TrIMPI: A Data Structure for Efficient Pattern Matching on Moving Objects

Tsvetelin Polomski
Christian-Albrechts-University at Kiel
Hermann-Rodewald-Straße 3
24118 Kiel
tpo@is.informatik.uni-kiel.de

Hans-Joachim Klein
Christian-Albrechts-University at Kiel
Hermann-Rodewald-Straße 3
24118 Kiel
hjk@is.informatik.uni-kiel.de

ABSTRACT

Managing movement data efficiently often requires the exploitation of some indexing scheme. Taking into account the kind of queries issued to the given data, several indexing structures have been proposed which focus on spatial, temporal or spatio-temporal data. Since all these approaches consider only raw data of moving objects, they may be well-suited if the queries of interest contain concrete trajectories or spatial regions. However, if the query consists only of a qualitative description of a trajectory, e.g. by stating some properties of the underlying object, sequential scans on the whole trajectory data are necessary to compute the property, even if an indexing structure is available.

The present paper presents some results of an ongoing work on a data structure for Trajectory Indexing using Motion Property Information (TrIMPI). The proposed approach is flexible since it allows the user to define application-specific properties of trajectories which have to be used for indexing. Thereby, we show how to efficiently answer queries given in terms of such qualitative descriptions. Since the index structure is built on top of ordinary data structures, it can be implemented in arbitrary database management systems.

Keywords

Moving object databases, motion patterns, indexing structures

1. INTRODUCTION AND MOTIVATION

Most index structures for trajectories considered in the literature (e.g. [8]) concentrate on (time dependent) positional data, e.g. R-Tree [9] or TPR*-Tree [17]. There are different approaches (e.g. [1], [12]) exploiting transformation functions on the original data and thereby reducing the indexing overhead through "light versions" of the trajectories to be indexed. In these approaches only stationary data is being handled. In cases where the queries of interest consist of concrete trajectories or polygons covering them, such indexing schemata as well as trajectory compression techniques (e.g. [1], [6], [10], [12], [13]) may be well-suited. However, there are applications [14] where a query may consist only of a qualitative description, e.g. *return all trajectories where the underlying object slowed down (during any time interval) and after that it changed its course*. Obviously, the motion properties *slowdown* and *course alteration* as well as their temporal adjustment can be computed using formal methods. The crucial point is that, even if an indexing structure is used, the stated properties must be computed for each trajectory and this results in sequential scan(s) on the whole trajectory data. Time consuming processing of queries is not acceptable, however, in a scenario where fast reaction on incoming data streams is needed. An example of such a situation with so-called *tracks* computed from radar and sonar data as input is the detection of patterns of skiff movements typical for many piracy attacks [14]. A *track* comprises the position of an object at a time moment and can hold additional information e.g. about its current course and velocity. Gathering the tracks of a single object over a time interval yields its trajectory over this interval.

To address the efficiency problem, we propose an indexing scheme which is not primarily focused on the "time-position data" of trajectories but uses meta information about them instead.

We start with a discussion of related work in Section 2. Section 3 provides some formal definitions on trajectories and their motion properties. In section 4 we introduce the indexing scheme itself and illustrate algorithms for querying it. Section 5 summarizes the present work and outlines our future work.

2. RELATED WORK

In this section we provide a short overview on previous contributions which are related to our approach. We start the section by reviewing classical indexing structures for moving objects data. Next to this, we show an approach which is similar in general terms to the proposed one and finally we review literature related to semantical aspects of moving objects.

