Indexing in-Network Trajectory Flows

Iulian Sandu Popa[#], Karine Zeitouni[#], Vincent Oria[*], Dominique Barth[#], Sandrine Vial[#]

[#]*PRiSM Laboratory, University of Versailles Saint-Quentin*

45 avenue des Etats-Unis, F-78035, Versailles, France

{Iulian.Sandu-Popa, Karine.Zeitouni, Dominique.Barth, Sandrine.Vial}@prism.uvsq.fr

[*]*New Jersey Institute of Technology*

Newark, NJ 07102, USA

Vincent.Oria@njit.edu

Abstract: Indexing moving objects (MO) is a hot topic in the field of moving objects databases since many years. An impressive number of access methods have been proposed to optimize the processing of MOs related queries. Several methods have focused on spatio-temporal range queries, which represent the foundation of MO trajectory queries. Surprisingly, only a few of them consider that the objects movements are constrained. This is an important aspect for several reasons ranging from better capturing the relationship between the trajectory and the network space to more accurate trajectory representation with lower storage requirements.

In this paper we propose T-PARINET, an access method to efficiently retrieve the trajectories of objects moving in networks. T-PARINET is designed for continuous indexing of trajectory data flows. The cornerstone of T-PARINET is PARINET, an efficient index for historical trajectory data. The structure of PARINET is based on a combination of graph partitioning and a set of composite B^+-tree local indexes. Because the network can be modeled using graphs, the partitioning of the trajectory data makes use of graph partitioning theory and can be tuned for a given query load and a given data distribution in the network space. The tuning process is built on a good quality cost model which is supplied with PARINET. The advantage of having a cost model is twofold: it allows a better integration of the index into the query optimizer of any DBMS and it permits tuning the index structure for better performance. The tuning process can be performed before the index creation in the case of historical data or on-line in the case of indexing data flows. In fact, massive on-line updates can degrade the index quality, which can be measured by the cost model. We propose a specific maintenance process that results into T-PARINET. We study different types of queries, and provide an optimized configuration for several scenarios. PARINET and T-PARINET can easily be integrated into any RDBMS, which is an essential asset particularly for industrial or commercial applications. The experimental evaluation under an off-the-shelf DBMS shows that our method is robust. It also significantly outperforms the reference R-tree based access methods for in-network trajectory databases.

Keywords: *Moving object database, Access method, in-Network trajectory, Data flows*

1. Introduction

With the proliferation of mobile devices capable of accurately reporting their positions in time, it has become possible to accumulate large amounts of trajectory data. Moreover, the data acquisition can be made in real-time by using the ubiquitous wireless communication systems. A wide range of applications in areas like transportation planning, traffic management, location-aware services, rely on these data. Subsequently, an important research effort went into the general field of moving objects databases (MOD). Most of these works can fit in one of the following two complementary classes: modeling spatio-temporal databases; and indexing techniques to efficiently process spatio-temporal queries.

The performance issue has become critical in spatio-temporal applications due to the large amount of data and the computation cost of geometric operators. An impressive number of access methods have been proposed for efficient processing of moving objects (MO) queries. We can classify these index methods from a temporal or a spatial point of view. From the temporal perspective, some techniques aim at indexing real-time application data with the objective of minimizing the update and retrieval costs. Examples include TPR*-tree [17], STRIPES [12] and ST2B-tree [4], to name a few. Some other techniques focus on indexing complete (past) trajectories of MOs and aim at reducing the retrieval costs in large datasets. Several access methods such as the MV3R-Tree [8], [16] have been proposed in this context.

From the spatial perspective, most MO access methods, e.g., [4], [8], [12], [16], [17], consider that the objects are moving freely in the space. However, in several real-life applications, the object movements are constrained (e.g., trains moving along a railroad network or vehicles moving along a road network). Taking into account the network can lead to specific models that are optimal for the data representation. A few works [1], [5], [14] have proposed access methods for objects moving in networks. We will present them in more detail in the related work section, as this paper is situated in the same context.

The existing indexing techniques for objects moving in networks decompose the network into roads, and then index the spatio-temporal location of the MOs on each road with a specific index, e.g., a 2D R-tree. One of the shortcomings of this approach is the way the space is decomposed: it is solely determined by the road network, and takes neither into account the distribution of the trajectory data, nor the queries on the data. Hence, more recent access methods for non-constrained MOs, e.g., [2], [4], have proposed to partition the 2D space according to the data distribution. Moreover, indexing both the spatial and temporal dimensions for a given road is not always useful, since the spatial dimension (i.e., relative positions) tends to be less selective than the temporal one in most cases.

Another important observation in the case of MO trajectory data is that the datasets can be very dynamic and can span over very long periods of time, which are expanding continuously. For example, it is not uncommon nowadays to continuously monitor the traffic in certain road networks or highway networks, and also to record all these data for future use. Indeed, numerous applications are based on analyzing the historic (trajectory) data, e.g., for location-based services, for traffic planning, for measuring the traffic impact on the environment, for infrastructure developments, etc.

A more general problem suggested by the above mentioned applications is to efficiently manage trajectory data flows. The existing techniques for indexing the current and near-future movements of MO focus on tracking the positions of a set of MO. The challenge for an index in this case is the ability to continuously adapt to the spatio-temporal distribution of the data and to find a balance within the update and query cost tradeoff. These methods discard the historical data. On the other hand, the methods that index past trajectory data consider mostly

static datasets that are known in advance (since the data is historical) and that are subject to little or no changes. The main issue in this case is to optimize the retrieval cost of spatio-temporal queries. More recently, Pelanis et al. [13] proposed an indexing technique for capturing the positions of moving objects at all points in time (past, present, and anticipated future). Nonetheless, the focus of this work is on indexing the transition from present states to recent-past states of the data, while indexing the whole past is not a concern.

Therefore, in a more general context, it would be interesting to have an access method that efficiently processes the spatio-temporal queries over the recorded history, while *continuously* recording the *history* of trajectory data *up to the current time*. Note that in the general context, the focus is still on the *snapshot (spatio-temporal) queries*, i.e., which are evaluated only once.

In this paper, we propose T-PARINET, an efficient method for indexing in-network trajectory data flows. The structure of T-PARINET is based on PARINET, i.e., a PARtitionned Index for in-NEtwork Trajectories. Thus, we first present PARINET, an access method to efficiently retrieve the (past) trajectories of objects moving in networks. Given a data set of trajectories, PARINET proceeds by partitioning the data and indexing the partitions with composite B^+-trees. This allows exploiting the built-in B^+-tree, a robust and efficient index structure that exists in every database system. Instead of using a 2D grid as in the previous methods, the partitioning of the data is based on graph partitioning theory in order to integrate the network topology. In addition, we proposed a cost model that allows tuning correctly the index structure for a given query load. The part of the work dealing with indexing constrained trajectory data has already been published in [24]. Similar to the existing approaches, the focus of that work was on indexing historical trajectory datasets, i.e., where the data are known in advance and are subject to little or no changes.

Then, we extend PARINET to solve the more general problem, i.e., indexing trajectory data flows. The Temporal PARINET (T-PARINET) provides an optimized handling of trajectory data flows. T-PARINET is configurable in a dynamic environment and to fulfill its goal, T-PARINET uses an on-line tuning process that creates periodically a new PARINET to index the trajectory data from the current moment to a future moment in time. The on-line tuning process is based on monitoring a set of parameters indicating the quality of the last built index in the structure of the T-PARINET. Hence, our approach is to propose a smooth between static indexes for continuous indexing of trajectory flows. More specifically, the contributions of this paper are the following:

- We propose a novel access method called PARINET to index datasets of in-network trajectories. PARINET is based on graph partitioning and time interval indexing.
- We present a cost model that combines the statistics on the data and the query workload to estimate the number of disk accesses for a given index configuration.
- We show how PARINET can automatically choose a good index configuration, based on the provided cost model, the data distribution and the query workload using well-known graph partitioning algorithms.
- We also propose Temporal PARINET for indexing continuously and efficiently in-network trajectory data flows. T-PARINET uses an on-line tuning process to automatically determine the index evolution in time. The tuning process is based on the cost model of PARINET adapted to the context of indexing trajectory data flows.
- We characterize the query types in the network constrained MO context and provide different test scenarios.

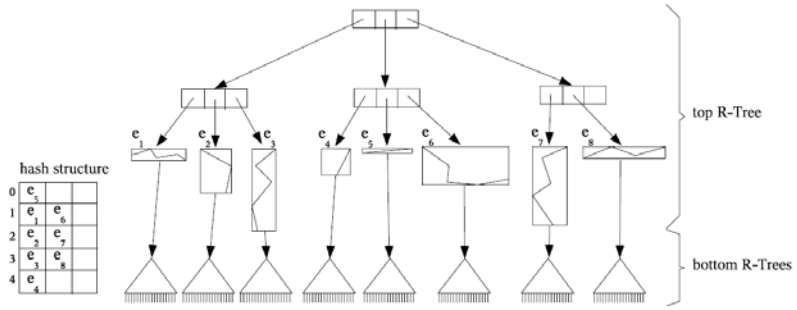

Figure 1: Example of the MON-tree index structure [1]

- We have implemented PARINET and T-PARINET using an off-the-shelf DBMS and validated our approach using an extensive experimentation that shows their efficiency and their scalability properties.

The rest of this paper is organized as follows: Section 2 presents the related work. Section 3 introduces the context for T-PARINET by defining the network model, the data model and the query types. Section 4 contains the description of PARINET along with the cost model and the tuning process. Section 5 introduces T-PARINET to continuously index in-network trajectory data flows. The experimental results are given in Section 6. Finally, we conclude and discuss some directions for future work in Section 7.

2. Related Work

2.1 Indexing Moving Objects in Networks

As pointed earlier, considerable attention has been paid to indexing methods for moving objects. Most of these works deal with indexing past, present or near-future positions of MOs that move freely in a two-dimensional space. There are only a few methods for indexing (past) trajectories of MOs in networks, which is the focus of this paper. The trajectories are represented with reference to a network, i.e., with the relative positions of the MOs on network edges [7]. The main idea in the previous works is to decompose a three-dimensional problem in two sub-problems in lower dimensions and then use a combination of two-level R-trees to index the trajectories.

The approach by Pfoser and Jensen [14] uses two 2D R-trees: one for indexing road edges and the other for accessing 2D transformed trajectory segments. The 3D (x, y, t) coordinates of a trajectory are mapped into a 2D (p, t) coordinate space using a Hilbert curve to linearize the network line segments. The same mapping is performed for queries. However, this generally leads to multiple sub-queries and may decrease the performances. For simplicity we will refer to this approach as PJ-tree in the rest of the paper.

The FNR-tree [5] utilizes a 2D R-tree to index road segments. For every leaf node in the 2D R-tree, there is a 1D R-tree to index the objects whose trajectories cross the segments included in the leaf node at a certain period of time. A major disadvantage of the FNR-tree is its limitation in trajectory modeling. Since only the time intervals are stored in the 1D R-tree, it is assumed that the objects cannot stop, change speed or direction in the middle of a road segment.

This limitation is addressed in [1] by the MON-tree. The MON-tree (see Figure 1) is composed of a 2D R-tree (the top R-tree) that indexes the network edges and a set of 2D R-trees (the bottom R-trees) that index the object movements along the edges. An additional hash structure used to map each edge to its corresponding tree helps speed up insertions.

Given a 3D spatio-temporal query, the top R-tree is used to find the precise intersection between the spatial part of the query and the network. Based on this intersection, a set of sub-queries is generated for each intersected part of each edge involved. Then, the corresponding bottom R-trees are accessed in order to respond to the sub-queries. MON-tree can handle two network models: an edge oriented model and a route oriented model (see Section 3.1). The experimental evaluation in [1] of the MON-tree against the FNR-tree shows that the first method always outperforms the second. The MON-tree on a route oriented network model shows better results.

We also would like to mention PIST [2] which indexes past positions of MOs in a 2D space. Although the application domain is different, it shares some similarity with the PARINET index as it combines partitioning and indexing as well. The data to be indexed consist of 2D points associated with time intervals, i.e., tuples (x, y, t_s, t_f) where x and y represent the coordinates in the 2D space, t_s and t_f represent a time interval. The type of queries considered is of spatio-temporal range type defined by a 2D spatial region and a time range. First, the 2D space is partitioned with a non-uniform grid, whose coarseness depends on the data distribution, data size and expected query size. Then, the data corresponding to each grid cell is indexed on (t_s, t_f) using a B^+-tree. The experimental results show a good performance of the method in comparison with R-tree based indexes.

2.2 Indexing MO Trajectory Data Flows

To the best of the authors' knowledge, no work in the MOD area considers the problem of continuously indexing data flows of trajectories (for constrained or non-constrained MOs) to optimize spatio-temporal queries. The closest works we found are the ones that focus on continuously tracking a set of non-constrained moving objects.

These works mainly index two types of queries. A first group of methods, such as TPR-tree [18], TPR*-tree [17], B^x-tree [26] or ST2-B-tree [4] have been proposed to optimize *snapshot spatio-temporal queries* that refer to present or near-future times. Typical examples of such queries are: *"Which MOs are within 1 km of my location right now?"* or *"What will be the number of MOs in the city center ten minutes from now?"*

A second group of methods, such as SINA [21], Q-index [22] and CNN [23], optimize *continuous spatio-temporal queries*. An example query in this context is: *"Continuously report the hotels within 5 km of my location"*. Both the first group and the second group of works mainly focus on spatio-temporal *range* queries, but these approaches remain applicable to a broader class of spatio-temporal queries, e.g., nearest-neighbor or aggregate queries.

In the above mentioned works, only the current positions (along with, in some cases, the current velocity vector used to predict near-future positions) of the tracked MOs are indexed. The past states representing the MOs' trajectories up to the current time are discarded. A few works deal with indexing the movements of non-constrained MOs at all points in time. These works inlcude the proposal of Sun et al. [19], the BB^x-tree [20] and the R^{PPF}-tree [13]. A common feature in these approaches is the focus on indexing the transition from present states to recent-past states of the data, while indexing the whole past is not a concern.

In [19] Sun et al. proposed a method to approximately answer aggregate spatio-temporal queries at all moments in time. The method is based on a multidimensional histogram representing the spatio-temporal evolution of the distribution of the moving objects. A main memory structure is used to keep the distributions corresponding to the current time and to the recent past. Older states of the histogram are migrated to the secondary storage and are simply indexed using a single packed B-tree or a 3D R-tree.

BB^x-tree [20] is an extension of the B^x-tree [26] (see above) and consists in an array of indexes that store the old phases of a rotating B^x-tree. For the past states, each index covers a time interval equal to $1.5T_{max}$, where T_{max} represents the anticipated maximum duration between two consecutive updates of any moving object. Also, the lifespans of two consecutive indexes overlap on an interval of length T_{max}. Since T_{max} can be very small in comparison with the length of the recorded history of the movements of moving objects, it is expected to have a large number of indexes in the structure of a BB^x-tree even for relatively short periods of time (e.g., of a few weeks). Moreover, the short lifespan of an index and the index overlap, make that normal range queries intersect several indexes, which will increase the query processing overhead. Another disadvantage of BB^x-tree is that it does not record the real position of the MOs, but only estimations of the MOs' location.

R^{PPF}-tree [13] is another proposal to index the positions of non-constrained moving objects at all points in time for timeslice queries. R^{PPF}-tree is based on the TPR-tree [18]. Pelanis et al. applied the partial persistence paradigm to the TPR-tree to enable it to retain and query past states of the indexed data. The focus here is on the application of partial persistence to the TPR-tree, which is not a trivial task. Once more, a single R-tree-like structure is used to index all the data spanning from the recording start instant in the past to the current time. This may lead to an important degradation of the index structure due to the high number of index entries.

3. The Context of PARINET and T-PARINET

The constrained movement requires a specific data representation but also specific query models [7]. It is important that the data representation be related to the network space instead of the 2D space in the case of constrained movement for several reasons. The first reason is that the 2D model does not capture the relationship between the trajectory and the network space, while this information is essential for the trajectory analysis. Queries relative to a network space (e.g., where are the gas stations on the A6 highway) are more suited to constrained moving objects. The second is the limit of the trajectory representations, estimated by linear interpolations between the reported positions, while the MO follows in fact the geometry of the network. In addition, the constrained model allows for dimensionality reduction by transforming the network in a 1D space by juxtaposing all the line segments [14]. Taking into account the network leads to a better storage and query performance than with the free trajectory model.

