Chapter 10

SENSING FOR MOBILE OBJECTS

Nicholas D. Larusso
UC Santa Barbara
Dept. of Computer Science
nlarusso@cs.ucsb.edu

Ambuj K. Singh
UC Santa Barbara
Dept. of Computer Science
ambuj@cs.ucsb.edu

Abstract Recent advances in affordable positioning hardware and software have made the availability of location data ubiquitous. Personal devices such as tablet PCs, smart phones and even sport watches are all able to collect and store a user's location over time, providing an ever-growing supply of spatiotemporal data. Managing this plethora of data is a relatively new challenge and there has been a great deal of research in the recent years devoted to the problems that arise from spatiotemporal data. This book chapter surveys recent developments in the techniques used for the management and mining of spatiotemporal data. We focus our survey on three main areas: (i) *data management*, which includes indexing and querying mobile objects, (ii) *tracking*, making use of noisy location observations to infer an object's actual or future position, and (iii) *mining*, extracting interesting patterns from spatiotemporal data. First, we cover recent advances in database systems for managing spatiotemporal data, including index structures and efficient algorithms for processing queries. Next, we review the problem of tracking for mobile objects to estimate an object's location given a sequence of noisy observations. We discuss some of the common approaches used for tracking and examine some recent work which focuses specifically on tracking vehicles using a road network. Then we review the recent literature on mining spatiotemporal data. We conclude by discussing some interesting areas of future research.

Keywords: Mobility, Spatiotemporal Data Management, Querying Mobile Objects, Movement Modeling,

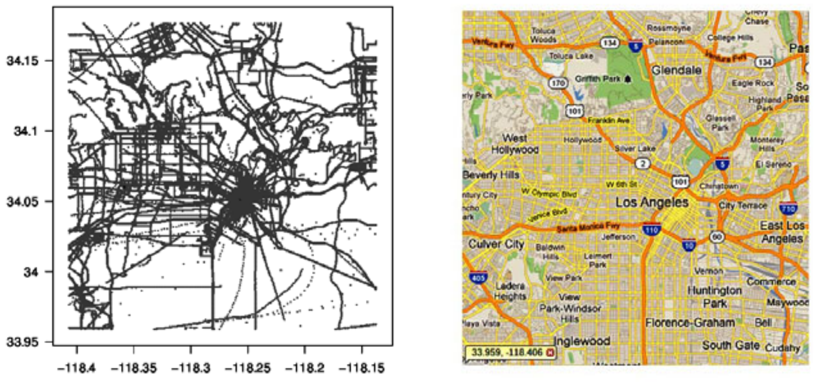
1. Introduction

The use of sensors that capture user location information over time is rapidly expanding and the various types of sensors are becoming ubiquitous. Location sensors are integrated into a large number of personal devices including PDAs, smartphones, and watches [104, 105]. Among the sensors used to acquire spatiotemporal data, the most popular is the Global Positioning System (GPS). GPS receivers are commonly embedded into vehicles for trip navigation, sports watches to track and monitor personal progress in hiking and running, and smart phones to provide general purpose location aware querying. The increased availability of positioning data has given rise to location based services (LBS), which utilizes a user's current position in order to personalize results. For instance, suppose a user searches for coffee shops on her smartphone. The query results will be filtered using both the relevancy from the search string as well as her position to return popular coffee shops which are physically close.

As an example of the availability of location data, figure 1 shows GPS data uploaded to [106] by users in a region of Los Angeles, CA along with a comparison to the underlying road network of the same area. From the figure, we can see two things: first, there is a large amount of spatiotemporal data available online. Second, the data is generated over several modes of transportation. Given how well the GPS trajectories outline the heavily traveled roads, it is clear that most of the data is collected while users are in vehicles. However, upon closer inspection, we can see that some trajectories cannot possibly be associated with an automobile and therefore must have been collected using some other mode of transportation (i.e. train, subway, or walking).

The adoption of new LBS has increased nearly as quickly as the technologies have become available. As of 2011, approximately 75% of smartphone users are using their devices for navigational purposes (e.g. driving directions) and 95% use their smart phones for location based searching [104, 105]. Coupled with the increasing number of smartphones year after year (approximately a 13% increase from 2010 to 2011 [104]), this signals a steep upward trend for LBS.

Given the availability of large quantities of spatiotemporal data, it is possible to ask several interesting questions about object movement. For example, assume we have a database containing the current positions of a



(a) GPS locations from trajectories (b) Corresponding map of Los Angeles, CA

Figure 10.1. Figure (a) shows a scatter plot of GPS points of approximately 200 GPS trajectories. Figure (b) is a road map of the underlying region of Los Angeles, CA, from which these points were collected. In addition to providing an outline of the highly traveled roads in LA, it can be seen that the GPS data represents multiple modes of transportation (i.e. automobile, train, and walking).

fleet of taxis picking up and dropping off customers throughout the city. There are several interesting queries and data management problems in this setting. First, given the current position of each taxi as well as the state (occupied or unoccupied), how can we efficiently direct the nearest unoccupied unit to respond upon receiving an incoming call? As taxis are continually moving, how can we keep the information in the database current in an efficient manner? Secondly, if updates occur only intermittently, what is the most accurate approach to answering queries when data may be stale? Third, can we identify any interesting movement patterns? Can we identify points of interest from common stops made throughout the city? Is it possible to infer the efficient routes for a particular origin and destination given historical movement patterns?

The examples presented above introduce problems in three different areas of managing spatiotemporal data: (i) querying and indexing, (ii) tracking, and (iii) mining. Querying mobile objects, like any temporal data, introduces challenges in defining expressive predicates that properly handle the time domain. Additionally, constructing and maintaining an index structure to efficiently process queries over mobile objects is difficult due to the high frequency of necessary updates. Because the values (i.e. location) are constantly changing, the query workload is skewed to become update-centric, forcing the index to update its structure more frequently. Unless the index is specially designed for such a query workload, frequent updates can be very costly and even outweigh any benefit the index structure provides for query processing.

Tracking is critical to managing both the spatial and temporal uncertainty in an object's position. Accurate tracking is challenging, especially at the database scale (i.e. tracking thousands to hundreds of thousands of objects), due to the computational constraints. Inferences and predictions about an object's position must be made quickly, and should use all of the data that has been observed thus far.

The difficulties in mining for patterns in spatiotemporal data are similar to those mentioned for querying. Core mining problems, such as that of identifying groups or clusters, is made significantly more difficult when the data change positions over time. New definitions and objectives must be defined which take into account not only the current data configuration, but also the past (or predicted) configurations.

Additionally, the problem of data uncertainty is inherent in all areas of managing spatiotemporal data. Despite technological improvements, the ability to localize mobile objects is still only available up to a degree of error. Due to the nature of dealing with inexact data, new approaches to indexing, querying, and mining are necessary to effectively account for ambiguities in the data [32, 15, 82, 31]. Although data uncertainty spawns from a variety of sources, it can be broadly categorized as one of two types: *spatial* and *temporal* uncertainty.

- **Spatial Uncertainty** is uncertainty in the location of an object at the instance an observation is made. That is, spatial uncertainty describes the limitations of a sensor to provide an accurate reading of an object's position. For example, high quality GPS sensors typically provide measurement accuracy in the range of $1 - 10$ meters, lower quality hardware is in the range of $10-50$ meters, and localization from cellular tower triangulation may resolve position to only within $100 - 2,000$ meters [6, 79].

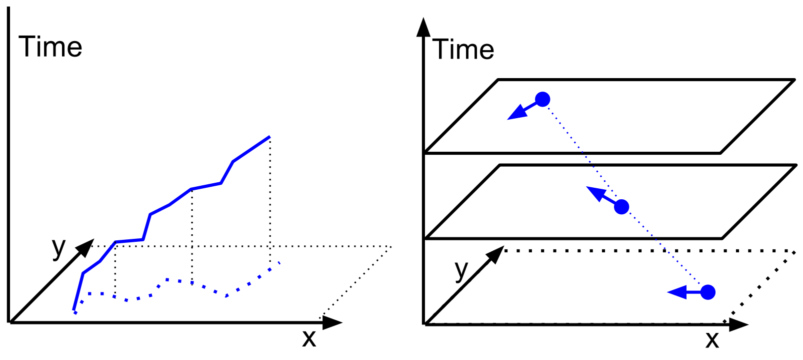
- **Temporal Uncertainty** is the uncertainty in an object's position since the previously received update. Temporal uncertainty arises due to the update schedule of how frequently an object will send information about its position to a database. Since objects may move continuously but only report their positions intermittently, there is a time-lag in which the database contains stale information. In several datasets, GPS traces have shown incredible variance in the frequency with which measurements are provided. The temporal resolution ranges from very high (1 second intervals) to very low (> 2 min. intervals) [97].

In this chapter, we introduce the types of problems that arise from managing spatiotemporal data and survey the recent research that addresses these issues. In our review, we attempt to cover work on data management, object tracking (processing updates of moving objects), and mining spatiotemporal data. Our aim in this chapter is twofold: (i) to provide a review of the recent developments in each of the different areas of study relating to spatiotemporal data and (ii) to introduce work on tracking and show how it relates to the database-centric research (e.g. querying, indexing, and mining). While we attempt to provide a broad overview of recent work in all of the mentioned areas, we pay special attention to work which explicitly manages uncertainty in the spatiotemporal data.

The rest of the chapter is organized as follows: first we will review work on data management for spatiotemporal data, including indexing and querying, in section 2. Next, we introduce the problem of tracking and review some core and recent developments in that area in section 3. In section 4, we review some recent work on mining spatiotemporal data in three broad categories: (i) clustering, (ii) popular route discovery, and (iii) identifying mobility patterns. We conclude by discussing directions for future research.

2. Data Management for Mobile Objects

Database management systems for spatiotemporal data can be characterized as one of two types: *spatiotemporal database systems* (STDB) and *moving object database systems* (MOD). Both are used to manage data collected from mobile objects, however, the specific problems each solves is quite different. STDBs store the complete historic trajectory of each object, and thus allows users to answer complex queries about user movement *over time.* For example, "find all users that passed through region A between 8 - 10am, through region B between 1 - 2pm and region C between 4 - 7pm". In contrast, MODs only maintain the current position of each object along with each object's velocity or heading information if it is available. Therefore, while STDBs contain more complete information, MODs provide access to location data that is consistently current (see figure 2). MODs are well suited to answer queries about the current (and near-future) configuration of mobile objects. Both types of database systems present a unique set of challenges due to the data they manage. In this section, we introduce both systems, as well as some recent work which addresses the challenges in efficient indexing and query processing. In addition to the general issues with managing spatiotem-



(a) Trajectory Object (b) Moving Object

Figure 10.2. Different types of data that may be extracted from mobile objects. In (a), a full trajectory is stored which describes an object's movement in space over a historic time interval. In (b), the position and velocity of an object are stored at the current time, however, historic information is not maintained.

poral data, we also introduce some specific problems in handling data uncertainty.

Many of the index structures for spatiotemporal trajectories are based on the R-tree [28, 5, 73]. For a comprehensive review of spatio-temporal indexing methods we refer the interested reader to [57] and [55].

2.1 Spatiotemporal Database Systems

A STDB allows the user to efficiently query the historic movements of a set of objects over a period of time (in the past). Although queries over spatiotemporal data can be quite complex [29], the basic queries that every STDB should answer are time-interval *range* and *nearest-neighbor* queries. A time-interval range queries answers the question "which objects traveled through region R between the times t_{start} and t_{end}?" (note that t_{end} is some time point in the past). For instance, this could be used to find out which vehicles traveled within a city during rush hour. The semantics for the time-interval nearest neighbor query may change slightly between systems, but the basic idea is to return the set of objects closest to some query point over a given time interval.

Pfoser et al. [66] distinguish between two types of queries on trajectories as **topological queries** and **navigational queries**. Topological queries are concerned with finding a set of trajectories that satisfy some spatial and temporal constraint. Range and nearest-neighbor queries over a time slice or interval are prime examples of topological queries. Navigational queries are based on derived information extracted from the trajectory and may involve dynamic information about the objects such as *speed* and *heading.* For example, it may be interesting to identify an object's top speed over a time interval, or find the set of objects with a particular heading at a specified time. Combining topological and navigational queries provide a powerful approach for analyzing complex spatiotemporal patterns.

To efficiently answer topological queries over trajectories, Pfoser et al. [66] introduce two index structures, the spatiotemporal R-tree (STR-tree) and the trajectory bundle tree (TB-tree). both structures naturally extend the basic R-tree [28] to handle trajectories in a more efficient manner. The problem with directly applying the R-tree to index trajectories is the amount of *dead space* typically incurred. To maintain a nearly $O(logN)$ access time, whole trajectories must be stored as single units; this reduces the discriminative power of the index structure due to the large area necessary to bound such a region. Alternatively, splitting each trajectory into a set of line segments improves the discrimination of each bounding box, however, access time is now dependent on the length of each trajectory (i.e. access time $O(klogN)$).

The approach utilized in the STR-tree is to segment trajectories while keeping parts of the same trajectory close within the index structure to improve the efficiency of queries over large time intervals. As compared to the R-tree, the major contributions of this index are new *insertion* and *split* methods which provide different optimization criteria. The authors assume that each trajectory is associated with a unique identifier and has already been partitioned so the index may already contain a partial trajectory for an object. The insertion algorithm first attempts to fit the new segment in the same minimum bounding box (MBR) as the previous segment from the same trajectory. We may split this index node if it is full as long as the parent is not full. Otherwise, the trajectory segment is inserted using the original *ChooseLeaf* algorithm, which locates the leaf node for which inserting the current segment will incur the lowest cost in terms of MBR overlap, area and dead space [28]. The split algorithm also considers how pairs of trajectory segments are related when optimizing the split. Segments that are not related to any other segments in the node (i.e. they come from different objects), may be placed into the new node, otherwise, if all segments are related the node is partitioned by time such that the newest segments will remain together. In general, the insert and split procedures first optimize for trajectory preservation (i.e. keeping segments of the same object nearby in the index structure) and then for spatial closeness (i.e. minimizing the increase in an MBR).