2.1 Indexing of Spatial, Temporal and Spatio-Temporal Data

The moving object databases community has developed several data structures for indexing movement data. According to [8], these structures can be roughly categorized as structures indexing only spatial data, also known as *spatial access methods* (*SAM*); indexing approaches for temporal data, also known as *temporal index structures*; and those which manage both - spatial and temporal data, also known as *spatio-temporal index structures*. One of the first structures developed for SAMs is the well-known R-Tree [9]. Several extensions of R-Trees have been provided over the years, thus yielding a variety of spatio-temporal index structures. An informal schematic overview on these extensions, including also new developments as the HTPR*-Tree [7] can be found in [11]. Since all of the proposed access methods focus mainly on the raw spatio-

25^{th} GI-Workshop on Foundations of Databases (Grundlagen von Datenbanken), 28.05.2013 - 31.05.2013, Illmenau, Germany.

temporal data, they are well-suited for queries on history of movement and predicting new positions of moving objects, or for returning most similar trajectories to a given one. If a query consists only of a qualitative description, however, all the proposed indexing structures are of no use.

2.2 Applying Dimensionality Reduction upon Indexing - the GEMINI Approach

The overall approach we consider in this work is similar to the GEMINI (GEneric Multimedia INdexIng method) indexing scheme presented in [6]. This approach was originally proposed for time series and has been applied later for other types of data, e.g. for motion data in [16]. The main idea behind GEMINI is to reduce the dimensionality of the original data before indexing. Therefor, representatives of much lower dimensionality are created for the data (trajectory or time series) to be indexed by using an appropriate transform and used for indexing. A crucial result in [6] is that the authors proved that in order to guarantee no false dismissals [12], the exploited transform must retain the distance (or similarity) of the data to be indexed, that is, the distance between representatives should not exceed the distance of the original time series.
In the mentioned approaches, the authors achieve encouraging results on querying most similar trajectories (or time series) to a given one. However, since the representatives of the original data are trajectories or time series, respectively, evaluating a query which only describes a motion behavior would result in the inspection of all representatives.

2.3 Semantical Properties of Movement

Semantical properties of movement data have been considered in various works, e.g. in [2], [5], and [15].
The authors of [2] propose a spatio-temporal representation scheme for moving objects in the area of video data. The considered representation scheme distinguishes between spatio-temporal data of trajectories and their topological information, and also utilizes information about distances between pairs of objects. The topological information itself is defined through a set of *topological relations operators* expressing spatial relations between objects over some time interval, including *faraway*, *disjoint*, *meet*, *overlap*, *is-included-by*/*includes* and *same*.
In [5], a comprehensive study on the research that has been carried out on data mining and visual analysis of movement patterns has been provided. The authors propose a conceptual framework for movement behavior of different moving objects. The extracted behavior patterns are classified according to a taxonomy.
In [15], the authors provide some aspects related to a semantic view of trajectories. They show a conceptual approach for how trajectory behaviors can be described by predicates that involve movement attributes and/or semantic annotations. The provided approach is rather informal and considers behavior analysis of moving objects on a general level.

3. FORMAL BACKGROUND

This section provides the formal notions as well as the definitions needed throughout the rest of the paper. We start with the term *trajectory* and then direct later our attention to motion properties and patterns.

3.1 Trajectories

In our approach we consider the trajectory τ_o of an object o simply as a function of time which assigns a position to o at any point in time. Since time plays only a role for the determination of temporal causality between the positions of an object, we abstract from "real time" and use any *time domain* instead. A *time domain* is any set which is interval scaled and countably infinite. The first requirement ensures that timestamps can be used for ordering and, furthermore, that the "delay" between two time assignments can be determined. The second requirement ensures that we have an infinite number of "time moments" which can be unambiguously indexed by elements of $\mathbb{N}$. In the following we denote a time domain by T.

Since objects move in a space, we also need a notion for a *spatial domain*. In the following, let S denote the spatial domain. We require that S is equipped with an adequate metric, such as the Euclidean distance (e.g. for $S = \mathbb{R} \times \mathbb{R}$), which allows us to measure the spatial distance between objects.

Having the notions of time and space we can define formally the term *trajectory*.

Definition 1. Let T, S and O denote a time domain, a space domain and a set of distinct objects, respectively. Then, *the trajectory* τ_o *of an object* $o \in O$ is a function $\tau_o : T \rightarrow S$.

