A Probabilistic Index Structure for Querying Future Positions of Moving Objects

Philip Schmiegelt[1], Andreas Behrend[2], Bernhard Seeger[3], and Wolfgang Koch[1]

[1] Department SDF
Fraunhofer FKIE, Wachtberg, Germany
{philip.schmiegelt,wolfgang.koch}@fkie.fraunhofer.de
[2] Institute of Computer Science III
University of Bonn, Germany
behrend@cs.uni-bonn.de
[3] Department of Mathematics and Computer Science
Philipps-Universität Marburg, Germany
seeger@informatik.uni-marburg.de

Abstract. We are witnessing a tremendous increase in internet connected, geo-positioned mobile devices, e.g., smartphones and personal navigation devices. Therefore, location related services are becoming more and more important. This results in a very high load on both communication networks and server-side infrastructure. To avoid an overload we point out the beneficial effects of exploiting future routes for the early generation of the expected results of spatio-temporal queries. Probability density functions are employed to model the uncertain movement of objects. This kind of probable results is important for operative analytics in many applications like smart fleet management or intelligent logistics. An index structure is presented which allows for a fast maintenance of query results under continuous changes of mobile objects. We present a cost model to derive initialization parameters of the index and show that extensive parallelization is possible. A set of experiments based on realistic data shows the efficiency of our approach.

1 Introduction

Due to the advances in GPS-technology, navigational devices are available in almost all vehicles and mobile phones today. These devices are primarily used for the computation of (optimal) routes and for giving instructions to the driver using his/her actual position on a route. Moreover, web services where devices periodically transmit their actual position are very popular. Surprisingly, there is an obvious mismatch between the available information and the information used by these services.

We are convinced that the information about future positions is also very valuable in many application scenarios, and that all of the available knowledge should be exploited. It is very simple to transmit the routes being computed by the navigational devices to these web services and utilize that information.

B. Catania, G. Guerrini, and J. Pokorný (Eds.): ADBIS 2013, LNCS 8133, pp. 112–125, 2013.
© Springer-Verlag Berlin Heidelberg 2013

In contrast to existing work, in this paper we focus on the efficient indexing of future positions for a high number of mobile objects given their preset trajectories and current positions. Our experimental evaluation confirms that a processing of this information in real-time is possible. Of course, the accuracy of the predicted position decreases over time. This can be modeled by a time variant probability density function, which can be efficiently handled by the methods proposed in this paper.

To illustrate the importance of using knowledge about future positions, let us consider a logistics company that manages a large fleet of trucks. Nowadays it is common that each parcels' position is known with high accuracy. Trucks delivering parcels have a rigid schedule, thus their future position is known with high accuracy. Still this knowledge is not yet exploited. A typical query would then be to select all trucks within a given region at the same time in the near future with a high probability:

```
select * from my_trucks where
position is within
   'my_company_headquarter'
[range (now + 1 hours)
    to (now + 2 hours)]
[probability 90%]
```

In order to optimize the delivery service of goods, it might be important to exchange goods among trucks dynamically. An optimal meeting point for a set of cooperating trucks has to be found:

```
select * from my_trucks t where
distance('truck_a', t) < 5 kilometers
[range 14:00 to 15:00]
[probability 75%]
```

Note that the results of these queries are not only important at the point in time when they will occur, but already earlier at the time when they have been computed. In fact, these "early" results are important for global planning and coordination of a fleet.

These examples show that the efficient determination of the future position of mobile objects represents a relevant problem. To this end, a framework is needed that supports continuous queries over a dynamic set of moving objects whose future travel routes are computed in advance. While the problem seems to be closely related to historical management of trajectories (like in [1]), the dynamic nature of the problem makes it substantially harder to address. Contrary to one-time queries, our problem is more related to continuous queries for the following reasons. First, mobile objects that start a new tour can influence the result sets of continuous queries. Second, traffic jams and other unforeseeable events will have a serious impact on traveling, and consequently also on continuous queries and their results. The difference to other approaches to continuous queries is that results already delivered to the user might become invalid later.

