation by comparing the query performance of trajectory data stemming from a variety of networks indexed by an R-tree in three vs. two-dimensions. Finally, Section 6 concludes and offers research directions.

2 The Trajectory Case

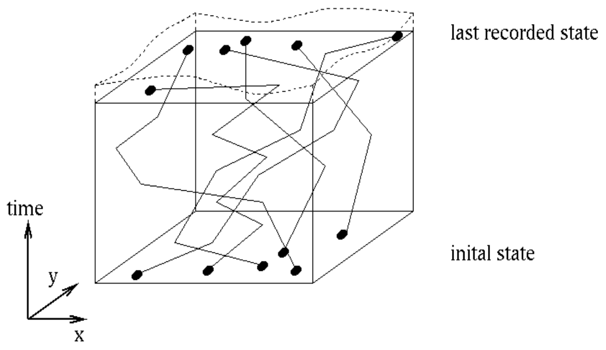
The central question in this work is how we can *ease the task of indexing point-object movements*. This section gives a brief description of the data and the related queries and access methods.

2.1 Trajectories

When sampling the position of a moving point across time, we obtain a trajectory. Consider the following application context. Optimizing transportation, especially in highly populated and thus congested areas, is a very challenging task that may be supported by an information system. A core application in this context is fleet management. Vehicles equipped with GPS receivers transmit their positions to a central computer using either radio communication links or mobile phones. At the central site, the data is processed and utilized. To record the movement of an object, we would have to know the position at all times, i.e., on a continuous basis. However, GPS and telecommunications technologies



(a)



(b)

Figure 1: Trajectories of moving point objects in 2+1 dimensional space

only allow us to sample an object's position, i.e., to obtain the position at discrete instances of time, such as every few seconds. A first approach to represent the movements of objects would be to simply store the position samples. This would mean that we could not answer queries about the objects' movements at times in-between those of the sampled positions. Rather, to obtain the entire movement, we have to interpolate. The simplest approach is to use *linear interpolation*, as opposed to other methods such as polynomial splines. The sampled positions then become the endpoints of line segments of polylines, and the movement of an object is represented by an entire *polyline in 3D space*. The solid line in Figure 1(a) represents the movement of an object. Space and time axes are combined to form a single coordinate system. The dashed line shows the projection of the movement into the 2D plane [23]. Figure 1(b) illustrates the spatiotemporal workspace (the cube in solid lines) and several trajectories (the solid polylines). Time moves in the upward direction. The wavy-dotted lines at the top symbolize the growth of the cube with time.

Semantically, the temporal dimension is different from the spatial dimensions. In classical spatial databases, only position information is available. In our case, however, we have also *derived information*, e.g., speed, acceleration, traveled distance, etc. Information is derived from the combination of spatial and temporal data. Further, we do not just index collections of line segments – these are parts of larger, semantically meaningful objects, namely trajectories. These semantic properties of the data are reflected in the types of queries that are of interest.

Such queries can be conceived as adapted spatial queries, e.g., range queries of the form "*find all objects within a given area at some time during a given time interval*" and queries with no spatial counterpart. Here, the so-called trajectory-based queries are classified in "*topological*" queries, which involve the entire movement of an object (*enter*, *leave*, *cross*, and *bypass)*, and "*navigational*" queries, which involve derived information, such as speed and heading. In [24], these queries, including combined queries, are discussed in greater detail.

2.2 Indexing Trajectories

Trajectories are three-dimensional spatial entities, and they can be indexed using spatial access methods. However, there are difficulties. Trajectories are decomposed into their constituent *line segments*, which are then indexed. The use of the R-tree [12] is illustrated in Figure 2. The R-tree approximates the data objects by Minimum Bounding Boxes (MBBs), here *three*-dimensional intervals. Approximating segments using MBBs proves to be inefficient. Figure 2 shows that we introduce *large amounts of "dead space*." It is evident that the MBBs cover large parts of space, whereas the actual space occupied by the trajectory is small. This leads to substantial overlaps between index entries and consequently to a small discrimination capability of the index structure. (Note that approximating trajectories with MBBs without prior decomposition into segments just makes things worse.)

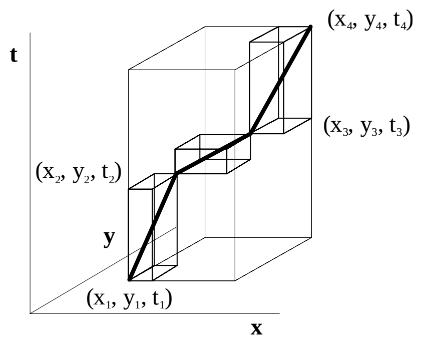


Figure 2: Approximating trajectories using MBBs

Other trajectory indexing problems include *trajectory preservation* and *skewed data growth*. As for the first problem, spatial indices cluster segments according to spatial proximity. However, in the case of trajectories, it is beneficial to some queries if the index preserves trajectories, i.e., clusters segments according to their trajectory and only then according to proximity (cf. [24]). The second problem refers to the fact that trajectory data grows mostly in the temporal dimension. The extents of the spatial dimensions are fixed, e.g., they are given by the city limits. Exploiting this property of the data can increase query performance. However, these problems are beyond the scope of this paper and are either treated elsewhere [24] or are subject to future work.

3 Reducing Dimensionality

The space as defined by a network is quite different from the Euclidean space the network is embedded into; intuitively, its dimensionality is lower. In the literature, the term *1.5 dimensional* has been used for a network embedded in 2D Euclidean space.

Given a network such as that shown in Figure 5(a), a mapping algorithm takes the edges of the network, the movement space, and transforms them into intervals of a one-dimensional space. When trajectories are constrained to such networks, e.g., cars move on roads, one can exploit this mapping to reduce the dimensionality of the trajectories. We translate 3D trajectories into two dimensions. This consequently reduces the indexing challenge in that a 2D spatial index suffices (cf. Figure 5(b)). Overall, we have to devise mappings for (i) *the network*, (ii) *the trajectories*, and (iii) *the queries*.

3.1 Mapping Networks and Trajectories

Mapping the network is a precursor to the mapping of the trajectories and the queries. Figure 3 proposes a simple algorithm that takes the network as its input[†]. In a first step, all network edges are sorted according to the Hilbert value

[†] We assume that the network is static. For this simple algorithm, a change in the network leads to recomputation.

of the center point of the edge. Figure 5(a) illustrates this step. A simple Hilbert curve is drawn on top of the network to illustrate the idea of sorting the edges accordingly.

```
Algorithm NetworkMapping (network)
LOCALS   range  //highest coordinate
         low    //lower coordinate of edge in 1D space
         up     //upper coordinate of edge in 1D space
NM1      sort edges by their Hilbert value, cf. Figure 5(a)
         FOR ALL edges
NM2          compute length of edge
NM3          low = range + 1
NM4          up = range + 1 + length
NM5          write edge(low, up)
NM6          range = up
         END FOR
```

Figure 3: Network mapping pseudocode

In a second step, all the edges are mapped sequentially according to their ordering to sub-intervals of a 1D interval. The first edge becomes the first sub-interval, which starts where the 1D interval starts and extends a distance that corresponds to its distance in the network. The second edge then starts where the first ends, etc. The end of the sub-interval corresponding to the last edge is the end of the 1D interval. More specifically, assuming we are dealing with integer coordinates, we map the start of the interval for a new edge to the end of the interval for the previous edge plus 1. This is done to reflect that the end point of one edge is not necessarily the start of the next edge in the ordering.