In contrast to the STR-tree, the TB-tree takes a more dramatic approach and groups segments of the trajectory to *ensure* that they are all 'bundled' together for fast retrieval. The leaf nodes of a TB-tree are forced to only contain segments belonging to the same trajectory, thus making much larger concessions in MBR overlap. The insertion strategy for the TB-tree is simply to find the previous segment of the trajectory being inserted and place this segment in the same MBR. If this is not possible, instead of splitting which would break apart segments from the same trajectory, a new node is constructed for the segment and is inserted in the first non-full parent. All of the leaf nodes containing a trajectory are connected through a doubly-linked list so that an individual object can be retrieved from any individual segment.

As compared to a 3-dimensional R-tree, both the STR-tree and the TB-tree result in more compact index structures with space utilization, or how tightly packed the nodes of the index structure are, approaching 100%. Additionally, both structure provide improved query times for range queries over a small numbers of moving objects, however, as the number of objects increased, the R-tree typically provided fewer node accesses. However, for *combined queries*, which retrieve trajectories through various means, the TB-tree outperformed the STR-tree and the R-tree by an order of magnitude even for a large number of mobile objects.

Chakka et al. [11] introduce a scheme for indexing spatiotemporal trajectories called *Scalable and Efficient Trajectory Index* (SETI). The idea proposed by the authors is that splitting the space dimensions from time, allows spatially close trajectory segments to be grouped together *and then* indexed by time, which provides a more compact and efficiently searchable structure. Their approach is to first statically partition the space over which objects move and then maintain an index structure only over the time intervals of segments contained within each disjoint spatial partition. Each trajectory segment in a space partition is indexed with an R-tree by a two dimensional point representing the start and end time of that segment.

Queries are processed using a spatial and temporal filtering followed by a refinement step to construct the answer set. The spatial filtering simply returns all of the spatial partitions contained within the spatial partitions that are contained within (or overlap with) the query region. The subsequent temporal filtering simply poses a range query to the R-tree in each spatial partition to retrieve the individual trajectory segments. A refinement step is necessary only over those spatial cells that partially intersect with the query region.

This conceptually simple idea of splitting the time and space dimensions was shown to provide significant improvements in query processing time over TB-tree. Experimental results show that SETI consistently provides lower query processing times for spatial range queries over time

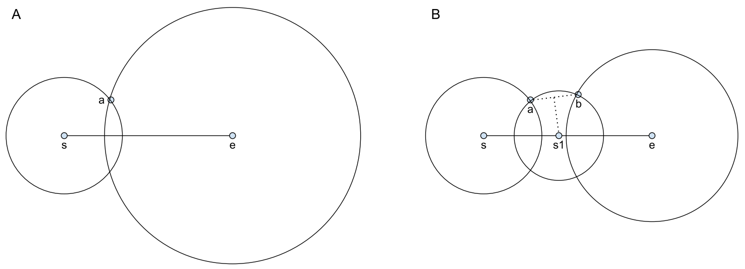


Figure 10.3. In (A), only point *a* has been processed and is thus the NN for both SPs *s* and *e*. In (B), after processing point *b*, we update the NN of *e* and create a new SP *s*1.

intervals (as well as time slices). However, it is not clear how SETI compares to TB-tree for *combined queries*, in which the query is both topological and navigational (mentioned in [66]), where processing may be required to access different segments of the same trajectory (which TB-tree does efficiently). A similar approach was simultaneously developed by Song et al. [75] in which the authors split the space and time dimensions.

In addition to the range query, nearest-neighbor and *k*-NN queries are also fundamental for any spatiotemporal data management system. For identifying nearby objects in the context of trajectory data, the continuous nearest-neighbor (CNN) query has been proposed in which a sequence of nearest neighbors are returned such that the nearest-neighbor (NN) is known for every time interval [27, 77]. Tao et al. [77] introduce the CNN query and develop an efficient query processing algorithm. The objective in processing a CNN is to identify, for every time range within the query interval, the nearest object to the query trajectory. The authors use a geometric approach in which a set of *split points*, locations at which the NN of the query trajectory changes, is maintained and incrementally updated during processing. Trajectories are processed by considering each line segment separately and aggregating the results through post-processing. The algorithm starts with the end points of the line segment being the two split points (SPs). Objects (i.e. spatial points) in the database are processed sequentially. For each object, we first check if it is the NN to any split points. This is done by maintaining a circle centered at each split point such that the radius is the distance to the closest (known) object. If a new object lies within this circle, then we must update the SPs and adjust the radii. Updates to SPs are made by computing the perpendicular bisector, the point at which this line intersects the trajectory will become a new SP. This process is shown in figure 2.1.

More recently, Guting et al. [27] introduce algorithms that provide more efficient query processing efficiency of CNN queries. The authors assume the trajectories are indexed using a $3d$ R-tree and they develop a *filter-and-refine* approach for processing queries which uses geometric pruning techniques. In the filtering step, we traverse the R-tree, pruning nodes from the candidate set based on the geometrical properties of the bounding box. The refinement step performs a modified plane-sweep algorithm over the candidate set of trajectories to test over what time intervals, if any, the trajectory is the nearest-neighbor to the query. The authors experimentally show that their method offers significantly faster query processing time and fewer disk accesses as both the size of the database and the query range increase as compared to other CNN processing algorithms.

Trajcevski et al. [81] also develop a query processing algorithm and index structure for the CNN query, however, they address this query when trajectories are uncertain. An uncertain trajectory is modeled as a cylinder in which the mobile object may have traveled anywhere within the enclosed area (i.e. the object's true location is assumed to be uniformly distributed within the cylinder). Further details about the uncertainty model for trajectories and how it is used in general query processing is provided in [80, 82]. The authors first extend the earlier work by allowing queries over pairs of objects in the database instead of known regions of interest by specifying a query as an uncertain trajectory [82]. The authors then show that they can order objects by their probability of being the nearest neighbor to the query (at any given time) by ranking objects according to their expected distances to the query location for any symmetric distribution function. This provides an ordering in which to process the objects (using a similar methodology as introduced in [15]).

To identify which objects are the nearest neighbors over a given time interval, the authors of [81] propose a hierarchical data structure such that for each node in the tree, that object has the highest probability, besides the parent, of being the nearest neighbor to the query within the time interval bounded by the parent. That is, the first level of the tree would be the nearest neighbors, each over its disjunct time interval. The second level would be nodes that are second nearest neighbors over disjunct time intervals bounded by their parent, and so on. They show how to construct this structure and use it to answer the NN query and several deviations.

In addition to range and nearest-neighbor queries, other, more complicated, queries have been defined on spatiotemporal data [4, 29]. For instance, Bakalov et al. [4] introduce a spatiotemporal join query that finds all pairs of objects with (nearly) matching subtrajectories over a given time interval. This query could be useful in clustering subtrajectories or simply identifying groups of mobile objects that traversed similar paths. More formally, the authors define the spatiotemporal join query to take two sets of trajectories, a distance threshold, ϵ, and time interval, δ_t, and returns all pairs such that there exists a subregion of each trajectory of length δ_t where the distance between the subregions is at most ϵ. The query is called a time relaxed spatiotemporal trajectory join (TRSTJ), because the query only constrains the length of the trajectory subregion, not the specific start time. To answer the TRSTJ, the authors propose a filter and refine approach where they use a compact trajectory representation and lower bound the Euclidean distance between trajectories.

2.2 Moving Object Databases

Unlike STDBs, a *moving object database* (MOD) only stores the *current* position of each object. MODs constantly contain up-to-date information about the location of each object and are therefore useful in real-time applications of managing a large number of mobile objects (e.g. navigation or emergency response dispatch). Similar types of queries are supported on MODs, however, the time intervals over which the data may be queries is limited to the present and future. Instead of asking where an object *has* been, a MOD answers the query "where is object A now"? or "where will it be in 5 minutes"? In order to answer such queries, the system must have some method to predict the future location of each object given its current location and its velocity. We will cover different approaches for updating and predicting location information in detail in section 3.

Similar to STDBs, a primary difficulty for MODs is constructing and maintaining an index structure. However, the *cause* of the difficulty in the two systems is quite different. Here, the problem is keeping the location information for all objects up to date, which requires frequent updates to the index. Continually modifying an index structure is likely to cause the discriminative capabilities of the structure to degrade over time unless special care is taken.

Cheng et al. [15] introduce a method for answering range queries and nearest neighbor queries with probabilistic guarantees when the location of objects is uncertain. The authors propose an uncertainty model that specifies a bounded region within which a mobile objects may be located with equal probability. The authors were the first to define and propose a solution to the probabilistic nearest neighbor query (PNNQ). They use a filter-and-refine approach in which objects located far away can be pruned (i.e. their shortest distance is larger than the longest distance of a closer object). After filtering objects that obviously do not satisfy the query, the space in which objects need to be evaluated (i.e. over which the integral needs to be performed) can be bounded. Lastly, the remaining objects are evaluated by computing the integral in Eq. 10.1.

$$p_i = \int_{n_i}^{f} p_i(r) \prod_{k \neq i}^{|S|} (1 - P_k(r)) dr \tag{10.1}$$

In Eq. 10.1, n_i is the shortest distance from object O_i to the query point q, and f is the bounded region over which objects must be evaluated. The integration can be interpreted as the probability that O_i is at a distance of r from q while all other points in the candidate set, S, are at a distance greater than r, which is evaluated over all possible values of r. The authors introduce a more efficient approach to evaluating this integral by sorting S and breaking up the integration limits to only the region in which $p_i(r)$ is non-zero. Furthermore, they also utilize an R-tree based index structure, called the velocity constrained index (VCI) [68], to improve processing time for large datasets.

Saltenis et al. [71] introduce a general index structure for efficiently processing queries on mobile objects called the time parameterized R-tree (TPR-tree). The idea behind the TPR-tree is to allow MBRs to grow and change position as a function of time in order to reduce the number of necessary updates to the index structure from objects moving (see figure 2.2). The authors propose *conservative bounds*, in which the MBR expands by the maximum velocity of the contained set of objects in each dimension. That is, considering the number line in which values to the left are smaller and those to the right are larger, the MBR expansions in both directions of one dimension are given in Eq. 10.1.

$$\begin{aligned} \overleftarrow{x} &= \min(o_i.\text{pos}(t)) - \min(o_i.\text{vel})(t_{\text{diff}}) \\ \overrightarrow{x} &= \max(o_i.\text{pos}(t)) + \max(o_i.\text{vel})(t_{\text{diff}}) \end{aligned} \tag{10.2}$$

Non-leaf level MBRs are constructed by aggregating the bounds from each of their children. For each dimension, the expansion in each direction is the maximum over all of its children.

Although the TPR-tree may be used to index mobile objects indefinitely, it is optimized only for a particular time horizon, after which the performance of the index may deteriorate. Specifically, the index structure requires two parameters: the querying window, W, which defines how far queries may look into the future and the index usage time, U, which defines how long users will query the index. Combining both of these values we get the time horizon, $H = U + W$, which is the time over which the index structure must be able to answer queries. Using these values, the index structure can be optimized since there is a limit as to how far into the future the index will be required to answer queries.

The TPR-tree uses the same optimization concepts from the R*-tree [5], specifically the area of MBRs, perimeter length, and overlap area, except they are minimized over a time horizon instead of simply minimizing the current state of the index. If we consider an objective function that minimizes overlap between MBRs $A(t)$, then we would like to optimize this function over the given time horizon: $\int_t^{t+H} A(t)dt$. The operations used in the TPR-tree are all analogous to the R*-tree, with the only difference being the function *splitNode*(), which determines how objects contained in full nodes should be partitioned. Since MBRs are dynamic, the TPR-tree split is based on optimizing over both the current positions *and* velocities.

Query processing in the TPR-tree is similar to the R-tree [28]. Since all of the MBRs are parameterized by time, answering a time-slice range query remains completely unchanged. To answer a time-interval range query or a moving range query, it is necessary to identify intersections between the MBR and the query region over the specified time interval in each dimension and check that the intersections happened over the same time interval in all dimensions. This is computed in a straight forward manner by comparing line segments of the query region and MBR boundaries in each dimension over time (the lines describing movement in the both the x and y dimensions over a time interval).

The simplicity of the TPR-tree has made it a popular choice for indexing MODs. Tao et al. [78] have extended the original work by introducing the TPR*-tree, where they improve upon the optimizations introduced in [71]. The TPR*-tree tightens the bounds of the MBRs thus making the index more discriminative. Experiments show substantial improvements in performance over the TPR-tree for large datasets. To handle spatial uncertainty in mobile objects, Hosbond et al. [31] adapt the TPR-tree to index moving objects with inexact location information. The authors essentially model location uncertainty as a Δ term in which each object can vary some amount from its stated position. They incorporate this error term into the MBR parameterization to guarantee each object is properly bounded over time.