This section presents the context for PARINET and T-PARINET. We first introduce the network model, the data model and the query model. Then, we conclude by giving the hypotheses that guided this proposal.

3.1 Network Model

The network model defined for PARINET is similar to those in [1] and [5]. We use two representations for the road network: a geometric view and a topologic view. The geometric view (or 2D view) captures the approximate geographic locations of the road network components. This is the base view of the road network. The topologic view uses a graph in order to represent the road sections and the intersections. It is useful in the partitioning of the network. The notations we use are similar to the ones in [15].

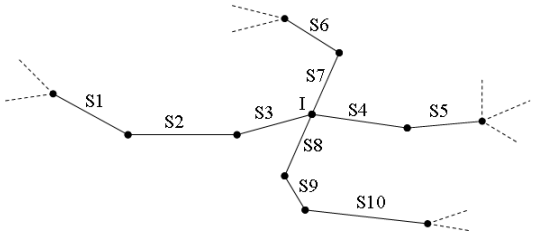

Figure 2: Example of a geometric representation of a network

The geometric representation of a *road network* is given by a tuple $RN^{2D} = (S,C)$, where S is a set of segments and C is a set of connections. A *road segment* $s \in S$ is a 2D line segment defined by (p_s, p_e), where $p_s = (x_s, y_s)$, $p_e = (x_e, y_e)$ and $p_s \neq p_e$; p_s and p_e are respectively the start and end points of the segment. A connection $c \in C$ is a tuple (p, S^c), where p is a geographical point that represents the location in the 2D space of the connection and S^c is a set of segments that meet at the connection. The list of segments in S^c should have p as one of their end points. Figure 2 gives a simple example of a geometric representation of a road network.

Definition 3.1: Given a road network RN^{2D} as described above, we define a *road* in RN^{2D} as $Road = (rid, S^c, start)$, where rid is a unique identifier, S^c is a set of connected segments that form a non self-intersecting polyline in RN^{2D} (which may be open or closed (a cycle)) and $start$ is one of the two endpoints of the polyline. Each segment belongs to one road only.

Definition 3.2: Given a road network RN^{2D} as described above, we define the set of junctions in RN^{2D} as $Junctions = \{ j \mid j \in C \wedge card(c(S^j)) \geq 3 \}$.

Different granularities can be superimposed to a road resulting in different network models.

Definition 3.3: For a given road network RN^{2D}, we define three possible network models:

- *Segment oriented network model*: each segment corresponds to a road.
- *Edge oriented network model*: each road is defined as the polyline between two *junctions*.
- *Route oriented network model*: the complete roads are considered without split. They can extend over the junctions. Notice that several configurations are possible for the route model on the same road network.

In the example in Figure 2, we have:

- 10 roads (S1,…,S10) in the segment oriented model;
- 4 roads: (S1, S2, S3), (S4, S5), (S6, S7) and (S8, S9, S10) using the edge oriented model;
- 2 roads: (S1, S2, S3, S4, S5) and (S6, S7, S8, S9, S10) in the route oriented model.

In the sequel, we will employ the general term *road network* to denote a road network modeled as one of the three possible network models. When necessary, we will indicate the specific network model for the given network.

Definition 3.4: Given a road network RN^{2D}, we define a position in the network space as a pair (rid, pos), where rid is a road identifier and $pos \in [0,1]$ is the relative position on the road measured from the $start$ end point of the road.

This is closely related to the concept of linear referencing widely used in GIS for transportation and available in DBMSs as Oracle Spatial or GIS tools as ArcGIS.

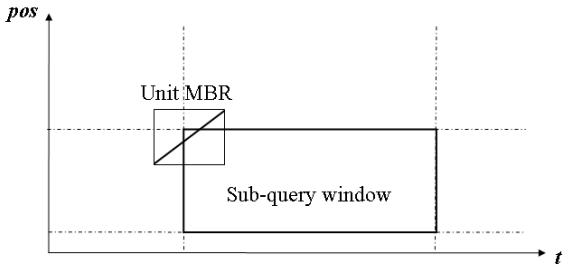

Figure 3: Example of unit and sub-query intersection

Definition 3.5: Given a road network RN^{2D}, we define a *road connection* rc as a tuple (p,R^c) where p is the geographical point location in 2D space of the road connection and R^c is the set of roads that meet at the connection. p has the same coordinates as one of the two end points of each road in the given connection.

Based on the 2D representation of a road network RN^{2D}, we construct the topologic representation of the network. In this representation, a network is defined as an undirected weighted graph $G=(V,E)$ with V a set of vertices and $E \subseteq V \times V \times \mathrm{N}$ a set of edges, where N is the set of natural numbers. Each $v \in V$ corresponds to one road connection in RN^{2D}. Given $v_1, v_2 \in V$, there is an edge $e=(v_1, v_2, w)$ in G iff there is a road in RN^{2D} between the corresponding road connections. The weight w is given by the function W, which depends on the data distribution and is defined in Section 4.3.3. Notice that in our network model, the roads are non-oriented. But taking into account the traffic orientation is a straightforward extension that can be achieved by splitting the two-way roads into two edges.

3.2 Data Model

As mentioned earlier, we intend to index the trajectories of the MOs in a network. An object moving on a road network reports its position at different moments in time. We assume that such an update is issued each time the MO changes its speed or passes on a different road in the network. An update contains the identifier of the MO, the network position (as given in Definition 3.4) and the associated time instant: $(moid,rid,pos,t)$. We define the *trajectory* of a moving object as a non-regulated sequence of *units* (i.e., the time intervals are not of equal size). Each unit is a tuple defined by two consecutive updates: $(moid,rid,[pos_1, pos_2],[t_1,t_2])$; t indicates a time instant, while pos gives the relative position on the road at the beginning and the end of the time interval [1], [7]. For each unit, it is assumed that the MO moves at constant speed, i.e., a linear interpolation is considered over each interval. Given a road, the relative position on the road and the time can be viewed as the two orthogonal axes of a 2D space. In this space, we denote by *unit segment* the 2D line segment bounded by the points (pos_1,t_1) and (pos_2,t_2). Also, a *unit minimum bounding rectangle* (unit MBR) is the rectangle that contains the unit segment (Figure 3).

3.3 Query Types

There are several types of queries that have been studied in the field of MOD, such as range queries, spatio-temporal join, nearest neighbors, within distance (or e-distance join) or skyline queries to name but a few. Among these query types, the range [1], [5] and the nearest neighbor queries [28] are, probably, the most studied in the context of MOD. In this paper, we consider these two types of queries and focus on the range queries since the nearest neighbor

queries can be brought down to a succession of range queries as it will be discussed in Section 4.2.2.

The range queries are composed of a spatial part and a temporal interval: $Q=(Q_s,Q_t)$. The queries return either all the MOs that have lied within the area of Q_s, at a certain time interval Q_t, or only the pieces of the trajectories that overlap the query. We consider two types of range queries: *2D queries* and *path queries*. The difference between the two types of queries lies in the spatial part Q_s.

The spatial component of the first type of queries is a 2D region. Hence, the *2D queries* represent "standard" range queries [1], [5] and [14]. Thus, Q_s is a 2D region (usually a rectangle). In the rest of the paper, we will refer to this type of queries as 2D queries. To support 2D queries, a transformation of Q_s is performed first. The exact intersection between the 2D region and the network is computed. Then the initial region in Q_s is replaced with the intersected network region. Formally, the new Q_s is a set of road sections: $Q_s=\{rs_1,rs_2,\dots rs_n\}$ where $rs_i=(rid_i,[pos^i_{11},pos^i_{12}],[pos^i_{21},pos^i_{22}],\dots[pos^i_{k1},pos^i_{k2}])$ and $\{rs_i \neq rs_j\}$ and $pos^i_{m1} \leq pos^i_{m2} \wedge pos^i_{m2} < pos^i_{(m+1)1}$. Each rs_i represents a set of disjoint and ordered intervals on one road [1]. Multiple intersection intervals with the query region are possible when the road is a polyline, which is the case for an edge or route network model. Usually, one can use a 2D R-tree over the network to speed up the computation of the mapping between a 2D region and a network region (see the top 2D R-tree in Figure 1).

The constrained movement suggests another type of useful query. For example, "find in a database all the MOs whose trajectories intersect a given MO trajectory", or "find the number of MOs that traverse a given road section at a certain time (interval)" are *path queries* that need to refer to the network. Path queries represent a new type of range queries that we introduce. In a path query, the spatial part, Q_s, represents a path in the network, i.e., a sequence of connected road sections. For this type of queries, no mapping is needed from the 2D space to the network space and Q_s has the same formalization as above.

Beside the range queries, the nearest neighbor (NN) queries are another popular type of query in MOD. Moreover, there are several types of NN queries, e.g., reverse NN, aggregate NN or continuous NN. For simplicity, we will only refer to the conventional NN queries in this paper. That is, given a (static) position in the network space and a time interval Q_t, the query returns the k MOs that were closer (w.r.t. the network distance, i.e., the shortest path between two network positions) to the network location during Q_t. Additionally, the list of returned MOs can be sorted on the minimum network distance from each MO trajectory to the network position.

3.4 Observations

In this subsection, we give a short informal intuition of the PARINET index structure. We use a filtering and refinement approach. The main idea of our proposal is that an approximate index search could deliver very good performances in terms of computation time, while offering at the same time good results in terms of physical accesses. The overall performance of such an access method can surpass the "exact" index search used in the existing methods.

Actually, in a network space, the spatial dimension is composed of a discrete component (the road identifier) and a continuous component (the relative position on the road). T-PARINET is based on the four following observations:

Observation 1: The relative position dimension is usually less selective than the temporal dimension. Using an index on time for filtering candidates followed by a refinement step should be more efficient than using an R-tree on the two dimensions.

The MON-tree and the PJ-tree fully index the bi-dimensional space (relative positions and time) with a 2D R-tree. Nevertheless, it is expected to have an important amount of overlapping of the indexed units in the spatial dimension, because in general, trajectories traverse entirely the road segments in their path. Moreover, except for queries on very small regions, the usual queries cover many road segments. Therefore, indexing only the temporal dimension might be more efficient, since time is more selective in this case. For this reason, we use a B^+-tree combined with sorted data on the time components. This offers an efficient sequential range scan of the tuples that intersect the temporal query interval Q_t.

Observation 2: The partitioning of the network space should not be made only on a road identifier basis, as it is the case for the existing methods. It should be based on the data distribution and the network topology.

Indeed, while the alternative of one index per road offers the advantage of an exact filtering on one component of the spatial dimension, it nevertheless has a few shortcomings. The partitioning is strictly related to the static road view of the network and does not consider the data statistics (distribution of MOs over the network). This is an important aspect and is even more relevant in a historical context. Moreover, the performance of the existing methods, e.g., MON-tree depends on the granularity of the employed network (section, edge, or route based model). Another argument is that a network can contain several thousand roads and having a separated index for each one could degrade the system performance even for small datasets.

Instead, we propose an index structure that takes into account the data distribution over the network and the network topology. The network will be partitioned in network regions that will be balanced with respect to the amount of data in each region. Therefore, the parts of the network with less traffic (e.g., the peripheral ones) will have larger extents than the busy zones (e.g., the central ones). Queries are most of the time defined on regions where road segments are close or connected. A general rule is to group together the objects that are close, which will help return more results in a few page accesses (for instance, R-trees are based on this rule). Because we are dealing with a network, the grouping should take into account the connectivity of road segments, i.e., the network topology, in addition to the data distribution.

Observation 3: The access method should be supplemented with a good quality cost model that will allow tuning the structure for better performances.

Most of the existing methods offer an empirical evaluation of the performance. Some of the recent works, e.g., [2], [4], propose analytical cost models, which are useful for the index (self)-tuning. Our objective is also to provide an administration tool for tuning the index.

Observation 4: In the context of continuous indexing of trajectory data flows, the access method should be able to adapt to the variation in time of the data distribution and density. Also, the access method should be efficient w.r.t. insertions and robust to massive index updating.

Trajectory data are inherently divers in space and time [4]. For example, the central part of a road network is more circulated than the peripheral parts. Also, the traffic can be denser during peak times on working days than during the week-end. In the case of indexing trajectory data flows, the access method should be able to adapt to these variations, since different index configurations are near-optimal for different periods of time. Moreover, the update efficiency of the index and its robustness to massive updates are essential in this context. Once again, having an access method that is based on the B^+-tree index appears to be the right choice due to the efficiency of the B^+-tree in performing update operations.

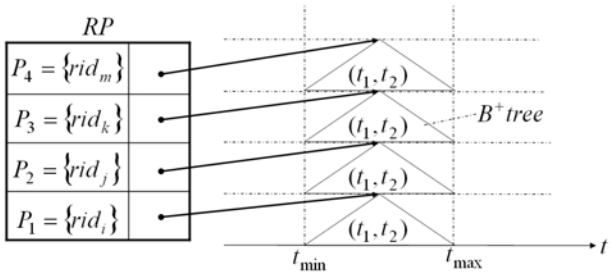

Figure 4: Example of PARINET index structure

4. PARINET Index

In this section we introduce PARINET for indexing datasets of in-network trajectories. PARINET constitutes the foundation for T-PARINET (see Section 5). PARINET is capable of answering two kinds of queries on historical constrained trajectories, namely range and nearest neighbor queries. We present first the index structure and its operations. Section 4.3 proposes a cost model based on query and data sizes, and formalizes PARINET tuning in terms of a graph partitioning problem. Finally, we show how one can automatically tune PARINET for a better performance, given a road network, the distribution of the data to be indexed, and an expected query workload.

4.1 Index Structure

Based on the above-mentioned observations, the intuition of our approach is to create a B^+-tree index on time intervals for the set of roads in each partition (retuned by the partitioning phase) rather than creating an index for each isolated road. The partitioning is based on both the data distribution and the network topology (cf. Observation 2), i.e., the partitions are balanced in terms of the amount of data and the partitions separate the network into regions (i.e., connected or close roads are grouped in the same partition).

The discussion on how one can choose a good number of partitions and how the partitioning is obtained is presented in Section 4.3. For now, we assume that this aspect is solved and a good partitioning can be obtained for a given network and a given data distribution. As a result of this operation, each road will be assigned to a certain partition (cluster).

Given a dataset D containing trajectories of MOs in a network as a set of trajectory units: $D = \{(moid, rid, [pos_1, pos_2], [t_1, t_2])\}$, the index is built in three steps: partitioning the trajectory units based on their road identifiers, sorting the partitions on the time intervals and indexing each partition using a composite B^+-tree on (t_1, t_2) interval. Note that an interval-based B-tree such as the RI-tree [10] could be used for indexing time intervals, but we chose a simple B^+-tree to allow an easier implementation. The index structure is quite simple. An example is given in Figure 4. A table RP (Road Partitioning) that contains one entry for each cluster keeps some basic information on the partitioning: the list of road identifiers for a cluster and a pointer to the B^+-tree index over the unit segments in the cluster. As we partition the data according to the spatial dimension, the time (t_{min}, t_{max}) represents the entire spanning time of the indexed trajectories. Therefore, only one *RP* table is necessary to report the relationship between the partition attribute (i.e., the *rid*) and the partition index.

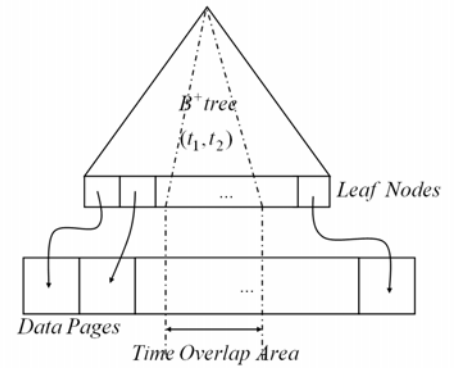

Figure 5: Example of index range scan

4.2 Index Operations

4.2.1 Range Queries

Given a (2D or path) spatio-temporal range query $Q=(Q_s,Q_t)$ where $Q_s=\{rs_1,rs_2,\ldots rs_n\}$ and $Q_t=[t_s,t_e]$ (see Section 3.3), PARINET can find all the objects that have traversed the road sections in Q_s during the time interval of Q_t, or simply return the trajectory units that intersect Q. Data retrieval is performed in three steps. First, we identify the partitions that contain the road identifiers in the query, i.e., the spatial filtering step. Then, we use the B^+-tree indexes of the selected partitions and look up candidate data, i.e., the temporal filtering step. Finally, we perform an exact match search among the candidates, i.e., the refinement step.