Having mapped the movement space, we must map the trajectories from the original space to the new space. The general idea behind the algorithm for this, as outlined in Figure 4, is that given a trajectory segment, such as segment 1 in Figure 5(b), we have to identify the network edge on which the movement occurred (TM1) as well as how much of this edge the object traversed (TM2). Then we find the respective portion of the edge in the 1D representation (TM3), and we record the respective coordinates (TM4). Figure 5(b) gives an example mapping of a trajectory consisting of four segments. Note that the temporal extent of the segment remains unaffected.

```
Algorithm TrajectoryMapping (trajectory, 2Dnetwork, 1Dnetwork)
        FOR ALL segments of the trajectory
TM1         find traversed network edge in 2Dnetwork
TM2         determine traversed portion of edge in 2Dnetwork
TM3         x0, x1 = respective 1Dnetwork coordinates
TM4         write segment(x0, t0, x1, t1)
        END FOR
```

Figure 4: Trajectory mapping pseudocode

In the pseudo code and its explanation, we assume that a trajectory segment does not extend beyond one network edge – whenever an object moves from one edge to another, a new segment is created in its trajectory. This simplifies TM1 through TM3. Further, together with a trajectory segment, we store an id that refers to the traversed network edge. This simplifies the lookup of the trajectory in TM1.

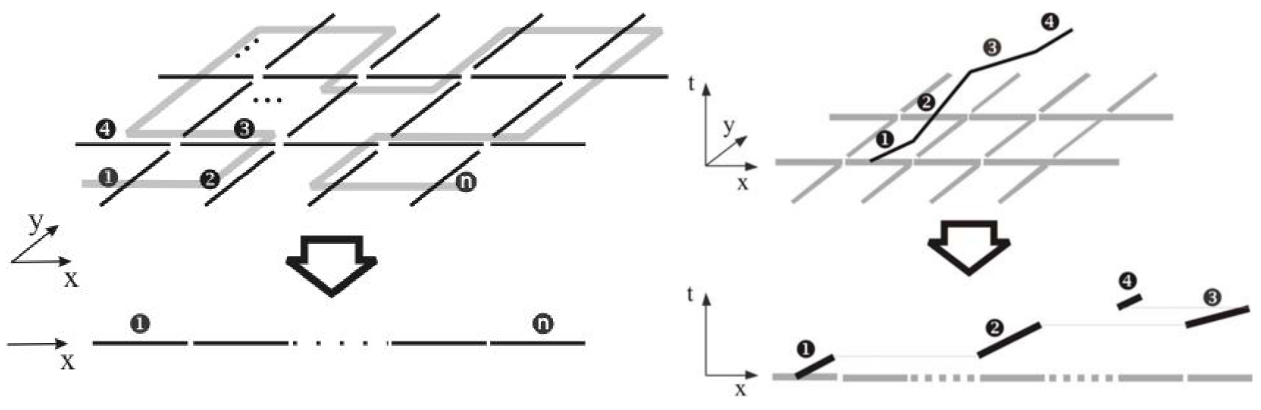


Figure 5: Mapping a (a) two-dimensional network and (b) a trajectory

3.2 Mapping Queries

After having transformed the data, the final step is to transform the queries. Although we mention several types of spatiotemporal queries in Section 2, we restrict ourselves to spatiotemporal range queries, i.e., given a spatiotemporal range, return the trajectory segments that intersect (and are contained in) it.

Assume that we want to process a range query against a database of trajectories. The question is then how we represent this query in the transformed space. Figure 6 illustrates the overall approach. The left-most picture illustrates a spatiotemporal range query in the untransformed space with a single trajectory intersecting it.

We eliminate the temporal extent of the query and apply the resulting query to an index that indexes the edges of the network. The result comprises all the edges that correspond to network portions of interest to the query. The result in the example is seven edges or intervals of edges. These sub-edges are then given the temporal extent of the original query. Something without temporal extent is “lifted” into the time dimension (cf. [11]). This is illustrated in the middle of Figure 6.

The final step is to map these 2D rectangles to the 2D space where the data resides. Information from the mapping of the network edges (cf. Section 3.1) is used to give the rectangles the correct displacement in the spatial dimension. We now have a set of 2D rectangle queries for the transformed data that match the data transformation, meaning that they retrieve the same trajectory segments from the transformed data representation, as does the original spatiotemporal range query on the untransformed data. In the example, query windows 1, 4, and 5 intersect the trajectory. Figure 7

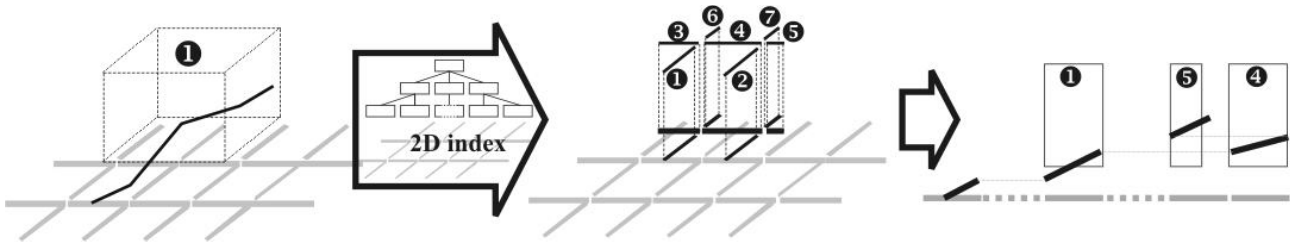


Figure 6: Mapping a range query window

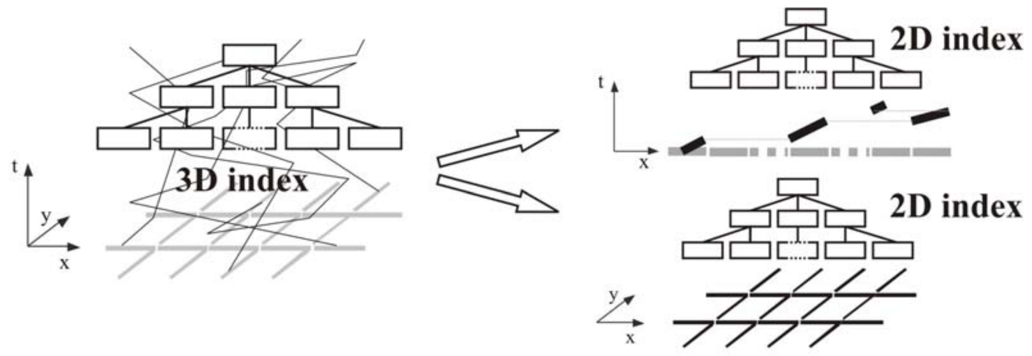


Figure 8: Replacing one 3D index with two 2D indices

provides a brief outline of this algorithm.

```
Algorithm QueryMapping(query, 2Dnetwork)
        //2Dnetwork access using an R-tree structure
QM1     given a query window, take the spatial extent and retrieve the portion of the 2Dnetwork
        contained in it
QM2     lift the retrieved edges by the temporal extent of the query window
```

Figure 7: Query-window mapping pseudo code

3.3 Summary

The original proposal was one of indexing 3D line segments using an index such as the R-tree. This causes problems in practice because commercial database management systems lack support for appropriate indexes. For example, although Oracle 9i provides 3D spatial data types, the spatial index support is limited for dimensions higher than two in that the number of permitted spatial operations is reduced to one [20].