For brevity, we can also write the trajectory of an object $o \in O$ in the form $(o, t_0, s_0), (o, t_1, s_1) \ldots$ for those $t \in T$ where $\tau_o(t) = s$ is defined. A single element (o, t_i, s_i) is called the *track of object o at time* t_i.

3.2 Motion Patterns

We consider a motion pattern as a sequence of properties of trajectories which reveal some characteristics of the behavior of the underlying moving objects. Such properties may be expressed through any predicates which are important for the particular analysis, such as *start*, *stop*, *turn*, or *speedup*.

Definition 2. Let T be a time domain, $\mathfrak{T}$ be the set of trajectories of an object set O over T, and I_T be the set of all closed intervals over T. A *motion property* on $\mathfrak{T}$ is a function $p : 2^{\mathfrak{T}} \times I_T \rightarrow \{true, false\}$.

That is, a motion property is fulfilled for a set of trajectories and a certain time interval if the appropriate predicate is satisfied. To illustrate this definition, some examples of motion properties are provided below:

- **Appearance**: Let $t \in T$. Then we define appear$(\cdot,\cdot)$ as follows: appear$(\{\tau_o\}, [t,t]) = true \Leftrightarrow \forall t' \in T : \tau_o(t') \neq$ undefined $\rightarrow t \leq t'$. That is, an object "appears" only in the "first" moment it is being observed.

- **Speedup**: Let $t_1, t_2 \in T$ and $t_1 < t_2$. Then speedup$(\cdot,\cdot)$ is defined as follows: speedup$(\{\tau_o\}, [t_1, t_2]) = true \Leftrightarrow v(\tau_o, t_1) < v(\tau_o, t_2) \wedge \forall t \in T : t_1 \leq t \leq t_2 \rightarrow v(\tau_o, t_1) \leq v(\tau_o, t) \leq v(\tau_o, t_2)$ where $v(\tau_o, t)$ denotes the velocity of the underlying moving object o at time t. That is, the predicate speedup is satisfied for a trajectory and a time interval if and only if the velocity of the underlying object is increasing in the considered time interval. Note that the increase may not be strictly monotonic.

- **Move away**: Let $t_1, t_2 \in T$ and $t_1 < t_2$. Then we define: moveaway$(\{\tau_{o_1}, \tau_{o_2}\}, [t_1, t_2]) = true \Leftrightarrow \forall t, t' \in T : t_1 \leq t < t' \leq t_2 \rightarrow dist(\tau_{o_1}, \tau_{o_2}, t) < dist(\tau_{o_1}, \tau_{o_2}, t')$ where the term $dist(\tau_{o_1}, \tau_{o_2}, t)$ denotes the distance between the underlying moving objects o_1 and o_2 at time t. That is, two objects are moving away from each other for a time interval, if their distance is increasing during the considered time interval.

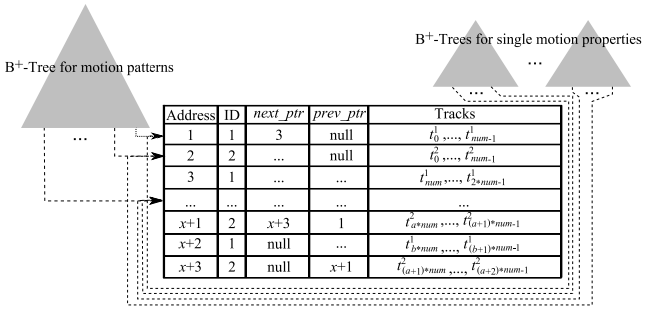

Figure 1: Overview of the index structure

Using motion properties, a *motion pattern* of a single trajectory or a set of trajectories is defined as *a sequence of motion properties ordered by the time intervals in which they are fulfilled*. It is important to note, that this common definition of a motion pattern allows multiple occurrences of the same motion property in the sequence. In order to get a well-defined notion it has to be required that the time intervals in which the motion properties are fulfilled are disjoint or that meaningful preferences on the motion properties are specified in order to allow ordering in case the time intervals overlap.