Based on the set of road identifiers $\{rid_{q1},rid_{q2},\ldots rid_{qn}\}$ in Q_s and on the distribution table RP, we determine the set of partitions $\{P_{p1},P_{p2},\ldots P_{pm}\}$ that include all the roads in a given query. Note that $m\le n$, but in general $m<n$ and we might also have $m\ll n$ depending on partition and query sizes. This means that the total number of searched partitions is smaller than the total number of accessed roads in general, as it is the case in a road oriented partitioning.

Then, for each accessed partition we perform a range scan by using the B^+-tree index in order to find the data pages that temporally overlap Q_t (see Figure 5). Note that this may lead to false positives, because the filtering is based only on time and does not consider the road identifiers or the relative positions on the road. However, the capability of accessing groups of roads that are likely to appear together in a query will lower the number of false positives.

Finally, at the refinement step, for each candidate data we determine if it truly intersects Q, i.e., the unit segment intersects one of the sub-query windows (as depicted in Figure 3). The actual intersection between the unit segment and the sub-query window is computed only if the unit MBR is not completely covered by the window.

4.2.2 NN Queries

Given a nearest neighbor query $Q_{NN}=(rid,pos,Q_t,k)$, where (rid,pos) is a network position (cf. Definition 3.4), $Q_t=[t_s,t_e]$ and k is a positive integer, one can use PARINET to efficiently find the k MOs that were closer to the network location during Q_t. To answer a NN query, we propose an approach that reduces a NN query to a sequence of range (path) queries. The proposed method is based on two phases, which can be repeated several times. In the first

step, the network graph is searched in a breadth-first manner starting from the query network position (query node). This is similar to the incremental network expansion proposed in [28]. Each time a new graph node is expanded, the algorithm verifies if the network distance from a new examined node to the start node is lower than a predefined threshold value th. If this condition is not verified, then the respective edge is split at the point situated at the distance th from the query node. The edge portion situated closer to the query node is included in the expansion and the graph search in this branch terminates. The first phase generates a set of network paths from the query node that have a length inferior to th. In the second step, the obtained paths will form the spatial part Q_s of a range query $Q=(Q_s,Q_t)$ that is processed using the index structure (cf. Section 4.2.1). If the number of MOs returned by Q is superior or equal to k, no additional search is necessary. The list containing the MO identifiers is sorted based on the minimum network distance to the query position and the top k MOs are returned as the final result.

Otherwise, the algorithm increases the threshold value (e.g., it doubles the last value) and repeats the two steps described above, i.e., it first continues with the network expansion and then evaluates a new range query. Note that the spatial part $Q_s^{'}$ of the new query should be computed as the difference between the previous and the current network expansion to avoid a repetitive search over the same network parts. The returned MOs are added to the current set of MO identifiers. The algorithm stops when the set contains at least k MO identifiers.

The above approach permits to transform a NN query into a succession of range queries, which can be in turn evaluated efficiently by PARINET. Nonetheless, there are two more aspects that need also to be considered here. A first issue is to find appropriate values for the threshold th. Assuming that the spatio-temporal distribution of the indexed data is known (see Section 4.3.2), the value th can be set inversely proportional to the spatial and temporal density of the data observed at the query location and the query time interval. Higher data densities indicate a higher probability to find the NN closer to the query point location and vice-versa. The second aspect concerns the efficiency of the network expansion step, since the graph search may be costly for large network graphs. Several existing solutions such as the proposals in [27], [28] can be employed to optimize the network expansion phase. For example, the connectivity-clustered access method proposed in [27] regroups the adjacency node lists of neighbor nodes on the same data pages to minimize the number of I/O.

4.2.3 Index Maintenance

Historical datasets are mainly static data. Therefore, the index construction can be done once and periodically updated. Practically, the data needs to be organized in partitions and the insertion of each unit in a bucket must be done with respect to (t_1,t_2) values. In a RDBMS for example, this can be done using a temporary view or table. Afterwards, a B^+-tree index is built for each bucket on (t_1,t_2). If additional data have to be added later, it can be directly appended to the existing partitions with the update of the corresponding index for each inserted unit. Usually, newly added trajectories are more recent than the existing ones, which have less impact on the B^+-tree. When the added trajectory units follow the same distribution as the initial ones and the amount of added data is small compared to the size of the existing data, the overall performance remains the same. Otherwise, a complete reconstruction of the index structure is needed in order to maintain the best performance. However, this is not feasible in the case of massive index updating. These aspects are thoroughly discussed in Section 5 and experimentally tested in Section 6.

4.3 Data Partitioning

4.3.1 Problem Statement

PARINET is based on the partitioning of the road network. Moreover, the partitioning must take into account the data distribution over the network (e.g., total number of unit segments for each road) and the network topology. It is clear that, for a given query load, different partitioning of the same data will lead to different performances. Our goal is to automatically find the best partitioning scenario for a given query load. This is possible as the network and the data to be indexed are known in advance.

This section presents a cost model that estimates the number of disk accesses necessary to answer a query load, given a certain configuration of the PARINET index. Then, using the cost model, we will rewrite the partitioning problem as an optimization problem and use a graph partitioning algorithm to resolve it. We assume that the overall performance (mainly the response time) of the index is directly related to the number of disk accesses.

4.3.2 PARINET Cost Model

In this section we present a cost model that estimates the number of physical disk accesses for a given query and index configuration. The notations used in this section are explained in Table 1.

The total number of disk accesses for a given query is the sum of the physical accesses in each accessed partition. We consider that the table RP, which gives the distribution of road identifiers in the partitions, is sufficiently small to fit in main memory. For each accessed partition, we have disk accesses for the range scan in that partition. A range scan comprises the index search and the data page scan (see Figure 5). We obtain Formula (1) for the total number of disk accesses: $DA_Q = \sum_{p \cap Q_S} (IA_p + PA_p)$ (1).

Table 1: Notations

DA_Q	Total number of disk accesses for a query
IA_p	Number of index accesses in a partition
IA_p^f	Number of fixed index accesses in a partition
IA_p^v	Number of variable index accesses in a partition
PA_p	Number of page accesses in a partition
N_p	Number of units (tuples) in partition p
$Pages_p$	Number of data pages in a partition
ρ_p^t	Temporal data distribution in a partition (percentage of $Pages_p$ per time unit)
T_{max}	Maximum length of the unit time intervals in the dataset
BS_i	Index block size (number of entries) per index page
BS_d	Data block size (number of entries) per data page

The data access cost is the number of pages containing the data that overlap with Q_t. Given the distribution of the data in time ρ_p^t, the number of pages read is:

$PA_p = Pages_p \times \int_{Q_t+T_{max}} \rho_p^t \cdot dt = \frac{N_p}{BS_d} \times \int_{Q_t+T_{max}} \rho_p^t \cdot dt$ (2). For simplicity, we consider a uniform temporal distribution such as $\rho_p^t = \rho_p = const$. In this case Formula (2) becomes: $PA_p = \frac{N_p}{BS_d} \times (|Q_t| + T_{max}) \times \rho_p$ (3) , where $|Q_t| = t_e - t_s$.

Note that T_{max} decreases the temporal selectivity of the query by enlarging the query time interval. The problem of long time intervals is well-known when indexing time related data. The usual solution is to decompose long time intervals into several smaller intervals. The drawback is that this will increase the data set size. However, this is not a problem with trajectory data sets because only a small percentage of the time intervals are long, i.e., there are few MO that are moving very slowly and that issue very rare updates. In [2] for example, they compute the optimal splitting of the long time intervals of a data set such as the data set size does not increase much. This kind of data preprocessing can also be applied to PARINET before the index construction. In general, constrained MOs such as vehicles moving in a road network, need to report their location frequently to have an accurate view of their trajectories. Hence, T_{max} is expected to be much smaller than Q_t and to have a limited impact on the query cost.

The number of index accesses is composed of a fixed cost and a variable cost. The fix cost comprises the accesses performed to reach the leaf nodes from the index tree root, which is equal to the tree height. The height of a B^+-tree is equal to the number of levels in the tree including the root level. This can be computed based on the number of index entries and the tree fanout: $IA_p^f = \lceil \log_{fan} N_p \rceil$ (4). A typical value for IA_p^f is 3 when $fan \approx 100$ and the number of index entries is in the millions of tuples. The variable index cost reflects the number of pages with leaf nodes that overlap with Q_t. Similar to PA_p, we obtain: $IA_p^v = \frac{N_p}{BS_i} \times (|Q_t| + T_{max}) \times \rho_p$ (5).

From Formulas (1), (3), (4) and (5) we obtain:

$$DA_Q = \sum_{p \cap Q_s} \left[\lceil \log N_p \rceil + N_p (|Q_t| + T_{max}) \rho_p \left(\frac{1}{BS_i} + \frac{1}{BS_d} \right) \right].$$

For the sake of simplicity, we consider that Q_t is implicitly enlarged with T_{max} in the following. The final formula for the number of disk accesses is:

$$DA_Q = \sum_{p \cap Q_s} \left[\lceil \log N_p \rceil + N_p \cdot |Q_t| \cdot \rho_p \left(\frac{1}{BS_i} + \frac{1}{BS_d} \right) \right] \quad (6).$$

One advantage of PARINET is that it allows a simple estimation of the disk accesses for a given query load, based on some statistics on the indexed data. This estimation can be used to automatically tune the index for a better performance. In short, we can modify the average area of network partitions by changing the total number of partitions n. Intuitively, given a query of a certain size, the number of disk accesses needed to answer the query will decrease with the partition size, because less false positives will be examined. However, increasing the number of partitions after a certain point will result in a performance loss. This is due to the fact that more partitions need to be considered, which increases the fixed index physical accesses and query overhead. The cost model is verified in the experimental evaluation presented in Section 6.

4.3.3 Using Graph Partitioning

Assuming that the above cost model is accurate, we can estimate the performance of the PARINET for a given configuration, without effectively constructing the index. Therefore, we can search among some of the possible configurations and materialize the best one with respect to the cost model. A possible index configuration corresponds to a network partitioning into a given number of parts that respects some given constraints (cf. Observation 2).

Graph partitioning is an important problem that has been extensively studied in the last decades. The problem is to partition the vertices of a graph in n roughly equal parts, such that the number of edges connecting vertices in different parts is minimized [9]. The problem was extended to graphs where each node and each edge can have weights. Therefore, the resulting partitions can be balanced in term of node weights instead of number of nodes, for example. The graph partitioning problem is NP-complete [6]. However, many algorithms have been developed to find high quality partitions extremely fast based on specific heuristics [9]. Public implementations are also available, e.g., METIS [11].

As formulated in Section 4.1, the constraints imposed by PARINET on the network partitioning, can be entirely satisfied by the graph partitioning algorithms. The formalization of the approach is the following: given an undirected network graph $G=(V,E)$ and a dataset D (as described in Section 3.2), we compute the weight function of the graph roads $W : E \to \mathrm{N}$. W associates for each road in G the number of units from D on that road. Let $L(G)$ be the line graph of G. W is a node weight function of $L(G)$. Let $P=\{P_1,P_2,\ldots P_n\}$ be the partitioning of $L(G)$ in n parts, such that the partitions are *contiguous* and *balanced* in terms of total weight. Let $Q_L=\{Q_1,Q_2,\ldots Q_k\}$ be a query load. We define the quality indicator of P over $L(G)$ as: $QI^n_{Q_L}=\sum_{i=1}^{k} DA_{Q_i}$ (7), where DA_{Q_i} is computed by (6).

Algorithm 1: Determining index partitioning

Input: Network graph $G=(V,E)$, trajectory dataset D, query load $Q_L=\{Q_1,Q_2,\ldots Q_k\}$

Output: Road Partitioning function $RP : E \to \{1,2,...,p\}$

1. Compute $W : E \to \mathrm{N}$ given G and D
2. Compute $L(G)$ of G
3. $QI^{optimal}_{Q_L}=\infty$
4. **for** $m=1$ **to** $card(E)$ **do**
5. $\quad RP_m \leftarrow METIS(L(G),m)$
6. $\quad QI^m_{Q_L}=\sum_{i=1}^{k} DA^m_{Q_i}$
7. $\quad$ **if** $QI^{optimal}_{Q_L} > QI^m_{Q_L}$ **then**
8. $\quad\quad RP \leftarrow RP_m$
9. $\quad\quad QI^{optimal}_{Q_L}=QI^m_{Q_L}$
10. **return** RP

The goal is to find the partitioning such that QI_{Q_L} is minimal (Algorithm 1). The idea is to implement a program that is based on METIS [11] and that returns the partitions with the best

QI_{Q_L} by iterating through the possible index configurations. METIS takes as input a weighted node graph and a number m of parts (line 5 in Algorithm 1). It partitions the input graph in m parts such that the partitions are fairly balanced and contiguous (although this is not guaranteed, non-contiguous portions are exceptions), which is conform to our demands for the partitioning of the road network (cf. Observation 2). By iterating with m from 1 to $card(E)$, we choose the partitioning with the best QI for the materialization of the index structure. Notice that our experimental results showed that a step of 100 for m in the iteration is sufficient because usually $QI_{Q_L}^m$ has small variations with m. Thus, the computation time for the optimal partitioning takes about one minute on our testing machine, which is negligible compared to the time necessary for testing several index configurations. For example, it takes several minutes to index about one million trajectory units. The time required to test the index performance needs also to be considered. Notice also that the partitioning algorithm can work with any network granularity, i.e., segment, edge or route (cf. Definition 3.3), since a graph representation can be built for the road network for each of the possible network granularities.

5. Temporal PARINET

Trajectory datasets can be very dynamic, i.e., characterized by frequent updating. In general, the updates do not concern the exiting data (although this type of update is possible, it is less probable), but rather about inserting new (parts of) trajectories to the dataset as time goes. New data can be added continually either as periodical large batches of updates or by continuously logging the individual updates coming from tracking a group of moving objects. Typically, the newer data are also the most recent from a temporal point of view.

In this context, a good access method should not only offer a good performance in term of querying the trajectories dataset, but it should also perform efficiently the updates and should have a robust performance in time (cf. Observation 4). Ideally, an index should be capable of integrating at a low cost the continuous incoming updates, while allowing to process queries over the complete dataset. Moreover, the query and update performances of the index should not degrade in time.

Clearly, using a single index structure is not an appropriate solution for two main reasons. First, the performance of most indexes degrades with the size of the dataset. The query performance of the R-tree (frequently used in this context) degrades with the number of indexed entries [25] due to an increase in the number of overlapping MBBs. Also, any tree-like index will continue to grow with the number of index entries. Although the increase in the tree height is logarithmic this aspect can not be neglected in the case of virtually infinite datasets such as trajectory datasets. Second, as indicated in the previous section, trajectory data are inherently divers in space and time. An index structure such as PARINET can balance the spatio-temporal diversity by choosing the best configuration for a given dataset, i.e., for a dataset corresponding to a time interval. However, the spatial distribution of the data can change between the observed time periods. Therefore, different index configurations are near-optimal for different periods of time.

In this section, we propose Temporal PARINET (T-PARINET) to continuously index the trajectories of in-network moving objects. Given a road network, T-PARINET periodically creates a new PARINET index that will span over a certain temporal window. The structure of the new index is determined based on an expected spatial distribution of the data and an expected query size by using an extended PARINET cost model. The construction of a new index is triggered based on two parameters that are continuously monitored, i.e., the *current index degradation* (due to the difference between the expected and the real data distribution

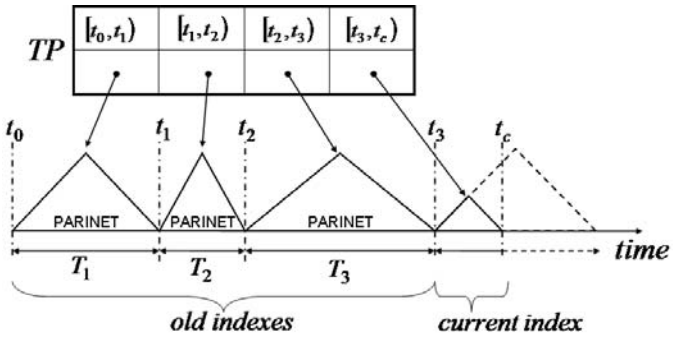

Figure 6: Example of T-PARINET index structure

and query size) and the *expected degradation* (due to an increase in the index height as the data accumulate).