The transformation approach replaces the single 3D index with trajectories with two 2D indexes (cf. Figure 8). One contains the network (i.e., it has two spatial dimensions), and the other contains the transformed trajectory data (i.e., it has one space and one time dimension). Thus, a higher-dimensional problem has been reduced to a lower-dimensional one.

The overall question remains, whether this mapping pays off in terms of more efficient query processing. We have reduced the dimensionality of the trajectory data at the cost of an additional index structure for the network and more complex, transformed queries. A brief cost-benefit analysis might look as follows. Query performance should *benefit* from the reduced size of the trajectory index. It is reduced by roughly a third since an index node contains four, not six, coordinates per segment. Mapping trajectories from three to two-dimensional space could reduce the dead space in the index. Dead space is defined to be the extra portion of the embedding space covered by the approximating bounding box with respect to the space covered by the original object [18]. Given a segment of length *b* approximated in three-dimensional space by a MBB of side length *a*, its dead space in 3D is $a^3 - b$ (assuming the segment has "thickness" 1), whereas in the 2D case, the dead space is $\sqrt{2}a^2 - b$. However, although smaller in absolute terms, one has to carefully evaluate this reduction in relation to the queries and their mapping to lower-dimensional space.

The occurrence of potentially many transformed query windows introduces *additional cost*, since the original query window may range over many network edges. We also need a second index for the network itself. However, this index is small and static. Thus, an optimized index structure can minimize the cost [17]. Overall, should this approach prove beneficial, it would allow us to use a simple index structure and, thus, off-the-shelf database management systems and their two-dimensional access methods.

4 Networks, Trajectories and their Fractal Dimensions

We have argued that reducing the dimensionality of our dataset might improve the query-processing performance. Our hypothesis is that this is not equally true for all types of networks. In the following, we introduce a measure for the complexity of a road network structure as a means to predict the performance of the mapping approach.

The overall cost of the mapping approach is largely determined by the number of query windows that are produced by the algorithm QueryMapping in Section 3.2 that uses the network to transform a query window by querying the edges contained in it. We argue that a network of lower complexity tends to produce fewer query windows in relation to a network of higher complexity. In the following, we define the fractal dimension of a network as a measure for its complexity of a network and give cost formulas to predict the query processing cost.

4.1 Fractals and Fractal Dimension, or Space, the not always Final Frontier

We use the notion of fractal dimension to quantify the inherent dimensionality of a road network. Generally, the fractal dimension of a network is between 1 and 2 (it is neither a line segment nor a surface).

One measure of *fractal dimension* is the Hausdorff dimension D_H ,

$$D_H = \lim_{r \to 0} \frac{\log N(r)}{\log(1/r)} \tag{1}$$

Equation 1 is based on an approach to approximate the fractal curve by using *N*(*r*) line segments (or disks for surfaces) of ever-decreasing length (diameter) *r*. Length *r* signifies the scale, or amount of detail, we expose the fractal to. Conceptually, as *r* approaches zero, we enter finer and finer wiggles of the fractal [29].

Taking a look at the fractal curve of Figure 9, showing a Hilbert curve at various levels of detail (decreasing *r*), or, conversely, at various stages of its construction process, we can observe that at the *n*'th construction step, the Hilbert curve consists of $N(r) = 2^{2n} - 1$ segments of length $r = 1/2^n$. Thus, it has Hausdorff Dimension $D_H = 2$. It is a "true" space-filling curve. Other examples of fractals include the Koch curve with $D_H = 1.26$ and the Sierpinski Gasket, a

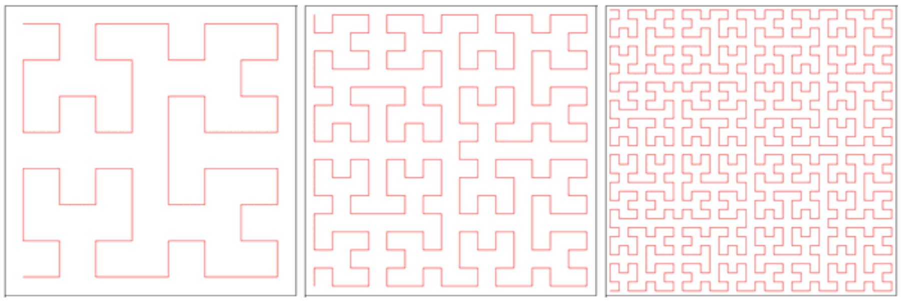

(a) (b) (c)

Figure 9: The Hilbert space-filling curve at various levels of detail or steps in its construction process

fractal point set with $D_H = 1.58$ [29].

In practice, the fractal dimension is measured using the *box-counting method* [29]. Superimposing a grid of spacing *r* over the fractal, we count the number of boxes (cells) *N(r)* that intersect the fractal.

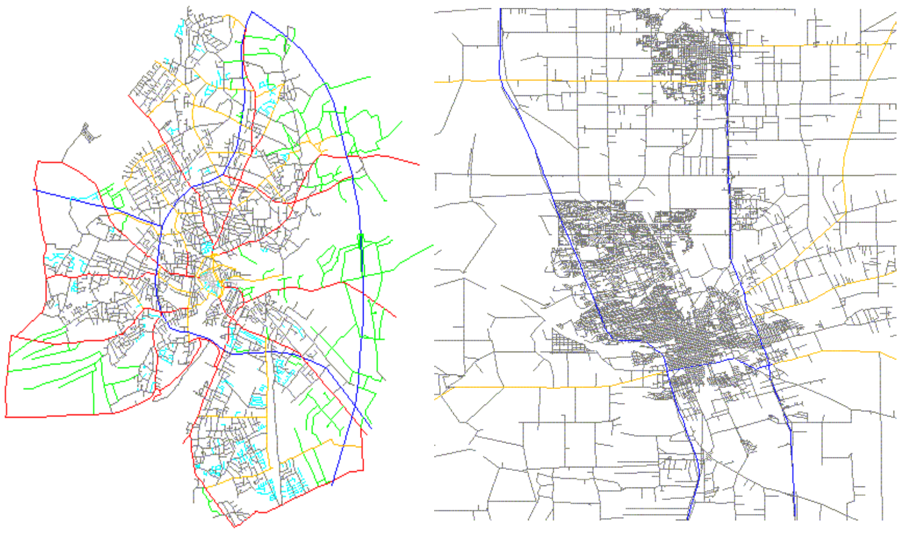
For practical ("finite") fractals, we can restate the above formula to return D_H for a given range $r \in (r_1, r_2)$ in which we assume the fractal is self-similar instead of using the limit $r \to 0$ [2].

$$D_H = \frac{\partial\left(\log N(r)\right)}{\partial \log\left(1/r\right)} \tag{2}$$

Plotting the numerator of Equation 2, $\log N(r)$, in terms of the denominator, $\log(1/r)$, we obtain what is known as a *box-count plot*. If the practical fractal is self-similar for $r \in (r_1, r_2)$, its box-count plot will be a straight line for this range. The slope of this line is the Hausdorff fractal dimension.