In [98], the authors present both a movement model for mobile objects with uncertainty as well as an index structure for efficient query processing. The proposed uncertain moving object model defines a probability distribution over location and velocity (independently). The model works by griding the space at some resolution and assigning each cell

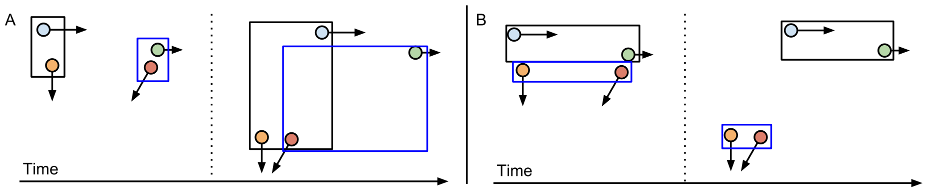


Figure 10.4. An example of how the TPR optimizes MBRs over time. The right panel (A) shows the MBR optimized for the current configuration, whereas the left (B) considers both the current object position as well as velocity.

a probability thereby constructing a discrete distribution over possible locations (velocities). Combining the current location and velocity distributions, it is then possible to predict an object's future location. Utilizing a gridded space technique allows the representation of arbitrary distributions, however, it also incurs a heavier cost for prediction since sampling must be employed. The authors then adapt the B^x-tree [35] to handle objects with uncertain locations. For each object, every non-zero probability grid cell of the object's location is inserted into the index structure, thus accounting for location uncertainty. Thus a trade-off is made between location accuracy and the space consumption and search time of the index structure.

Chung et al. [17] develop query processing and indexing techniques to efficiently answer probabilistic range queries when uncertainty in the trajectories is described with Gaussian noise. The authors assume that objects are moving along a known path and thus treat each object as moving in 1-dimensional space over time. The movement uncertainty for each object is modeled by Brownian motion in which the object's velocity is normally distributed. To avoid the linear scan of the database, the authors apply the Hough transform, which maps lines to points, to the expected trajectory of an object and use an R-tree to index these points in the dual space. Queries are then transformed in the same manner, although some additional work is required since the trajectories contain some location uncertainty. The query region is expanded depending on the probability threshold issued in the query. Although this approach provides efficient processing of probabilistic range queries, it makes the limiting assumption that objects move in 1-dimensional space.

Chen et al. [12] perform an experimental investigation of the effectiveness of several index structures for MODs under various conditions. The authors compare the TPR-tree [71], TPR*-tree [78], B^x-tree [35], B^{dual}-tree [95], STRIPES [65], and RUM*-tree [92]. The R-tree with Update Memo (RUM*-tree) employs main-memory memos to help avoid frequent disk accesses for deleting old record entries. This reduces the cost of an updating an object's position simply to that of inserting a new object into the R-tree and old entries are deleted in batches. The B^{dual}-tree and STRIPES are both indexes that utilize dual space. The B^{dual}-tree maps an objects location and velocity to a 1-dimensional point using a Hilbert curve. STRIPES maps a mobile object to a four dimensional point and applies a quad-tree based structure to index the space. The experimental evaluation provides a thorough comparison between the different index structures on several aspects of updating and querying mobile objects. Although each index performs well under certain circumstances, the B^{x}-tree consistently performed near the top over all experiments never costing more that 2X the best index structure. While the TPR-trees were *the* most efficient indexes for the querying experiments (in terms of time and I/O), they also performed the worst for updates. Overall, this work highlights the need to clearly identify the expected query workload distribution in order to select the most efficient structure.

A problem that has received far less attention in the spatiotemporal data management community is that of how to most efficiently update the database with new positions for mobile objects. Wolfson et al. [90] address this problem by framing it as an optimization problem. The authors provide a model of the cost to poll an object in order to update its location and a cost for answering a query given the current (estimated) location uncertainty (i.e. how much error are we willing to tolerate) and aim to minimize their total costs while handling a query workload. The authors also introduce a spatial indexing scheme for moving objects that considers the possible location error. For this, they use an R^{+}-tree [73] over three dimensions (the x, y plane and time). The index is updated only when objects report their position to the database (using the derived update policy).

2.3 Mobile Objects on Road Networks

The queries and index structures discussed thus far consider the case of unrestricted movement, however, movement is often restricted to a transportation network (i.e. road network). If the structure of the underlying transportation network is known, then incorporating this additional information can provide more accurate tracking and prediction of object locations (and therefore more meaningful queries). Restricting movement to the network structure brings about its own set of challenges.

Zheng et al. [100] address the problem of managing moving objects on road networks where the specific path traveled by an object is uncertain. The authors represent the uncertain locations of objects as time-dependent probability distributions and develop methods to efficiently query the objects. They represent a road network as a graph in which the edges have two attributes, the length of the road and the maximum allowed speed along that path. There are two kinds of uncertainty in this work: *path uncertainty*, that is, which path (sequence of edges) did the object take to get from the last position to the next, and *location uncertainty*, given the path, the actual location of the object at time t. Path uncertainty arises from the object selecting a specific route due to some criteria (i.e. traffic, road work, landmarks) and location uncertainty arises from the fact that we don't know how fast the object was moving the entire time, though we can bound this by the maximum allowable speed on each segment.

In [100], the authors propose a novel index structure for processing range queries on uncertain objects over road networks called the Uncertain Trajectory Hierarchy (UTH). The UTH consists of three levels: (i) a network edge hash table, (ii) a movement R-tree, and (iii) a trajectory list. The edge hash table allows fast retrieval of any specific road segment. The movement R-tree is an index over time for a specific road segment which allows us to identify the set of objects traveling along a road over a given time interval. Lastly, the trajectory list stores the actual trajectory for each object. A filter and refinement approach to answering uncertain path queries on road networks is developed by the authors by leveraging the UTH index.

Hua and Pei [32] also study the problem of answering path queries on road networks, however, in this case it is the edge speeds (or flow) that are uncertain, not the positions of the moving objects. The authors introduced a network model in which adjacent edges in the road network were correlated and they introduce exact and more efficient approximate methods for computing the probability of paths. They also provide an A^*-like algorithm for efficiently finding the most probable paths under specific speed constraints.

Kim et al. [43] develop an index structure for objects with movements that are restricted to a road network called *Indexing Moving Objects on road sectors* (IMORS). The index is composed of a static component, that is made up of an R^*-tree over the road segments, and a dynamic component, which contains a mapping from road segments to mobile objects. The dynamic module of the index structure is essentially a table indexed by object identifiers' that contains attributes of the object as well as its location and a pointer to the road segment it currently occupies. To process an update, the object record is found, its location is updated and we check if it has changed road segments. If so, we search the R-tree based on the new coordinates and fix the pointer in the object record to point to the updated road segment. The IMORS provides significant efficiency gains in processing updates over the TPR-tree (to which it was compared in the experiments), by utilizing the static nature of the road network. Since locations of roads do not move, object positions are modified by updated the road segment to which they point, however, all searching is done over the road network indexed R^*-tree.

When dealing with mobile objects that have restricted movement, such as automobiles on a road network, this gain in efficiency in indexing and query, and increased accuracy in predicting future locations is common. The idea of integrating all available information about the object movement is important as it allows us to identify object positions and movement more reliably. Utilizing extra information can help combat against the problems of spatial and temporal uncertainty when managing spatiotemporal data.

Additionally, there is a plethora of work in the area of efficiently processing routing queries for navigation queries. For instance, computing the quickest (or most efficient) route between two points considering the dynamics of the road network is of great interest. Incorporating extra information like speed limits, predicted traffic patterns, and road closures can greatly improve routing and therefore help relieve traffic congestion.

Nikolova et al. [63] show several theoretical results related to problems of optimal route planning. The authors show the surprising result that the problem of finding an optimal route *and* start time can be solved using standard shortest path algorithms for certain cost functions while the optimal route planning problem when the start time is fixed is NP-hard. Similarly, Wilkie et al. [89] consider the routing problem on a stochastic traffic network, however, they integrate the effect of the planner into future predictions. That is, vehicles will query the planning system to ask for directions, assuming the vehicles stick to these routes, the planner has additional information about the state of the traffic in the future. Malviya et al. [54] introduce an approach for continuously monitoring a set of shortest path for mobile objects. The authors first precompute a set of *good* routes using a road network and historic travel times and then re-rank these paths by integrating information about real-time traffic. Gonzalez et al. [24] developed a fastest-path algorithm which utilizes historic information about traffic and weather conditions to provide reliable routes. The authors also introduce a network partitioning algorithm which reduces the search space by focusing on the

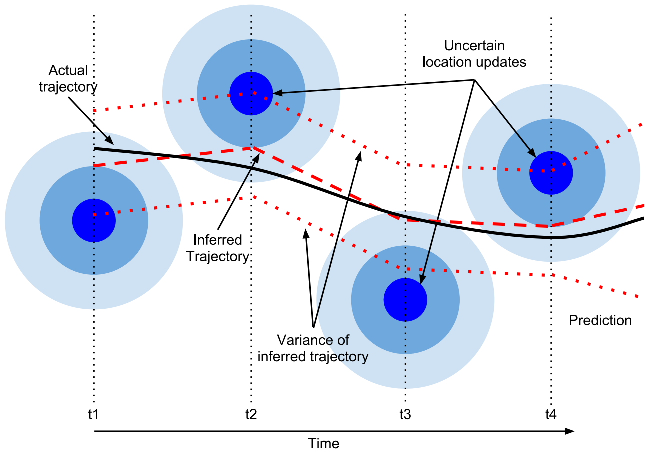


Figure 10.5. An example of turning noisy location observations into a trajectory. The filtering model attempts to identify the most likely path that would have generated the noisy observations given a specific movement model (linear in this case: $X_t = X_{t-1} + \dot{X}_{t-1}\ (\Delta t)$).

most traveled roadways (i.e. highways) and only expanding extra edges when they have historically exhibited improved performance.

3. Probabilistic Models for Tracking

Processing location updates from mobile objects is a crucial component of managing spatiotemporal data because the raw locations obtained from a sensor are often noisy. Even GPS has been shown to contain errors on the order tens or hundreds of meters [6]. Because locations at adjacent time steps are not independent (see figure 3), it is possible to incorporate information about the dynamics of the mobile object in order to improve upon its current position. This is exactly what *tracking* accomplishes. By explicitly modeling the dependencies between locations observed at adjacent time steps, we can filter the raw data to produce a cleaner estimate of the object's trajectory which accounts for noise in the sensor as well as the system dynamics (e.g. friction). Additional constraints over an object's possible movements (e.g. road network) can also be incorporated to further improve the filtered trajectory.

Due to the inherent uncertainty in the problem of tracking mobile objects, probabilistic models such as dynamic Bayesian networks (DBNs) have been applied to several tracking scenarios with great success [50, 60]. In this section, we will first introduce the basic problems involved in tracking mobile objects. We pose the tracking problem as Bayesian

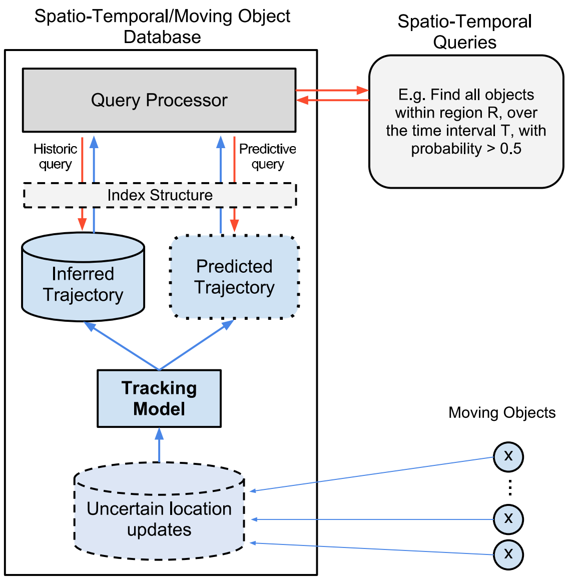


Figure 10.6. The (general) architecture of a STDB or a MOD. Both systems require an algorithm to turn the raw location estimates from the moving objects into a usable trajectory. In the case of the MOD, this conversion happens in real-time as new locations are made available (filtering) and in the the case of STDB, the complete sequence of locations and time stamps are available (smoothing). The quality of the inferred locations directly affects query accuracy, thus the tracking algorithm is a vital component of the data management system.

filtering task, then we review the Kalman filter model (KFM) [39, 88] in detail. Lastly, we discuss some methods which address the tracking problem when objects are constrained to move on a road network. Table 10.1 contains a list of notation used throughout this section.

3.1 The Tracking Problem

The general task of tracking can be formulated as a Bayesian filtering problem where we would like to estimate the value of an unobserved random variable x, given an observation z. Because x defines the state of a mobile object (position, velocity, altitude, etc.), this value will change over time and we would like to re-estimate it each time a new observation becomes available. The general problem of Bayesian filtering is then to update our beliefs about x_t, incorporating all available information (i.e. $z_{1:t}$) by computing the posterior distribution $p(x_t|z_{1:t})$. To keep our presentation clear, we will assume that the state of a mobile object, x, is described by a vector containing the object's current position and ve-

Table 10.1. Notation

Notation	**Explanation**
x_t	Belief state (hidden) for at time t
X	Location component of belief state
$\dot{X}$	Velocity component of belief state
$z_{0:t}$	Observations from time $0..t$
A	Deterministic transition matrix
H	Deterministic observation mask
Q	Covariance matrix for the movement dynamics. Determines the amount of uncertainty we have in our transition model.
R	Observation covariance matrix. Determines the amount of uncertainty we have in our observations.
$\mathcal{N}(\mu, \Sigma)$	Gaussian distribution with expected value μ and covariance matrix Σ.

locity ($x_t = [X, Y, \dot{X}, \dot{Y}]$) and the location measurement, z_t is described by a position ($z_t = [X, Y]$).

$$\begin{aligned} p(x_t|z_{1:t}) &= \frac{p(z_t|x_t)p(x_{t-1}|z_{1:t-1})}{p(z_{1:t})} \\ &\propto p(z_t|x_t)p(x_{t-1}|z_{1:t-1}) \end{aligned} \tag{10.3}$$

The first term in Eq. 10.2 is the likelihood function which describes the relationship between x_t and z_t (i.e. describes the sensor error). For example, GPS sensors are typically assumed to have error that is normally distributed about the true location. In this case, $p(z_t|x_t) = \mathcal{N}(z_t; x_t, \sigma)$, where σ describes how much variation we expect to see in the measurement.