This section is organized as follows: we introduce the index structure and operations of T-PARINET in Section 5.1. In Section 5.2 we propose a simple solution to optimize the cost of the updates and thus, to increase the index throughput. Section 5.3 presents the extended cost model and the on-line tuning algorithm used to decide on the development of the index structure in time.

5.1 T-PARINET Structure and Operations

T-PARINET consists of a sequence of PARINET indexes that are associated with different time intervals covering the index lifespan from t_0, corresponding to the oldest data in the trajectory dataset, to the current time (t_c). An example of T-PARINET is given in Figure 6. Each component index PARINET$_i$ is associated to a time interval $T_i = [t_{i-1}, t_i)$. The time intervals partition the lifespan $[t_0, t_c)$ of the global index. The time intervals of the component indexes are disjoint. A *Time Partitioning* table (*TP*) has one entry for each component index, which contains the corresponding lifespan and a pointer to the index.

There are two types of component indexes in a T-PARINET. There is one *current index*, which is the component index covering the data in the most recent time interval. The current index is affected by both queries and massive updates. The lifespan of the current index is continuously expanding to the right with the current time. The rest of the component indexes represent *past indexes*. A past index is usually only queried and has a fixed lifespan. Updates are rare.

5.1.1 Search Algorithm

Given a spatio-temporal range query $Q = (Q_s, Q_t)$, where $Q_s = \{rs_1, rs_2, \ldots rs_n\}$ and $Q_t = [t_s, t_e]$, the query processing for T-PARINET has two steps. First, the intersection between Q_t and the time intervals in the table *TP* is computed, resulting in a set of time intervals $\{Q_t^1, Q_t^2, ..., Q_t^k\}$. Then, the initial query is mapped to a set of *k* queries, where $Q^i = (Q_s, Q_t^i)$ and $i = \overline{1,k}$. Each query Q^i is then evaluated by using the corresponding component index PARINET$_i$, where the search operation is described in Section 4.2.1. Note that the number of mapped queries k is expected to be low, i.e., $k = 1$ or $k = 2$ in most cases, since the length of the query time interval Q_t is normally smaller that the lifespan of a component index (see Section 5.3.2).

5.1.2 Index Evolution in Time

Given a dataflow of moving object updates *(moid,rid, pos,t)* in a road network, we can build a T-PARINET to index the historical trajectory data. At time t_0 the structure of the index is initialized with an empty PARINET and by inserting the first time interval $[t_0,t_c)$ in the *TP* table. As presented in Section 4, the structure of a PARINET index is determined based on the distribution (i.e., the temporal density of trajectories on each road) of the trajectory dataset that we want to index and on an expected query load. In the case of continuous indexing a dataflow of trajectories the data (distribution) is not exactly known in advance. Hence, the structure of a new component index in T-PARINET will be computed based on an expected distribution of the data.

There are several ways of anticipating the spatio-temporal distribution of the data. For example, this can be based on statistics of the traffic from previous observations in the road network. If this type of information is not available, one could consider, for instance, a uniform trajectory distribution (although more elaborated models could be easily devised) for the first component index. Then, as the time passes and new component indexes are instantiated, the past distribution of the data can be used to foresee a possible future trajectory distribution. Nonetheless, the index should be robust w.r.t. both the data distribution and the query size.

Once the current component index is instantiated, it will be continuously updated with new trajectory units and also queried. After a period of time, when the index degradation due to the difference between the expected and the real data distribution and density, exceeds a certain threshold, the construction of a new component PARINET index is triggered. This aspect is thoroughly discussed in Section 5.3. Also, the time interval in the TP table is updated for the past index, e.g., from $[t_0,t_c)$ to $[t_0,t_1)$, and a new entry $[t_1,t_c)$ is inserted for the new current index. This process can then continue indefinitely. Note that the initialization of a new component index is not costly since only the declaration of the index configuration is effectively processed and stored into the database.

5.1.3 Index Maintenance

In the context of T-PARINET, i.e., indexing trajectory flows, we expect the delete operations to be rare. However, in some cases, the database administrator might want to remove (parts of) trajectories from the indexed data. There are mainly two types of delete operations. First, one would want to remove all the trajectory units that overlap with a spatio-temporal window. The delete operation in this case consists of two steps, i.e., a search operation is performed first to determine the data units that overlap with the spatio-temporal window and then the selected units are removed from the dataset and from the index entries. Second, one would want to delete an entire trajectory from the indexed dataset. In this case, all the trajectory units that belong to the indicated trajectory need to be found and removed. This can require a complete data scan. Alternatively, to optimize the deleting of an entire trajectory, an additional B^+-tree index on the MO identifiers of the trajectory units can be used to speed up the search for the units belonging to a trajectory. Note that this will lead to an increase in the cost of the insert operation, since we need to update both the additional B^+-tree index and the T-PARINET index.

An interesting observation concerning T-PARINET is that the index structure is adapted to archiving operations. As time passes, the oldest component indexes in a T-PARINET may not longer be of interest. For example, several applications may need only the most recent data, whereas the aged data may be interesting only from a statistical point of view.

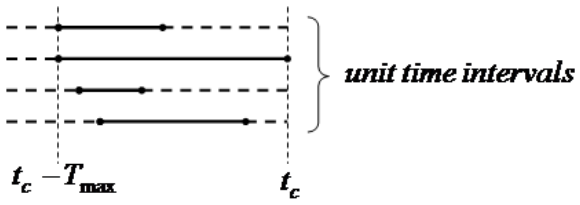

Figure 7: Example of sorted trajectory units in the last *Tmax* time interval

Therefore, the oldest component indexes in a T-PARINET can be periodically archived and removed from the index structure, along with the corresponding entries in the *TP* table. The archiving process could also include the creation of some data models that keep aggregate spatio-temporal information about the past, e.g., the traffic density, the traffic distribution and the past user queries.

5.2 Optimizing Update Operations

Given a T-PARINET structure that continuously stores and indexes a flow of trajectory updates, only the current index in the structure is modified by the updates. The current index, which is a PARINET index, consists in a forest of B^+-trees over clusters of trajectory units (see Section 4.1). The trajectory units in each cluster need to be kept sorted on the units' time interval $[t_1, t_2]$ to ensure an optimal query performance of the index. This constraint is easy to preserve since the updates are inherently chronologically ordered. However, an in-memory buffer that stores all the updates in the time interval $[t_c - T_{max}, t_c)$ is needed to be certain that the new trajectory units are inserted sorted on the units' time interval (cf. Figure 7). The reason is that, as shown in Figure 7, the time intervals can have different lengths, but insertions occur once t_2 is known.

In the context of adding intensively and continuously new trajectory units to the indexed dataset, the cost of the insert operation becomes crucial for the index throughput. Moreover, the robustness of the index with regard to the insert operation is also important. PARINET is based on the B^+-tree index. This type of approach has an important advantage over the existing methods that are based on the R-tree index (see Section 2.1). The index operations in a B^+-tree (e.g., search, insertion, and deletion) can be performed more efficiently than in an R-tree. The difference in performance between the two structures is even more significant in a concurrent environment. This represents an essential aspect for real-time applications, where frequent queries and updates arrive simultaneously. Moreover, in our context the insert operation can be performed even more efficiently, since the new data can be directly appended to the existing data, i.e., the index is expanding only to the right as seen on the time axis. In addition, this type of append-only insertion will indirectly provide good index robustness. This is not the case for the related methods that are based on the 2D R-tree index (except for the FNR-tree that uses a 1D R-tree to index only the time interval).

Although we expect PARINET to offer a good performance w.r.t. the cost of the insert operation, some simple optimizations, yet having potentially an important impact on the update cost, can still be considered in this context. Recall that PARINET partitions the dataset over a number of clusters (corresponding to network regions) that are each indexed with a B^+-tree. Let us consider a data page of updates buffered in main memory that needs to be inserted into the index. In the case of a non-partitioned index, the buffered data page is copied to disk at the cost of one I/O operation and then the index is updated at the cost of one or several I/O operations. On the other hand, since the index is partitioned in our case, the buffered updates will generally fall in different partitions. Therefore, one page of buffered updates will require

several disk accesses to copy to disk, i.e., equal to the number of involved partitions. Moreover, all the local indexes in the affected partitions need to be updated, increasing even more the insertion cost.

Hence, the partitioning used by PARINET, which greatly improves the query performance in a static environment, can have a reverse effect on the insertion operations. This shortcoming can be easily avoided. Instead of buffering the updates page-wise, i.e., gathering one page of updates before copying to secondary storage, one could use a partition-wise buffering, i.e., one in-memory data page for each partition. A simple in-memory hash structure having one package (e.g., of a disk page size) for each partition can be used. The updates are committed to disk only when a package overflow occurs. The insertion of the trajectory units in a package will only affect one partition, minimizing thus the operation cost.

5.3 Temporal Partitioning

5.3.1 Problem Statement

In the context of continuous indexing of trajectory data flows, T-PARINET creates periodically a new PARINET structure to handle the data that will be collected in a temporal window. Three main reasons explain this process. First, it permits to better adapt the index structure to the possible spatio-temporal diversity of the data in the dataflow and also to the query size and query distribution over the road network. For example, if we can anticipate the spatio-temporal distribution of the data and the query size for a certain time period, based on past observations, then we can build a PARINET index that offers an optimal performance w.r.t. the cost model presented in Section 4.3. Second, it allows limiting the degradation of the current index in a T-PARINET. Since the structure of every component index is determined based on an expected data distribution and an expected query size, the differences between the actual and the expected data will lead to a degradation of the index performance. However, by monitoring the index performance loss, we can bind it to the process of periodic indexing. Third, the same process has a positive effect on the index maintenance. Given a past index in a T-PARINET that has a degradation exceeding a certain accepted limit, we can rebuild off-line the component index to a near-optimal configuration. In this case, only the local index is concerned, which greatly limits the cost of the operation.

Since the process of indexing a trajectory dataflow is continuous, the index should be able to determine automatically when it is the best moment to trigger the construction of a new component index. Therefore, a temporal partitioning model, which considers the index degradation, is needed. At the same time, the temporal partitioning model should include some parameters such as the lifespan of a component index in addition to the degradation factor. The time interval covered by a component index needs to be larger than the time interval of the queries in general. Otherwise, the queries will overlap several component indexes, which will lead to a degradation of the query performance, since more indexes need to be traversed to answer a query. These aspects are developed in the following two subsections.

5.3.2 T-PARINET Cost Model

In this section we revisit the cost model of PARINET proposed in Section 4.3 in the context of T-PARINET. We define the constraints and the parameters needed by T-PARINET for an on-line tuning of the evolution of the index structure in time. The main idea is that the construction of a new component index is triggered whenever the degradation of the current

index exceeds a certain defined limit. However, the time interval covered by each component index should not be too small compared to the time interval of the queries as indicated above.

Given a spatio-temporal query $Q=(Q_s,Q_t)$ over a T-PARINET index and assuming that the query time interval $Q_t=[t_s,t_e]$ is inside the lifespan of a component index PARINET$_i$, the number of disk accesses needed to answer this query is:

$$DA_Q = \sum_{p \cap Q_s} \left[\left\lceil \log N_p^i \right\rceil + N_p^i |Q_t| \left(\frac{1}{BS_i} + \frac{1}{BS_d}\right) \int_{t_s}^{t_e} \rho_p^i(t) \cdot dt \right] \quad (7).$$

N_p^i and ρ_p^i have the same significance as N_p and ρ_p defined in Section 4.3.2 and they are measured relatively to the component index PARINET$_i$.

Assuming that the temporal interval of a query can be situated with equal probability at any instant within the lifespan of the T-PARINET, we can use an average temporal data distribution instead of the local temporal distribution $\rho_p^i(t)$. The average temporal distribution is: $\tilde{\rho}_p^i = \frac{\int_{t_{i-1}}^{t_i} \rho_p^i(t) \cdot dt}{|T_i|} = \frac{1}{|T_i|}$, where $|T_i| = t_i - t_{i-1}$ is the lifespan of the local index. Note that this assumption does not limit the generality of the cost model. In the case of non-uniform temporal distribution of the queries, $\tilde{\rho}_p^i$ can be estimated based on the specific distributions.

Then, Formula (7) can be rewritten as: $DA_Q = \sum_{p \cap Q_s} \left[\left\lceil \log N_p^i \right\rceil + N_p^i \left(\frac{1}{BS_i} + \frac{1}{BS_d}\right) \frac{|Q_t|}{|T_i|} \right]$ (8),

where $|Q_t| = t_e - t_s$. To simplify the formulas, we use the notation $\sigma_p^{Q_t} = N_p^i \left(\frac{1}{BS_i} + \frac{1}{BS_d}\right) \frac{|Q_t|}{|T_i|}$ representing the (average) temporal selectivity of a query in a partition. Using this notation the number of disk accesses for given a query is:

$DA_Q = \sum_{p \cap Q_s} \left(\left\lceil \log N_p^i \right\rceil + \sigma_p^{Q_t} \right)$ (9). Also, given a query load $Q_L = \{Q_1, Q_2, \ldots Q_k\}$ the quality indicator of PARINET$_i$ is $QI_{Q_L}^i = \sum_{j=1}^{k} DA_{Q_j}^i$ (cf. with the cost model defined in Section 4.3.3).

We define the following parameters that measure the quality of a component index PARINET$_i$ in a T-PARINET.

Definition 5.1: The *global cumulated degradation* of a component index PARINET$_i$ w.r.t. a query load Q_L is defined as: $GCD_{Q_L}^i = \frac{QI_{Q_L}^i - QI_{Q_L}^{i\text{-optimal}}}{QI_{Q_L}^{i\text{-optimal}}}$, where $QI_{Q_L}^{i\text{-optimal}}$ represents the quality indicator of the optimal configuration of PARINET$_i$ w.r.t. the cost model.

Definition 5.2: The *local cumulated degradation* of a component index PARINET$_i$ w.r.t. a query load Q_L is defined as: $LCD_{Q_L}^i = \frac{stdev(DA_{Q_L}^i) - stdev(DA_{Q_L}^{i\text{-optimal}})}{stdev(DA_{Q_L}^{i\text{-optimal}})}$, where

$stdev(DA_{Q_L}^i) = \sqrt{\frac{1}{k} \cdot \sum_{j=1}^{k} (DA_{Q_j}^i - \overline{DA}_{Q_L}^i)^2}$ and $\overline{DA}_{Q_L}^i = \frac{\sum_{j=1}^{k} DA_{Q_j}^i}{k}$.

Definition 5.3: The *load factor* of a component index PARINET$_i$ having m partitions is defined as: $LF^i = \frac{\sum_{j=1}^{m} N_j^i}{m \cdot fan^{\lceil \tilde{h} \rceil}}$, where fan is the fanout of the B$^+$-trees indexes and $\tilde{h} = \frac{\sum_{j=1}^{m} \lceil \log N_j^i \rceil}{m}$ (10) is the average height of the trees.

The defined parameters indicate the current degradation of a component index (e.g., the current index or a past index) or the expected index degradation. GCD measures the percentage of query performance loss of a component index compared with the optimal index configuration w.r.t. the cost model. This parameter offers a global view of the performance loss, since it considers the aggregated cost over the query load. LCD is a parameter intended to measure the unbalance in the query cost across the indexed data space. Recall that PARINET partitions the data into several clusters that are balanced w.r.t. the amount of data in each cluster. This will also help balancing the cost of queries. Due to the difference between the expected and the real distribution of the data, the partition weights can be unbalanced. LCD indicates a local degradation, since it measures the unbalance in the query cost.

The first two parameters measure a degradation that is already presented in the index. LF indicates an imminent degradation of the current index. A drop in the query performance is also caused by an increase in the tree height of the B$^+$-trees since the data continuously accumulate. When LF approaches 1, the average height of the B$^+$-trees is expected to augment to include the new entries. This will lead to an increase in the query cost. However, the overhead can be avoided by an earlier "closing" of the current index and triggering the creation of a new current index.