The networks in our application context are examples of such practical fractals. Figure 10 shows the road networks of Oldenburg, a city in Germany, and San Jose, California. Using a box-counting algorithm as described in [33], the respective fractal dimensions are as indicated in the figure.

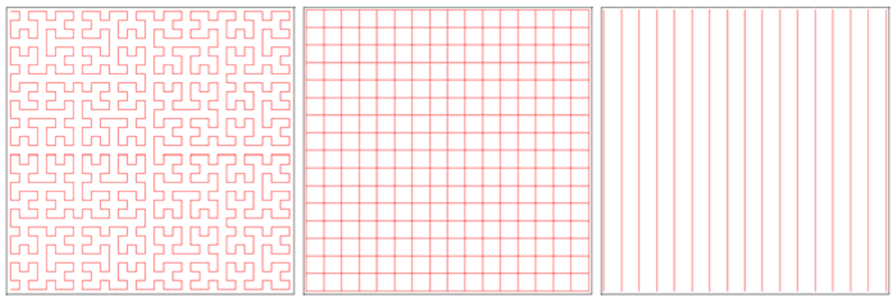
In the performance study described in Section 5, we will use the three additional synthetic fractals shown in Figure 11. The first network is based on the Hilbert curve. Its fractal dimension is 2. The dimensionality of the Hilbert curve used in our experiments measured using a box-counting algorithm yields 1.98. Figure 11(b) shows a Raster network similar to the road networks found in some US cities. This curve is similar to the Hilbert curve in that the *n*'th step in its construction process consists of $N = 2^{2n} - 2^n$ segments of length $r = 1/2^n$. Its fractal dimension is thus 2. The measured dimension for a curve used in the experiments is 1.95. Finally, Figure 11(c) shows a Parallel network. This curve is the network equivalent of a smooth curve. Thus, in keeping the number of segments finite, this curve has a fractal



$D_H = 1.64$ (a)

$D_H = 1.72$ (b)

Figure 10: Practical fractals: (a) Oldenburg, Germany and (b) San Jose, CA road networks [3]



$D_H = 1.98\ (2)$ $D_H = 1.95\ (2)$ $D_H = 0.98\ (1)$

(a) (b) (c)

Figure 11: Synthetic (fractal) networks and their measured and, in parenthesis, actual fractal dimension: (a) a Hilbert curve, (b) a Raster network, and (c) a Parallel network

dimension of 1. The box-count algorithm computed 0.98 for the curve used in the experiments.

4.2 Query Windows and Fractal Dimension

One objective of this work is to determine how the fractal dimension of the network that constrains the object movement correlates with query performance when using our mapping approach.

Figure 9 shows a Hilbert curve at various stages of its construction process. Considering the self-similar nature of fractals, another interpretation of Figure 9(a) and (b) is that they are merely *enlarged* sections of image (c). Figure 12 shows that (b) is the upper right quarter of (c), and (a) is the upper right quarter of (b). In the spirit of Section 3.2, on the mapping of queries, assuming that the side length of the total fractal is 4*s*, we need query windows of side length *s* and 2*s* to "query" the scaled versions of the fractals corresponding to Figure 9(a) and (b), respectively. We can state the following. *Subsequent steps in the construction process relate to doubling the extents of a query window ranging over the fractal.*

To see why this argument is important, recall Section 3.2, where we map one 3D range query to a set of 2D range queries. The *number* of resulting query windows directly determines the query processing cost in the two-dimensional space – the larger the number of queries, the higher the query processing cost. Thus, it would be interesting to know the number of 2D queries in advance, or at least to know it in relative terms. Put differently, given a number of queries for a query window A, we would like to know how many queries we can expect for a query window B that has double side lengths, or extents, in comparison to A (quadrupled in area).

In terms of edges contained in a query window, doubling its extent, according to the above argument, relates to making another step in the fractal construction process. By transforming Equation 2 we can state the following equation that may be used for determining the number of edges *N*(*r*) found for a given level of detail determined by *r*.

$$\partial \log N(r) = D_H \partial \log(1/r) \tag{3}$$

Substituting now $1/r$ with the query window extent *s* (the number of edges is inversely proportional to *r*, but directly proportional to *s*), the following power law follows from Equation 3.

$$\partial N(s) = \partial s^{D_H} \tag{4}$$

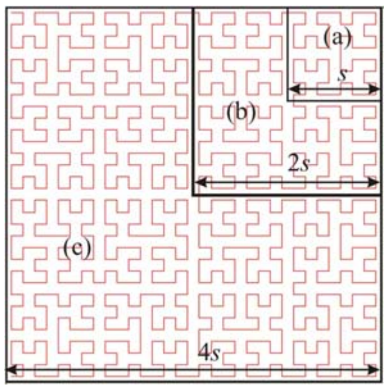


Figure 12: Self-similarity: parts of fractals and query windows

Given the example of a Hilbert curve with $D_H = 2$, we may expect four times the number of edges by doubling the size of a query window. Note that Equation 4 is invariant to scaling. Thus, we only get an estimate for the number of edges, but not for their lengths.

The algorithm QueryMapping in Section 3.2 uses the network to transform a query window by querying the edges contained in it. Thus, the number of transformed query windows is governed by Equation 4. In the following section, we empirically evaluate the above results.

5 Performance Studies

The objective of the following study is twofold. First, we want to validate, if this is indeed possible, the fractal-dimension based formulas from Section 4.2. Second, we want to gain insight into in which situations the mapping approach is useful. To achieve this objective, we compare the I/O costs of processing spatiotemporal range queries in the native and transformed spaces. We use trajectory data generated based on three synthetic and two real world networks.

The index structure used for the three- and two-dimensional trajectory data as well as the network data is an R-tree [12] implementation in the C programming language. In the experiments, the page size of leaf and non-leaf nodes is 1024 bytes, which results in maximum fanouts of 36 for 3D indices and 51 for 2D indices. Larger node sizes do not change the ratio between these fanouts.

5.1 Datasets and Queries

While several real spatial datasets are available for experimentation purposes (e.g., the TIGER-Line files of geographic features, such as roads, rivers, lakes, and boundaries, covering the entire United States), accessible and widely accepted, real moving-object datasets are missing. Due to the lack of real data, our performance study uses synthetic datasets. We utilize a network-based data generator [3] to create trajectories of moving objects. In particular, we use three *synthetic networks* of varying complexity and of similar lengths (the sum of the lengths of all edges), a Hilbert network, “h,” a Raster network, “r2,” and a Parallel network “p” (cf. Figure 11 – note that the numbers of edges were kept low for illustration purposes). The networks comprise 1023, 544, and 33 edges, respectively. To illustrate the impact of a varying number of edges, we use the networks “r1,” “r2,” “r3,” and “r4” representing sequential steps in the recursive construction process of a Raster network and comprising 144, 544, 2112, and 8320 edges, respectively. In addition, we use the two *real-world road networks* of San Jose and Oldenburg comprising 24123 and 7035 edges, respectively (cf. Figure 10). The network labels given in quotation marks are used in the figures of the performance study. Identical labels denote identical networks. Unless stated otherwise, the following parameters apply. For each of the networks, we generate trajectory datasets for 500 moving objects, whose positions are sampled 250 times each. Thus, each dataset should consist of 125k trajectory segments each. Given this amount of data and the parameters of the index structure from above, the sizes of the 2D and 3D segment indexes are typically 2.5MB and 3.35MB, respectively. In each experiment, we use sets of 500 quadratic query windows, each with spatial extents of 0.25%, 0.5%, 1%, 2%, 4%, and 8% of the extent of the quadratic 2D space that the data and networks are embedded into. The temporal extent of the query was kept constant at 10% of the respective data space. In our experiments, we found that varying the temporal extent of the query does not affect the query processing cost significantly when comparing the 2D to the 3D approach. This is plausible, since varying the temporal extent of a query does not affect the number of query windows produced in the mapping approach (cf. Section 3.2).