The second term, $p(x_{t-1}|z_{1:t-1})$, is the posterior distribution of x_{t-1}. That is, this term is the result of filtering at the previous time step. From this equation, we see that it is possible to recursively update our belief about the state of a mobile object online (as new data arrive). All of the information about x_{t-1} is captured in $p(x_{t-1}|z_{1:t-1})$, thus there is no need to revisit old datum. Lastly, the denominator is the marginal probability of the sequence of observations. Since the observations remain fixed (this data is observed), this term may be considered a normalizing constant and ignored for our purposes. For a readable introduction to Bayesian statistics in a more general context, we refer the interested reader to [30].

While tracking is an *online* problem (i.e. updates must be made as data arrive), in general there are three types of inference we may be interested in for any DBN: (i) prediction, (ii) filtering and (iii) smoothing.

A graphical representation of each type of inference for a simple chain model with a single hidden variable is shown in figure 3.1. Prediction and filtering are computable online, while smoothing requires observations from future instances in order to *correct* our estimate of an object's state given more information. In the context of data management systems, filtering and prediction would be used in a MOD, while smoothing would be used to obtain complete historic trajectories of mobile objects for a STDB.

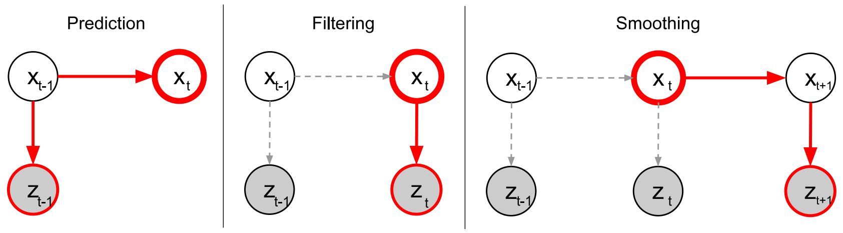


Figure 10.7. KFM Inference steps: **Prediction** predicts the value of x_t from the last known position of the object, z_{t-1} and the movement model. This is computed in the graphical model by integrating out the unobserved value of x_{t-1}. **Filtering** *corrects* the value of x_t by incorporating the latest uncertain observation, z_t. *Smoothing* updates the filtered estimate for x_t by also incorporating information from later observations ($z_{1..T}$).

In general, it is rarely possible to evaluate Eq. 10.2 exactly, and approximate methods have become quite common [13, 19, 59]. However, under certain modeling assumptions, exact inference is tractable. Next, we introduce the Kalman filter model (KFM), a popular model for which exact inference is tractable. Due to the popularity of the KFM and its widespread adoption in tracking and sequential data processing [40, 41, 74, 91, 51, 94], we discuss this model in some depth. Our objective in the following section is to briefly introduce the Kalman filter to unfamiliar readers, including some intuition as to how and why the model works.

3.2 Kalman Filter

The Kalman filter model [39] (KFM) is a linear dynamic system that offers an efficient exact inference procedure. The efficiency stems from the fact that all of the variables in the model are assumed to come from a joint Gaussian distribution. As a result, both the observation noise (Eq. 10.4) and the system dynamics (Eq. 10.5) are assumed to be Gaussian distributions. The observation noise describes how observations are related to the actual belief state. In this case, we expect observations to be distributed normally about the true state with the degree of uncertainty given by the covariance matrix, R. The system dynamics describe how the state changes between time steps. In the KFM, we assume the current state is a linear function of the previous state (e.g. $x_t = x_{t-1} + \dot{x}_t$).

$$\begin{aligned} p(z_t|x_t) &= \mathcal{N}(x_t, R) & (10.4) \\ p(x_t|x_{t-1}) &= \mathcal{N}(Ax_t, Q) & (10.5) \end{aligned}$$

Eq. 10.5 describes the model dynamics and Eq. 10.4 describes the noisy observation process. The transition probability of the previous belief state x_{t-1} to the current belief state x_t depends on the deterministic transition matrix A and the transition covariance matrix Q. To explain the ideas behind the Kalman filter we consider a simple example in which we are tracking a moving object in one-dimension. The representation of our belief state, x, contains the object's current location and velocity. Therefore, the transition matrix $\mathbf{A}$, is defined to encode $X_t = X_{t-1} + \dot{X_{t-1}}t$ and $\dot{X}_t = \dot{X_{t-1}}$, that is $\mathbf{A} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$. The covariance matrix Q describes the noise in the dynamic process, such as head/tail winds, loss of power due to friction, changing altitudes, etc. Suppose that we are only able to observe the object's location at discrete time steps, but have no way of acquiring the velocity directly. In this case the deterministic observation mask $H = \begin{bmatrix} 1 & 0 \end{bmatrix}$, and we must infer velocity completely from the location observations. The covariance matrix R, describes how much uncertainty exists in these observations. For example, if we were using a GPS sensor to track the object, R would be relatively small, allowing the observations to vary only a couple of meters from the actual location. In contrast, if we relied on cellular tower triangulation to track the moving object, R would be very large, allowing observed locations to be hundreds of meters from the true location.

Since the state variable, x_t is assumed to be normally distributed, we need only maintain the mean and variance of the distribution at each time step to completely describe our knowledge about x_t. Because of this, and some nice analytic properties of the normal distribution, simple update equations for the necessary parameters are tractable. This simplicity makes the KFM a popular choice for modeling dynamic, real-valued data. Because of its popularity, in this section we will describe the KFM filtering and smoothing algorithms and provide the reader with some intuition as to how the update equations work. Before getting started on the KFM, we first digress slightly to review some important properties of the Gaussian distribution.

3.2.1 Joint, Marginal, and Conditional Gaussians. In this section we interchange the terms normal and Gaussian distribution. In both cases, we are referring to the same density function (shown in Eq. 10.6).

$$p(x_1, ..., x_\rho) = \frac{1}{(2\pi)^{\rho/2}|\Sigma|^{1/2}} exp\left(-\frac{1}{2}(\mathbf{x}-\mu)^T\Sigma^{-1}(\mathbf{x}-\mu)\right) \qquad (10.6)$$

For any set of random variables that are jointly normally distributed, all marginal and conditional distributions associated with any individual or subgroup of variables are also normally distributed. For example, consider the simple bivariate normal distribution $p(x, y)$ with parameters given in Eq. 10.7.

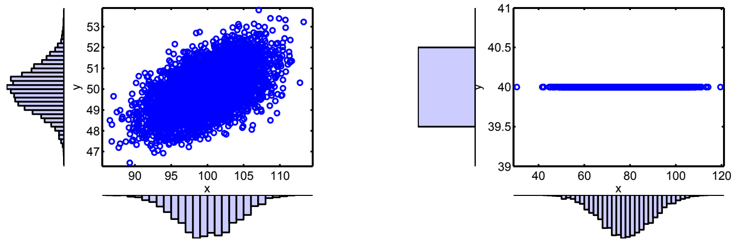
$$p(x,y) = \mathcal{N}(\begin{bmatrix} \mu_x \\ \mu_y \end{bmatrix}, \begin{bmatrix} \Sigma_x & \Sigma_{x,y} \\ \Sigma_{y,x} & \Sigma_y \end{bmatrix})$$

The marginal distribution for x is computed by integrating over all possible values of y. That is, $p(x) = \int p(x,y)\ dy$. For a joint Gaussian, marginalization or *integrating out* variables is a simple procedure, since the result is a normal distribution we must only find the mean and covariance matrix that specify the distribution. In this case, we simply take the mean and covariance sub-matrix corresponding only to the variable(s) of interest and the marginal is again normally distributed. For example, using the joint in Eq. 10.7, $p(x) = \mathcal{N}(\mu_x, \Sigma_x)$.

Marginalization is the process of simply removing a variable from our distribution. However, this process does not provide us with any additional information about the variable of interest, it only serves to simplify our distribution form by reducing dimensionality. Alternatively, it may be possible to observe the value of a variable may provide us with information about our variable of interest. In this case we are interested in computing the *conditional distribution.* That is, the distribution over x given that you have observed a specific value for y. In the case of jointly distributed Gaussian variables, the conditional distribution is again a Gaussian. The parameters for a conditional Gaussian are shown below.

$$\begin{aligned} p(x|y) &= \mathcal{N}(m_{x|y}, P_{x|y}) \\ m_{x|y} &= \mu_x + \Sigma_{x,y}\Sigma_y^{-1}(y-\mu_y) && (10.7) \\ P_{x|y} &= \Sigma_x - \Sigma_{x,y}\Sigma_y^{-1}\Sigma_{y,x}^T && (10.8) \end{aligned}$$

The interpretation of these updated parameter values is quite intuitive. For instance, Eq. 10.7 says that the updated mean is the marginal mean of x corrected by some value. This correction term depends on the coupling between the two variables which is encoded in the covariance term,



(a) Joint Gaussian (b) Conditional Gaussian

Figure 10.8. A joint Gaussian distribution of two random variables is shown in figure (a). In (b), we show the result of the distribution over x after conditioning on the value of y.

the original variance term for the observed variable as well as the shift of the observation from the expected value. Notice that if our observation matches the expected value, μ_y, or the covariance between x and y is small, then the correction term is small and thus observing y provides little information about x. Similarly, the covariance is corrected according to the covariance and variance term of the observed variable. Notice here that the correction term is subtracted from the original variance. Since the covariance matrix is positive semi-definite, conditioning on an observed value is guaranteed to decrease variance and therefore reduce our uncertainty about the variable of interest.

Figure 3.2.1 shows an example of two variables that are jointly normally distributed. There is strong correlation between the two variables and thus when we condition on y in figure 10.8(b), the marginal distribution over x changes by shifting (correction to the mean) and scaling (reduction in variance).

3.2.2 Filtering. There are three types of inference we will be interested in computing with the KFM: prediction, filtering, and smoothing. Figure 3.1 shows each of the different procedures, highlighting the variables and connections which are used in each. We will first discuss the filtering problem, updating our parameters of interest upon the arrival of new observations, which subsumes the task of prediction. Then, we will introduce smoothing, estimating parameters given past *and future* observations, which is an offline task that typically provides more accurate estimates with reduced uncertainty.

The objective of filtering is to update our estimates by incorporating the most recent observation. For the KFM, the posterior takes the form of the Gaussian distribution. This means that to describe our current belief state, x_t, we only need to compute and store the mean vector and covariance matrix. Rewriting the posterior in Eq. 10.9, we derive the functional forms of the distributions to show how we end up with a normally distributed posterior.

$$p(x_t|z_{0:t}) \propto p(z_{0:t}|x_t)p(x_t|z_{0:t-1})$$

From our model assumptions (Eq. 10.4), we have that $p(z_{0:t}|x_t)$ is normally distributed. The second term is the *predicted* belief state given all observations up to the previous time step. This distribution can be rewritten in terms of the model dynamics and a recursion term as shown in Eq. 10.8. We denote the posterior parameters of x from the previous time step as μ_{t-1} and Σ_{t-1}

$$\begin{aligned} p(x_t|z_{0:t-1}) &= \int p(x_t|x_{t-1})p(x_{t-1}|z_{0:t-1})dx_{t-1} \\ &= \int \mathcal{N}(x_t; Ax_{t-1}, Q)\mathcal{N}(x_{t-1}; \mu_{t-1}, \Sigma_{t-1}) \\ &= \mathcal{N}(A\mu_{t-1}, A\Sigma_{t-1}A^T + Q) \qquad (10.9) \end{aligned}$$

To predict the state at time t, we simply apply the model dynamics to the estimate of the state at $t-1$, integrating over all possibilities. The integration over the previous state is necessary since we are actually uncertain of the true value of x at any given time and thus must consider all possibilities. We use the second term, the posterior of x from the previous time step, weight each guess of the previous state based on our posterior distribution for x_{t-1}. We denote the predicted parameters for x_t as shown in Eq. 10.10 and 10.11.

$$\begin{aligned} m_t &= A\mu_{t-1} \qquad (10.10) \\ P_t &= A\Sigma_{t-1}A^T + Q \qquad (10.11) \end{aligned}$$

Combining the observation and prior distributions of the belief state (Eq. 10.4 and 10.8), we can reconstruct the joint distribution over x_t and z_t.

$$p(x_t, z_t|z_{t-1}) = \mathcal{N}(\begin{bmatrix} m_t \\ Hm_t \end{bmatrix}, \begin{bmatrix} P_t & P_tH^T \\ HP_t & HP_tH^T + R \end{bmatrix})$$

Algorithm 5 Kalman filtering algorithm

```
Input: $\mu_{t-1}, \Sigma_{t-1}, z_t$
  // predict the current state from previous values
  $m_t = A\mu_{t-1}$
  $P_t = A\Sigma_{t-1}A^T + Q$
  // compute Kalman gain
  $K = (P_t H^T)(H * P_t * H^T + R)^{-1}$
  // apply correction to predictions using new observation
  $\mu_t = m_t + K(z_t - Hm_t)$
  $\Sigma_t = (I - KH)P_t$
```

Conditioning on z_t, we have the measurement update step from 10.7 and 10.8.

$$\begin{aligned} p(x_t|z_{0:t}) &= \mathcal{N}(\mu, \Sigma) \\ K &= HP_t(HP_tH^T + R)^{-1} \\ \mu &= m_t + K(z_t - Hm_t) \qquad (10.12) \\ \Sigma &= P_t - K(HP_t)^T \qquad (10.13) \end{aligned}$$

Where K is referred to as the Kalman Gain. The inference algorithm for the KFM proceeds by first predicting the $t+1^{st}$ state of x which updates the parameters as described in Eq. 10.8. Then, upon observing a new measurement, z_t, we perform the *measurement correction* step, which updates the parameters for x_t according to equations 10.12 and 10.13. We show the complete filtering inference algorithm for the KFM in algorithm 5 for completeness. From the update equations above, we see how updating belief states upon the arrival of new observations can be computed efficiently, using matrix multiplications and a matrix inverse operation.