Hence, by monitoring these parameters, the construction of a new component index in a T-PARINET can be automatically triggered whenever they exceed certain predefined threshold values. Nevertheless, two conditions regarding the lifespan of the current index need to be verified before creating a new index.

Lemma 5.4: Assuming the maximum time interval of the queries $|Q_t^{\max}|$, the lifespan of a component index should verify the following inequality (11) in order to have sargable temporal predicates in the queries.

$|T_i| > |Q_t^{\max}| \cdot \frac{BS_i + BS_d}{BS_i}$ (11).

Proof. Each partition in a PARINET index uses a B$^+$-tree to index the temporal dimension of the data. The query optimizer will make use of these indexes only if the estimated number of disk accesses for an index based search is lower than the number of disk accesses in a full partition data scan. Therefore, building a B$^+$-tree index in each partition is useful only if:

$\lceil \log N_p^i \rceil + N_p^i (\frac{1}{BS_i} + \frac{1}{BS_d}) \frac{|Q_t|}{|T_i|} < \frac{N_p^i}{BS_d}$ (cf. with Formula (8)). Since $\lceil \log N_p^i \rceil << \frac{N_p^i}{BS_d}$, the above inequality leads to Formula (11). □

Lemma 5.5: Assuming the maximum time interval of the queries $|Q_t^{\max}|$, the creation of a new component index indicated by the *load factor* of the current index is beneficial only if the lifespan of the current index verifies the following inequality: $|T_i| > |Q_t^{\max}| \cdot \tilde{h}$ (12), where $\tilde{h}$ is computed by (10).

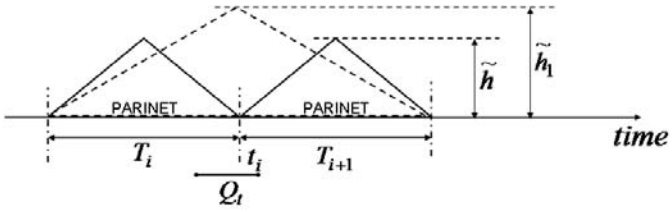

Figure 8: Example of T-PARINET index structure

Proof. Let us consider the component index PARINET_i in Figure 8, which has the lifespan T_i. At time t_i, LF^i is close to 1 indicating thus an imminent increase in the average index height. Therefore, a new component index is built for the data arriving after t_i. To keep the formulas tractable, we consider that PARINET_{i+1} has the same configuration as PARINET_i. Creating a new index will keep the cost of the queries situated in the interval T_i from augmenting, except for the queries having Q_t overlapped with t_i. For these queries the cost will increase since we need to visit two PARINET indices to evaluate them. Globally, this is not too penalizing if the percentage of queries in T_i that intersect t_i is low. This percentage can be estimated as $\frac{|Q_t|}{|T_i|}$, conform with the temporal uniformity assumption considered earlier. Hence, the average cost of a query in T_i is equal to:

$DA_Q = \sum_{p \cap Q_s} \left[(1+\frac{|Q_t|}{|T_i|})\tilde{h} + \sigma_p^{Q_t} \right]$ (13). This is based on Formula (9) in which the cost of traversing the index is increased with the probability that a query overlaps two indexes.

Another option would be to continue indexing the data arriving after T_i with the same index regardless of the increase of the average index height. In this case the average cost of a query is $DA_Q^1 = \sum_{p \cap Q_s} (\tilde{h}_1 + \sigma_p^{Q_t})$ (14), where $\tilde{h}_1 = \tilde{h} + 1$. From Formula (13) and (14), we conclude that is beneficial to create a new index at t_i only if $\frac{|Q_t|}{|T_i|}\tilde{h} < 1$. □

Corollary 5.6: The lifespan T_i of a component index in T-PARINET has a minimum value of: $\max\left\{ \left|Q_t^{\max}\right| \cdot \frac{BS_i + BS_d}{BS_i}, \left|Q_t^{\max}\right| \cdot \tilde{h} \right\}$.

Proof. This is directly deduced from Lemma 5.4-5.5. □

Proposition 5.7: The lifespan T_i of a component index in T-PARINET has a minimum value of $2 \cdot \left|Q_t^{\max}\right|$.

Since the format of the trajectory units is known (see Section 3.2), one can easily estimate the ratio $\frac{BS_i + BS_d}{BS_i}$ being approximately 1.6. On the other hand, due to the large amount of data, which characterizes trajectory datasets, a minimum average height $\tilde{h}$ of 2 appears to be reasonable for a PARINET index (e.g., a PARINET with 100 partitions and $\tilde{h} = 1$ can index only up to $3 \cdot 10^4$ trajectory units, whereas at $\tilde{h} = 2$ it can go up to $9 \cdot 10^6$). Therefore, the maximum value defined by Corollary 5.6 representing the minimum lifespan is expected to be $2 \cdot \left|Q_t^{\max}\right|$. Note that $\left|Q_t^{\max}\right|$ refers to the maximum size of the time interval of most of the queries. This does not mean that the queries that have a time interval larger than $\left|Q_t^{\max}\right|$ will

not be processed by the database. Moreover, a minimum index lifespan above $2 \cdot \left|Q_t^{\max}\right|$ can be considered to avoid component indexes with short lifespan when $\left|Q_t^{\max}\right|$ has small values.

5.3.3 Temporal Partitioning Algorithm

In the previous section, we extended the cost model of PARINET in the context of T-PARINET, i.e., for continuously indexing trajectory datasets. Proposition 5.7 gives the minimum lifespan of a component index in T-PARINET based on the maximum expected query time interval. In addition, we defined some quality factors, i.e., GCD, LCD and LF, to measure the performance of a component index. By simply monitoring the evolution in time of these parameters in the current index of a T-PARINET, an on-line tuning process can automatically decide when is an appropriate moment to trigger the construction of a new current index. Note that these parameters only take into account the query cost since the updates can be performed at approximately constant cost regardless of the index configuration as discussed in Section 5.2. Also, the instantiation of a new component index is considered to have a small cost and is neglected.

The on-line tuning process of T-PARINET is based on a simple algorithm (Algorithm 2). The process computes and continuously updates a few global statistics on the current index. Then, it verifies based on these statistics if the index quality indicators are situated within some predefined limits. A new component index is created when one of the parameters exceeds a threshold value. The configuration of the new index is determined based on an expected data distribution and density. Note that the creation of a new component index can be triggered only if the lifespan of the current index is greater than a predefined value (cf. with Proposition 5.7). Afterwards, the tuning process continues monitoring the new index in T-PARINET.

Algorithm 2: T-PARINET On-line Tuning

Input: Road Partitioning function RP_i of the current index PARINET_i, global statistics $Stat_i$ of PARINET_i, query load $Q_L = \{Q_1, Q_2, \ldots Q_k\}$ and $\left|Q_t^{\max}\right|$, thresholds GCD^{th} and LCD^{th}

1. current index $ci = i$
2. **while** *true*
3. update $Stat_{ci}$
4. **if** $t_c - t_{ci-1} > 2 \cdot \left|Q_t^{\max}\right|$ **then**
   if $Stat_{ci}.GCD_{Q_L}^{ci} > GCD^{th}$
5. **or** $Stat_{ci}.LCD_{Q_L}^{ci} > LCD^{th}$
   or $Stat_{ci}.LF^{i} > 0.9$ **then**
6. Create new index PARINET_{i+1}
7. Compute $Stat_{i+1}$
8. current index $ci = i + 1$

Given a component index PARINET_i, $Stat_i$ includes the following information on the index structure: the current number of trajectory units for each road (needed to compute $L(G)$ in Algorithm 1) and the total current number of trajectory units in each partition of PARINET_i (used to estimate DA_Q and $\tilde{h}$). This is sufficient to compute GCD^i, LCD^i and LF^i.

Note that the tuning process itself can be improved in certain cases. For instance, if good prediction models for the traffic are available, an important change in the distribution and the density of the data flow can be foreseen and the construction of a new current index can be triggered in advance, i.e., without having to wait for a degradation of the current index.

6. Experimental Evaluation

In this section, we experimentally evaluate both PARINET and T-PARINET. PARINET is devised to index (static) trajectory data sets. T-PARINET extends PARINET in order to optimize the handling of trajectory data flows. The two access methods have several points in common, since T-PARINET is a sequence of PARINET indexes. For example, the query processing is quite similar for the two methods and thus, the query performance is expected to be the same. Nevertheless, there are some particularities of one index compared to the other index. For example, the update performance and the index throughput are characteristics that are important mainly in the context of T-PARINET. Since the experimental evaluation presented in this section covers the two indexes in a uniform way and to avoid any confusion, we summarize in the following the points that apply to each method. Thus, the comparison of the query performances in Section 6.2.1 is valid for both PARINET and T-PARINET. The update performances and the index throughput measures in Section 6.2.2 and 6.2.3 are applicable to T-PARINET. The in-depth query performance evaluation (Section 6.3.1) and the cost model evaluation (Section 6.3.2) are valid for both methods. The index robustness with the variations of the query size (first part of the Section 6.3.3) is important in the context of PARINET. The index robustness with the variations of both the query size and the data distribution and density (second part of the Section 6.3.3), is a feature of T-PARINET. Finally, Section 6.4 compares PARINET and T-PARINET to underline the benefits of T-PARINET in a dynamic environment.

We compare our approach with the reference access methods for in-network trajectories, i.e., PJ-tree [14], MON-tree [1] and FNR-tree [5]. As PARINET, PJ-tree, MON-tree and FNR-tree were devised to index trajectory data sets. Hence, we use these methods directly for comparison with PARINET for historical data. Since there is no other work, to our knowledge, which deals with continuous indexing of in-network trajectory data flows, we also use PJ-tree, MON-tree and FNR-tree as reference in a dynamic environment for comparison with T-PARINET.

All the experiments were conducted under an off-the-shelf DBMS implementation. We used Oracle 11g Enterprise Edition installed on a Pentium 4, 3.2 GHz machine with 2.5 GB memory (note: the tests do not need so much memory) running Windows XP. Implementing PARINET under Oracle is straightforward using the available table partitioning mechanism. A given dataset is stored in a relational table where each tuple represents a unit having the following attributes: $(moid, rid, pos_1, pos_2, t_1, t_2)$. The table is partitioned based on the rid value. Each partition contains a list of rid values. Oracle allows creating an index for each partition. Table T that keeps the mapping between the road identifiers and the partitions is implicit because the DBMS internally manages the partitions based on the table metadata.

The evaluation is divided in two parts. First, we compare our method against the reference access methods for in-network trajectories in Section 6.2. We test the query performance, the update performance and the throughput of the three methods. Second, we lead a more elaborated set of experiments on T-PARINET in Section 6.3. The second part aims to validate the proposed cost model and to assess the index robustness with the variation of the query size and the (spatio-temporal) variation of the data distribution and density.

6.1 Datasets and Queries

Available real trajectories data, such as INFATI (www.cs.aau.dk/TimeCenter), are not representative enough in terms of trajectory variety and data size. Moreover, the underlying road network required by the experimentation is rarely available free of charge. Therefore, we used the generator for moving objects in networks proposed in [3] to create synthetic datasets. The generator is available with several network examples and we used two of them: the road networks for the city of Oldenburg (Germany) and the city of Stockton (San Joaquin County, CA). The networks are represented in a segment oriented model, i.e., each line segment represents a road and has a unique identifier. This is the smallest granularity for a network representation. We used the networks directly in this format for T-PARINET tests and transformed Oldenburg for the comparison between T-PARINET and the R-tree based access methods. Oldenburg has 7035 segments and 6105 nodes, while Stockton has 24123 segments and 18496 nodes. Stockton contains more than three times the number of roads of Oldenburg. With regard to the distribution of the generated MOs, we set the generator for a region-based distribution. In this approach, the network regions with higher node density have a higher probability of containing a starting point for a MO. The position of each MO is reported each time it passes a node. We generated 10 classes of MOs, each class corresponding to a maximum speed. The generator also simulates weather conditions or similar events that impact the motion and speed of the MOs.

Table 2: Tested Datasets Statistics for PJ-tree, FNR-tree, MON-tree and T-PARINET

Dataset name	**# of units**	**# of MO**	**# of time units**	**# of MO created per time unit**
Old 1	124079	3929	400	10
Old 2	273543	8890	600	15
Old 3	510761	15823	800	20

Table 3: Tested Datasets Statistics for T-PARINET

Dataset name	**# of units**	**# of MO**	**# of time units**	**# of MO created per time unit**
Oldenburg 1	685515	15964	800	20
Oldenburg 2	3489751	79785	800	100
Stockton 1	690890	8285	830	10
Stockton 2	3448008	41475	830	50

We have two collections of datasets. One collection is composed of small datasets that we use to compare T-PARINET against the R-tree based indexes. The other one consists of larger datasets and is used for a deeper analysis of T-PARINET. The reason for having two different datasets is that the R-tree based indexes do not scale well for large datasets. Table 2 presents the statistics for the first dataset collection. They are all based on a transformed map of Oldenburg (see Section 6.2). Table 3 gives the statistics for the second dataset collection based on the original Oldenburg and Stockton maps.

The datasets have different number of units, trajectories, trajectory length or time span, and map size. In average, a MO trajectory in Stockton has twice the number of units of a trajectory in Oldenburg, because of the network size. Hence, for the same number of units, we have twice as many MOs in Oldenburg than in Stockton. This will allow testing the index behavior according to the map and the dataset sizes in terms of total number of units, of MO and of trajectory length.

We tested the two types of queries: 2D and path. For each type of query and for each map, we generated three scripts, each script containing queries of fixed size. The statistics for the generated query sets are given in Table 4 and 5. For the 2D queries, we first randomly generated a 2D square window and a time interval. The intervals have the same relative size in all the dimensions. Then, we transformed the query as presented in Section 3.3 and generated the final script. For the path queries, we randomly selected some trajectories from the dataset and used them to generate the spatial interval of the queries. We generated queries where the size of the spatial interval is 0.25%, 0.5% and 1% respectively of the total number of roads in the network. Each road section in a query contains the entire road segment. For each of the three spatial windows, the time interval was fixed to 2.5%, 5% and 10% of the total time of the dataset. We chose a smaller spatial window due to the large number of roads in a network. The temporal interval is randomly chosen within the temporal interval of the dataset.

Table 4: Tested 2D Query Statistics

Query set name	**Spatial interval in each dimension**	**Avg. # of intersected roads by the queries**	**Temporal interval**	**# of queries in the set**
Old 2D Q1	2.5%	9.5	2.5%	105
Old 2D Q2	5%	34	5%	64
Old 2D Q3	10%	60.2	10%	49
Sto 2D Q1	2.5%	23.4	2.5%	108
Sto 2D Q2	5%	96.3	5%	100
Sto 2D Q3	10%	156	10%	98

Table 5: Tested Path Query Statistics

Query set name	**Spatial interval**	**# of roads in the query**	**Temporal interval**	**# of queries in the set**
Old P Q1	0.25%	17	2.5%	84
Old P Q2	0.5%	35	5%	75
Old P Q3	1%	70	10%	63
Sto P Q1	0.25%	60	2.5%	71
Sto P Q2	0.5%	120	5%	66
Sto P Q3	1%	241	10%	34

For a given query set, dataset and index configuration, we measured the average time per query and the average number of disk accesses per query. Similarly, given a large batch of updates that need to be executed, we measured the average time and the average number of I/Os per one thousand processed updates. We used the default page size in Oracle, i.e., 8KB. The resulted fanout is 340 for the B^+-tree index. Oracle logically implements the R-tree as a tree and physically using tables inside the database. Hence, the fanout does not depend on the page size. We used the default value of the fanout, i.e., 35, for the R-tree under Oracle 11g, since the R-tree index is already tuned for an optimal performance. Oracle uses a LRU buffer cache. We set the size of the buffer cache to 32MB, which allows for good performances of the tested indexes (e.g., the minimum allowed size of the buffer cache under Oracle 11g is 8MB). In all the tests that measure the query performance, we emptied the cache between each query run to limit the influence of the cache on query processing evaluations. In the tests that measure the update performance (Section 6.2.2), we clear the buffer cache and commit after each 32 thousand processed updates. There is no intervention on the cache memory for the tests that measure the index throughput (Section 6.2.3).