5.2 Network Complexity

This set of experiments aims at validating the fractal-dimension-based formulas of Section 4.2. As stipulated, the query processing cost in the mapping approach is largely determined by the number of transformed query windows, which, in turn, is governed by the fractal dimension of the network. Equation 4 states that *the higher the fractal dimension* of a network is, the *higher* the relative number of transformed query windows and thus the associated *query processing cost* is.

In Figure 13 to Figure 15 this claim is empirically verified. For each case of a network, Figure 13(a) to Figure 15(a), we determine the factor by which (i) the number of query windows and (ii) the resulting I/O cost for processing these queries increases by *each time doubling the query window size*. For example, using a query window extent of 4% as the basis and doubling it to 8%, the number of transformed query windows is increased 2, 2.8, and 3 times for the Parallel, the Raster, and the Hilbert network, respectively (cf. Figure 13(a), query window extent 8%). Consequently, the I/O cost for processing these queries increases by 2.3, 2.9, and 3 times.

Using Equation 4 and the fractal dimensions of Figure 11 (1, 2, and 2), the number of query windows should be increased by 2, 4, and 4 times, respectively. This difference between measured and empirically determined costs may be explained by the query windows not covering the whole network uniformly. Especially for small query windows in "sparse" networks, the numbers may be biased, i.e., the number of retrieved network edges is not representative. Thus, by increasing the query window size or, alternatively, by making the network "denser" and increasing the number of edges such as in the experiment shown in Figure 14(a), the values are more reliable and converge towards their theoretical bounds. In Figure 14(a), for the Raster network with the largest number of edges, "r4," the I/O cost is increased 3.5 times when increasing the query window size from 4% to 8% (query window extent 8%), which is closer to its theoretical value of 4. Doubling a smaller query window, e.g., from 0.5% to 1% results only in a 1.8 time increase. Figure 15(a) shows similar results for the two real-world networks. Using Equation 4 and the measured fractal dimensions of Figure 10 (1.64 and 1.72), the number of query windows should be increased by 3.12 and 3.3 times for the Oldenburg and the San Jose network, respectively. In our experiments, by using a query window extent of 4% as basis; and doubling it to 8%, the number of query windows is increased by 3.5 and 4 times.

An alternative interpretation for the number of transformed query windows is presented in Figure 13(b) to Figure 15(b). The three figures show a modified box-count plot based on the size of the query window and the number of retrieved edges and thus transformed queries (cf. Section 3.2). Our box-count plot differs from the respective "classical" plot in that we replace a grid of varying spacing with a set of query windows of varying size. In contrast to the grid, the set of query windows covers portions of the network redundantly and does not guarantee its complete coverage. The slopes of these curves correspond to the Hausdorff fractal dimensions of the networks (indicated in parenthesis in the figures).

The plot of Figure 13(b) gives a clear result for the Parallel network with a $D_H = 1$, the values for the Raster and the Hilbert network are 1.46 and 1.45, respectively, as opposed to their theoretical value of 2. The box-count plot of Figure 14(b) for the "r2, …, r4" networks reveals that the measured fractal dimension converges to 2 at higher constructions steps. For "r4," the fractal dimension is measured as 1.72 as opposed to its theoretical value of 2. The plot of Figure 15(b) shows Hausdorff dimensions of $D_H = 1.82$ and 1.99 for the Oldenburg and San Jose network, respectively. Table

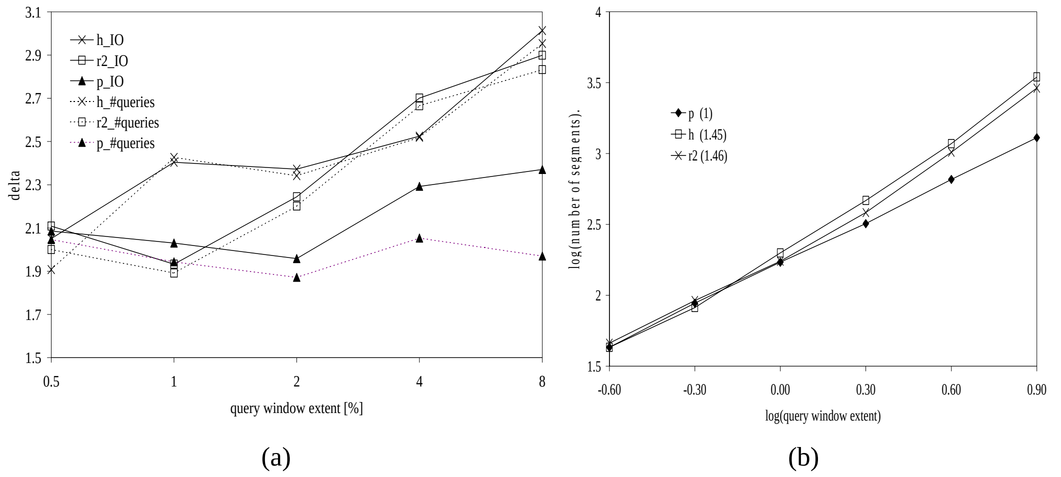


Figure 13: Fractal dimension and query processing cost for *Hilbert*, *Raster*, and *Parallel networks*: (a) I/O cost and (b) a modified box-count plot

