Cairo University

Egyptian Informatics Journal

www.elsevier.com/locate/eij
www.sciencedirect.com

FULL-LENGTH ARTICLE

A generic trajectory similarity operator in moving object databases

Nehal Magdy [*], **Mahmoud A. Sakr**, **Khaled El-Bahnasy**

Faculty of Computer and Information Sciences, Ain Shams University, Cairo, Egypt

Received 18 January 2016; revised 21 June 2016; accepted 21 July 2016

KEYWORDS

Trajectory similarity;
Moving objects databases;
Secondo;
TWED

Abstract Evaluating similarity between trajectories of moving objects is important for wide range of applications. The existing similarity measures typically define some meaning of similarity and propose algorithms for computing it. We think that the meaning of similarity is application dependant, and should only be determined by the user. Therefore, there is a need for a generic approach where users can define the meaning of similarity. In this paper, we propose a parametrized similarity operator, based on the time warped edit distance, where the meaning of similarity is generic and left for user to define. Our proposed operator is implemented in Secondo and evaluated using both synthetic and real datasets. The results were promising and as expected.

1. Introduction

The increase in devices that can be used to track moving objects such as GPS devices causes a rapid growth in movement data. Moving Objects can be hurricanes, cars, animals, athletes, suspected terrorists, or network data packets. They change their location/value with time, and tracking them produces a sequence of observations that digitally represent the movement history in the form of the so-called trajectories. Due to the large amount of movement data, there has been a deep interest in proposing new methodologies in classification, clustering, indexing, approximating and simplifying such data. All these works collaborate toward building a generic Moving Objects Database system (MOD).

There are two flavors of MOD. The first focuses on the current movement of objects and the prediction of the near future values [1]. In that flavor no historical movement data are stored. In the second flavor, historical movement data are stored and represented. These systems focus on querying and analyzing historical movements [2]. In this paper we focus on the second flavor which is called trajectory databases.

A trajectory in real life is a continuous function in time [3,4]. Due to limitations of devices, tracking such a continuous function is observed in a discrete manner, typically as a sequence of time stamped values. This finite sequence can be observed in many different ways [3]:

[*] Corresponding author.
E-mail addresses: nehalmagdy@cis.asu.edu.eg (N. Magdy), mahmoud.sakr@cis.asu.edu.eg (M.A. Sakr), khaled.bahnasy@cis.asu.edu.eg (K. El-Bahnasy).

Peer review under responsibility of Faculty of Computers and Information, Cairo University.

Production and hosting by Elsevier

http://dx.doi.org/10.1016/j.eij.2016.07.001
1110-8665 © 2016 Production and hosting by Elsevier B.V. on behalf of Faculty of Computers and Information, Cairo University.

- A time based approach where values are observed at regularly spaced time moments e.g., every 2 min.
- Value based approach where a new value is recorded when a significant change from the previous value occurs.
- Location based approach where values are observed when the object comes close to specific location.
- Event based approach where values are observed when certain events occur.
- Finally any combinations of these approaches.

This observation of object's movement can be utilized to induce the actual (or approximate) movement trajectory.

An important function in a MOD system is to measure the similarity between trajectories [5,6]. It is important for many application domains such as surveillance systems, recommender systems, and stock market analysis. Using a similarity measure, it is possible to find suspicious movements, rare movements, and frequent patterns, predict future occurrence of phenomena such as hurricanes, and recommend traveling routes. The similarity meaning varies from one application to another. It could be the similarity of one of the movement attributes such as speed, direction and acceleration, or the raw values of movement. All previously proposed similarity measures propose their technique and implicitly define the meaning of similarity in their implementation. They restrict user to that meaning with no ability to define the meaning of similarity from his own perspective. This previous disability is one of the research gaps. In this paper, a generic similarity operator is proposed where the meaning of similarity is passed by user as a parameter based on application domain. So, our contribution is twofold:

- Proposing a parametrized TWED based similarity operator where the meaning of similarity is passed as a parameter.
- Implementing that operator in the open source MOD, Secondo, and evaluating our operator based on a set of experiments using synthetic and real datasets.

The rest of paper is organized as follows: Section 2 describes the motivation and the related work in the field of measuring similarity between trajectories. In Section 3, we give a background on MODs and discuss in depth the TWED similarity measure. In Section 4, our proposed operator is presented with a detailed description. Then, our implementation, experiments, and the output results are discussed in Section 5. Finally, Section 6 concludes the paper, and sets ideas for future research work.