3.2.3 Smoothing.

Before we get into the smoothing equations for the KFM, we first provide some notation below to identify the different versions of parameters. The first line shows the filtered probability distribution for x_t (explained in the previous section) which we still identify with the parameters μ and Σ. The second line shows the smoothed probability distribution, for which we use the parameters ν and Φ for the mean and covariance to differentiate from the filtered parameters.

$$\begin{aligned} p(x_t|z_{1:t}) &= \mathcal{N}(x_t; \mu_t, \Sigma_t) \\ p(x_t|z_{1:T}) &= \mathcal{N}(x_{t+1}; \nu_{t+1}, \Phi_{t+1}) \end{aligned}$$

Smoothing utilizes observations from the past, present, and future to provide an improved estimate of the belief state. Inference for smoothing consists of a forward pass through the chain (i.e. filtering) followed by an additional backward recursion in which we consider future observations as well. Specifically, we wish to compute the conditional distribution shown in equation 10.13. Note that in the second step, conditioning on x_{t+1} makes $z_{t+1:T}$ independent of x_t, which is why the extra observations are dropped from this term.

$$\begin{aligned} p(x_t|z_{0:T}) &= \int_{x_{t+1}} p(x_{t+1}, x_t|z_{0:T}) \\ &= \int_{x_{t+1}} p(x_t|x_{t+1}, z_{0:t})p(x_{t+1}|z_{0:T}) \qquad (10.14) \end{aligned}$$

We recognize that $p(x_{t+1}|z_{0:T})$ defines our backward recursion, since this is the smoothed estimate for x_{t+1} given all observations. Therefore, we must derive the parameters for $p(x_t|x_{t+1}, z_{0:t})$ before we can solve the integral in Eq. 10.13. Since we are not given this distribution directly, we will start with the joint distribution over two timesteps, and continue by conditioning on x_{t+1} to get the final distribution.

$$p(x_t, x_{t+1}|z_{0:t}) = \mathcal{N}(\begin{bmatrix} \mu_t \\ A\mu_t \end{bmatrix}, \begin{bmatrix} \Sigma_t & \Sigma_t A^T \\ A\Sigma_t & A\Sigma_t A^T + Q \end{bmatrix})$$

Conditioning on x_{t+1}, we derive the following.

$$\begin{aligned} p(x_t|x_{t+1}, z_{0:t}) &= \mathcal{N}(x_t; m_{t|t+1}, P_{t|t+1}) \\ J &= \Sigma_t A^T(A\Sigma_t A^T + Q)^{-1} \\ m_{t|t+1} &= \mu_t + J(x_{t+1} - A\mu_t) \\ P_{t|t+1} &= \Sigma_t - J\Sigma_t A^T \end{aligned}$$

Plugging these values into equation 10.13, we can now solve the integral.

$$\begin{aligned} p(x_t|z_{0:T}) &= \int_{x_{t+1}} p(x_t|x_{t+1}, z_{0:t})p(x_{t+1}|z_{0:T}) \\ &= \int_{x_{t+1}} \mathcal{N}(x_t; m_{t|t+1}, P_{t|t+1})\mathcal{N}(x_{t+1}; \nu_{t+1}, \Phi_{t+1}) \\ &= \mathcal{N}(\mu_t + J(\nu_{t+1} - A), \Sigma_t - JA\Sigma_t) \qquad (10.15) \end{aligned}$$

Equation 10.14 shows the final distribution and the parameters for the smoothed estimate of x_t given $z_{0:T}$. The smoothing algorithm works by correcting the filtered estimate of x_t by trying to minimize the difference between the predicted value of the next state and the smoothed estimate for the $(t+1)^{st}$ time step. Finally, the smoothing algorithm initializes the

Algorithm 6 Kalman smoothing algorithm

```
Input: $\mu_t, \Sigma_t, \nu_{t+1}, \Phi_{t+1}$
  // predict state using filtered estimate
  $m_+ = A\mu_t$
  $P_+ = A\Sigma_t A^T + Q$
  // compute Kalman smoother gain
  $J = (\Sigma_t A^T)P_+^{-1}$
  // apply correction to filtered estimate
  $\nu_t = \mu_t + J(\nu_{t+1} - m_+)$
  $\Phi_t = \Sigma_t + J(\Phi_{t+1} - P_+)J^T$
```

smoothed parameters for the x_T to be the same as the filtered estimate for the state at that time, then proceeds backwards through the chain starting at $t = T - 1$ updating each belief state and covariance matrix using the update equations in Eq. 10.16 and 10.17.

$$\nu_t = \mu_t + J(\nu_{t+1} - A\mu_t) \tag{10.16}$$

$$\Phi_t = \Sigma_t - J\Sigma_t A^T \tag{10.17}$$

Lastly, algorithm 6 provides the update algorithm (for a single time step) of the basic Kalman smoother. The algorithm takes as input the filtered parameters of the current time step as well as the smoothed estimates from the $t + 1^{st}$ estimate and produces a smoothed estimate for state t. Similar to the filtering algorithm, we see that the updates are quite efficient, requiring only a few matrix operations.

To conclude this section, we provide a tracking example in figure 3.2.3. The red line represents the true trajectory, the blue points are the observations at each time step. The figure also plots the filtered and smoothed estimates of the trajectory along with a dashed line at a distance of $1\sigma^2$ to represent the estimate uncertainty. It is clear that the filtered trajectory is a substantial improvement over simply connecting the observations. Furthermore, the smoothed estimate improves upon the filtered estimate, resulting in a very close match to the actual trajectory with significantly reduced uncertainty (showing more confidence in the smoothed estimate).

This figure illustrates the importance of applying tracking algorithms when sensors provide noisy data. Simply using the raw sensor data may result in inaccurate trajectories which, if used in a MOD or STDB, will subsequently result in erroneous query results.

Unfortunately, the assumptions upon which the Kalman filter is based (i.e. linear dynamics, Gaussian measurement and system noise) may be too restrictive for some applications and therefore more general techniques are required. In these situations, exact inference becomes in-

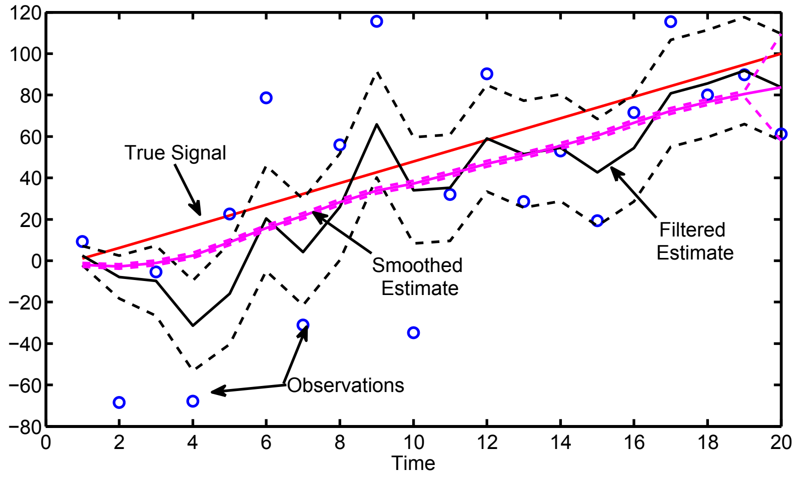


Figure 10.9. An example output of the Kalman filtering and smoothing estimates given a set of noisy observations.

tractable and approximate techniques must be utilized. In the case of non-linear movement, two extensions to the KFM have been introduced which provide local approximations of non-linear movement while (nearly) maintaining the simple filtering update equations. The Extended Kalman Filter (EKF) locally linearizes the state estimation, using partial derivatives of the model dynamics and measurements to approximate updates. Similarly, the Unscented Kalman Filter (UKF) [38] attempts to maintain the efficient update equations of the KFM in the case of non-linear system dynamics by applying the unscented transform, a deterministic sampling technique, to propagate the state distribution through the non-linear dynamics before recovering the parameters of the normal distribution.

Although these approximations provide the convenience of the simple KFM update equations, they typically fail when the dynamics or observation errors result in multi-modal distributions [3]. This is due to the fact that both the EKF and UKF both represent the posterior distribution over x_t as a Gaussian. To represent more complicated densities, a different representation scheme is required as well as approximate inference methods. One of the most popular methods for approximate inference for non-linear dynamics is known as particle filtering [19, 20, 3, 26, 10, 13]. Particle filtering is a generic framework for computing inference in dynamic models in which no special structure exists. The main idea is to represent the probability density function (pdf) describing our belief state of the world as a finite set of weighted point masses. Each

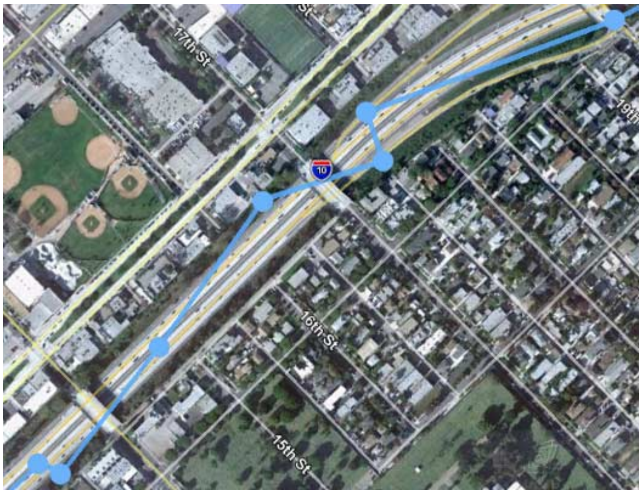


Figure 10.10. An example of the raw trajectory observations overlaid on a (given) road network. Incorporating extra structure, we are able to provide a more accurate path over which the object is likely to have traveled.

point mass is propagated through the model dynamics to predict future states, and is then re-weighted according to the observation likelihood given the predicted value for each. As the number of point masses tends toward infinity, this representation will tend toward the underlying density function and thus can provide a very accurate representation of the system. Several excellent and practical introductions to particle filtering can be found in [20, 10, 3].

3.3 Tracking with Road Networks

Incorporating road network structure into tracking algorithms to improve accuracy has been a popular area of research lately [86, 18, 56, 44]. One such example is the work by Agate and Sullivan [2] in which the authors develop a model for tracking mobile objects that utilizes a road network to constrain object movement and hence improve tracking accuracy. The authors focus on tracking when both ground moving target indicator (GMTI) and high-range resolution (HRR) radar readings are available and thus their observation likelihood models are specific to these measurements. The dynamic model encodes the restriction of the object to only move along the known road segments. Given the road segment upon which the object is currently located, the probability of the next state is a function of the structure of the network since the object is limited to transition to adjacent roads. The state variable then maintains the current road segment (a discrete ID), a position on that road segment, and a deviation from the road to allow for errors in the road network. Inference for the proposed model is computed using a particle filter which is typical for these complex, non-linear dynamic models. The future position of an object is computed by allowing each particle to take a random walk along the road network for a limited time. Each road segment has a distribution over the amount of time it would take to traverse this segment, thus the point at which the particles stop when time runs out provides a reasonable estimate of the object's next state. The results show substantial improvements over a basic KFM for tracking.

A similar problem is that of map-matching [69], in which an object's noisy position observation is aligned with a known restricted movement surface (e.g. road network). The difficulty in map-matching is the uncertainty in an object's observed location at a given time. Additionally, the road network may be uncertain as well (e.g. user generated maps). Lastly, the problem typically needs to be solved in real-time so the object can identify its true current position and continue navigating to its destination.

A natural model for the task of map-matching is the hidden Markov model (HMM) since it combines information about the distribution over the current state of an object with new (noisy) observations. Newson and Krumm [60] apply an HMM for the map-matching problem using GPS as the observed value and individual road segments as the hidden states. The authors defined the transitions between roads to be based on the distance and connectivity between segments. For instance, a vehicle is more likely to transition to a connected road segment than one that is far away. A similar model is also introduced by Pink and Hummel [67]. In this case, the authors utilize inferred heading information about the vehicle paired with a more accurate representation of the road network based on smooth polynomials instead of linear segments to improve accuracy. Both of these methods rely on consistent and high frequency GPS measurements.

In practice, GPS observations are often obtained irregularly and at low-sampling rates (i.e. 1/120Hz or lower). In these situations, the map-matching problem becomes that of inferring the specific route (sequence of roads traveled) between two GPS observations in an offline setting. Several approaches have been developed specifically for this scenario.

Similar to previous works, Lou et al. [52] address the low sampling frequency map-matching problem by introducing an algorithm that combines a spatial and temporal analysis into an HMM-like model. The Spatio-Temporal-matching (ST-matching) algorithm first attempts to match each GPS reading to a road segment, then considers the location of the surrounding readings to correct the matched road segment. The underlying assumption behind this approach is that the most direct route is typically the correct one. The temporal analysis utilizes the average speed along each road link. Yuan et al. [97] develop a voting-based map-matching algorithm, called interactive voting map-matching (IVMM), specifically for low sampling rate GPS data. Mapped points are allowed to influence neighboring points with a weight inversely proportional to their distances. The algorithm uses dynamic programming to find the best scoring path given the observations.