4. TRAJECTORY INDEXING USING MOTION PROPERTIES

In this section we explain how the proposed index is being created and used. Index creation starts with the determination of the motion pattern of each trajectory to be indexed. For this purpose, the motion predicates specified by the user are computed. The resulting motion patterns are indexed with references to the original trajectories.

The resulting index is schematically depicted in Figure 1. TrIMPI consists mainly of a data structure holding the raw trajectory data, and secondary index structures for maintaining motion patterns. Thereby, we differentiate between indexing single motion properties and indexing motion patterns.

A query to the index can be stated either through a motion pattern or through a concrete trajectory. The index is searched for motion patterns containing the given one or the computed one, respectively. In both cases, the associated trajectories are returned. The following subsections consider the outlined procedures more precisely.

4.1 Indexing Trajectory Raw Data

Since the focus of TrIMPI is not on querying trajectories by example, the index structure for the raw trajectory data can be rather simple. For our implementation, we considered a *trajectory record file* as proposed by [3]. This structure (Figure 1) stores trajectories in records of fixed length. The overall structure of the records is as follows

ID_o	*next_ptr*	*prev_ptr*	$\{track_0, \ldots, track_{num-1}\}$

ID_o denotes the identifier of the underlying moving object, *next_ptr* and *prev_ptr* are references to the appropriate records holding further parts of the trajectory, and $\{track_0, \ldots, track_{num-1}\}$ is a list of tracks of a predefined fixed length *num*. If a record r_i for a trajectory τ_o gets filled, a new record r_j is created for τ_o holding its further tracks. In this case, $next\_ptr_{r_i}$ is set up to point to r_j, and $prev\_ptr_{r_j}$ is set up to point to r_i.

Using a trajectory record file, the data is not completely clustered, but choosing appropriate record size leads to partial clustering of the trajectory data in blocks. This has the advantage that extracting the complete trajectory requires only loading as much blocks as needed for storing a trajectory.

4.2 Indexing Motion Patterns

For the maintenance of motion patterns we consider two cases - single motion properties and sequences of motion properties. Storing single motion properties allows the efficient finding of trajectories which contain the considered motion property. This is advantageous if the searched property is not often satisfied. Thus, for each motion property p a "list" DBT_p holding all trajectories satisfying this property is maintained. As we shall see in Algorithm 4.3, we have to combine such lists and, thus, a simple unsorted list would not be very favourable. Therefore, we implement these lists through B$^+$-Trees (ordered by the trajectory/object identifiers). An evaluation of union and intersection of two B$^+$-Trees with m and n leaves can be performed in $O(m \log \frac{m+n}{m})$[4].

The search for motion patterns with more than one motion property can be conducted through the single DBT_p structures. However, if the query motion pattern is too long, too many intersections of the DBT_p structures will happen and the resulting trajectories will have to be checked for containing properties that match the given order, as well. To overcome this problem, sequences of motion properties are stored in an additional B$^+$-Tree structure DBT. The elements of DBT have the form (p, τ_o) where p is a motion pattern, and $o \in O$. To sort the elements of DBT, we apply lexicographical ordering. As a result, sequences with the same prefix are stored consecutively. Thus, storing of motion patterns that are prefixes of other motion patterns can be omitted.

4.3 Building the Index

The algorithm for the index creation is quite simple. It consists primarily of the following steps:

- Determine the motion properties for each trajectory τ_o. Consider, if needed, a sliding window or some reduction or segmenting technique as proposed in [1], [6], [10], [12], [13], for example. Generate a list f of the motion properties of τ_o, ordered by their appearance in τ_o.
- Store τ_o into the trajectory record file.
- Apply Algorithm 4.1 to f to generate access keys relevant for indexing.
- For each generated access key, check whether it is already contained in the index. If this is not the case, store it in the index. Link the trajectory record file entry of τ_o to the access key.