6.2 PARINET vs. PJ-tree vs. MON-tree vs. FNR-tree

In this section, we compare PARINET with PJ-tree, MON-tree and FNR-tree. Implementing the reference access methods under Oracle is also straightforward, since Oracle offers an R-tree index within the Oracle Spatial data cartridge. PJ-tree uses a single 2D R-tree to index the data which makes it easy to implement under an off-the-shelf DBMS supplied with an R-tree. MON-tree and FNR-tree follow the same technique as for PARINET with regard to data partitioning. Note also that in the case of MON-tree there is one difference from the method presented in [1]. The MON-tree in [1] uses a modified version of an R-tree that is capable of handling multiple query windows in one index scan. Oracle implements an R-tree index, but, to the best of our knowledge, does not allow changes to the built-in indexes. Besides, all the reference methods use a (top) 2D R-tree to perform the mapping of the 2D queries (cf. Section 3.3). In our tests, the mapping of the 2D queries is performed apart and not considered in the index performances.

Moreover, as reported in [1], the MON-tree performs better on route-oriented network models. We concatenated the segments of the Oldenburg map and generated longer routes. From the 3328 segments that constitute the core of the Oldenburg network, we generated 186 routes, i.e., an average of 17.892 segments per route. Then, three datasets (see Table 2) were generated on this transformed map. Note, that the data generation was done as previously explained, i.e., a MO reports its position each time it traverses a segment node. This means that several units are generated for a MO that traverses a route. This aspect is important for the MON-tree and the PJ-tree, because the selectivity on the relative positions on a route is significant for this network model.

As the longer routes can be seen as segment clusters, we consider them as separate partitions for PARINET. This kind of partitioning also helps validate our first observation (see Section 3.4). A proper partitioning based on the software METIS is used in the next section.

FNR-tree [5] can only be employed with a segment oriented network model. The 2D R-tree indexing the 3328 segments has 104 leaf nodes. Based on this clustering of the network segments, we partitioned the datasets into 104 partitions and index each partition with a 1D R-tree on the unit time intervals. The R-tree index implemented in Oracle can index spatial data having the dimensionality in the interval from two to four. To simulate a 1D R-tree, we create a 2D R-tree over a data space that has the extent of the second dimension equal to zero.

We measure first the query performance of the three tested methods. In the context of T-PARINET, the update performance becomes equally or even more important than the query performance. Therefore, we also measure the update performance and the throughput (i.e., including updates and queries) in a concurrent environment.

6.2.1 Query Performance

Figure 9 presents the comparative results for the three methods for Old 1, 2 and 3 datasets (see Table 2) and the two types of queries. PJ-tree outperforms MON-tree for most of the tests, except for the execution time of the largest 2D queries and the largest dataset. Using several indexes as in MON-tree instead of one as in PJ-tree leads to an increased overhead which can be significant for small datasets. We observe that MON-tree shows better performance than PJ-tree for 2D queries on large datasets. The performance of the FNR-tree is dual. On the one hand, FNR-tree requires the highest number of disk accesses to answer a query among the tested methods. On the other hand, the average execution time per query is lower in general than the other R-tree based indexes. We think that this is due, similar to PARINET, to the sequential type of disk accesses as only the time interval is indexed.

Nevertheless, this advantage is outbalanced by the significant increase of the number of disk accesses for large queries and large datasets.

The performance of PARINET is always better than the R-tree based indexes from both execution time and disk access point of views. The difference increases with the query size. The average time per query, which is sometimes one order of magnitude smaller for PARINET came as a surprise. This is certainly due to the sequential type of disk accesses in PARINET as the data is clustered and sorted.

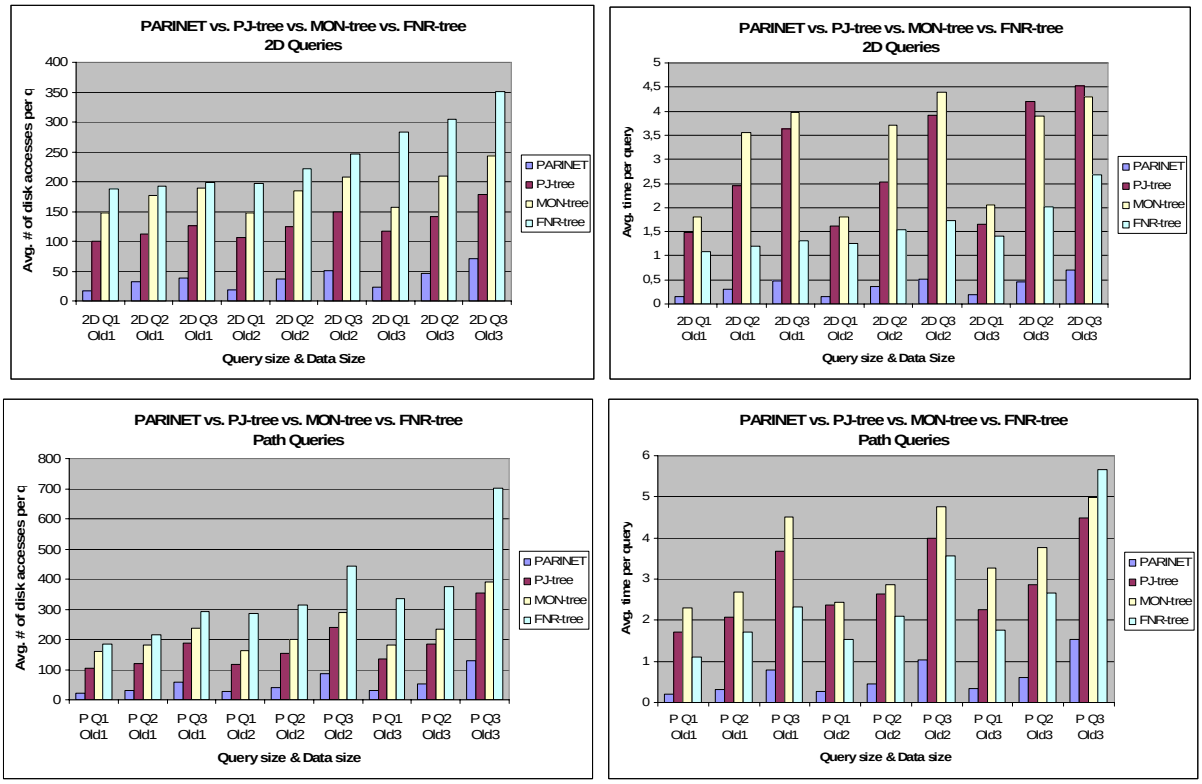

Figure 9: Query performance comparison between (T-) PARINET, PJ-tree, MON-tree and FNR-tree

6.2.2 Update Performance

The update performances of the three methods are presented in Figure 10. We proceeded by building first the index structures without any data. Then, we inserted all the trajectory data units available in Old 1, 2 and 3 datasets and measured the total number of disk accesses and the execution time. The values in Figure 10 are averaged over one thousand updates. Moreover, we also tested the method proposed in Section 5.2, i.e., to optimize the update operation for PARINET, and employed it equally to optimize the update cost for the R-tree based methods.

As expected, PARINET offers excellent update performance, since it is based on the B^+-tree index (see Section 5.2). PJ-tree, MON-tree and FNR-tree are using the R-tree index that is less efficient for update operations. Furthermore, the update cost increases with the number of updates due to the degradation of the index structure in time. This aspect can be observed on PJ-tree which offers better update cost that MON-tree and FNR-tree on Old 1 and Old 2 datasets, i.e., for less than 273 thousand updates. However, the performance degrades greatly for PJ-tree on Old 3 dataset, i.e., for 510 thousand updates, and falls behind of MON-tree's and of FNR-tree's update performances. MON-tree and FNR-tree use a forest of R-trees,

which can incur a higher update cost locally, but will offer a higher robustness on the long run since each local index is affected by only a fraction of the total number of updates.

Another observation is that the simple method that we proposed to optimize the update cost improves greatly the update performances of PARINET, MON-tree and FNR-tree (since these methods are based on partitioning). It also has a positive impact on the robustness of PJ-tree.

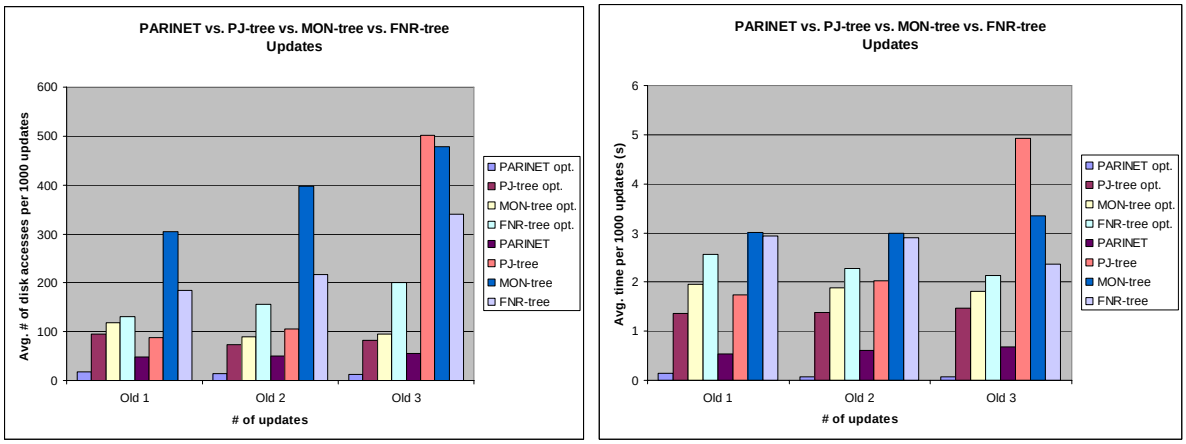

Figure 10: Update performance comparison between (T-) PARINET, PJ-tree, MON-tree and FNR-tree

6.2.3 Throughput

An access method for continuously indexing MOs' trajectories (e.g., the current index in T-PARINET) needs to process both queries and updates in a concurrent environment. The throughput represents the average number of operations executed in a unit time (e.g., each second). Several factors influence the throughput. An important element is the query-update ratio. Updates are normally less expensive than queries. However, in real applications, the number of updates is expected to be much larger than the number of queries. Also, whereas several queries can be processed simultaneously since they hold shared lock on the traversed nodes, the queries will block the updates operating in the same index parts and vice-versa.

The number of concurrent operations that are executed at a certain time is another important factor on the index throughput. Intuitively, the throughput will reduce with an increasing number of concurrent operations being executed. This is due to the fact that the same resources will be shared among more execution threads and each thread will wait longer to gain access to the index structure.

To evaluate the throughput, we implemented a multi-thread program in Java, which connects to the database and executes tasks from two task pools. One task poll contains updates and the other contains queries. We varied the update-query ratio from 1:1 to 10000:1, and the number of execution threads from 2 to 64. Among the working threads, there is one dedicated thread that executes tasks only from the update pool. The rest of the threads execute the queries. The throughput is measured from a point in time where each index contains 510 thousand trajectory units (i.e., the data from Old 3). We use 2D and path queries (see Table 3 and 4) of different sizes having the time interval randomly chosen from the index creation time up to the current time.

The results are presented in Figure 11. The curves in the first graphic represent the throughput of the tested methods when varying the update-query ratio with the number of threads fixed to 8. The throughput increases with the update-query ratio, since the update operations are less expensive than the queries. Note that the throughput axis is logarithmic in

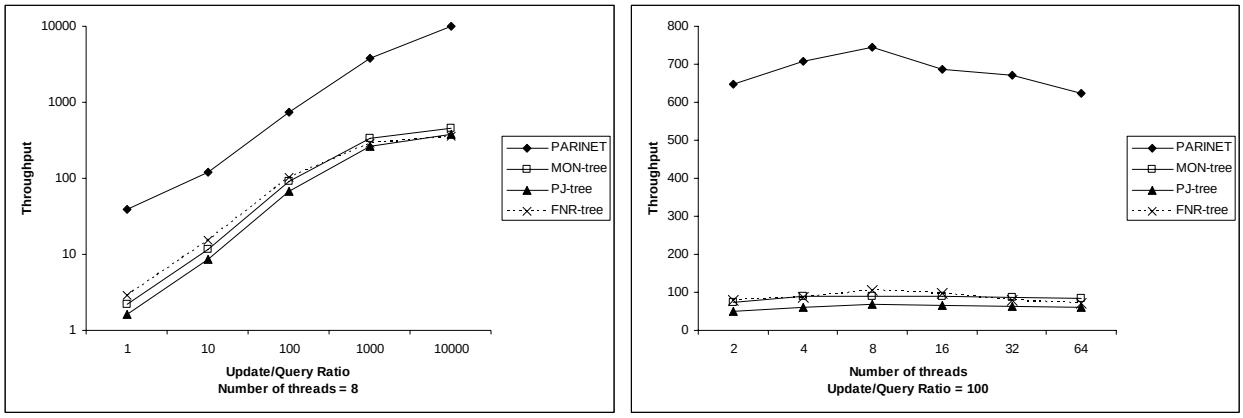

Figure 11: Throughput performance comparison between (T-) PARINET, PJ-tree, MON-tree and FNR-tree

this graphic. FNR-tree, MON-tree and PJ-tree have similar throughput values. MON-tree shows a marginal advantage over PJ-tree. FNR-tree has a slightly better throughput than MON-tree for update-query ratios smaller than 100. As the number of updates increases, MON-tree offers the highest throughput among the R-tree based methods. As expected from the results presented in the previous two subsections, PARINET has the largest throughput. The gain over the reference access methods is significant, being approximately one order of magnitude greater in favor of PARINET.

The curves in the second graphic indicate the variation of the throughput with the number of threads when the update-query ratio is fixed to 100:1. All the tested indexes show a maximum throughput for a number of 8 threads. The throughput increases with the number of threads, i.e., from 2 to 8 threads, and then decreases as the number of threads increases.

In this first part of the experimentation we compared PARINET with the reference R-tree based access methods for in-network trajectories. The experimental evaluation showed that PARINET performs significantly better regarding both the query processing and the update processing or their combination in a concurrent environment. It also showed good robustness under intense updating. Another important advantage of PARINET is that it proposes a cost model that, if sufficiently accurate, will allow tuning the index structure for better performance for both historical data or in a dynamic environment. This is studied in the following section.

6.3 PARINET in Depth

In this section we report the results of a more elaborated set of experiments on PARINET and T-PARINET. The main objectives are to evaluate the accuracy of the proposed cost model and to test the index robustness with the variation of the query size and the variation of the data distribution and density. We used the largest datasets (see Table 3) for the tests. We use METIS [11] for computing the partitions and generate the indexes for 100, 200, 500, 1000, 1500 and 3000 partitions. We analyze the index performance with respect to the map, the dataset and the query sizes. Then, we evaluate the proposed cost model. Finally, we discuss the index robustness with the query size and data size variations.