However, these early results are very important to the user for the purpose of planning. The key question of the paper is therefore how to manage the results of a set of continuous queries efficiently and how to update the results due to the occurrence of unforeseeable events. This paper is based on the previous work ([2] and [3]). However, we did not take into account the decreasing probability of the knowledge of an object position over time. This made the approach quite unrealistic for many scenarios. Furthermore, the parameters for the creation of the index had to be determined manually, whereas in this paper a cost model is proposed. We also show that an efficient parallelization of the algorithms introduced in the previous papers is possible. In sum, the additional contributions of this paper are the using of a probability density function (pdf) to approximate an objects position. Second, the introduction of a cost model to allow a precise determination of the parameters of the index structure. And third the added capability to parallelize the algorithms.

2 Preliminaries

In this section we introduce the formal definitions of moving objects, trajectories, and queries. We assume that the moving objects are bounded in a two-dimensional universe, e.g., the unit square. Our approach is based on a continuous timeline that allows us to compute all results, even if they are valid only between two discrete timestamps. This is a major difference to the discrete-time model commonly used in data stream management systems like [4].

2.1 Trajectories

Within the universe, a path can be specified as a sequence of two-dimensional points $P = (p_1, \ldots, p_n)$, termed waypoints throughout this paper. Associated to P is a series of points in time $T = (t_1, \ldots, t_n)$. A trajectory $traj = (P, T)$ consists of a sequence of points and a sequence of time stamps with equal length $|P| = |T|$. Note that the trajectories of different objects do not need to and in general will not have the same length. Furthermore, if two consecutive waypoints are equal whereas the corresponding points in time are not, idleness is modeled.

For sake of simplicity, we assume that the movement between two waypoints p_i and p_{i+1} can be linearly interpolated. Note that our approach can be generalized to more advanced interpolation functions.

Note that in many scenarios trajectories are confined to road networks. This is, however, not necessary in our approach where arbitrary trajectories are handled.

2.2 Moving Objects

A moving object can be any locatable device, e.g., a GPS-enabled mobile phone or a trackable truck. With each object o, a trajectory $traj_o$ is associated. Similar to [5], we require that an object is aware of its current position, but also of its

future positions and their certainty, not only its current position, direction, and speed. However, this only disqualifies very simple GPS loggers, but even cheap navigational devices have enough storage capacity and computational power to comply with this requirement.

The decreasing accuracy of the predicted position in the course of time is included in our design by using a probability density function (pdf), which allows us to model the uncertainty of the objects' position in a mathematically precise way. It is also possible to include uncertainty about the trajectory. However, the semantically more sound approach is to have a certain trajectory, where the objects current position is uncertain. This also allows for more flexibility, as the trajectories are independent of the type of moving object they are used by.

A moving object is therefore defined as a tuple $M = (traj, \mathcal{P})$, where $traj$ is the associated trajectory and $\mathcal{P}$ can be an arbitrary pdf.

In the remainder of the paper, we will discuss Gaussian functions with expectation $\mathbb{E}$ and covariance matrix $\mathbb{C}$ as pdfs only, for the sake of simplicity. However, any probability density function could be employed. In the two-dimensional case, $\mathbb{E}$ is a two-dimensional vector and $\mathbb{V}$ is a 2x2 matrix.

In the case of moving objects, the mean is defined by the trajectory, whereas the covariance matrix is a function growing with the time difference to the last reported accurate position. The uncertainty of the position of a moving object at time t is therefore modeled by

$$\mathcal{N}\big(p_{uncertain}(t);\, p(t),\, \mathbb{V}(t - t_0)\big)$$

where $p_{uncertain}(t)$ is the random variable denoting the uncertain position of the object, $p(t)$ is the predicted position according to the given trajectory and $\mathbb{V}(t - t_0)$ is a covariance matrix, increasing with distance to time t_0 where the last certain position was reported. For the sake of simplicity we will assume that $\mathbb{V}(t_0)$ is a diagonal matrix obtained by multiplying the identity matrix with a scalar factor. By that the error of the expected value is the same regardless of the direction. For practical purposes we cut all probability values which are below a predefined threshold θ.

An example is given in Figure 1, where an object moves on the road. The uncertainty at the first timestamp in the foreground of the figure is very small. This is indicated by a high peak of the pdf and a narrow base. On the planned path (further in the background) the uncertainty has increased, shown by a larger covariance and a smaller absolute value for the expected position. Note that the knowledge of a road network will have a great impact on the shape of the pdf. This is, however, beyond the scope of this paper and does not affect the methods used for indexing. Using such meta information will be addressed in future work.