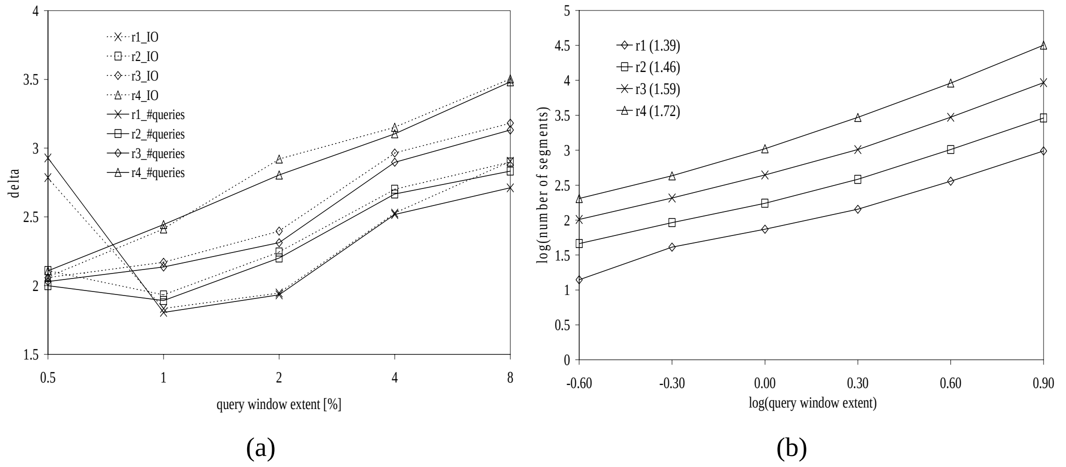
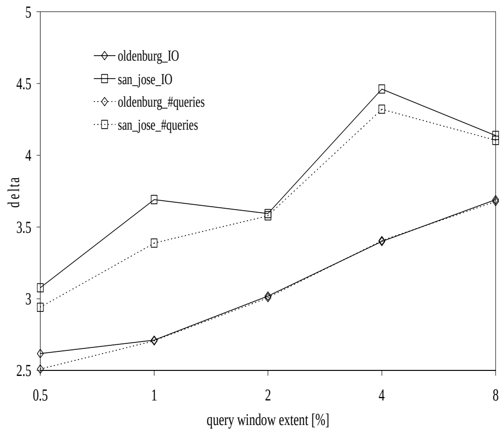
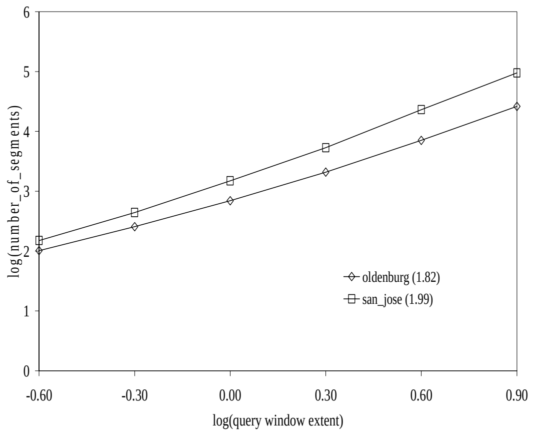


Figure 14: Fractal dimension and query processing cost for *Raster networks* of varying construction steps: (a) I/O cost and (b) a modified box-count plot



(a)



(b)

Figure 15: Fractal dimension and query processing cost for *real-world networks*:
(a) I/O cost and (b) a modified box-count plot

1 gives an overview of the various measures of fractal dimensions obtained with different methods.

Figure 13(b) to Figure 15(b) are alternative representations for the same data that was used for Figures 14(a) to 16(a). Consequently, the explanation of the differences between the theoretical values and the values determined by the modified box-count plot is the same. The main cause is the non-uniform coverage of the network by the query windows as opposed to the coverage of a grid for the regular box-count plot. Using the latter method, we obtain fractal dimensions very close to their theoretical values (cf. Table 1).

Network	Hausdorff fractal dimension		
	theoretical	box-count [33]	modified box-count
p	1	0.98	1
h	2	1.98	1.45
r1	2	1.95	1.39
r2	2	1.98	1.46
r3	2	2	1.59
r4	2	2	1.72
Oldenburg	-	1.64	1.82
San Jose	-	1.72	1.99

Table 1: An overview of the various fractal dimension measures

5.3 Query Processing Cost – Synthetic Networks

The query processing cost in the 2D approach is largely determined by the numbers of queries that result from the query window mappings. Overall, a trade-off exists between increased cost caused by a large number of queries vs. reduced cost caused by increased fanout. Further, mapping the queries requires us to access an additional index structure containing the network (cf. Section 3.2). This adds additional cost, which is included in the following charts unless stated otherwise.

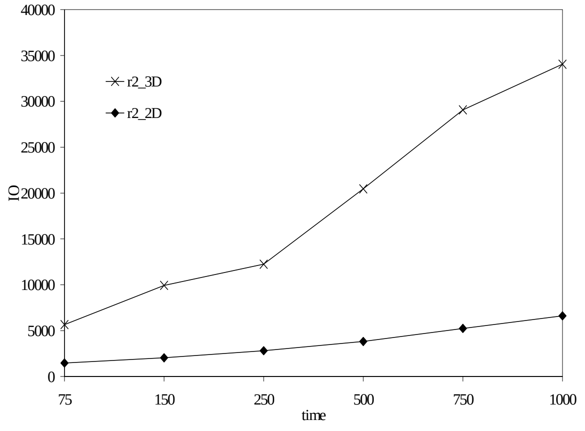
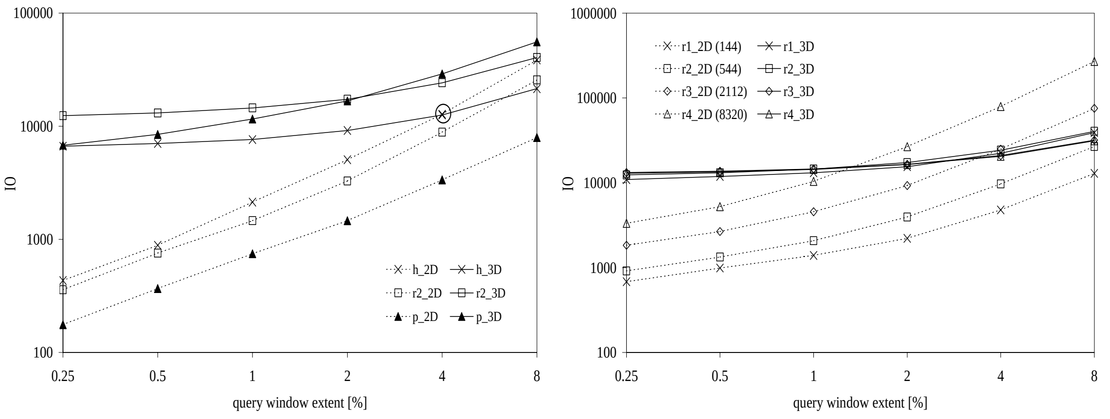


Figure 16: Query performance varying temporal extent for the Raster network

The experiments in Figure 17 aim at comparing the query processing of trajectory data sets stemming from networks (i) of different types and (ii) networks of the same type but varying lengths and numbers of edges.

Figure 17(a) compares the processing of trajectory data from different types of networks. The break-even point between 2D and 3D query processing cost is reached earlier or later depending on the type of network, e.g., network "h" reaches it at a 4% query window extent, "r2" shortly after 8%, and "p" well beyond the 8% mark. Thus, given two networks of similar length, the more complex one has a higher query processing cost in the mapping approach. Section 5.2 establishes that the number of query windows produced is positively correlated to the network complexity. This confirms the results shown in Figure 17(a), where the break-even point for the most complex network ("h") occurs for the smallest respective query window size.

Figure 17(b) shows the result of an experiment with varying Raster networks. The 3D query processing cost remains nearly constant for all networks. This is plausible, since in 3D, we index the trajectories, and the shape of the underlying network is of little influence. However, the 2D cost varies greatly and is for a given query window size lowest for the dataset linked to a network with the *lowest number of edges*. For small query windows, the 2D cost is generally lower than the 3D query processing cost. With an increased query size, and thus an increased number of transformed query



(a) (b)

Figure 17: Query performance varying query size: (a) Hilbert, Raster, and Parallel networks and (b) varying Raster networks

windows, the 2D cost increases over the 3D cost. Again, this break-even point is shifted towards larger query windows for networks with a smaller number of edges.