2. Motivation and related work

There exist in the literature measures for evaluating the similarity between trajectories. They can be classified into spatial, spatiotemporal and temporal similarity measures [4]. Spatial measures focus on measuring similarity based on the spatial dimension while ignoring the time dimension. Spatiotemporal similarity considers both spatial and time dimensions. Temporal similarity measures focus only on time dimension.

The spatial similarity measures are based on spatial movement attributes, such as path of the moving object (raw representation), geometric shape representation of object's trajectory and direction of data representation [4]. For example, the work in [7], is based on spatial raw representation where trajectories' elements are aligned at the same position, trajectories must have same number of elements, local time shifting is not taken into consideration and its efficiency decreases with the existence of noise. The works in [8–11], are based on geometric shape of trajectories. They consider local time shifting, and compare trajectories of possibly different number of elements. In [8,9], they consider local time shifting. Other works such as [12,13] conclude that the similarity measures that are based on raw representation of trajectories are sensitive to rotation, shifting and scaling. Therefore, they proposed a measure based on movement direction. The work in [12] builds on the work in [13], and takes into account local time shifting, robustness to noise and the possibility of comparing trajectories of different lengths.

The spatiotemporal similarity is based on spatiotemporal movement attributes such as speed, and the time series representation of object's trajectory [4]. Measures based on time series representation of a trajectory are usually associated with a distance function that can either be metric or be non-metric [4]. Any distance function is said to be metric if it satisfies non-negativity, uniqueness, symmetry and triangle inequality. Based on the used distance function, these measures can be divided into two classes. The first class uses L_1-norm and L_2-norm as a distance measure such as dynamic time warping [14], edit distance with real penalty [15], and time warp edit distance [16]. The second class scores similarity based on a matching threshold such as longest common subsequence [17], and edit distance on real sequences [18]. The accuracy of the first class is subject to the existence of noise, while the second class is more robust to noise. Both classes handle local time shifting, and trajectories can have different lengths. Another work is proposed in [19], where similarity is based on both the speed and the path of moving objects. It follows a warping approach based on dynamic time warping. This approach works on trajectories of different lengths and handles local time shifting, but it is not robust to noise.

The temporal similarity measures are based on temporal attributes such as time instance, time interval, and time duration [4]. In [20,21], the similarity measures are based on the time instances on which object exists. The first work analyzes the migration of different salmon in the rivers. They conclude that some populations enter their habitat before others. In the second, they analyze migration of raptors, and conclude that adult honey buzzards migrate after their juvenile species. Another temporal movement parameter is the temporal interval that can be represented as an ordered set of time instances. The fundamental work of Allen [22] proposes an algebra for reasoning over time intervals. Based on Allen interval relations, migratory movement of birds can be expressed as that the faster birds' movement occurs during the movement of the slower ones. Temporal duration is another temporal movement parameter which represents the time difference between two time instances. In [23], they track the movement of adult and juvenile sea eagles. They conclude that migratory movement of adults lasts shorter than their younger species.

All previously mentioned similarity measures implicitly define the meaning of similarity. Some assumed it as the speed or direction or time interval of the moving objects while others worked on the raw data of trajectories or combination of them. They also defined what distance functions are used for assessing similarity in their implementation, like L_1-norm.

These works lack the genericness, and therefore are not suitable in the context of a MOD system. A MOD system would rather require a generic operator that can satisfy the requirements of a wide variety of applications. Therefore we were motivated in this paper to propose such a generic operator, where the meaning of similarity can be controlled by the MOD user.

3. Background

The proposed similarity operator in this paper is based on the time-warp edit distance [16], and on the moving objects model in [2]. It is also fully implemented in the open source MOD Secondo [24]. In this section, these works are presented as a necessary background for the rest of the paper.

3.1. Time warp edit distance

We adopt the representation of a moving object trajectory as a discrete time series. Thus discrete time series similarity algorithms can be used. By far, varieties of elastic matching methods have been proposed, e.g., edit distance [7], dynamic time warping [14], edit distance with real penalty [15].

We choose to use the recently developed distance measure, TWED [16], for measuring the distance between moving object trajectories. The reason to choose TWED is based on its properties that are favorable to measuring distances on trajectories, as follows:

- Superior performance in dealing with local time shifting.
- Being a metric, as it satisfies the triangular inequality. Thus it is suitable for classification and clustering.
- It is an elastic metric (a property which is inherited from the edit distance). TWED, however, introduces a stiffness parameter to control the elasticity of the metric, and thus more flexible for time series matching.