Introducing probability in the context of moving objects introduces a broad variety of challenging tasks like tracking, filtering, or retrospection (see [6]). These problems are, however, beyond the scope of this paper.

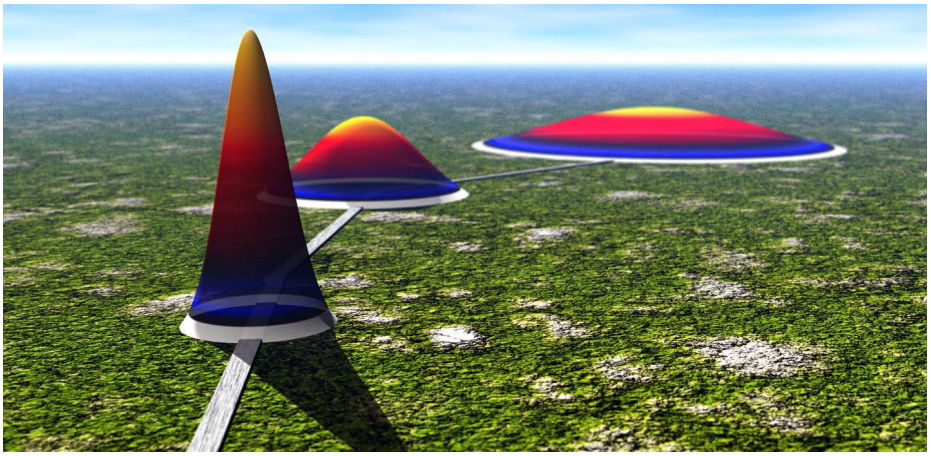


Fig. 1. Object Moving with Uncertain Position

2.3 Queries

A continuous query q over mobile objects is formally defined as

$$q = (pred(o), [t_{start}, t_{end}], \Theta_{(\text{optional})}),$$

where $[t_{start}, t_{end}]$ denotes the time interval within the query result is of interest to a user. $pred(o) \mapsto \{[s_0, e_0], \ldots [s_n, e_n]\}$ is a function returning for every moving object o the set of time intervals when o qualifies for the query predicate *pred*. These intervals are non-overlapping, and adjacent intervals are required to be coalesced. Once a query is created, results are received continuously. Θ is an optional probability threshold for this query. It is possible to specify that, e.g., only results having a probability greater then 95% should be reported. It is worthwhile noting that this cannot be specified within the query predicate as the predicate ultimately decides whether a moving object can be a result of the query, e.g., if the object might be in the query region at any point in time. However, the probability can dynamically change when after the transmission of the expected trajectory the correct position is reported. The probability of query result changes after the transmission, and previously discarded results might have an updated probability which is higher than the query threshold Θ.

3 A Probabilistic Index Structure

3.1 Index Structure

For indexing the spatial domain, a simple grid index [7] is used. The basic idea of a grid index is to partition the spatial data space into partitions (cells) using a two dimensional quadratic grid. Each cell of the grid index contains pointers

to the objects that intersect this cell. The advantage of such a simple structure is that the uncertainty of the position of a moving object can be included in an intuitive way. Pointers are not only generated for cells intersecting the (certain) trajectory, but for all cells the object might be within. The probability threshold θ is used to avoid having too many cells affected with a very low probability of the object actually being there. The computation of the overlap with the object's pdf is straightforward if a Gaussian pdf is used and the covariance matrix is monotonically increasing. The first is assumed in this paper for sake of simplicity, the second is a quite natural thought. This implies that the covariance matrix, dependent on time t, is defined as

$$\mathbb{C} = \begin{Bmatrix} a & 0 \\ 0 & a \end{Bmatrix} \cdot t = t \cdot a \cdot \mathbb{I}.$$

As the error ellipses are circles for sake of simplicity, as defined in the beginning of the paper, the radius r has to be calculated as the only parameter of the circle. As in our two dimensional setting both spatial coordinates (p_x and p_y) are normally distributed, the circle radius $r^2 = p_x^2 + p_y^2$ is χ^2 distributed. Thus, for each desired confidence interval p, e.g. 95%, the value of the χ^2 distribution can be conveniently looked up in a χ^2 table, or be computed on the fly. The radius r is therefore given by $r = \sqrt{\chi^2(p) \cdot a}$. In case of $p = 95\%$, a lookup in the chi^2 table shows that $chi^2(95\%) = 0.103$

As time passes, uncertainty on positions grows. In this case, it is sufficient to calculate the extent of the Gaussian pdf which is above the threshold θ for each waypoint. The resulting trapezoid is a superset of the region the object might be in. Note that a superset is sufficient for the index to function properly, as the actual intersection of the object's pdf and a query region always has to be calculated. Only performance might suffer if the superregion is too large. It is important to note, however, that the index is independent of the actual pdf used. Other distributions unlike the Gaussian could be employed. Only the calculation of the confidence region has then to be adapted, the index itself remaining unchanged.