Algorithm 4.1 is used to generate index keys of a pattern. An index key is any *subpattern* $p' = (p'_j)_{j=0}^{m-1}$ of a pattern $p = (p_i)_{i=0}^{n-1}$ which is defined as follows:

- For each $j \leq m-1$ exists $i \leq n-1$ such that $p'_j = p_i$
- For each j, k such that $0 \leq j < k \leq m-1$ exist i, l such that $0 \leq i < l \leq n-1$ and $p'_j = p_i$ and $p'_k = p_l$.

To generate the list of index keys, algorithm 4.1 proceeds iteratively. At each iteration of the outer loop (lines 3 to 16) the algorithm considers a single element p of the input sequence f. On the one hand, p is being added as an index key to the (interim) result (lines 14 and 15) and on the other hand it is being appended as a suffix to each previously generated index key (inner loop - lines 5 to 13). Algorithm 4.1 utilizes two sets whose elements are lists of

motion properties - `supplist` and `entries`. The set `supplist` contains at each iteration the complete set of index keys, including those which are prefixes of other patterns. The set `entries` is built in each iteration of the inner loop (lines 5 to 13) by appending the current motion property of the input sequence to any element of `supplist`. Thereby, at line 14 `entries` holds only index keys which are no prefixes of other index keys. Since the resulting lists of index keys are stored in a B^+-Tree by applying a lexicographical order, sequences of motion properties which are prefixes of other sequences can be omitted. Therefore, the set `entries` is returned as final result (line 17).
Since the given procedure may result in the computation of up to 2^{k_0} different indexing keys for an input sequence with k_0 motion properties, a global constant G is used to limit the maximal length of index keys. Using an appropriate value for G leads to no drawbacks for the application. Furthermore, the proposed querying algorithm can handle queries longer than G.

Algorithm 4.1 Building the indexing keys

```
Require: f is a sequence of motion properties
Require: G is the maximal length of sequences to be indexed
 1 function createIndexKeys(f)
 2     supplist ← empty set of lists
 3     for all a ∈ f do
 4         entries ← empty set of lists
 5         for all l ∈ supplist do
 6             new ← empty list
 7             if |l| ≤ G then
 8                 new ← l.append(a)
 9             else
10                 new ← l
11             end if
12             entries ← entries ∪ {new}
13         end for
14         entries ← entries ∪ {[a]}
15         supplist ← entries ∪ supplist
16     end for
17     return entries
18 end function
```

4.4 Searching for Motion Patterns

Since the index is primarily considered to support queries on sequences of motion properties, the appropriate algorithm for evaluating such queries given in the following is rather simple. In its "basic" version, query processing is just traversing the index and returning all trajectories referenced by index keys which contain the queried one (as a subpattern). This procedure is illustrated in algorithm 4.2. There are, however, some special cases which have to

Algorithm 4.2 Basic querying of trajectories with a sequence of motion properties

```
Require: s is a sequence of motion properties; |s| ≤ G
Require: DBT is the index containing motion patterns
1 function GetEntriesFromDBT(s)
2     result ← {τ_o | ∃p s.t. s ≤ p ∧ (p, τ_o) ∈ DBT}
3     return result
4 end function
```

be taken into account. The first of them considers query sequences which are "too short". As stated in Section 4.2, it can be advantageous to evaluate queries containing only few motion properties by examination of the index structures for single motion properties. To be able to define an application specific notion of "short" queries, we provide besides G an additional global parameter α for which holds $1 \leq \alpha < G$. In algorithm 4.3, which evaluates queries of patterns of arbitrary length, each pattern of length shorter than α is being handled in the described way (lines 3 to 8). It is important that each trajectory of the interim result has to be checked whether it matches the queried pattern (lines 9 to 13).
The other special case are queries longer than G (lines 16 to 24). As we have seen in algorithm 4.1, in such cases the index keys are cut to prefixes of length G. Thus, the extraction in this case considers the prefix of length G of the query sequence (lines 17) and extracts the appropriate trajectories (line 18). Since these trajectories may still not match the query sequence, e.g. by not fulfilling some of the properties appearing on a position after $G-1$ in the input sequence, an additional check of the trajectories in the interim result is made (lines 19 to 23).
The last case to consider are query sequences with length between α and G. In these cases, the index *DBT* holding the index keys is searched through a call to algorithm 4.2 and the result is returned. Finally, the function `Match` (algorithm 4.4) checks whether a tra-