Given two time series R, S representing a pair of trajectories, as in Fig. 1, TWED edits the two series using three operations: $delete_R$, $delete_S$, and *match*. Each of the three operations has a penalty, and the distance is sum of penalties of the minimal-cost sequence of editing operations needed to transform one time series into the other.

A graphical illustration of the TWED operations is given in Fig. 2. The *Delete* operation inside R or S involves dragging and dropping the sample to be deleted to its previous one. The cost associated with this delete operations is the length of the vector from the deleted sample to its previous one, as determined by the user defined function d_{lp} (e.g., L_n-norm). An extra associated constant penalty called Lambda λ is added to this delete cost. So, the cost of deleting element m from the series R would be $d_{lp}(r_m, r_{m-1}) + \lambda$. The *Match* operation involves dragging and dropping the segment between two samples in the first trajectory to the matching segment in the second trajectory. The cost associated with the match operation is the sum of lengths of two vectors connecting the start and end of both segments, which is $d_{lp}(r_m, s_n) + d_{lp}(r_{m-1}, s_{n-1})$.

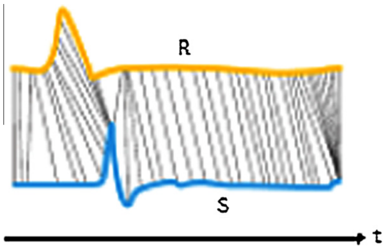


Fig. 1 Sequence matching.

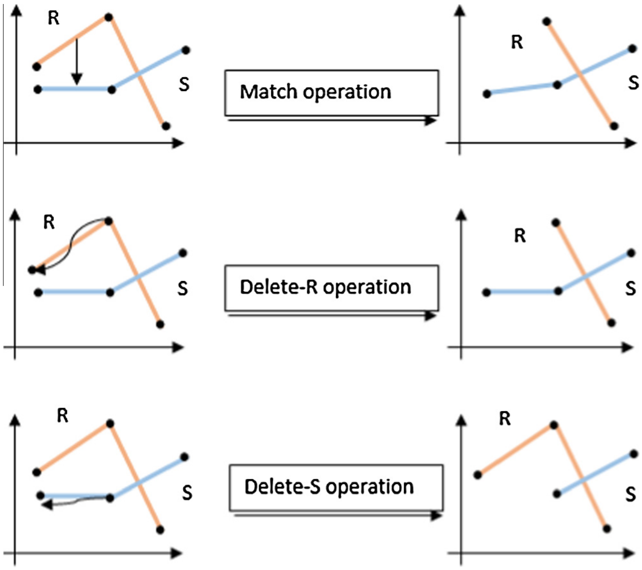


Fig. 2 TWED operations.

To provide the controlled temporal elasticity, TWED includes the time stamp difference in all its cost functions. These time differences are multiplied by a stiffness parameter γ to control the elasticity. TWED, hence, accepts four parameters $R_1^m, S_1^n, \lambda, \gamma$ and returns the minimum of:

$$\begin{cases} \mathrm{TWED}_{\lambda,\gamma}(R_1^{m-1}, S_1^n) + \gamma \cdot (t_{r_m} - t_{r_{m-1}}) + d_{lp}(r_m, r_{m-1}) + \lambda & delete - R \\ \mathrm{TWED}_{\lambda,\gamma}(R_1^m, S_1^{n-1}) + \gamma \cdot (t_{s_n} - t_{s_{n-1}}) + d_{lp}(s_n, s_{n-1}) + \lambda & delete - S \\ \mathrm{TWED}_{\lambda,\gamma}(R_1^{m-1}, S_1^{n-1}) + d_{lp}(r_m, s_n) + \gamma \cdot (t_{s_n} - t_{r_m}) + d_{lp}(r_{m-1}, s_{n-1}) + \gamma \cdot (t_{s_{n-1}} - t_{r_{m-1}}) & Match \end{cases} \tag{1}$$

The Lambda and Stiffness values are dataset dependent. They can be learned using cross validation such as leave one out procedure in any dataset [16]. Using leave one out procedure, the stiffness parameter is adapted as follows: we use the dataset to select the best (γ) value as well as the best λ value from a set of γ and λ values, namely the ones leading to the minimal error rate on the dataset, according to a leave-one-out procedure (that consists of iteratively selecting one trajectory from the dataset and then considering it as a test against the remaining trajectories within the dataset itself). After evaluating error rate for all (γ, λ) combinations, we choose the one with the minimum error rate. If different (γ, λ) values lead to the minimal error rate then the pairs containing the highest γ value are selected first. Finally, then the pair with the highest λ value is selected.