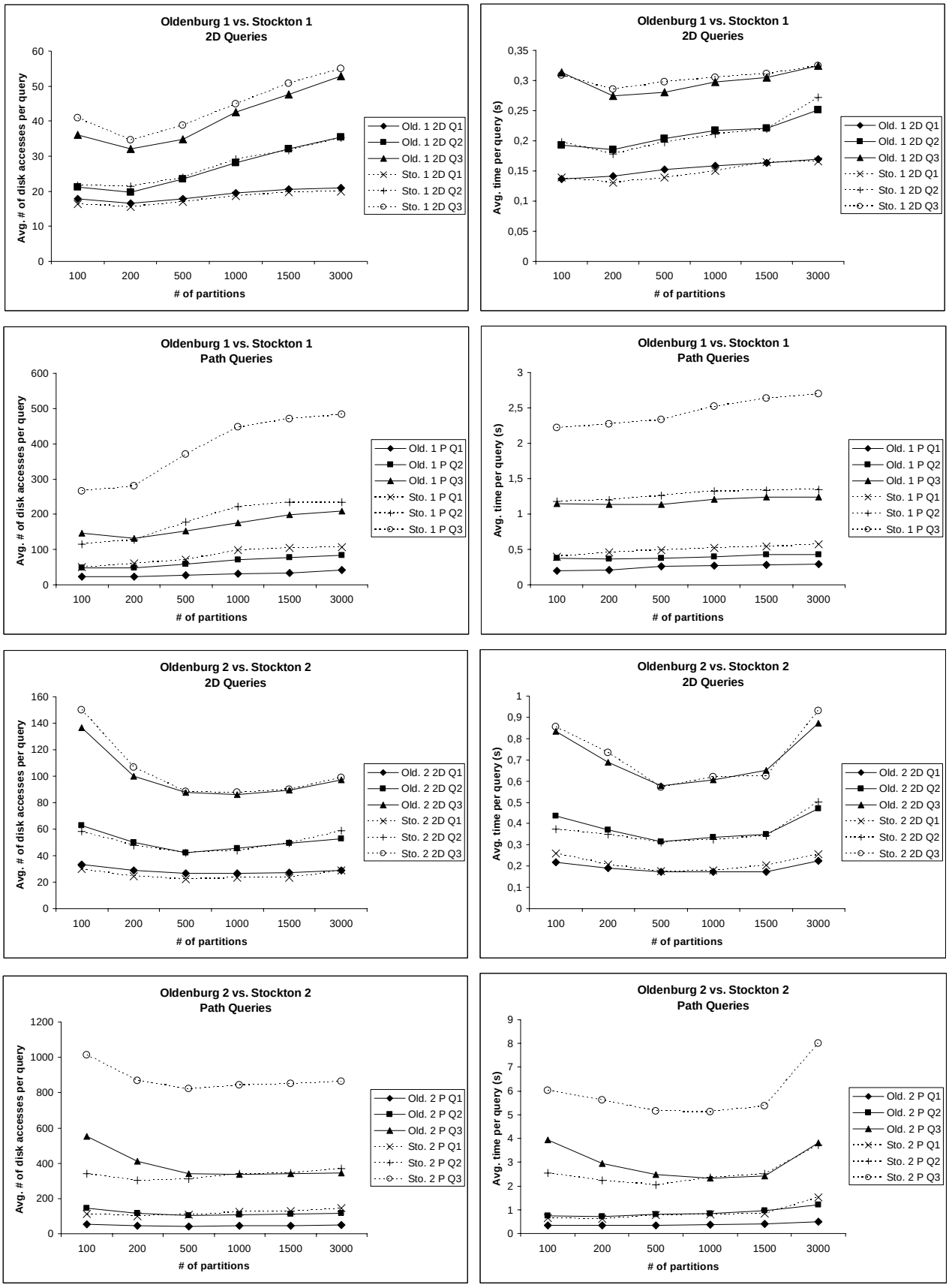

Figure 12: (T-) PARINET query performance

6.3.1 PARINET Query Performance

Figure 12 presents the results obtained for 2D and path queries over Oldenburg 1 and 2 and Stockton 1 and 2 datasets. The tests confirm our second observation (Section 3.4), i.e., the index performance depends on the dataset size, the query size and the query type. A smaller number of partitions are necessary for an optimal performance when dealing with smaller datasets. For our datasets, 100-200 partitions offer the best performances for the smaller datasets, and 500-1000 partitions are needed for the largest.

The index performance also depends on the query size and the query type. For the 2D queries, when the query size is large, we need to reduce the number of partitions in order to maintain an optimal index performance. For example, the optimal performance is obtained for 1000 partitions for the small 2D queries over the Oldenburg 2 and Stockton 2 datasets and for 500 partitions for the largest query sets over the same datasets. For the path queries the opposite holds true.

The query type influences the index performance. Path queries are more demanding than 2D queries, which is foreseeable considering the fact that they extend over more network regions for a given partitioning scheme and a spatial query size. The performance for the 2D queries depends on the degree of data density over the network. We can see that for the same amount of data (Oldenburg 2 and Stockton 2) and similar queries, the indexes yields similar results. As expected, the execution time depends on disk accesses. However, the execution time is more sensitive to the number of partitions. For higher partition numbers, the variations for the disk accesses are minor, but they can be significant for the execution time. For instance, in Figure 12, we observe small variations in the disk access values for 500 partitions or more (left part) whereas the execution time becomes worse (right part). Therefore, since the cost model only estimates the disk accesses, a good technique will be to choose a break point where increasing the number of partitions becomes useless in term of disk access.

Overall, we find that PARINET has a solid performance record when correctly tuned and scales much better with the data and query sizes than the R-tree based methods. The query performance remains good both for disk accesses and execution times, for queries containing from 17 up to 150 road identifiers. Also, we did not observe any influence of the number of MOs or the trajectory lengths on the query performances. Only the query size and the number of units in the dataset affect the performances.

6.3.2 PARINET Cost Model Evaluation

The above tests on the PARINET performances confirm our observations, i.e., given a dataset, only some configurations of the index offer near-optimal performance for a query load. The tuning of the number of partitions has a relatively small impact on disk accesses needed to process a query, for relatively small datasets and small query sizes. This aspect was observed for Oldenburg 1 and Stockton 1 datasets where the data density is smaller (i.e., 700 thousand units in approximately 800 time units). Nevertheless, choosing a near optimal number of partitions can significantly improve the query execution time for large queries, even for small datasets. On the other hand, the number of partitions is important for the query performance for large datasets. This aspect was observed for Oldenburg 2 and Stockton 2 datasets where the data density is larger (i.e., 3.5 million units in approximately 800 time units).

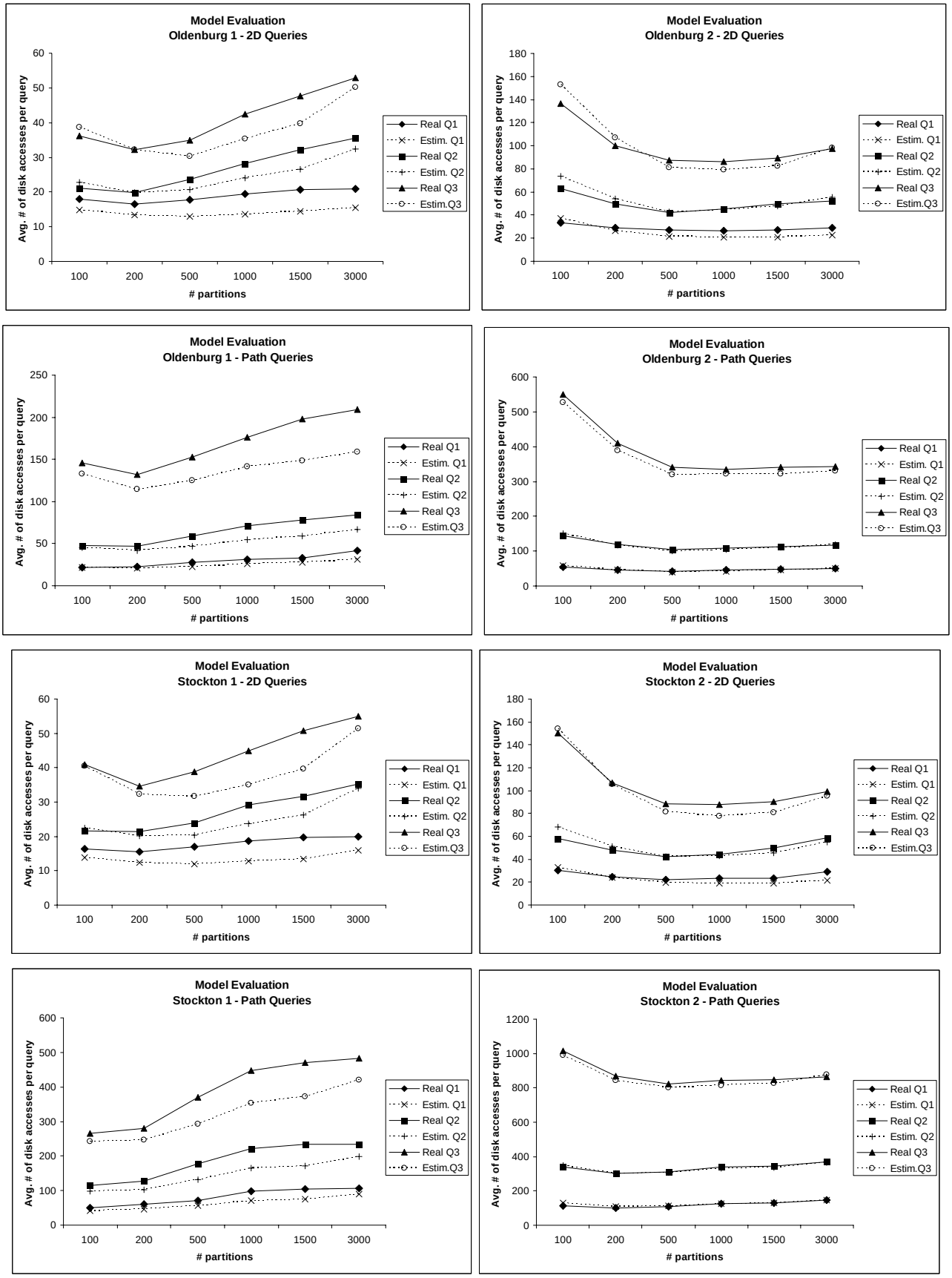

Figure 13: (T-) PARINET cost model evaluation

In Section 4.3.2, we presented a cost model that estimates the number of disk accesses for a query load and a given index configuration. The tests show that the execution time of a query usually depends on the number of disk accesses. Therefore, the cost model can estimate the performance of our access method for a given configuration, without effectively constructing the index. This is very important considering that the index creation is costly, i.e., it is not an option to actually test all the possible configurations in order to choose the best one. If the cost model is accurate, we can automatically find and materialize a good configuration among all the possible ones. Moreover, the cost model can be further employed in the context of continuously indexing MOs trajectories in order to optimize the index evolution in time (see Section 5.3.2). In this case, its role is to permit monitoring the efficiency of the current index and to compare it with a near-optimal index configuration.

In this section, we experimentally evaluate the proposed cost model. For all the tests presented in Section 6.3.1, we also calculated the number of disk accesses by using Formula (6) defined in Section 4.3.2. Practically, we implemented a program that takes as input the network graph with the data distribution for each road, a query load and a number of partitions, and outputs an estimated number of disk accesses. We consider a uniform temporal distribution of the data, which is a good approximation for the generated datasets. For non uniform data distribution, the real temporal distribution must be used, as in (2), in order to obtain a good estimation of disk accesses.

The results are presented in Figure 13. As we can see from the figure, the cost model offers a good estimation for the number of disk accesses.

Another observation is that the cost model is more accurate for the larger datasets and also offers good estimations for the smaller datasets when the number of partitions is smaller or equal to 200. The reason the cost model is less accurate for small datasets and large number of partitions lays in the way the data partitioning is implemented under the Oracle Server. Each partition of a partitioned table has allocated a minimum of eight data pages regardless of the amount of data in that partition. For small datasets distributed over a large number of partitions, the partitions become under-occupied, i.e., they contain less than eight pages of data. This will lead to an increase in the number of disk accesses to answer a query. The cost model can be easily adapted to take into account this specific case. Nevertheless, for the sake of generality, we used the cost model as initially proposed in Section 4.3.2.

In conclusion, the experiments show that the cost model is good enough to be used for tuning the PARINET index.

6.3.3 Robustness of PARINET and T-PARINET

The access method proposed in this paper is devised to be employed for two main (complementary) usages.

Firstly, PARINET can be used to index historical datasets. In this case, the data are known in advance. Therefore, PARINET can be automatically tuned for near-optimal performance given a dataset and an expected query load. However, even if the data is historical, it does not mean that the queries are known in advance. Moreover, the queries at a given time may differ a lot (different users may pose totally different queries), and the queries may change across time (during the day or week or month). In such cases, the index itself should be able to handle very different queries at the same time and should be able to adapt to changes over time. We are interested in a robust index structures whose performance that does not degrade much with reasonable variations between the expected and actual query sizes.

Secondly, the extended version of PARINET, i.e., T-PARINET, is intended to efficiently and continuously index trajectory data flows. In this case, the index structure is built in

advance based on anticipated values for the data distribution and data density and the query size. Hence, the index should feature good robustness with the combined variation of both the data size and the query size. In this section we analyze the index robustness for the two case scenarios.

- *Query Robustness*: Given a dataset of in-network trajectories, PARINET can be tuned to offer near-optimal performance with regard to a query load, e.g., containing queries of a certain size and a certain spatio-temporal distribution. However, as indicated above, the actual user queries can be different from the expected queries and can also cover a wide range regarding the query size.

Table 6: PARINET Query Robustness

Query set name	Avg. # of intersected roads by the queries	Temporal interval	*GCD* (disk accesses)	*GCD* (exec. time)
Old 2D Q1	9.5	2.5%	0	0
Old 2D Q2	34	5%	0.055	0.062
Old 2D Q3	60.2	10%	0.018	0.048
Old P Q1	17	2.5%	0	0
Old P Q2	35	5%	0.045	0.111
Old P Q3	70	10%	0.016	0.027
Sto 2D Q1	23.4	2.5%	0	0
Sto 2D Q2	96.3	5%	0.038	0.041
Sto 2D Q3	156	10%	0.032	0.087
Sto P Q1	60	2.5%	0	0
Sto P Q2	120	5%	0.026	0.084
Sto P Q3	241	10%	0.051	0.093

To measure the index robustness with regard to the query size, we use the tests in Section 6.3.1. We consider that the index is tuned for the query load Q1. We observe that the performance degrades when the query load changes to Q2 and Q3, since different index configurations are near-optimal for these query loads. For example, for a given dataset 1500 partitions offer a near-optimal performance for Q1, while 1000 for Q2 and 500 for Q3 are near-optimal.

We measure global cumulated degradation (*GCD*) in query performance as: $GCD_{Q_L} = \frac{QI_{Q_L} - QI_{Q_L}^{optimal}}{QI_{Q_L}^{optimal}}$ (see Definition 5.1 in Section 5.3.2). The results for Oldenburg 2 and Stockton 2 (largest datasets) are given in Table 6. In most cases the performance degradation is more than acceptable (i.e., less than 10%), for a variation between the expected and real query size that is important (up to 6 times more roads in the spatial part of the query and 4 times larger the extent of the time interval). The results indicate good robustness of PARINET to query variations.

- *Robustness with the variation of the data distribution and density and the query size*: T-PARINET creates periodically a new component index to manage the trajectory data that will be gathered from the moment of construction to a future point in time. At building time there is no data. Hence, T-PARINET creates a new index which is optimized for a predicted data distribution and density and an expected query load. Since the predictions might not be accurate, we also need to study the index robustness in this case scenario. Testing the robustness of T-PARINET has two objectives. First, it gives an idea about actual index degradation values (given as *GCD* and *LCD* values) under different case scenarios. Second, it allows comparing the real *GCD*/*LCD* values with their estimated values. It is important to

have fairly accurate estimations, since the tuning process (Algorithm 2) is based on estimated *GCD*/*LCD* values.

To measure the index robustness with the variation of the data density and data distribution we proceeded as follows. We created a new PARINET component index that is optimized for a given data density (e.g., 700 thousand trajectory units in 800 time units) representing the expected data. Also, we considered that the distribution of the data is uniform over the road network and use the query load Q2 as reference. Then, we used as real data a dataset that has different characteristics from the expected values (e.g., the data density is 3.5 million trajectory units in 800 time units and the data distribution is non-uniform, i.e., the data density is higher in the central part of the network than in the peripheral parts). Finally, we measured the query performance for different query loads, including the reference query load, and compared it with the near-optimal values.

Specifically, we used the Oldenburg network and created an index optimized for a data density of 700 thousand trajectory units in 800 time units, i.e., the smaller dataset, but for a uniform distribution. Also, the index is optimized for the query load 2D Q2. Then, we used as real data the larger dataset which has a data density five times greater than expected and a non-uniform data distribution. We measured the query performance for all the query loads (i.e., Q1, Q2 and Q3, 2D and path) and computed the index degradation w.r.t. the near-optimal index configuration. We did the same tests for the Stockton network. The sole difference was to consider the larger dataset as predicted data and to use the smaller dataset as real data.