Algorithm 4.3 Querying trajectories with a sequence of arbitrary length

```
Require: s is a sequence of motion properties
Require: G is the maximal length of stored sequences
Require: DBT_p is the index of the property p
Require: 1 ≤ α < G maximal query length for searching single property indexes
 1 function GetEntries(s)
 2     result ← empty set
 3     if |s| < α then
 4         result ← 𝔗
 5         for all p ∈ s do
 6             suppset ← DBT_p
 7             result ← result ∩ suppset
 8         end for
 9         for all τ_o ∈ result do
10             if ! match(τ_o, s) then
11                 result ← result\{τ_o}
12             end if
13         end for
14     else if |s| ≤ G then
15         result ← GetEntriesFromDBT(s)
16     else
17         k ← s[0..G − 1]
18         result ← GetEntriesFromDBT(k)
19         for all τ_o ∈ result do
20             if ! match(τ_o, s) then
21                 result ← result\{τ_o}
22             end if
23         end for
24     end if
25     return result
26 end function
```

jectory τ_o fulfills a pattern s. For this purpose, the list of motion properties of τ_o is being generated (line 2). Thereafter, s and the generated pattern of τ_o are traversed (lines 5 to 14) so that it can be checked whether the elements of s can be found in the trajectory pattern of τ_o in the same order. In this case the function `Match` returns `true`, otherwise it returns `false`.

5. CONCLUSIONS AND OUTLOOK

In this paper we provided some first results of an ongoing work on an indexing structure for trajectories of moving objects called TrIMPI. The focus of TrIMPI lies not on indexing spatio-temporal data but on the exploitation of motion properties of moving objects. For this purpose, we provided a formal notion of motion properties and showed how they form a motion pattern. Furthermore, we showed how these motion patterns can be used to build a meta index. Algorithms for querying the index were also provided. In the next steps, we will finalize the implementation of TrIMPI and perform tests in the scenario of the automatic detection of piracy attacks mentioned in the Introduction. As a conceptual improvement of the work provided in this paper, we consider a flexibilisation of

Algorithm 4.4 Checks whether a trajectory matches a motion pattern

```
Require: τ_o is a valid trajectory
Require: s is a sequence of motion properties
 1 function MATCH(τ_o, s)
 2     motion_properties ← compute the list of motion properties of τ_o
 3     index_s ← 0
 4     index_props ← 0
 5     while index_props < motion_properties.length do
 6         if motion_properties[index_props] = s[index_s] then
 7             index_s ← index_s + 1
 8         else
 9             index_props ← index_props + 1
10         end if
11         if index_s = s.length then
12             return true
13         end if
14     end while
15     return false
16 end function
```

the definition of motion patterns including arbitrary temporal relations between motion predicates.

6. ACKNOWLEDGMENTS

The authors would like to give special thanks to their former student Lasse Stehnken for his help in implementing TrIMPI.

7. REFERENCES

[1] R. Agrawal, C. Faloutsos, and A. N. Swami. Efficient similarity search in sequence databases. In D. B. Lomet, editor, *Proceedings of the 4th International Conference on Foundations of Data Organization and Algorithms, FODO'93, Chicago, Illinois, USA, October 13-15, 1993*, volume 730 of *Lecture Notes in Computer Science*, pages 69–84. Springer, 1993.

[2] J.-W. Chang, H.-J. Lee, J.-H. Um, S.-M. Kim, and T.-W. Wang. Content-based retrieval using moving objects' trajectories in video data. In *IADIS International Conference Applied Computing*, pages 11–18, 2007.

[3] J.-W. Chang, M.-S. Song, and J.-H. Um. TMN-Tree: New trajectory index structure for moving objects in spatial networks. In *Computer and Information Technology (CIT), 2010 IEEE 10th International Conference on*, pages 1633–1638. IEEE Computer Society, July 2010.