Alternatively, if the user knows enough about the data, the values of lambda and stiffness can be intuitively set. Since λ is the deletion penalty, it needs to be set with the value difference threshold, after which two values cannot be considered a match. For example, two coordinates on a road network cannot match if they are more than 50 m apart (assuming a maximum GPS error of 50 m). In such a case, lambda should be set to 50 m. The intuition is that below 50 m distance, a match is more favored than a deletion and the other way around.

Stiffness γ, on the other hand, is multiplied by the temporal difference penalty. Its role is to normalize the value difference and the temporal difference scales. The user should then decide what is the maximum expected/possible value difference in the dataset, and what is the maximum expected/possible temporal difference. The stiffness value is then set to the division of the former over the later. In such a way, the penalties of value difference and temporal difference will have similar effects on the overall similarity. Thus, none of the two scales will dominate the other.

The TWED's dynamic programming implementation in [16], accepts data of arbitrary dimension along with a suitable distance function. The distance function used is the L_n-norm where $n = 1, 2, \ldots$ and it is possible to pass its degree n as a parameter. TWED, hence, supports genericness of both the dimension of data and the degree of the used L_n-norm in its implementation. These favorable properties are inherited in the proposed trajectory distance operator.

A TWED example is shown in Fig. 3 that shows how TWED implementation works between two 1-D trajectories R and S, where four matrices are defined: Delete-R, Delete-S, Match and Final Distance Matrix.

Delete-S and Delete-R at position 'index' is measured as follows:

$$delete[index] = \begin{cases} 0, & if\ index = 0 \\ d_{lp}(sample_{i-1}, 0), & if\ index = 1 \\ d_{lp}(sample_{i-1}, sample_{i-2}), & if\ index > 1 \end{cases}$$

Match cost between the two trajectories R and S is measured as follows:

$$Match[i,j] = \begin{cases} d_{lp}(R_{i-1}, S_{j-1}) + d_{lp}(R_{i-2}, S_{j-2}), & if\ i\&j > 1 \\ d_{lp}(R_{i-1}, S_{i-1}), & otherwise \end{cases}$$

where d_{lp} is the user defined function. After building deletion and match matrices, the final distance matrix holds the accumulative cost required (added to γ^* the time stamp differences) to superimpose both trajectories where $distanceMatrix[n+1, m+1]$ is the TWED distance.

3.2. SECONDO

Our operator is implemented in SECONDO [24], a generic DBMS where various implementations of different data models can reside. With its genericness, it is possible to implement the various database models, such as object oriented, XML, and the relational model. It consists of three components: the kernel, the query optimizer, and the graphical user interface (GUI). The Kernel is the place where new algebra modules, new types, and new operators can be implemented.

SECONDO has currently several Algebras supporting moving object operations. It contains an almost complete implementation of the MOD model in [2,25]. This model defines ADTs for spatial and non-spatial moving objects. The *mpoint* type, for instance, represents a moving point object that changes its location with time (e.g., car, train). An *mreal* represents a real value that changes with time (e.g., temperature, speed of a car), and so on.

SECONDO can store moving object data in relations, and has a rich set of query operators to manipulate relations, and the moving objects inside. It supports two query languages: an SQL-like language provided by the query optimizer, and the *executable language* directly supported by the kernel. In the rest of this paper, the executable language is used for its flexibility. Our proposed operator is also embed in it. The following example illustrates the language. Let *Champs* be a relation with the type:

```
rel(tuple(<(Champ: string), (Year: string)>))
```

A query that finds the champions in year 2000 is as follows:

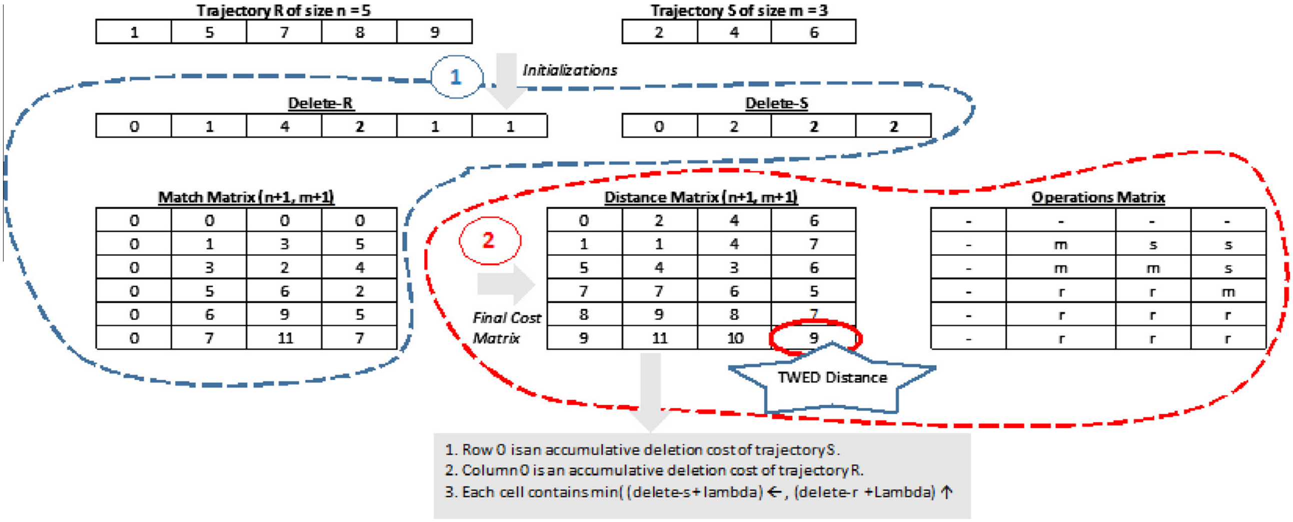
```
query Champs feed
  filter[.Year = "2000"] consume;
```

The *feed* operator loads a disk relation and converts it into an in-memory tuple stream. The *consume* operator does the opposite. The *filter* operator excludes the tuples that do not fulfill its Boolean condition. The signatures of these operators are as follows:

$\underline{rel}(tuple) \rightarrow \underline{stream}(tuple)$ **feed** _#

$\underline{stream}(tuple) \rightarrow \underline{rel}(tuple)$ **consume** _#

$\underline{stream}(tuple) \times (tuple \rightarrow \underline{bool}) \rightarrow \underline{stream}(tuple)$ **filter** _# [_]



Trajectory R of size n = 5

1	5	7	8	9

Trajectory S of size m = 3

2	4	6

Delete-R

0	1	4	2	1	1

Delete-S

0	2	2	2

Match Matrix (n+1, m+1)

0	0	0	0
0	1	3	5
0	3	2	4
0	5	6	2
0	6	9	5
0	7	11	7

Distance Matrix (n+1, m+1)

0	2	4	6
1	1	4	7
5	4	3	6
7	7	6	5
8	9	8	7
9	11	10	9

Operations Matrix

-	-	-	-
-	m	s	s
-	m	m	s
-	r	r	m
-	r	r	r
-	r	r	r

1. Row 0 isan accumulative deletion cost of trajectory S.
2. Column 0 is an accumulative deletion cost of trajectory R.
3. Each cell contains min((delete-s + lambda) ←, (delete-r +Lambda) ↑

Fig. 3 TWED example using L_1-norm, $\lambda = 1$ and $\gamma = 0.0$.

The last column in the signature shows the operator syntax, where # denotes the operator, and _ denotes an argument. It is a procedural language, in which the user step-by-step calls the operators to produce the desired output.

4. The proposed operator

As mentioned in the introduction section, we focus on the trajectory MOD flavor where the history of movement is stored for analysis and querying. A trajectory T of length m can be expressed as follows: $T = [(v_1, t_1), (v_2, t_2), \ldots, (v_m, t_m)]$, where each pair represents a time stamped value. We propose a TWED based similarity operator for evaluating the similarity between pairs of moving objects represented in the previous form and called it TWEDistance. It accepts a pair of moving objects, and yields the cost/distance of converting one of them into the other. The operator is carefully designed to allow the user defining the intended meaning of similarity through the parameters. Formally, let D_v denote the domain of a type v. Our TWEDistance operator is a mapping:

$$[D_{m(x)} \times D_{m(x)} \times \mathbb{R} \times \mathbb{R} \times ((D_x, D_x) \rightarrow \mathbb{R})] \rightarrow \mathbb{R}$$

Syntax wise the operator looks as follows:

$$\text{TWEDistance}(m(v), m(v), \gamma, \lambda, f : (v, v) \rightarrow \mathit{real}) \rightarrow \mathit{real}$$

where the first two parameters are the moving objects, γ, and λ are required by TWED, and the last parameter is a function mapping a pair of elements from the two series representing the moving objects into a real value. This last parameter is a user defined function that replaces the L_n-norm in TWED. For example, this function can be absolute difference $|v_1 - v_2|$. The values of γ, λ must be $\geqslant 0$. The operator yields a real value representing the distance.