The temporal index does not store any probability information, as time is assumed to be certain, i.e., all clocks are synchronized. An anomaly regarding time has to be taken into account, however. If the end time of the trajectory is closely before the start time of a query, and both the last waypoint and the border of the query are in close spatial vicinity, the query will be ignored. Due to the uncertainty in space, however, the query predicate might still be fulfilled with a high probability. This anomaly can be circumvented by adding 'dummy' elements to the trajectory, where the last waypoint is repeated. That is, the moving object is not immediately regarded as deleted, but treated as a non-moving object for some time, until the probability decreased below the threshold θ. As this can be done programmatically, the described anomaly can automatically be handled and correct results are delivered.

In Figure 2, the interaction of the spatial and the temporal index is illustrated. A trajectory and a temporally bounded range query are stored in this example. The range query is shown in light green. As it touches four cells of the spatial

grid, a reference to the query is stored in each cell. Also, references of the query are added to the temporal index in each cell overlapping with the lifetime of the query. The same is done for the trajectory. All cells where the probability of the object being in this cell is greater then θ, a reference to the object is stored. These cells are marked in grey.

3.2 Cost-Based Parameters

This subsection introduces a cost model that can be used to set the grid resolution of the spatial grid indexes appropriately. This means that once the approximate lengths of the routes and the sizes of the queries are known, the optimal resolution of the index can be calculated with a simple formula. We assume here the case of a general update and a constant pdf for each object, which leads to a pessimistic model.

Let us consider the unit square to be the universe. The grid resolution r is the number of slices in each dimension of the grid index. We assume that the resolution is constant for every dimension. However, we distinguish the resolution of the query index (r_q) and the resolution of the object index (r_o). Note that the side length of a grid cell is then $\frac{1}{r_o}$ and $\frac{1}{r_q}$, respectively. Overall, there are $|Q|$ queries and $|O|$ objects being distributed uniformly across the universe. We assume a constant number of objects and a constant number of queries. The average length of a trajectory is given by $l_{trajectory}$. We first examine range queries, all of them being rectangular with extension l_{query} in every dimension.

In the worst case, the trajectory of an object crosses $l_{object} \cdot r_o$ grid cells. Analogously, $l_{query}^2 \cdot r_q^2$ cells are occupied by a single query. This means that a cell contains approximately $\frac{|O| \cdot (l_{object} \cdot r_o)}{r_o^2} = \frac{|O| \cdot l_{object}}{r_o}$ objects and $\frac{|Q| \cdot l_{query}^2 \cdot r_q}{r_q^2} = \frac{|Q| \cdot l_{query}}{r_q}$ queries.

The cost of an insertion of a moving object can therefore be estimated by

$$\underbrace{l_{object} \cdot r_o}_{\text{cells to consider}} \cdot \underbrace{\frac{|Q| \cdot l_{query}}{r_q}}_{\text{queries per cell}}$$

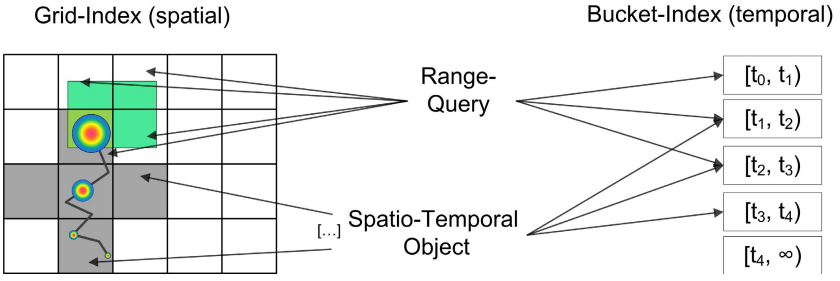
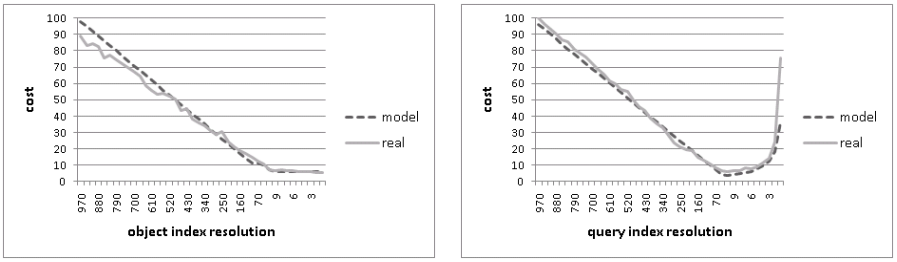