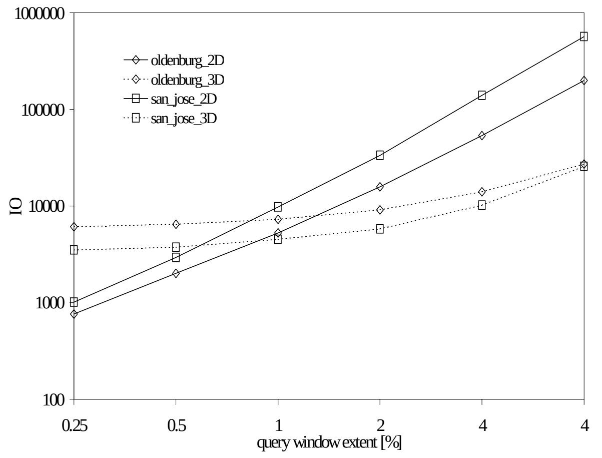
Figure 16 compares the processing of trajectory data ("r2" network) with a varying temporal extent ranging, from 75 to 1000 samples, stemming from 250 moving objects. The size of the query windows (2% spatial extent, 10% temporal extent) was kept constant for all datasets. Both, the 3D and 2D query processing cost is increased for larger datasets. However, the rate of increase for the 3D approach is higher. This overall favors the 2D approach, since trajectory data typically grows with respect to time rather than space ("objects keep on moving, but not necessarily further away").

5.4 Query Processing Cost – Real Networks

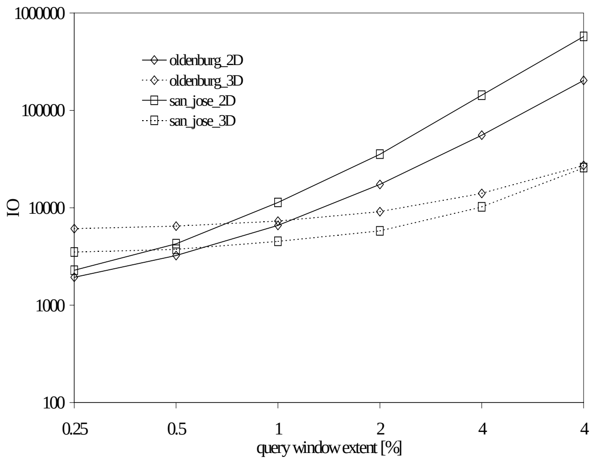
Using real-world networks, we want to determine whether the results from the previous section still hold. We additionally want to examine the effect of the added cost of query window mapping. Figure 18(a) shows for a varying query size the number of I/Os needed in the three- and two-dimensional cases to process the respective query *without* considering the mapping cost. We reach the break-even point for the San Jose dataset at a query size of 0.7%. For the Oldenburg dataset, break-even is at 1.4%. Considering the cost of querying the network in Figure 18(b), the break-even point becomes 0.4% and 1.1%, respectively. The Oldenburg dataset reaches this break-even point with larger query sizes because of its lower complexity and the smaller number of network edges.

5.5 Summary

The results of the performance study are as follows. Mapping trajectory data to lower dimensions prove beneficial in terms of query processing given the right circumstances! The number of query windows in the 2D case determines its cost and competitiveness with respect to the 3D approach. This number is determined by the density and complexity of the underlying road network. The experiments show that the lower the complexity of a network, the larger the query window is that can be supported by the mapping approach with competitive cost. Further, when considering the typical temporal growth of the trajectory data, the 2D approach scales better. The rate of increase of the 2D query processing cost is smaller when compared to the 3D case. Finally, the experiments show that the number of transformed query



(a)



(b)

Figure 18: Query performance varying query size: (a) without and (b) with network query cost

windows is correlated with the fractal dimension of the network. Thus, Equation 4 was validated empirically.

6 Conclusions and Future Work

Much work in the context of spatiotemporal query processing has dealt with the indexing of trajectories by proposing new access methods. In particular, the present paper aims at using off-the-shelf methods for the indexing of trajectories of objects moving in transportation networks. The idea is to exploit the restriction of the movement to networks in order to reduce the dimensionality of the trajectory data. More specifically, this paper proposes a mapping approach that exploits the possibility of mapping two-dimensional networks to one-dimensional intervals. Thus, the dimensionality of trajectories can be reduced from three to two dimensions. The important question then becomes the following: given two indices, one for the original 3D data and one for the mapped 2D data, which one is superior in terms of query processing cost? The mapping of queries to suit the 2D data becomes the critical aspect. In particular, the number of 2D queries that result from the mapping of a 3D query is crucial. The larger this number is, the less likely it is that the mapping approach outperforms querying the data in the original space. An important parameter that determines this number of 2D queries is the fractal dimension of the network. In our case, we use the Hausdorff fractal dimension. We derive a power law based on the Hausdorff dimension that estimates the number of mapped query windows relatively to changing the original query window extent. The performance study presents results for varying sets of queries and trajectory data. The data is produced using a network-based generator [3]. Using five different types of networks, three synthetic and two real-world networks, the experiments confirm what is given analytically by the above power law. The lower the fractal dimension of a network, the more likely it is that the mapping approach proves to be beneficial over indexing the data in 3D space.

This work points to several future research directions. One obvious extension is to consider alternative access methods for indexing the trajectory data in the 3D and the 2D spaces. It would be especially interesting to adapt the TB-tree [24] to the 2D data. Another extension is to investigate this approach for other types of queries proposed previously [24]. One has to carefully investigate the implications of the mapping approach with respect to indexing in general, e.g., dead space in 3D space vs. 2D space. The trajectory mapping algorithm assumes a Hilbert ordering of the segments. With respect to optimal index structures, more elaborate schemes, such as using the trajectory data for determining the approach for mapping the network edges, might improve the cost of indexing trajectories in 2D space. Also, there are existing approaches to the indexing of moving object data in the 2D case. Comparing them to the performance of the 3D case using R-tree based methods is of interest. Finally, a challenging focus of future work is to apply this method in a practical setting, e.g., using off-the-shelf database technology and applying the presented approach in an application context such as location-based services for mobile devices.

Acknowledgements

The authors would like to thank Thomas Brinkhoff for providing us with the use of his network-based data generator as well as for being ever-present with help and support. We also would like to thank Ouri Wolfson for discussions on this subject during a visit at Aalborg University.

This work is supported by the DB-Globe project, funded by the European Commission under contract number IST-2001-32645, the IXNILATHS project funded by the Greek General Secretariat of Research and Technology, and the Wireless Information Management research training network, funded by the Nordic Academy for Advanced Study.

References

[1] Agarwal, P. K., Arge, L., and Erickson, J.: Indexing Moving Points. In *Proc. of the 19th ACM Symposium on Principals of Database Systems*, pp.175-186, 2000.

[2] Belussi, A. and Faloutsos, C.: Estimating the Selectivity of Spatial Queries Using the ‘Correlation’ Fractal Dimension. In *Proc. of the 21th Int’l Conference on Very Large Databases*, pp. 299-310, 1995

[3] Brinkhoff, T.: Generating Network-Based Moving Objects. In Proc. *of the 12th Int’l Conference on Scientific and Statistical Database Management*, pp. 253-55, 2000.