The TWEDistance operator accepts moving objects of any type (e.g., *mpoint*, *mreal*, *mint*). Both of them must have the same type. These moving objects may represent the spatiotemporal route of cars, their speeds, their headings, etc. Thus, the user is able to pass his attributes of interest on which the similarity should be evaluated. The fifth parameter is a user defined function, that accepts a pair of instances, from the two input trajectories, and computes the cost of matching them. Assume, for instance, the two input trajectories are of type *mstring* (i.e., a string value that is changing over time). The matching function would accept a pair of strings and returns a real value. It might, for instance, return their edit distance.

Our proposed operator is generic and can be arbitrarily applied in different domains:

- It might, for instance, be applied to moving objects representing the prices of the stocks to cluster those that have similar trends.
- It might also be applied to Web server logs to spot the users that have similar navigation patterns.
- One can represent a soccer game as a trajectory, such that it stores the number of the player possessing the ball and the time it was passed to him. On such a dataset, one can identify the common pass/attack/defense patterns for a given team.

Algorithm 1 illustrates the evaluation of TWED-distance. It takes as an input two moving objects' trajectories divided into two sequences the spatial values and the time-stamp values, TWED's Stiffness and Lambda, and a user defined distance function. It returns as an output the TWEDistance that represents the sum of penalties of the minimal-cost sequence of editing operations needed to transform one trajectory into the other (TWED's implementation details are shown in Section 3.1). For sake of clarity, the technical detail of converting the Secondo representation of moving objects into time series is hidden.

```
Algorithm 1: TWEDistance Algorithm
input  :
    $R$, the sequence of values of the first object
    $S$, the sequence of values of the second object
    $t_r$, the sequence of time stamps for the first object
    $t_r$, the sequence of time stamps for the second object
    $d_{lp}$ the user defined distance function
    Lambda $\lambda$, Stiffness $\gamma$
output: the cost/distance between two moving objects
R$_{Size}$ ← Length($R$);
S$_{Size}$ ← Length($S$);
TWED [0..R$_{Size}$, 0..S$_{Size}$ ];
for $i$ ← 1 to S$_{Size}$ do
    TWED$[0, i]$ ← $\infty$;
for $j$ ← 1 to R$_{Size}$ do
    TWED$[j, 0]$ ← $\infty$;
TWED$[0, 0]$ ← 0;
for $i$ ← 1 to R$_{Size}$ do
    for $j$ ← 1 to S$_{Size}$ do
        Delete-R ← TWED [i-1,j ] + $d_{lp}$( $R[i-1]$ , $R[i]$) + $\gamma$ * ( $t_r[i]$ - $t_r[i-1]$ ) + $\lambda$ ;
        Delete-S ← TWED$[i, j-1]$ + $d_{lp}$( $S[j-1]$ , $S[j]$) + $\gamma$ * ( $t_s[j]$ - $t_s[j-1]$ ) + $\lambda$;
        Match ← TWED$[i-1, j-1]$ + $d_{lp}$( $R[i]$ , $S[j]$) + $\gamma$ * (|$t_r[i]$ - $t_s[j]$ |) ) + $d_{lp}$( $R[i-1]$ ,
          $S[j-1]$) + $\gamma$ * (|$t_r[i-1]$ - $t_s[j-1]$ |);
        TWED$[i, j]$ ← Minimum(Delete-R, Delete-S, Match);
return TWED[R$_{Size}$, S$_{Size}$];
```

Our operator is built on top of the moving objects model in [2,25], and is implemented within the MOD system Secondo. As a consequence, our operator leverages their query capabilities. Moving object data can be preprocessed (e.g., smoothed, re-sampled), existing functions can be used to compute arbitrary motion attributes on which the similarity is to be evaluated, and new functions can be extended. Therefore, this operator is generic enough that it can work with any moving types. New moving types and new operators (e.g., trend operator) will also fit, if they are introduced in the same model, and our operator can use them.