Zheng et al. [101] introduce the first data-driven method for resolving the inherent uncertainty of a trajectory collected using a very low GPS sampling rate. The idea is to utilize a collection of historic trajectories and find popular (partial) paths between the sporadic GPS observations. The authors introduce two algorithms for solving the local path problem, one based on greedy-like search process and the other which first extracts a traversal graph containing all of the relevant nodes and edges between two observations and performs a shortest path search in the reduced space. Complete trajectories are then constructed using a dynamic programming algorithm (and a decomposable scoring function). In their experiments, the authors show that their approach significantly outperforms previous methods for dealing with low-sampling-rate trajectories.

Although GPS observations are the most popular type of data for tracking and identifying an object's position, there are other options as well. In fact, continuous collection of GPS can be quite expensive (in terms of power consumption) for a mobile sensor. Therefore, Thiagarajan et al. [79] aim to utilize only the signal from cellular towers, which requires much less energy to collect, to perform map-matching. The authors pose this as a supervised learning problem. In this context the training data is pairs of cellular tower fingerprints (tower IDs and their respective signal strengths) and their corresponding GPS locations (considered to be the labeled data). That is, using the cellular fingerprints as a feature vector, and the GPS location of the user as the actual location, they pose map-matching as a classification problem. Their approach grids the area of interest and uses a HMM to determine the grid after observing the cell tower fingerprint. The authors introduce several additional methods to clean and refine the signal, including integrating information from other sensors on the cell phone (e.g. accelerometer or compass). The experimental results show their method to be a very accurate, energy efficient alternative to constantly using GPS.

Additionally, several other works on map-matching have been introduced that assume spatially and temporally high resolution GPS data is available [6, 7, 25, 34]. These models are similar in that they assume a small degree of error in the observations which allows them to use relatively simple nearest-neighbor approaches to map the object's GPS observation to a road segment on the known road network.

3.4 Tracking for External Sensing

The work mentioned up until this point has all assumed that the mobile objects were providing their location willingly in order to navigate or be queried. However, once this assumption is removed, the problem becomes significantly more challenging. At the core of these challenges is the fact that we do not know which observations belong to which mobile objects, referred to as the *data association* problem. For instance, if two objects are nearby, their observations may get switched, thus the trajectory we obtain would actually be composed of the movements of two different objects.

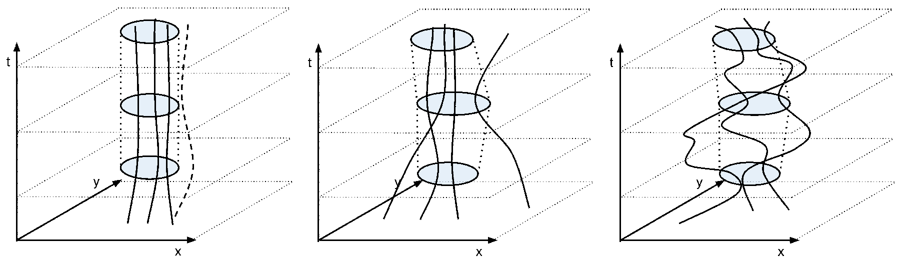
Although the data association problem makes external sensing a much more complicated problem, it is not the only issue in this scenario. Because sensing occurs in an incredibly noisy environment, we may detect false positives as well as miss the detection of actual objects (false negatives). Moreover, the total number of mobile objects is considered to be unknown. New tracking algorithms have been developed for this scenario, making use of particle filtering methods and finite set statistics (FSS) [42, 53, 61, 64, 86, 85]. The problem scenario of external sensing has not been addressed in the database community, mainly because the current solutions only scale to managing $5 - 10$ objects, as high dimensional filtering is known to be an open problem.

4. Mining Mobility Data

Querying spatiotemporal data is able to provide answers to simple questions, such as *what are the closest coffee shops to me?* or *how many objects have passed through this area over a given time interval*? However, extracting semantically higher-order information from such low level data is a difficult problem. For instance, it may be of interest to identify those mobile objects that behave similarly (e.g. travel to similar locations), identify popular or efficient routes, or just to be able to quantitatively characterize and predict user movements.

We partition this section into three major areas that cover recent work in mining spatiotemporal data: (i) clustering, (ii) route detection, and (iii) movement patterns. The first, covers work on clustering moving objects or grouping similar trajectories. Clustering is a crucial task in many data management and analysis tasks since it can be used for compression and indexing as well as understanding similarities in the movements of objects. Second, we cover work on popular route detection. This section reviews some recently developed methods for identifying often-traveled paths. The intuition is that the transportation network may not represent all of the factors that influence the decision to take one path over another (e.g. long stop lights, traffic congestion, etc.). By mining previous travel patterns, it is possible to identify the frequently traveled paths. This information can then be used for managing traffic congestion, finding efficient routes, or simply studying the effectiveness of the given road network. The third section focuses on problems related to quantitatively assessing individual user movement patterns. Instead of looking at aggregate behavior, as the work in popular route detection does, the work in this section focuses on the individual. Here the interesting problems are predicting future locations and high-order understanding of user movement (e.g. is the user going to work or to the store?).

Clustering. In the spatiotemporal data setting, clustering aims to group together objects which are within close proximity of one another *and* will remain so over time. Li et al. [47] propose a technique for clustering moving objects by extending the ideas of micro-clustering [99] to handle data that changes over time. To maintain good clusters over time, the authors propose computing a minimum bounding rectangle (MBR) for each cluster. When the MBR reaches a predefined threshold, a split event occurs in which the object furthest from the center of the cluster is removed and reassigned to the nearest microclucster. The resulting time complexity of the clustering approach is $O((N+T)log^2(N+T)log(N))$, where N is the number of mobile objects and T is the total time over which the clustering is to be maintained. Similarly, Jensen et al. [36] also propose an online method for efficiently clustering moving objects based on [99]. The authors introduce a dissimilarity measure for mobile objects which takes the weighted sum of the differences between the locations of two objects over m time steps. The weights are monotonically decreasing as they become further into the future i.e. the current time is weighted more heavily than future positions. Utilizing the BIRCH clustering framework [99], the authors extend the clustering feature vector to include object positions and velocities in a format that is efficient to update. Computing a radius for each cluster (at each time step), future necessary cluster split points can be predicted and then processed efficiently by reassigning.



(a) Flocks (b) Convoys (c) Swarms

Figure 10.11. An example of (a) flocks, (b) convoys, and (c) swarms. Each of these patterns captures groups of objects that tend to move together over time, though they each specify slightly different constraints as highlighted in this figure.

In addition to the basic notions of clustering for mobile objects, newer definitions have also been introduced which provide a stronger notion that objects must *move* together. Recently, the notion of *flocks*, *convoys* and *swarms* have been introduce which impose varying constraints on how tightly packed objects must remain over time. Specifically, in [87], a flock is defined as a set of at least μ trajectories that are located within a given disk with radius $\frac{\epsilon}{2}$, for δ or more time steps. A flock query returns all *sets* of trajectories such that the predicate $flock(\mu, \epsilon, \delta)$ is met. To answer the flock query, the authors propose first griding the space such that each grid is a square with edge lengths ϵ. This reduces the necessary number of comparisons between points and allows the authors to provide an exact answer to the flock query in polynomial time. The authors provide a basic query processing algorithm along with several filtering approaches to improve the algorithm efficiency.

Jeung et a. [37] relax the definition of a flock by using the notion of density connected groups of objects over time. The new spatiotemporal groups are called *convoys* and the authors introduce a filter-and-refine algorithm called *convoy discovery using trajectory simplification* (CuTS) to search for convoys in a given database. The authors first simplify trajectories using linear approximations of subtrajectories such that a maximum error bound is maintained. The simplified trajectories contain fewer points than the originals and are thus easier to manage. The filtering step in CuTS involves a trajectory simplification followed by a density based clustering. In the refinement step, the full trajectories are run through the density clustering again to account for the δ error introduced in the trajectory simplification. The resulting set of clustered trajectories are guaranteed to be convoys.

Furthermore, Li et al. [48, 49] define a *swarm*, which again relaxes the notion of a convoy by removing the restriction that objects must remain together over consecutive time steps. A swarm is defined as a set of at least min_o objects that remain clustered for at least min_t time steps over a given interval. That is, a swarm is defined by a set of objects, $\mathcal{O}$ and a set of time steps, T, over which the objects remain clustered. To avoid repeatedly identifying subsets of the same objectset, the authors define a *closed swarm* to be a swarm such that the objectset and the timeset are maximal (i.e. adding another object will shrink the timeset and adding another time step will shrink the valid objectset).

Li et al. [48, 49] develop an algorithm for finding swarms which is based on the Apriori (frequent itemset) framework. To manage the exponential search space, three pruning rules are introduced. The first rule says that if $|T| < min_t$, then there is no superset of $\mathcal{O}$ in which the time constraint will be satisfied, thus we can stop growing this set. The second pruning rule, backward pruning, states that if the maximum time set of an objectset, $\mathcal{O}$, does not decrease by adding one more object, then $\mathcal{O}$, may be pruned and we can continue expanding the new objectset $\mathcal{O}' = \{\mathcal{O} \cup o_i\}$. Lastly, forward pruning, by similar means to backward pruning, allows us to determine if an objectset is not closed. Using these pruning rules, the *ObjectGrowth* algorithm performs a depth-first search (on the space of swarms) and identifies all closed swarms.

Popular Route Discovery. Closely related to the problem of clustering, is that of discovering popular routes. Whereas clustering is an object-centric task, the goal of popular route discovery is to identify specific *paths* that are heavily traveled. We review two recent approaches to this problem, the first finds heavily traveled paths conditioned upon a specific origin and destination and the second finds all paths such that the amount of traffic is greater than a given threshold.

Chen et al. [14] introduce a new approach for discovering popular routes only given a set of GPS trajectories. The authors assume that a road network is not available and construct one directly from the data. They use a density based clustering routine for identifying the underlying road network (specifically the intersections) from a set of individual GPS trajectories. The clustering algorithm is an adaptation of DBSCAN [21] with a different connectivity metric which is based on the angle of intersection between trajectories (since roads typically intersect at nearly right angles).

Using the constructed road network, Chen et al. [14] a random walk based approach for identifying popular paths with respect to a specific destination node. The authors assign a transition probability at each intersection by counting the number of objects that took each path from the GPS trajectories. Since the objective is to identify popular routes between a given source and destination, each trajectory is weighted by the likelihood that it is heading toward the specified destination. That is, separate random walk probabilities are constructed for each specified destination. Using this network, the authors run a random walk using the destination node as an absorbing state to compute a score for each node. The paths are scored by computing the product of the individual node popularity scores (i.e. $\prod_{v \in V} p(v)$) and the path with the maximum popularity is computed using an adaptation of Dijkstra's shortest path search. Although it is not clear if the algorithms will scale to large networks or answering queries online (due to the preprocessing costs), the experimental results presented in [14] are promising.

Similarly, Li et al. [46] propose a method for identifying all of the heavily traveled routes in a given road network, independent of a specific starting and stopping point. The authors introduce a new algorithm, called *FlowScan*, which combines ideas from both individual and aggregate-level analyses over trajectories to identify popular routes. FlowScan iteratively expands a route starting with a given road edge, r, by identifying those edges that are within ϵ hops of r and satisfy the minimum *traffic* support (i.e. number of trajectories that pass through that road segment). The traffic support is computed as $|traffic(r) \cap traffic(s)| \geq MinTraffic$, which says that the two roads must *share* some minimum amount of traffic (i.e. the same trajectories must travel through both road segments). Starting from an edge satisfying the minimum traffic threshold, the algorithm proceeds by adding edges until the conditions are not satisfiable.

The proposed algorithm manages finding popular routes even in difficult instances where routes may partially overlap with one another and there may be sparse regions in which objects may choose from several roads to connect two portions of the same 'route'. The authors experiment with their method on simulated data on a real road network to show the quality of the identified routes obtained by their approach and compare it to prior work. They show that FlowScan is able to identify correct popular routes when the other methods failed, either by grouping overlapping routes together or missing parts of routes due to gaps.

Mobility Patterns. The last group of work we discuss is broadly categorized as mining mobility patterns. By mobility patterns, we refer to the common movements exhibited either by the same object over a long history, or movements repeated by several different objects. For instance, it may be common for residents to use a similar path during their morning commute to get from the suburbs where they live to the downtown area in which they work. Identifying such patterns may be useful in designing an efficient highway system or estimating road wear over time.

Perhaps the most basic problem involving movement patterns is that of predicting an object's future position given its current location. Tao et al. [76] introduce the *recursive motion function* (RMF) for predicting the future position of a moving object. It formulates an object's location at time t as $\text{loc}_t = \sum_i^f K_m(i)\text{loc}_{t-i}$, where $K_m(i)$ is a constant matrix that represents the type of motion performed by the object and f, called retrospect, is the minimum number of the most recent timestamps which are required to compute the elements of all $K_m(i)$. The $K_m(i)$ matrices represent $m = 1..M$ different motion types (e.g. linear, circular, sine, polynomial, ...) and the motion estimation is done by determining which pattern results in the lowest summed squared distances between the object's actual and predicted trajectory. To manage the large set of potential motion patterns, the authors also propose the spatiotemporal prediction tree (STP-tree), an R-tree based index mobility functions. The STP-tree maintains a set of polynomial curves which represent object movement over time, in the case when the all of the predicted patterns produce linear functions, the STP-tree reduces to the TPR-tree. Although RMFs have performed very well at predicting future locations of mobile objects with complex mobility patterns, they have some weaknesses. Because predictions are based on past movements, RMFs are not able to capture sudden changes in direction (such as a U-turn). Additionally, predictions made several time steps into the future tend to loose accuracy since objects tend to only follow a given motion type for a limited time.