[4] Chakka, V. P., Everspaugh, A., Patel, J. M.: Indexing Large Trajectory data sets with SETI. In *Proc. CIDR*, 2003.

[5] Chakrabarti, K. and Mehrotra, S.: Local Dimensionality Reduction: A New Approach to Indexing High Dimensional Spaces. In *Proc. of the 26th Int’l Conference on Very Large Databases* , pp. 89-100, 2000.

[6] Di Pasquale, A., Forlizzi, L., Jensen, C. S., Manolopoulos, Y., Nardelli, E., Pfoser, D., Proietti, G., Saltenis, S. Theodoridis, Y., Tzouramanis, T., Vassilakopoulos, M.: Access Methods and Query Processing Techniques. In *Spatio-Temporal Databases: The CHOROCHRONOS Approach*, Sellis, T. et al. (eds), pp. 203-261, Springer Verlag, Berlin, 2003.

[7] Faloutsos, C. and Kamel, I.: Beyond Uniformity and Independence: Analysis of R-Trees Using the Concept of Fractal Dimension. In *Proc. of the 13th ACM Symposium on Principals of Database Systems*, pp. 4-13, 1994.

[8] Faloutsos, C. and Lin, K.-I.: *FastMap*: A Fast Algorithm for Indexing, Data-Mining and Visualization of Traditional and Multimedia Datasets. In *Proc. of ACM-SIGMOD Conference on the Management of Data*, pp. 163-174, 1995.

[9] Faloutsos, C., Seeger, B., Traina, A., and Traina, C.: Spatial Join Selectivity Using Power Laws. *Proc. of ACM-SIGMOD Conference on the Management of Data*, pp. 177-188, 2000.

[10] Frentzos, E.: Indexing Objects Moving on Fixed Networks. In *Proc. of the 8th SSTD conference*, pp. 289-305, 2003.

[11] Güting, R.H., Böhlen, M., Erwig, M., Jensen, C.S., Lorentzos, N., Schneider, M., and. Vazirgiannis, M..: A Foundation for Representing and Querying Moving Objects. *ACM TODS* 25(1):1-42, 2001.

[12] Guttman, A.: R-trees: a Dynamic Index Structure for Spatial Searching. In *Proc. of ACM-SIGMOD Conference on the Management of Data*, pp. 47-57, 1984.

[13] Hadjieleftheriou, M., Kollios, G., Tsotras, V., and Gunopulos, D.: Efficient Indexing of Spatiotemporal Objects. In *Proc. of the 8th Int’l Conference on Extending Database Technology*, pp. 251-268, 2002.

[14] Hage, C., Jensen, C.S., Pedersen, T.B., Speicys, L., and Timko, I.: Integrated Data Management for Mobile Services in the Real World. In Proc*. of the 29th Int’l Conference on Very Large Data Bases*, pp. 1019-1030, 2003.

[15] Kollios, G., Gunopulos, D., and Tsotras, V., Delis, A., and Hadjieleftheriou, M.: Indexing Animated Objects Using Spatiotemporal Access Methods. *IEEE TKDE*, 13(5):758-777, 2001.

[16] Lazaridis, I., Porkaew, K., and Mehrotra, S.: Dynamic Queries Over Mobile Objects. In *Proc. of ACM-SIGMOD Conference on the Management of Data*, pp. 269-286, 2002.

[17] Leutenegger, S., Lopez, M., and Edington, J.: STR: A Simple and Efficient Algorithm for R-Tree Packing. In *Proc. of the 12th Int'l Conference on Data Enginnering*, pp. 497–506, 1997.

[18] Manolopoulos, Y., Theodoridis, Y., and Tsotras, V.: *Advanced Database Indexing*. Kluwer Academic Publishers, Boston/Dordrecht/London, 2000.

[19] Nascimento, M., Silva, J., and Theodoridis, Y.: Evaluation of Access Structures For Discretely Moving Points. In *Proc. of Int'l Workshop on Spatio-Temporal Database Management*, pp. 171-188, 1999.

[20] Oracle Corporation: *Oracle Spatial User's Guide and Reference, Release 9.2*. 2002.

[21] Pagel, B.-U., Korn, F., and Faloutsos, C.: Deflating the Dimensionality Curse using Multiple Fractal Dimension. In *Proc. of the 16th Int'l Conference on Data Engineering*, pp. 589-598, 2000.

[22] Pfoser, D.: Indexing the Trajectories of Moving Objects. *IEEE Data Engineering Bulletin*, 25(2): 3-9. 2002.

[23] Pfoser, D. and Jensen, C.: Capturing the Uncertainty of Moving-Object Representations. In *Proc. of the 6th Int'l Symposium on Spatial Databases*, pp.111-132, 1999.

[24] Pfoser, D., Jensen, C., and Theodoridis, Y.: Novel Approaches to the Indexing of Moving Object Trajectories. In *Proc. of the 26th Int'l Conference on Very Large Databases*, pp.395-406, 2000.

[25] Pitoura, E., Abiteboul, S., Pfoser, D., Samaras, G., and Vazirgiannis, M.: DB-Globe, A Service-Oriented P2P System for Global Computing. *SIGMOD Record*, 32(3), 2003.

[26] Porkaew, K., Lazaridis, I., and Mehrotra, S.: Querying Mobile Objects in Spatio-Temporal Databases. In *Proc. of the 7th Int'l Symposium on Advances in Spatial and Temporal Databases*, pp. 55-78, 2001.

[27] Saltenis, S. and Jensen, C. S.: Indexing of Moving Objects for Location-Based Services. In *Proc. of the 18th Int'l Conference on Data Engineering*, pp. 463-472, 2002.

[28] Saltenis, S., Jensen, C. S., Leutenegger, S., and Lopez, M.: Indexing the Positions of Continuously Moving Objects. In *Proc. of ACM-SIGMOD Conference on Management of Data*, pp. 331-342, 2000.

[29] Schröder, M.: *Fractals, Chaos, Power Laws: Minutes from an Infinite Paradise*. W.H. Freeman and Company, New York, 1991.

[30] Sistla, A., Wolfson, O., Chamberlain, S., and Dao, S.: Modeling and Querying Moving Objects. In *Proceedings of the 13th International Conference on Data Engineering*, pp. 422-432, 1997.

[31] Tao, Y. and Papadias, D.: MV3R-Tree: A Spatio-Temporal Access Method for Timestamp and Interval Queries. In *Proc. of the 27th Int'l Conference on Very Large Databases*, pp. 431-440, 2001.

[32] Tayeb, J., Ulusoy, Ö., and Wolfson, O.: A Quadtree-Based Dynamic Attribute Indexing Method. *The Computer Journal* 41(3), pp. 185-200, 1998.

[33] Traina, C., Traina, A., Wu, L., and Faloutsos, C.: Fast Feature Selection Using Fractal Dimension. In *Proc. of the XV Brazilian Symposium on Databases*, 2000.

[34] Tzouramanis, T., Vassilakopoulos, M., and Manolopoulos, Y.: Overlapping Linear Quadtrees: A Spatio-Temporal Access Method. In *Proceedings of the 6th International Symposium on Advances in Geographic Information Systems*, pp. 1-7, 1998.