5. Experimental evaluation

We extended the Secondo kernel with a new Algebra and named it Similarity Algebra. It contains only the TWEDistance operator. The operator signature in Secondo is

$$\text{mapping} \times \text{mapping} \times \text{real} \times \text{real} \times \text{fun}((\text{basic} \cup \text{spatial}) \times (\text{basic} \cup \text{spatial})) \rightarrow \text{real}$$

This operator is also made available in the Secondo executable language. We list here some examples of calling it. Assume *obj1*, *obj2* are moving objects of type *mpoint* (e.g., representing a pair of vehicles):

```
let L1 = fun(x: real, y: real)
  abs(x - y);
let L2 = fun(x: point, y: point)
  sqrt(pow(getx(x) - getx(y), 2) +
  pow(gety(x) - gety(y), 2));
...TWEDistance(obj1, obj2, 1.0, 1.0, L2)
...TWEDistance(speed(obj1), speed(obj2), 0.0,
      1.0, L1)
...TWEDistance((2.0 * speed(obj1)) +
      (3.0 * direction(obj1)),
    (2.0 * speed(obj2)) +
      (3.0 * direction(obj2)),
    1.0, 1.0, L1)
```

In the syntax above, we start by defining function objects for the L_1-norm, and the L_2-norm. In the remaining lines, three different calls to the TWEDistance operator are illustrated, that can occur in the middle of a query.

Our TWEDistance operator was evaluated using set of experiments. The first three experiments are intended to evaluate how far the similarity values returned by the operator reflect the objects' similarity as expected in the experimental setting. The fourth experiment is dedicated to evaluate the scalability, and the last experiment evaluates the operator on a real dataset. We use two datasets: the synthetic dataset *berlintest*, which is contained in the standard installation of SECONDO, and the GEO-life real dataset [26–28].

Berlintest dataset contains the *Trains* relation. This relation was created by simulating the underground trains in Berlin. Simulation was based on the real train schedule and the real underground network of Berlin. The period of simulation is about 4 h in one day. It contains 9 lines identified by an ID (Line attribute) and group of trains travel per each line and identified also by an ID. Each train's trajectory is stored in an *mpoint* attribute. This relation contains 562 trajectories, consisting of a total of 54,595 observations.

Geo-Life is a GPS trajectory dataset that was collected in the Geolife project (Microsoft Research Asia) by 182 users in china in a period of over three years (from April 2007 to August 2012). It contains 17,621 trajectory with different sampling rates. Each trajectory is represented by a sequence of time-stamped points, each of which contains the information of latitude, longitude and altitude. We filtered the dataset to year 2008 that contains 7334 trajectory with 5,474,814 observations.

5.1. Similarity based on the direction

In this experiment, the *Trains* relation of the *berlintest* dataset was used. It contains trajectories of trains traveling at different lines. We selected the subset of trains that travel on *TrainsLine1* as follows.

```
let TrainsLine1=
  Trains feed filter[.Line = 1] consume;
```

This new relation has a total of 58 trajectories, from which 29 are heading on one direction, and the rest are heading on the opposite direction. This relation is then self-joined to produce a new relation *TrainPairs*, where every tuple contains a pair of trains:

```
let TrainPairs=
  TrainsLine1 feed addcounter[Cnt, 1] {a}
  TrainsLine1 feed addcounter[Cnt, 1] {b}
  symmjoin[.Cnt_a < ..Cnt_b]
  consume;
```

For each tuple, the similarity of its pair is computed based on the similarity of their heading. The goal of this experiment is to test the correctness of our implementation. A pair of trains going in the same direction, should be more similar than a pair of trains going on opposite directions. Moreover, a pair of trains on the same direction, that are close in terms of the trip start time, should also be more similar than a pair whose temporal difference is bigger. We run this experiment twice: once with the stiffness is set to zero, and once with the stiffness is set to one.

A zero stiffness allows for infinite temporal elasticity. In other words, the temporal difference does not affect the distance. So the distance is affected only by the differences in the heading. The query used in the first run of this experiment is as follows:

```
let DirDistNoTime=
  TrainPairs feed
  extend[Dist: TWEDistance (
    direction(.Trip_a),
    direction(.Trip_b),
    0.0,
    1.0,
    L1)]
  sortby[ID_a asc, Dist asc]
  consume;
```

Please cite this article in press as: Magdy N et al., A generic trajectory similarity operator in moving object databases, Egyptian Informatics J (2016), http://dx.doi.org/10.1016/j.eij.2016.07.001

The parameters passed to the TWEDistance operator are the two moving objects representing the headings of the two trains in the tuple, a stiffness value of zero, and Lambda of one. The last parameter is an L_1-norm for comparing a pair of heading values.