To address these issues, Jeung et al. [37] use previous trajectories from objects to provide a method that is able to accurately predict an object's location multiple time steps in the future. The authors utilize the object's past behavior by performing frequent item-set mining to find common locations (i.e. *given that the object is at the mall at* 4*pm, she will be at the beach at* 5*pm with confidence* c). This work addresses the problem of answering prediction queries by assuming each object has an underlying repetitive pattern. The proposed algorithms are experimentally shown to outperform RMFs, previously the state-of-the-art method for predicting future locations.

Another important aspect of mobility is periodicity as users tend to exhibit many regularities in their movements (e.g. going to work every morning). Li et al. [48, 49] develop a novel technique for identifying periodic movements of a user where the period lengths are also extracted from the data. To robustly identify periodic patterns at various resolutions, the authors propose the idea of using references spots for each individual. Reference spots are highly visited areas in space which are found using density estimation. Given a reference spot, the movement of a user can be described as a binary sequence in which the user is either at that spot or not. Periodic movement may then be detected by applying a Discrete Fourier Transform (DFT) and selecting all frequencies higher than a threshold. This procedure is repeated for each reference location, thus identifying various period lengths which are robust to noise in the spatial movement of the user.

Periodic movement patterns are then described as a probability distribution (computed via maximum likelihood) over reference locations at each time step within a given period. Using these probability distributions, the next step is to identify the specific patterns. This is accomplished through a hierarchical clustering of the probability distributions, using KL-divergence as the distance measure. Patterns are combined and an overall clustering score is maintained such that when the score increases too much, the clustering stops and hence picks k, the number of clusters, automatically. Experimental evaluation found the proposed methods to be able to accurately identify interesting periodic movement behaviors. Additionally, the authors applied their method to a real data set, the location of a bald eagle over three years, and they were able to identify the migration patterns of the eagle over that time.

In addition to utilizing the large quantities of available trajectory data to identify interesting patterns, it is also possible to identify object movements that *do not* follow the expected behavior. That is, given enough data we can identify common behavior patterns and use this to detect anomalies. To this end, Li et al. [46] introduce a rule and motif based anomaly detection method for moving objects (ROAM). The idea in this work is to partition trajectories into several prototypical submovements and use features extracted from these movements to classify each trajectory as normal or abnormal. In this work, the authors pose anomaly detection as a supervised learning problem and thus assume a training set of labeled trajectories is available. The proposed algorithm is composed of three steps: (i) motif extraction, (ii) feature generation and (iii) classification.

The motif extraction phase slides a window (of fixed length) over each trajectory and the set of subtrajectories (i.e. each window) is clustered. The cluster centroids are referred to as *motifs*, and a second pass through the dataset determines which trajectories *express* each motif (i.e. have a subtrajectory that matches with error less than ϵ). Next, features are generated for each motif and the values are discretized (or clustered) to ensure generalization (e.g. (right-turn, speed, 11mph) or (right-turn, time, 2am)). Additionally, a hierarchy is built over the value space for each specific feature (e.g. ‘right-turn speed’) to provide a multi-resolution view of the data. Lastly, to utilize this feature hierarchy, the authors propose classification using hierarchical prediction rules (CHIP) which is based on FOIL [70]. CHIP learns a set of rules by exploring features in a top down manner, such that it will first attempt to classify based on high level features in the hierarchy. To determine if it should expand a lower feature level, CHIP computes the information gain from using the higher resolution features. The experiments show that the proposed technique significantly outperforms a basic SVM classification. In fact, ROAM is shown to consistently outperform competing methods as both the number of trajectories and the number of motifs increases.

Combining several of these ideas, [8, 50] build high-order models of user movements in which subtrajectories may be identified as completing specific tasks. By high-order modeling, we refer to the ability of a model to assign semantically meaningful labels to segments of a trajectory (e.g. “going to work”) and include more abstract attributes (e.g. mode of transportation) over the raw trajectory data. In this work, the objective is to be able to answer queries *not* about the specific trajectory, but about the purpose of the movement (e.g. given a trajectory is it more likely that the subject is going to work or the grocery store?). Specifically, Liao et al. [50] introduce a dynamic Bayesian network model which incorporates high level goals, such as “going to work” or “going to the supermarket” into the model which may be inferred from low level location data (e.g. GPS). The result is a model which is capable of querying a variety of aspects of a user’s movement. The authors show that their proposed model can be learned in a completely unsupervised manner, though applying the semantic categories such as “going to work” must be supervised, and is able to identify locations of interest and abnormal behavior in a real data trace.

In addition to mining patterns of *movement*, a related problem is to identify those *locations* that are visited by a large number of trajectories. Using the abundant amount of GPS trace data available over a region, it is possible to find specific locations that are of interest (e.g. Statue of Liberty, good restaurants, popular bars, etc.) [9, 84, 102]. For instance, Zheng et al. [102] develop a PageRank-like algorithm for mining interesting locations by considering each user’s travel experience. The basic idea is that users that are well traveled within a region of interest will likely know more of the relevant locations and thus a visit from one of these *travel authorities* should be weighted more than a visit from a tourist who does not know the local area well. Alternatively, Uddin et al. [84] identify regions of interest (ROIs) from trajectory data by looking for dense areas (at least N mobile objects in a fixed area) in which mobile objects spend some minimum amount of time. The authors index trajectories by velocity, then extend the Pointwise Dense Region (PDR) [62] method to identify ROIs.

Similarly, the work by Giannotti et al. [22] combine aspects of mining interesting locations, with understanding and predicting movement patterns. The authors extend prior work on mining frequent patterns and define a spatiotemporal sequence (ST-sequence), which is a sequence of locations visited by a set of users with a given level of support. The goal in this work, much like in the frequent itemset mining, is to identify a frequently visited sequence of locations over time. The authors address the problem when the locations, or regions-of-interest (ROI), are known as well as when they must be extracted from the data. More recent work has built upon these concepts of pattern mining in the spatiotemporal domain as well [58, 72].

5. Discussion and Future Research Directions

Over the past two decades, we have seen significant advancements in all areas related to managing spatiotemporal data. In this chapter, we attempt to cover the state-of-the-art solutions to the current challenges specific to managing spatiotemporal data. Our review focused on data management techniques as these are fundamental to nearly all applications involving mobility data. Additionally, we also covered some of the core and recent work on tracking mobile objects. Lastly, we introduced some of the recent applications of mining spatiotemporal data to extract interesting patterns.

Despite the multitude of work in these areas, new and challenging problems are constantly being introduced. Below we briefly outline a few potentially interesting areas of future research.

- **Combining spatiotemporal information with social networks:** Work on social network analysis in the recent years has been plentiful due to the explosion of the availability of social data from sites such as Facebook, Twitter, MySpace, and various other relationship or communication networks. The analysis of users and their connections has largely focused on the concept of homophily, which is the tendency of individuals to connect to others that are similar to themselves. However, physical space is another significant factor which influences how users interact with one another. Combining information about a user's movement with her social network presents an exciting research direction [16, 96].

- **Data-driven Techniques:** Large quantities of spatiotemporal data are becoming readily available through several research efforts and sharing sites [106–108]. For example, the California Department of Transportation has made the data it collects from highway loop counters publicly available (for free) [107]. Utilizing these data stores and developing data-driven techniques to tackle core problems such as map-matching [101], identifying locations of interest [102] or traffic analysis [33] continues to be a promising area of research.

- **Communication Efficiency:** Communication between mobile objects, as well as between mobile objects and a central database, remains an expensive operation in terms of power and bandwidth consumption. Making use of low quality sensors which use less power, or scheduling the transfer of data in a more effective manner can both help to reduce these costs [79, 83].

- **Information Movement:** In this chapter, we have focused on managing the mobility data of physical objects traveling through space, however, studying the flow of information offers a similar set of challenges. With the growth of the internet, there has been an astronomical increase in the availability and sharing of information. Only recently have researchers started to ask questions about how information gets disseminated over time [1, 23, 45, 93]. Additionally, combining the challenges invovled in tracking information and moving objects, which are used to transfer information, results in yet another set of interesting problems known as *data ferrying* [103].

Acknowledgements

This work was partially supported by the National Science Foundation under grant IIS-0917149.

References

[1] E. Adar and L. Adamic. Tracking information epidemics in blogspace. In *Web Intelligence*, pages 207–214. Ieee, 2005.

[2] C. S. Agate and K. J. Sullivan. Road-Constrained Target Tracking and Identification Using a Particle Filter. In *SPIE*, number 805, 2003.

[3] M. Arulampalam, S. Maskell, N. Gordon, and T. Clapp. A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking. *IEEE Transactions on Signal Processing*, 50(2):174–188, 2002.

[4] P. Bakalov, M. Hadjieleftheriou, and V. Tsotras. Time relaxed spatiotemporal trajectory joins. In *GIS*, pages 182–191. ACM, 2005.

[5] N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger. The r*-tree: an efficient and robust access method for points and rectangles. 19(2), 1990.

[6] M. Bierlaire, J. Chen, and J. Newman. A Probabilistic Map Matching Method for Smartphone GPS data. Technical report, EPFL, 2010.

[7] S. Brakatsoulas, D. Pfoser, R. Salas, and C. Wenk. On Map-Matching Vehicle Tracking Data. In *VLDB*, pages 853–864, 2005.

[8] H. H. Bui. A General Model for Online Probabilistic Plan Recognition. In *IJCAI*, 2003.

[9] X. Cao, G. Cong, and C. Jensen. Mining significant semantic locations from gps data. *Proceedings of the VLDB Endowment*, 3(1-2):1009–1020, 2010.

[10] O. Cappe, S. J. Godsill, and E. Moulines. An Overview of Existing Methods and Recent Advances in Sequential Monte Carlo. *Proceedings of the IEEE*, 95(5):899–924, May 2007.

[11] V. P. Chakka, A. C. Everspaugh, and J. M. Patel. Indexing Large Trajectory Data Sets With SETI. In *CIDR*, 2003.

[12] S. Chen, C. S. Jensen, and D. Lin. A Benchmark for Evaluating Moving Object Indexes. In *PVLDB*, pages 1574–1585, 2008.

[13] Z. Chen. Bayesian Filtering: From Kalman Filters to Particle Filters, and Beyond. Technical report, Adaptive Systems Lab, McMaster University, Hamilton, Ontario, 2003.

[14] Z. Chen, H. T. Shen, and X. Zhou. Discovering Popular Routes from Trajectories. In *ICDE*, 2011.

[15] R. Cheng, D. V. Kalashnikov, and S. Prabhakar. Querying Imprecise Data in Moving Object Environments. *TKDE*, 16(9):1112–1127, 2004.

[16] E. Cho, S. Myers, and J. Leskovec. Friendship and mobility: User movement in location-based social networks. In *SIGKDD*, pages 1082–1090. ACM, 2011.

[17] B. S. E. Chung, W.-C. Lee, and A. L. P. Chen. Processing probabilistic spatio-temporal range queries over moving objects with uncertainty. In *EDBT*. ACM Press, 2009.

[18] A. Civilis, C. S. Jensen, and S. Pakalnis. Techniques for Efficient Road-Network-Based Tracking of Moving Objects. *IEEE Trans-*

actions on Knowledge and Data Engineering, 17(5):698–712, May 2005.

[19] A. Doucet, S. Godsill, and C. Andrieu. On sequential Monte Carlo sampling methods for Bayesian filtering. *Statistics and Computing*, 10:197–208, 2000.

[20] A. Doucet and A. M. Johansen. A Tutorial on Particle Filtering and Smoothing : Fifteen years later. Technical Report December, 2008.

[21] M. Ester, H. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data mining*, volume 1996, pages 226–231. AAAI Press, 1996.

[22] F. Giannotti, M. Nanni, D. Pedreschi, and F. Pinelli. Trajectory Pattern Mining. In *KDD*, pages 330–339, 2007.

[23] M. Gomez-Rodriguez, J. Leskovec, and A. Krause. Inferring networks of diffusion and influence. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 5(4):21, 2012.

[24] H. Gonzalez, J. Han, X. Li, M. Myslinska, and J. Sondag. Adaptive fastest path computation on a road network: A traffic mining approach. In *VLDB*, pages 794–805. VLDB Endowment, 2007.

[25] J. S. Greenfeld. Matching GPS Observations to Locations on a Digital Map. *Environmental Engineering*, 1(3):164–173, 2002.

[26] F. Gustafsson, F. Gunnarsson, N. Bergman, U. Forssell, J. Jansson, R. Karlsson, and P.-J. Nordlund. Particle filters for positioning, navigation, and tracking. *IEEE Transactions on Signal Processing*, 50(2):425–437, 2002.

[27] R. H. Guting, T. Behr, and J. Xu. Efficient k-nearest neighbor search on moving object trajectories. *The VLDB Journal*, 19(5):687–714, Apr. 2010.

[28] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *ACM SIGMOD Conference*, 1984.

[29] M. Hadjieleftheriou, G. Kollios, P. Bakalov, and V. J. Tsotras. Complex spatio-temporal pattern queries. In *VLDB*, 2005.

[30] P. Hoff. *A first course in Bayesian statistical methods.* Springer Verlag, 2009.

[31] J. Hosbond, S. Saltenis, and R. Ortoft. Indexing uncertainty of continuously moving objects. In *Database and Expert Systems Applications, 2003. Proceedings. 14th International Workshop on*, pages 911–915. IEEE, 2003.

[32] M. Hua and J. Pei. Probabilistic Path Queries in Road Networks: Traffic Uncertainty Aware Path Selection. In *EDBT*, 2010.