Fig. 2. Decoupled Indexes



(a) Object Index Comparison (b) Query Index Comparison

Fig. 3. Confirmation of Cost-Model

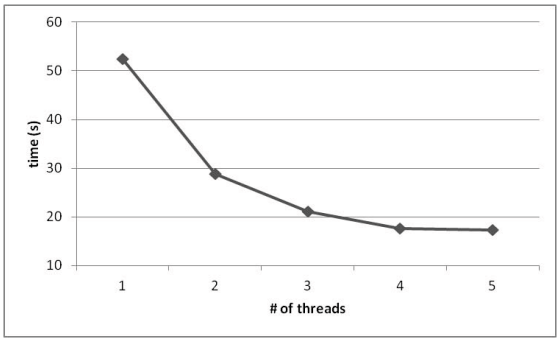


Fig. 4. Sizes of Thread Pool

$$= l_{object} \cdot |Q| \cdot l_{query} \cdot \frac{r_o}{r_q}$$

By using the same parameter set, we now examine the cost of a general update. First, the grid index has to be updated. The object has to be found and deleted, before the new trajectory can be inserted. $(r_o \cdot l_{object}) \cdot log(\frac{|O| \cdot l_{object}}{r_o})$. In the next step, the cells intersecting with the query have to be retrieved. $r_q \cdot l_{object} \cdot (\frac{|Q| \cdot l_{object} \cdot l_{query}}{r_q})$

Finally, new events have to be calculated for each qualifying query. Since this cost is constant for a given instance, it is not included in the cost model.

We also experimentally confirmed our model. The trajectories for the experiments were created synthetically to match the underlying assumptions of the cost model. However, we also found that the cost model is sufficiently accurate for realistic data sets, created by Brinkhoffs generator[8]. In order to demonstrate the accuracy we compared the runtimes with the cost model, too. This, however, is only possible by not using the pure numbers of operations as a cost measure, but to introduce an additional weighting factor for each operation to express the required CPU-time.

Figure 3(a) shows the time needed to access the object index as a function of its resolution. The resolution of the query index was kept constant in this

experiment. We conducted the same experiments with a varying resolution of the query index, while keeping the resolution of the object index constant. The results are shown in Figure 3(b). The results demonstrate that the cost model is sufficiently accurate (with a relative error less than 10% for almost all cases). Moreover, the cost model shows that a global minimum for the resolution of the grid indexes exists. In particular, this observation is useful for setting this important parameter.

3.3 Parallelization

For the developed algorithms synchronization is essential due to the extremely high concurrency with respect to the number of objects and queries. Our goal was to provide flexibility and parallelism as much as possible, while still guaranteeing correctness. Each object has its own mutual lock, allowing multiple readers but only a single writer. The grid index also has such a lock, guaranteeing the integrity of the grid cells. In case of an update of a moving object, both a write-lock for the grid index and the updated object have to be acquired, preventing concurrent changes of an object or concurrent insertions of multiple queries (objects) into the grid index. The efficiency can be proven by measuring the total runtime of the algorithms with a given set of parameters. Figure 4 depicts the total runtime as a function of the number of threads available to the algorithms. As the experiments where executed on a quad core processor, having more than four threads does not have an effect on the runtime. The index structures and algorithms are also designed to be executed in a shared-nothing environment[9]. As this task is purely engineering due to the inherent ability of all methods to run in parallel it is not discussed in this paper.

4 Experiments

A set of experiments was conducted to validate the performance of our proposed approach to supporting continuous queries.