Table 7: T-PARINET Robustness

Query set name	***GCD* (estimated)**	***GCD* (real)**	***LCD* (estimated)**	***LCD* (real)**
Old 2D Q1	0,463	0,501	0,035	0,135
Old 2D Q2	0,419	0,339	0,196	0,256
Old 2D Q3	0,473	0,415	0,058	0,089
Old P Q1	0,355	0,315	0,495	0,697
Old P Q2	0,356	0,333	0,529	0,654
Old P Q3	0,452	0,393	0,339	0,558
Sto 2D Q1	0,231	0,221	0,04	-0,024
Sto 2D Q2	0,343	0,194	0,55	0,538
Sto 2D Q3	0,296	0,157	-0,038	-0,064
Sto P Q1	0,355	0,279	-0,285	-0,308
Sto P Q2	0,185	0,129	-0,352	-0,37
Sto P Q3	0,037	0,074	-0,22	-0,174

For al the tests, we measured both the global (*GCD*) and the local index (*LCD*) degradation (see Section 5.3.2). We also estimated the values for *GCD* and *LCD* by using the cost model. The results are presented in Table 7. The first observation is that the *GCD* values do not exceed 0.5. This indicates that the average number of I/Os for a query will increase with less than 50% w.r.t. the near-optimal configuration when the real data density is different from the expected data density with a factor of 5 and when the real query size is different from the expected query size with a factor of 2 in each dimension. Furthermore, the *LCD* values, which indicate the relative increase in the unbalance of the query cost across the indexed data space, are lower than 0.7 (i.e., 70%). The second observation is that the accuracy of the cost model is again confirmed. The estimated values for both *GCD* and *LCD* are close to the real values. The negative values for the *LCD* indicate that in the initial (predicted) index configuration the query cost in a query load is more balanced that in the near-optimal

configuration. Note also that this is more apparent for the path queries (for 2D queries the values are very small). This is not unusual since the path queries are not uniformly distributed over the network space, as for the 2D queries, but are following the distribution of the trajectories (see Section 6.1).

6.4 PARINET vs. T-PARINET

To underline the benefits of T-PARINET over PARINET in a dynamic environment, we compare the two methods in this section. To this end, we employ the following scenario. We consider a trajectory data flow having a data density, i.e., the average number of trajectory units per time unit, which alternates between two values every 800 time units. Alternating the data density in the dataflow is intended to simulate the changes in the real traffic, which also periodically oscillates between high and low density values. In our scenario, these values are of approximately 700 thousand trajectory units in 800 time units and 3.5 million trajectory units in 800 time units. They correspond to Oldenburg 1 and Oldenburg 2 datasets (see Table 3). We refer in this section to the *time window of 800 time units* having one of the two possible values of the data flow density as a *time interval*. Then, for both indexes we need to create index structures in advance, based on the expected data density and the expected query size. Since the predictions are rarely accurate, we consider that the index performance is inferior to the near-optimal index performance. In our scenario, we randomly choose global index performances that are inferior to the near-optimal performance from 10% to 30%., i.e., the *GCD* of each generated (component) index is in the interval $[0.1;0.3]$.

In the case of PARINET a single index structure is generated, which will be tuned for the lower data density. Moreover, since the structure of PARINET is tuned to a configuration close to a near-optimal configuration for the lower data density, it means that the index structure of PARINET will not be adapted to the data flow density only half of the time. This makes the scenario fair for PARINET, since it reduces the global index degradation w.r.t. the near-optimal configuration.

In the case of T-PARINET, the index evolution in time is decided by the tuning process. The tuning process uses three user-defined parameters, i.e., $\left|Q_t^{\max}\right|$, GCD^{th} and LCD^{th}. When the lifespan of the current index exceeds $2 \cdot \left|Q_t^{\max}\right|$, the tuning process continuously monitors the GCD, LCD and LF values and compares them with the threshold values. If one of these three parameters exceeds the user-defined values, a new component index is created. For simplicity, we considered in our scenario that $2 \cdot \left|Q_t^{\max}\right|$ is equal to the time interval of the data flow. Consequently, the tuning process will start verifying if a new component index needs to be created every time a change in the data flow density occurs (line 4 in Algorithm 2). Moreover, we set the value for the GCD^{th} parameter to be zero, such as even a small degradation cumulated in the current index will trigger the creation of a new component index (line 5 in Algorithm 2). By "forcing" the creation of new component indexes, one can expect to have a T-PARINET that approaches a near-optimal configuration. Note that conform to the line 5 of the algorithm, a single parameter (in this case GCD) is sufficient to trigger the creation of a new component index, regardless of the values of the other two parameters (in this case LCD and LF).

Then, we compare the average query cost of the two index structures at different moments in time, i.e., after a certain number of elapsed time intervals. The results corresponding to a simulation run are presented in Figure 14. The query performance of PARINET and T-PARINET are the same for the first time interval since they use the same index configuration.

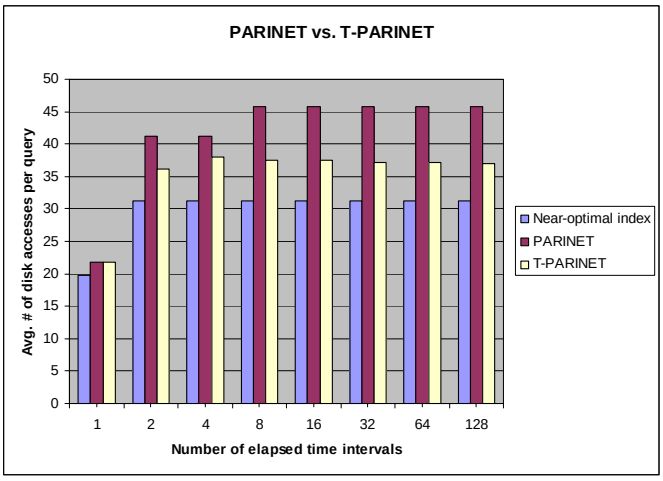

Figure 14: PARINET vs. T-PARINET

The near-optimal values represent the query performance of a T-PARINET index with each component index having a *GCD* equal to zero. The query cost is about 10% greater than the near-optimal cost after the first time interval. The average query cost increases after two time intervals due to an increase in the data flow density in the second time interval. However, the degradation for PARINET is more important than the degradation presented by T-PARINET. This is due to the fact that PARINET uses the same index structure to index the data flow with a different data density in the second time interval, whereas T-PARINET creates a new component index each time interval.

The average query cost of T-PARINET changes after each time interval since a new component index is added to index the data in the next time interval. The average query cost will decrease or increase in function of the quality of the new component index w.r.t. the near-optimal configuration. Clearly, this depends on the quality of the prediction of the spatio-temporal density in the data flow. In our case, the query cost will stabilize around 15% above the near-optimal cost, because the considered degradation factors are randomly generated in the interval $[0.1;0.3]$.

On the other hand, the query performance of PARINET continues to degrade after eight time intervals due to an increase in the index height since the data accumulate continuously. This type of degradation will continue to appear in the future if the same index is used to index the data flow. Although the increase in the index height is logarithmic with the data size it can not be neglected. Overall, after eight time intervals the degradation of the query performance of PARINET is around 46%, whereas T-PARINET presents a degradation of about 20%.

There are three observations that we can draw from this scenario. First, when using PARINET in a dynamic context, there are no guaranties regarding the index query performance degradation. Since the index configuration is fixed, no adaptation to the spatio-temporal distribution and density of the data is possible. T-PARINET corrects this shortcoming by periodically creating a new component index. The degradation of the query performance w.r.t. the near-optimal index configuration will only depend on the quality of the predictions of the spatio-temporal distribution and density of the data. Second, with T-PARINET we can also avoid a degradation of the query performance due to massive accumulation of the data. Third, even in the case of poor predictions of the spatio-temporal distribution and density of the data, the index structure of T-PARINET can be adjusted off-line since only some component indexes will be affected. Those past component indexes showing important degradation factors can be replaced with component indexes having near-

optimal index configurations since the exact data distribution will be known at the reconstruction time. This type of off-line maintenance can be hardly performed for PARINET, because PARINET has a single component index which is continuously modified by the updates in the indexed data flow. Moreover, the cost of the reconstruction will be much higher, since it will affect all the indexed data up to the reconstruction time.

6.5 Summary of the Experiments

We experimentally evaluated in Section 6 the proposed method for indexing in-network trajectory data. The first part of the evaluation (i.e., Section 6.2) focused on measuring the performances of T-PARINET relatively to the reference access methods, i.e., PJ-tree, MON-tree and FNR-tree. The experiments showed that our approach is more efficient for both the query processing (Section 6.2.1) and the update processing (Section 6.2.2) or for processing a mix of queries and updates in a concurrent environment (Section 6.2.3).

The second part of the evaluation consisted in a set of more elaborate experiments on T-PARINET. T-PARINET is supplied with a cost model that estimates the number of disk accesses for a query load given a certain index configuration. The experimental evaluation in Section 6.3.2 indicated that the cost model has a good accuracy in estimating the number of I/Os for a query load. Since the overall query performance is directly related to the number of I/Os (cf. Section 6.3.1), one can use the cost model to tune the index structure for a near-optimal performance given an expected query size and an expected data distribution and density. Nevertheless, in a day-to-day usage scenario, some of the actual user queries may be very different from the expected queries. In this case, the index should be able to handle very different queries at the same time. Moreover, the predictions on the data distribution and density might not be accurate. Hence, the index should be equally robust with regards to this factor. The experimental results presented in Section 6.3.3 showed a good robustness of T-PARINET with the variation of the above mentioned factors. Also, T-PARINET was designed to continuously index trajectory data flows. The benefits of the on-line tuning process over the static version of the index were demonstrated in Section 6.4.

7. Conclusions and Future Work

In this paper we propose a new access method called T-PARINET (Temporal PARtitionned Index for in-NEtwork Trajectories), for efficient retrieval of the trajectories of objects moving in networks. T-PARINET can be used as an access method for either a classic situation, i.e., for indexing historical data, or in a dynamic context, i.e., for continuously indexing trajectory data flows. The index structure is based on graph partitioning and on indexing the partitions with composite B^+-trees, which are ubiquitous in the database world. T-PARINET can be easily integrated into any DBMS, which is an essential feature particularly for industrial or commercial applications.

T-PARINET has two important advantages over the existing R-tree based approaches. The first advantage lies on the side of the performance. The experimental evaluation under an off-the-shelf DBMS shows that our approach significantly outperforms the reference R-tree based indexes. T-PARINET offers both superior query performance and update performance. It also operates well in a concurrent and dynamic environment. Furthermore, T-PARINET shows robust performance with the query size, the data size and under massive updating.

The second important advantage of T-PARINET is that it is supplied with a good quality cost model. The benefit is twofold. First, the cost model allows a better integration of the index into the query optimizer of any DBMS. Second, as discussed in the paper, it permits

tuning the index structure for better performance both for historical data and in a dynamic context by adapting the index structure to any data distribution and query size.

As future work, we intend to work on an extension of T-PARINET to handle queries about the current and near-future position of objects moving in networks. Also, optimizing continuous spatio-temporal queries for constrained moving objects appears to be a challenging and useful task. In addition, we plan to use a T-PARINET base access method for indexing several data types related to in-network MO trajectories. Indeed, we are currently extending an off-the-shelf RDBMS to support spatio-temporal data generated by objects with embedded sensors that move in networks. T-PARINET is intended to be a base access method for several data types defined in the system such as trajectories and mobile sensor measures.

Acknowledgment

This work was partially supported by grants from Région Ile-de-France and partially supported by a grant from DoD-ARL through the KIMCOE center of Excellence. We also thank the authors of [1] for giving their permission to use Figure 1.

References

[1] Almeida, V.T. de, Guting, R.: Indexing the Trajectories of Moving Objects in Networks. GeoInformatica 9(1): 30–60 (2005)
[2] Botea, V., Mallett, D., Nascimento, M.A., Sander, J.: PIST: An Efficient and Practical Indexing Technique for Historical Spatio-Temporal Point Data. GeoInformatica 12(2): 143-168 (2008)
[3] Brinkhoff, T.: A framework for generating network-based moving objects. GeoInformatica 6(2): 153-180 (2002)
[4] Chen, S., Ooi, B.C., Tan, K.L., Nascimento, M.A.: The ST2B-tree: A Self-Tunable Spatio-Temporal B^+-tree Index for Moving Objects. Proc. ACM SIGMOD, pp. 29–42 (2008)
[5] Frentzos, E.: Indexing objects moving on fixed networks. Proc. SSTD, 289–305 (2003)
[6] Garey, M.R., Johnson, D.S.: Computers and Intractability; A Guide to the Theory of NP-Completeness, New York (1990)
[7] Güting, R.H., Almeida, V.T. de, Ding, Z.: Modeling and Querying Moving Objects in Networks. VLDB Journal 15(2): 165-190 (2006)
[8] Hadjieleftheriou, M., Kollios, G., Tsotras, J., Gunopulos, D.: Indexing spatiotemporal archives. VLDB Journal 15(2): 143-164 (2006)
[9] Karypis, G., Kumar, V.: A Fast and Highly Quality Multilevel Scheme for Partitioning Irregular Graphs. SIAM Journal on Scientific Computing, Vol. 20, No. 1, pp. 359—392 (1999)
[10] Kriegel, H.-P., Pötke, M., Seidl, T.: Managing Intervals Efficiently in Object-Relational Databases. Proc. VLDB (2000)
[11] METIS - Family of Multilevel Partitioning Algorithms. [On-line]. Available: http://glaros.dtc.umn.edu/gkhome/views/metis
[12] Patel, J.M., Arbor, A., Chen, Y., Chakka, V.P.: STRIPES: an efficient index for predicted trajectories. Proc. ACM SIGMOD 635–646 (2004)
[13] Pelanis, M., Saltenis, S., Jensen, C.S.: Indexing the past, present, and anticipated future positions of moving objects. ACM Trans. Database Syst. 31(1): 255-298 (2006)
[14] Pfoser, D., Jensen, C.S.: Indexing of Network-Constrained Moving Objects. Proc. ACM-GIS, 25–32 (2003)
[15] Speicys, L., Jensen, C.S.: Enabling Location-based Services - Multi-Graph Representation of Transportation Networks. GeoInformatica 12(2): 219-253, (2008)
[16] Tao, Y., Papadias, D.: MV3R-Tree: A spatio-temporal access method for timestamp and interval queries. Proc. VLDB, pp. 431-440 (2001)
[17] Tao, Y., Papadias, D., Sun, J.: The TPR*-tree: an optimized spatio-temporal access method for predictive queries. Proc. VLDB 790–801 (2003)
[18] Saltenis, S., Jensen, C.S., Leutenegger, S.T., Lopez, M.A.: Indexing the positions of continuously moving objects. Proc. ACM SIGMOD 331–342 (2000)
[19] Sun, J., Papadias, D., Liu, B.: Querying about the Past, the Present and the Future in Spatio-Temporal Databases. Proc. ICDE 202–213 (2004)

[20] Lin, D., Jensen, C.S., Ooi, B.C., Saltenis, S.: Efficient indexing of the historical, present, and future positions of moving objects. Proc. MDM, 59–66 (2005)
[21] Mokbel, M.F., Xiong, X., Aref, W.G.: SINA: Scalable Incremental Processing of Continuous Queries in Spatio-temporal Databases. Proc. SIGMOD: 623-634 (2004)
[22] Prabhakar, S., Xia, Y., Kalashnikov, D.V., Aref, W.G., Hambrusch, S.E.: Query Indexing and Velocity Constrained Indexing: Scalable Techniques for Continuous Queries on Moving Objects. IEEE Trans. on Computers, 51(10), (2002)
[23] Tao, Y., Papadias, D., Shen, Q.: Continuous Nearest Neighbor Search. Proc. VLDB (2002)
[24] Sandu Popa, I., Zeitouni, K, Oria, V., Barth, D., Vial, S.: PARINET: A tunable access method for in-network trajectories. Proc. ICDE: 177-188 (2010)
[25] Theodoridis, Y., Stefanakis, E., Sellis, T.K.: Efficient Cost Models for Spatial Queries Using R-Trees. IEEE Trans. Knowl. Data Eng. 12(1): 19-32 (2000)
[26] Jensen, C.S., Lin, D., Ooi, B.C.: Query and update efficient B^+-tree based indexing of moving objects. Proc. VLDB: 768–779 (2004)
[27] Shekhar, S., Liu, D: CCAM: A Connectivity-Clustered Access Method for Networks and Network Computations. TKDE, 19(1), 102-119 (1997)
[28] Papadias, D., Zhang, J., Mamoulis, N., Tao, Y.: Query Processing in Spatial Network Databases. Proc. VLDB 802-813 (2003)