[33] A. Ihler, J. Hutchins, and P. Smyth. Adaptive Event Detection with Time-Varying Poisson Processes. In *KDD*, 2006.

[34] A. Jawad and K. Kersting. Kernelized Map Matching for Noisy Trajectories. In *SIG SPATIAL*, 2010.

[35] C. Jensen, D. Lin, and B. Ooi. Query and update efficient b+-tree based indexing of moving objects. In *VLDB*, pages 768–779. VLDB Endowment, 2004.

[36] C. Jensen, D. Lin, and B. Ooi. Continuous clustering of moving objects. *Knowledge and Data Engineering, IEEE Transactions on*, 19(9):1161–1174, 2007.

[37] H. Jeung, Q. Liu, H. T. Shen, and X. Zhou. A Hybrid Prediction Model for Moving Objects. In *ICDE*. IEEE, Apr. 2008.

[38] S. Julier and J. Uhlmann. A new extension of the kalman filter to nonlinear systems. In *Int. Symp. Aerospace/Defense Sensing, Simul. and Controls*, volume 3, page 26. Spie Bellingham, WA, 1997.

[39] R. Kalman et al. A new approach to linear filtering and prediction problems. *Journal of basic Engineering*, 82(1):35–45, 1960.

[40] B. Kanagal and A. Deshpande. Online Filtering, Smoothing and Probabilistic Modeling of Streaming data. In *ICDE*, 2008.

[41] U. Khan and J. Moura. Distributing the kalman filter for large-scale systems. *Signal Processing*, 56(10):4919–4935, 2008.

[42] Z. Khan, T. Balch, and F. Dellaert. MCMC-based particle filtering for tracking a variable number of interacting targets. *IEEE transactions on pattern analysis and machine intelligence*, 27(11):1805–19, Nov. 2005.

[43] K.-S. Kim, S.-W. Kim, T.-W. Kim, and K.-J. Li. Fast indexing and updating method for moving objects on road networks. In *Web Information Systems Engineering Workshops*, pages 34–42. Ieee, 2003.

[44] W. Koch. On Bayesian Tracking of Extended Objects. In *Multisensor Fusion and Integration for Intelligent Systems*, pages 209–216. Ieee, Sept. 2006.

[45] J. Leskovec, L. Backstrom, and J. Kleinberg. Meme-tracking and the dynamics of the news cycle. In *SIGKDD*, pages 497–506. ACM, 2009.

[46] X. Li, J. Han, J. Lee, and H. Gonzalez. Traffic density-based discovery of hot routes in road networks. *Advances in Spatial and Temporal Databases*, pages 441–459, 2007.

[47] Y. Li, J. Han, and J. Yang. Clustering Moving Objects. In *KDD*, 2004.

[48] Z. Li, B. Ding, J. Han, and R. Kays. Swarm: Mining relaxed temporal moving object clusters. *VLDB Endowment*, 3(1-2):723–734, 2010.

[49] Z. Li, J. Han, M. Ji, L. Tang, Y. Yu, B. Ding, J. Lee, and R. Kays. Movemine: Mining moving object data for discovery of animal movement patterns. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(4):37, 2011.

[50] L. Liao, D. J. Patterson, D. Fox, and H. Kautz. Learning and inferring transportation routines. *Artificial Intelligence*, 171(5-6):311–331, Apr. 2007.

[51] H. Loose, U. Franke, and C. Stiller. Kalman particle filter for lane recognition on rural roads. In *Intelligent Vehicles Symposium, 2009 IEEE*, pages 60–65. IEEE, 2009.

[52] Y. Lou, C. Zhang, Y. Zheng, X. Xie, W. Wang, and Y. Huang. Map-matching for low-sampling-rate GPS trajectories. In *GIS*, number c. ACM Press, 2009.

[53] R. P. S. Mahler. Multitarget bayes filtering via first-order multitarget moments. *IEEE Transactions on Aerospace and Electronic Systems*, 39(4):1152–1178, Oct. 2003.

[54] N. Malviya, S. Madden, and A. Bhattacharya. A Continuous Query System for Dynamic Route Planning. In *ICDE*, 2011.

[55] Y. Manolopoulos, A. Nanopoulos, and Y. Theodoridis. *R-trees: Theory and Applications.* Springer-Verlag New York Inc, 2006.

[56] O. Mazhelis. Using Recursive Bayesian Estimation for Matching GPS Measurements to Imperfect Road Network Data. In *Intelligent Transportation Systems*, pages 1492–1497, 2010.

[57] M. F. Mokbel, T. M. Ghanem, and W. G. Aref. Spatio-temporal Access Methods. *IEEE Data Engineering Bulletin*, 2010.

[58] A. Monreale, F. Pinelli, R. Trasarti, and F. Giannotti. Wherenext: a location predictor on trajectory pattern mining. In *SIGKDD*, pages 637–646. ACM, 2009.

[59] K. P. Murphy. *Dynamic Bayesian Networks: Representation, Inference, and Learning.* Phd, University of California, Berkeley, 1994.

[60] P. Newson and J. Krumm. Hidden Markov Map Matching Through Noise and Sparseness. In *SIG SPATIAL*, 2009.

[61] W. Ng, J. Li, S. J. Godsill, and S. K. Pang. Multitarget Initiation, Tracking and Termination Using Bayesian Monte Carlo Methods. *The Computer Journal*, 50(6):674–693, Sept. 2007.

[62] J. Ni and C. Ravishankar. Pointwise-dense region queries in spatio-temporal databases. In *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*, pages 1066–1075. IEEE, 2007.

[63] E. Nikolova, M. Brand, and D. R. Karger. Optimal Route Planning under Uncertainty. In *International Conference on Automated Planning and Scheduling*, 2006.

[64] S. K. Pang, J. Li, and S. J. Godsill. Detection and Tracking of Coordinated Groups. *IEEE Transactions On Aerospace And Electronic Systems*, 47(1), 2009.

[65] J. Patel, Y. Chen, and V. Chakka. Stripes: an efficient index for predicted trajectories. In *SIGMOD*, pages 635–646. ACM, 2004.

[66] D. Pfoser, C. S. Jensen, and Y. Theodoridis. Novel Approaches to the Indexing of Moving Object Trajectories. In *VLDB*, pages 395–406, 2000.

[67] O. Pink and B. Hummel. A statistical approach to map matching using road network geometry , topology and vehicular motion constraints. In *Intelligent Transportation Systems*, pages 862–867, 2008.

[68] S. Prabhakar, Y. Xia, D. Kalashnikov, W. Aref, and S. Hambrusch. Query indexing and velocity constrained indexing: Scalable techniques for continuous queries on moving objects. *IEEE Transactions on Computers*, 51(10):1124–1140, 2002.

[69] M. A. Quddus, W. Y. Ochieng, and R. B. Noland. Current map-matching algorithms for transport applications: State-of-the art and future research directions. *Transportation Research Part C*, 15(5):312–328, Oct. 2007.

[70] J. Quinlan and R. Cameron-Jones. Foil: A midterm report. In *ECML*, pages 1–20. Springer, 1993.

[71] S. Saltenis, C. S. Jensen, S. T. Leutenegger, and M. A. Lopez. Indexing the positions of continuously moving objects. In *SIGMOD*, pages 331–342, June 2000.

[72] S. Scellato, M. Musolesi, C. Mascolo, V. Latora, and A. Campbell. Nextplace: a spatio-temporal prediction framework for pervasive systems. *Pervasive Computing*, pages 152–169, 2011.

[73] T. Sellis, N. Roussopoulos, and C. Faloutsos. The R+-Tree: a Dynamic Index for Multi-Dimensional Objects. In *VLDB*, 1987.

[74] D. Simon. Kalman filtering with state constraints: a survey of linear and nonlinear algorithms. *Control Theory and Applications, IET*, 4:1303 – 1318, 2010.

[75] Z. Song and N. Roussopoulos. Seb-tree: An approach to index continuously moving objects. In *Mobile Data Management*, pages 340–344. Springer, 2003.

[76] Y. Tao, C. Faloutsos, D. Papadias, and B. Liu. Prediction and indexing of moving objects with unknown motion patterns. In *SIGMOD*. ACM Press, 2004.

[77] Y. Tao, D. Papadias, and Q. Shen. Continuous nearest neighbor search. In *VLDB*, pages 287–298. VLDB Endowment, 2002.

[78] Y. Tao, D. Papadias, and J. Sun. The TPR*-Tree: An Optimized Spatio-Temporal Access Method for Predictive Queries. In *VLDB*, 2003.

[79] A. Thiagarajan, L. Ravindranath, H. Balakrishnan, S. Madden, and L. Girod. Accurate, Low-Energy Trajectory Mapping for Mobile Devices. In *Networked Systems Design and Implementation (NSDI)*, 2011.

[80] G. Trajcevski. Probabilistic range queries in moving objects databases with uncertainty. In *MobiDE*, pages 39–45. ACM Press, 2003.

[81] G. Trajcevski, R. Tamassia, P. Scheuermann, and I. F. Cruz. Continuous Probabilistic Nearest-Neighbor Queries for Uncertain Trajectories. In *EDBT*, 2009.

[82] G. Trajcevski, O. Wolfson, K. Hinrichs, and S. Chamberlain. Managing uncertainty in moving objects databases. *ACM Transactions on Database Systems (TODS)*, 29(3):463–507, 2004.

[83] H. Tsai, D. Yang, and M. Chen. Mining group movement patterns for tracking moving objects efficiently. *Knowledge and Data Engineering, IEEE Transactions on*, 23(2):266–281, 2011.

[84] M. Uddin, C. Ravishankar, and V. Tsotras. Finding regions of interest from trajectory data. In *Mobile Data Management (MDM), 2011 12th IEEE International Conference on*, volume 1, pages 39–48. IEEE, 2011.

[85] M. Ulmke, F. Fkie, and P. Willett. Gaussian Mixture Cardinalized PHD Filter for Ground Moving Target Tracking. In *Information Fusion*, number 3, 2007.

[86] M. Ulmke and W. Koch. Road-map Assisted Ground Moving Target Tracking. *IEEE Transactions on Aerospace and Electronic Systems*, 42(4):1264–1274, Oct. 2006.

[87] M. Vieira, P. Bakalov, and V. Tsotras. On-line discovery of flock patterns in spatio-temporal data. In *SIGSPATIAL*, pages 286–295. ACM, 2009.

[88] G. Welch and G. Bishop. An Introduction to the Kalman Filter. Technical report, University of North Carolina at Chapel Hill, 2006.

[89] D. Wilkie, J. V. D. Berg, M. Lin, and D. Manocha. Self-Aware Traffic Route Planning. In *Artificial Intelligence*, pages 1521–1527, 2011.

[90] O. Wolfson, S. Chamberlain, S. Dao, L. Jiang, and G. Mendez. Cost and imprecision in modeling the position of moving objects. In *Data Engineering, 1998. Proceedings., 14th International Conference on*, pages 588–596. IEEE, 1998.

[91] S. Won, W. Melek, F. Golnaraghi, et al. A kalman/particle filter-based position and orientation estimation method using a position sensor/inertial measurement unit hybrid system. *Industrial Electronics*, 57(5):1787–1798, 2010.

[92] X. Xiong and W. Aref. R-trees with update memos. In *ICDE*, pages 22–22. IEEE, 2006.

[93] J. Yang and J. Leskovec. Modeling information diffusion in implicit networks. In *ICDM*, pages 599–608. IEEE, 2010.

[94] J. Yim, J. Joo, and C. Park. A kalman filter updating method for the indoor moving object database. *Expert Systems with Applications*, 38(12):15075 – 15083, 2011.

[95] M. Yiu, Y. Tao, and N. Mamoulis. The b dual-tree: indexing moving objects by space filling curves in the dual space. *The VLDB Journal*, 17(3):379–400, 2008.

[96] X. Yu, A. Pan, L. Tang, Z. Li, and J. Han. Geo-friends recommendation in gps-based cyber-physical social network. In *ASONAM*, 2011.

[97] J. Yuan, Y. Zheng, C. Zhang, X. Xie, and G.-Z. Sun. An Interactive-Voting Based Map Matching Algorithm. In *Mobile Data Management*, pages 43–52, 2010.

[98] M. Zhang, S. Chen, and C. S. Jensen. Effectively Indexing Uncertain Moving Objects for Predictive Queries. In *VLDB*, 2009.

[99] T. Zhang, R. Ramakrishnan, and M. Livny. Birch: an efficient data clustering method for very large databases. In *ACM SIGMOD Record*, volume 25, pages 103–114. ACM, 1996.

[100] K. Zheng, G. Trajcevski, X. Zhou, and P. Scheuermann. Probabilistic range queries for uncertain trajectories on road networks. In *EDBT*, pages 283–294. ACM, 2011.

[101] K. Zheng, Y. Zheng, X. Xie, and X. Zhou. Reducing Uncertainty of Low-Sampling-Rate Trajectories. In *ICDE*, 2011.

[102] Y. Zheng, L. Zhang, X. Xie, and W.-y. Ma. Mining Interesting Locations and Travel Sequences from GPS Trajectories. In *WWW*, pages 791–800, 2009.

[103] Y. Zhu, W. Wu, and V. Leung. Energy-efficient tree-based message ferrying routing schemes for wireless sensor networks. *Mobile Networks and Applications*, 16(1):58–70, 2011.

[104] `http://www.gstatic.com/ads/research/en/2011\_TheMobileMovement.pdf`.

[105] `http://www.idc.com/getdoc.jsp?containerId=prUS23112511`.

[106] `http://www.openstreetmap.org/`.

[107] `http://pems.dot.ca.gov/`.

[108] `http://www.movebank.org/`.