4.1 Dataset and Experimental Setup

Unfortunately, no real world dataset is available due to the novelty of our approach. Nevertheless, we tested the proposed data structures and algorithms on a real road network. Due to its popularity, we choose to use Brinkhoff's moving object generator[8] to simulate the moving objects. In our experiments, we restrict the discussion of the results to the network generated from the street map of the San Francisco Bay Area. The usage of other maps showed similar effects, and therefore, are omitted.

Our default setting of the generator results in a simulation of 10,000 moving objects and 1,000 randomly distributed range queries, each of them covered 1% of the data space as these numbers match with a typical application in the area of fleet management. Note that the algorithms are easily parallelizable and

therefore using a larger experimental setting, e.g., a typical one in the domain of social networks, can be done. The experimental results for this are given in Figure 4. A more detailled description can be found in section 3.3 of this paper.

The algorithms presented here were implemented in JAVA and were performed on an Intel Dual Core Xeon 3.4 GHz with 6GB RAM running CentOS 5. The JAVA virtual machine occupied 3GB RAM.

4.2 Comparison

In the following, we compare our approach with the one presented in [5]. Let us first describe it briefly. Similar to our approach, the queries are stored in a grid index. There the moving objects are assumed to move along a straight line using the speed and the direction as reported at the last position. In fact, moving objects are required to report their position whenever their route or speed changes. This information is then used to compute the current results of each query. Note that future results are obtained by a simple extrapolation of the linear movement of the object. To limit the length of this predicted line, a *maximum-update-interval (MUI)* is defined. The period between two updates of a moving object must be within the MUI. An example is given in Figure 5. The trajectory is implied in the center of the figure, the necessary transmissions using a linear extrapolation approach above it, the single transmission of the trajectory approach beneath. The third transmission is necessary because the MUI has been reached, although the route did not change and the linear prediction is still correct.

We are aware that there are fundamental differences between this approach and ours, especially as it does not take probabilistic values into account. Recall that our approach delivers all results of a continuous query starting at the time the query is issued and results are updated when changes occur. Nonetheless we opted to use this competitive approach for our comparison, since it is still the one that is closest to our approach among those published so far. To keep the comparison as fair as possible, we compared the average total processing time of each object in both approaches. This means we added up the costs of all operations performed on a single object.

To model the dynamic changes of a trajectory, caused by a traffic jam, at each waypoint an object is updated with a given probability. This probability is called the *update rate*. We examined update rates between 0% and 15% in the experiments. In most real-life applications, this update rate will presumably be very small as we assume that a user follows the suggested track of the navigation system. The higher the update rate the higher the costs of our approach, while the update rate has no impact on the competitive approach.

Figure 6 depicts the average time for applying all updates to a moving object as a function of the MUI. It provides the costs of temporal updates for update rates 1%, 5%, 10% and 15% , In addition, the curve is plotted for the competitive approach. The costs of the competitive approach increase with increasing MUI, while the curves of our approach are obviously independent from MUI. Note that the range of the MUI is chosen in accordance with the recommendations