[4] E. D. Demaine, A. López-Ortiz, and J. I. Munro. Adaptive set intersections, unions, and differences. In *Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*, SODA '00, pages 743–752, Philadelphia, PA, USA, 2000. Society for Industrial and Applied Mathematics.

[5] S. Dodge, R. Weibel, and A.-K. Lautenschütz. Towards a taxonomy of movement patterns. *Information Visualization*, 7(3):240–252, June 2008.

[6] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast subsequence matching in time-series databases. In R. T. Snodgrass and M. Winslett, editors, *Proceedings of the 1994 ACM SIGMOD international conference on Management of data*, SIGMOD '94, pages 419–429, New York, NY, USA, 1994. ACM. 472940.

[7] Y. Fang, J. Cao, J. Wang, Y. Peng, and W. Song. HTPR*-Tree: An efficient index for moving objects to support predictive query and partial history query. In L. Wang, J. Jiang, J. Lu, L. Hong, and B. Liu, editors, *Web-Age Information Management*, volume 7142 of *Lecture Notes in Computer Science*, pages 26–39. Springer Berlin Heidelberg, 2012.

[8] R. H. Güting and M. Schneider. *Moving Object Databases*. Data Management Systems. Morgan Kaufmann, 2005.

[9] A. Guttman. R-Trees: a dynamic index structure for spatial searching. In *Proceedings of the 1984 ACM SIGMOD international conference on Management of data*, SIGMOD '84, pages 47–57, New York, NY, USA, 1984. ACM.

[10] J. Hershberger and J. Snoeyink. Speeding Up the Douglas-Peucker Line-Simplification Algorithm. In P. Bresnahan, editor, *Proceedings of the 5th International Symposium on Spatial Data Handling, SDH'92, Charleston, South Carolina, USA, August 3-7, 1992*, pages 134–143. University of South Carolina. Humanities and Social Sciences Computing Lab, August 1992.

[11] C. S. Jensen. TPR-Tree Successors 2000–2012. `http://cs.au.dk/~csj/tpr-tree-successors`, 2013. Last accessed 24.03.2013.

[12] E. J. Keogh, K. Chakrabarti, M. J. Pazzani, and S. Mehrotra. Dimensionality reduction for fast similarity search in large time series databases. *Journal Of Knowledge And Information Systems*, 3(3):263–286, 2001.

[13] E. J. Keogh, S. Chu, D. Hart, and M. J. Pazzani. An online algorithm for segmenting time series. In N. Cercone, T. Y. Lin, and X. Wu, editors, *Proceedings of the 2001 IEEE International Conference on Data Mining, ICDM'01, San Jose, California, USA, 29 November - 2 December 2001*, pages 289–296. IEEE Computer Society, 2001.

[14] T. Polomski and H.-J. Klein. How to Improve Maritime Situational Awareness using Piracy Attack Patterns. 2013. submitted.

[15] S. Spaccapietra and C. Parent. Adding meaning to your steps (keynote paper). In M. Jeusfeld, L. Delcambre, and T.-W. Ling, editors, *Conceptual Modeling - ER 2011, 30th International Conference, ER 2011, Brussels, Belgium, October 31 - November 3, 2011. Proceedings*, ER'11, pages 13–31. Springer, 2011.

[16] Y.-S. Tak, J. Kim, and E. Hwang. Hierarchical querying scheme of human motions for smart home environment. *Eng. Appl. Artif. Intell.*, 25(7):1301–1312, Oct. 2012.

[17] Y. Tao, D. Papadias, and J. Sun. The TPR*-tree: an optimized spatio-temporal access method for predictive queries. In J. C. Freytag, P. C. Lockemann, S. Abiteboul, M. J. Carey, P. G. Selinger, and A. Heuer, editors, *Proceedings of the 29th international conference on Very large data bases - Volume 29*, VLDB '03, pages 790–801. VLDB Endowment, 2003.