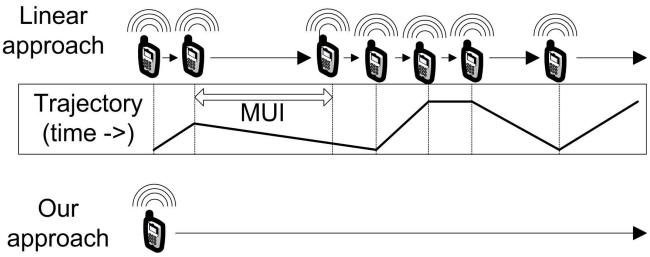


Fig. 5. Necessary Transmissions

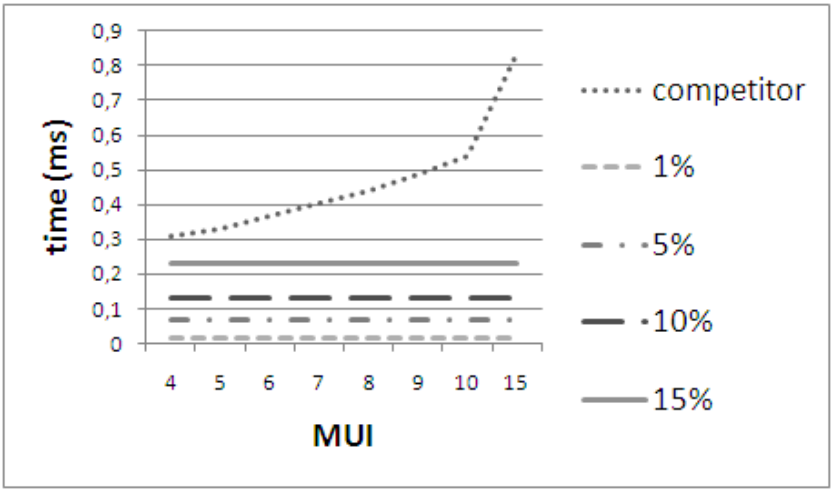


Fig. 6. Update Performance

in [5]. The results show that the performance of the competitive approach is superior to our approach only in case of general updates and small settings of MUI. For MUI > 6, our approach with general updates only is even superior to the competitive approach, If temporal updates are applicable, a large difference in the performance can be observed.

5 Related Work

There has been a large interest in supporting queries on moving objects, but only few papers address the problem of continuous queries on these objects. Most similar to our work is the QMOS (Query Moving Object Stream) system [5]. However, the fundamental difference to our work is that the knowledge about future trajectories is not exploited for query processing. Instead, the author predicts the future movement by only using a linear extrapolation. This requires that objects permanently transfer their current positions, an overhead that can

be avoided in our approach. Moreover, the temporal horizon of continuous queries is very limited, while there is no limitation in our approach on how far a query is in the future.

Most related work (e.g. [10,11]) deals with indexing of historical data. In general however, these methods are not applicable to continuous queries as they lack the option to update predicted future trajectories. As emphasized before, the efficient processing of updates is of utmost importance in the context of continuous queries.

Using stream processing technology in the context of moving objects is suggested by different papers. The main idea of SCUBA [12] is to group objects into clusters. Moving objects are constrained to a road network and an update of the direction can occur when a crossroad is reached. PLACE [13] also offers the evaluation of moving queries on moving objects. The moving objects have to submit their position continuously to the system, which then decides whether the information is processed or discarded. Though queries are constantly reporting changing results, continuous queries are not supported. SINA (Scalable INcremental hash based Algorithm) [14] is similar to PLACE, but uses a different data structure.

The problem of managing trajectories led to the development of different kinds of index structures. Most are based on the TPR*-tree[15], an index structure of the R-tree family enhanced to better support the temporal domain. STRIPES [10] is based on dual transformations, but its limitation is that its efficiency is only acceptable in case of a small time-horizon, whereas a larger horizon leads to a rapid degeneration of query performance. The horizon has to be set in advance and cannot be adapted to the incoming data. As a consequence, moving objects have to report their position frequently, even if neither the path nor the speed has changed. Another approach for moving objects is to keep the objects in a one-dimensional index structure. Space filling curves are used by the authors of [11] for their index structures in combination with the popular B-Tree. In general, these kinds of adapted B-trees can handle updates more efficiently than methods derived from R-trees and are therefore the preferred choice in dynamic settings. However, as objects are still assumed to move in a linear fashion, the limitation of these approaches is that only the near future of moving objects can be indexed.

There are a few papers dealing with future trajectories. In [16], trajectories are considered uncertain within a given bound. Queries are defined on these trajectories, allowing the user to specify, for example, whether an object is possibly or definitely inside a given region. Based on a common relational database system, [17] proposes methods for bulk updating trajectories when a road obstruction occurs. The processing is based on triggers and relational database technology. This severely limits performance and does not scale well. [18] focuses on insertion and deletion of trajectories without considering the problem of updating results.

While there are indexing methods for historical trajectories, as for example [19,20], we are not aware of approaches indexing future trajectories. For indexing historical data, updates on past states are not allowed. The proposed methods

are therefore not applicable to the problem of efficiently processing future trajectories, where updates frequently occur.

Related to our approach have been methods from data stream processing that are designed for processing predictive queries [21]. However, these methods are not designed for managing trajectories, but for supporting current and near future results only.

6 Conclusion and Future Work

In this paper we extended index structures and algorithms for processing the future trajectories of moving objects. The decreasing accuracy of the prediction was accounted for by using a probability density function. A cost model for determining the index parameters was introduced. It was also shown that the proposed methods can be efficiently parallelized.

In our present and future work, we address more advanced continuous queries like nearest neighbor queries. The predicates of these queries cannot be evaluated on the base of a single object, but require the inspection of multiple objects. We will also include the efficient processing of multi-modal pdf, where, e.g., at an intersection, many alternative routes are possible.

References

1. Patroumpas, K., Sellis, T.K.: Managing trajectories of moving objects as data streams. In: STDBM 2004, pp. 41–48 (2004)
2. Schmiegelt, P., Seeger, B.: Querying the future of spatio-temporal objects. In: ACM GIS 2010, pp. 486–489 (2010)
3. Schmiegelt, P., Seeger, B., Behrend, A., Koch, W.: Continuous queries on trajectories of moving objects. In: IDEAS 2012, pp. 165–174 (2012)
4. Krämer, J., Seeger, B.: Semantics and implementation of continuous sliding window queries over data streams. TODS 34(1), 1–49 (2009)
5. Lin, D., Cui, B., Yang, D.: Optimizing moving queries over moving object data streams. In: Kotagiri, R., Radha Krishna, P., Mohania, M., Nantajeewarawat, E. (eds.) DASFAA 2007. LNCS, vol. 4443, pp. 563–575. Springer, Heidelberg (2007)
6. Koch, W.: On bayesian tracking and data fusion: A tutorial introduction with examples. IEEE AESS Magazine 25(7), 29–52
7. Samet, H.: The Design and Analysis of Spatial Data Structures (Addison-Wesley). Addison-Wesley Pub. (Sd)
8. Brinkhoff, T., Str, O.: A framework for generating network-based moving objects. Geoinformatica 6 (2002)
9. DeWitt, D., Gray, J.: Parallel database systems: the future of high performance database systems. Commun. ACM 35, 85–98 (1992)
10. Patel, J.M., Chen, Y., Chakka, V.P.: Stripes: An efficient index for predicted trajectories. In: SIGMOD 2004, pp. 637–646 (2004)
11. Jensen, C.S., Lin, D., Ooi, B.C.: Query and update efficient b + -tree based indexing of moving objects. In: VLDB 2004, pp. 768–779 (2004)

12. Nehme, R.V., Rundensteiner, E.A.: Scuba: Scalable cluster-based algorithm for evaluating continuous spatio-temporal queries on moving objects. In: Ioannidis, Y., Scholl, M.H., Schmidt, J.W., Matthes, F., Hatzopoulos, M., Böhm, K., Kemper, A., Grust, T., Böhm, C. (eds.) EDBT 2006. LNCS, vol. 3896, pp. 1001–1019. Springer, Heidelberg (2006)
13. Mokbel, M.F., Xiong, X., Hammad, M.A., Aref, W.G.: Continuous query processing of spatio-temporal data streams in place. Geoinformatica 9(4), 343–365 (2005)
14. Mokbel, M.F., Xiong, X., Aref, W.G.: Sina: Scalable incremental processing of continuous queries in spatio-temporal databases. In: SIGMOD 2004, pp. 623–634 (2004)
15. Tao, Y., Papadias, D., Sun, J.: The tpr*-tree: An optimized spatio-temporal access method for predictive queries. In: VLDB 2003, pp. 790–801 (2003)
16. Trajcevski, D., et al.: Managing uncertainty in moving objects databases. TODS 29(3), 463–507 (2004)
17. Ding, H., Trajcevski, G., Scheuermann, P.: Efficient maintenance of continuous queries for trajectories. Geoinformatica 12(3), 255–288 (2008)
18. Chon, H.D., Agrawal, D., El Abbadi, A.: Range and knn query processing for moving objects in grid model. Mob. Netw. Appl. 8(4), 401–412 (2003)
19. Hadjieleftheriou, M., Kollios, G., Tsotras, J., Gunopulos, D.: Indexing spatiotemporal archives. The VLDB Journal 15(2), 143–164 (2006)
20. De Almeida, V.T., Güting, R.H.: Indexing the trajectories of moving objects in networks. Geoinformatica 9(1), 33–60 (2005)
21. Dittrich, J., Blunschi, L., Vaz Salles, M.A.: Indexing moving objects using short-lived throwaway indexes. In: Mamoulis, N., Seidl, T., Pedersen, T.B., Torp, K., Assent, I. (eds.) SSTD 2009. LNCS, vol. 5644, pp. 189–207. Springer, Heidelberg (2009)