We analyzed the results of this query and they were as expected. For all trains, they were closer (have smaller distance values) to the ones traveling at the same direction than to those traveling in the opposite direction.

A similar query is used in the second run, but with a nonzero stiffness as follows:

```
let DirDistWithTime=
  TrainPairs feed
  extend[Dist: TWEDistance (
    direction(.Trip_a),
    direction(.Trip_b),
    (360.0 * 6.0),
    1.0,
    L1)]
    sortby[ID_a asc, Dist asc]
    consume;
```

Because this query has a nonzero stiffness, the results should be the following:

1. similar to the previous query: for every train, the trains in the same direction should appear before the trains in the opposite direction, and
2. for every train, the trains in the same direction should be sorted by how close their start times are to this train.

The results of the query showed that these expectations were met in 100% of the trains.

In the query above, a stiffness value of (360 * 6) was chosen in order to normalize the two scales of value difference and time difference. This value is chosen because the maximum difference in direction is 360°,[1] and the maximum time difference in this dataset is $\frac{1}{6}$ day. Clearly, setting the values of Lambda and stiffness requires knowledge of the dataset.

5.2. Similarity based on the spatial proximity

In this experiment we again evaluate the correctness of the proposed operator and our implementation. We experiment with a meaning of similarity, which is different from the previous experiment, the spatial proximity. That is, trajectories have smaller distances if they traverse similar paths. In the experiment's run, we generate exact replicas of some trains, and test whether the replicas are closer to their original trajectories than other trains. We add a bounded random disturbance to the spatial path of the replicas, and evaluate whether the results remain the same.

We generated a new relation, *DisturbedNewTrains*, that contains data of 36 train where one train was picked from each line. Each of these trains was replicated 3 times, adding a bounded random disturbance to the trajectory of every replica, with different upper bounds. The disturbance is in the form of random spatial translations within some upper bound, applied to every observation in the trajectory.

We then self-joined the *DisturbedNewTrains* relation, so that every tuple will contain a pair of trains. The main query of this experiment applied the TWEDistance operator to every pair of trains to measure their similarity based on their spatial route. The goal is to check whether the replicas appear similar to their original, than other trains.

In this query, we used zero stiffness as we focus on spatial proximity. We used Euclidean distance as a distance function, and evaluated results based on different disturbance values. We are not going to show queries to keep the discussion focused. The results were as expected, with zero disturbance, replicas are the closest trains to its original one and with increased disturbance, the results' accuracy decreased as shown in Fig. 4, where the disturbance axis shows the upper bound in meters.

5.3. Similarity based on the spatiotemporal proximity

In this experiment, it is similar to the one above, except that it takes time into account. We used the *TrainPairs* relation that was used in the direction similarity experiment. The TWEDistance was measured on the pairs of trains. The stiffness value is again chosen to normalize the scales. The used Query is as follows:

```
query TrainPairs
  feed
  extend[Dist:
    TWEDistance(.Trip_a,.Trip_b,
      (15783.0 * 6.0), 1.0, L2)]
  consume;
```

The results again came as expected. For every train, the closest trains were those that go on the same line and direction, in a sorted manner by how close their start times are to that train.

5.4. Scalability

In this experiment, the operator scalability is evaluated by applying it to sequences of multiple lengths. We start by two random trajectories from the trains relation. These trajectories

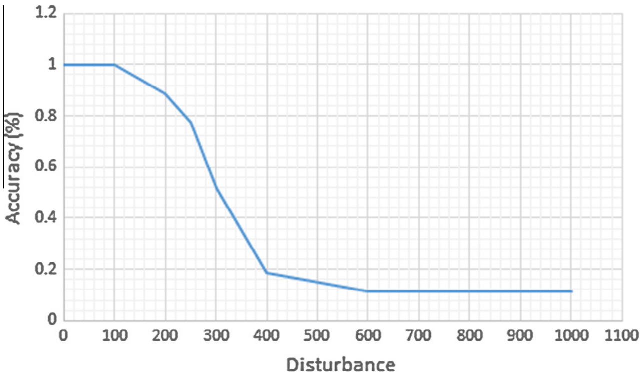


Fig. 4 Accuracy with respect to disturbance.

[1] To be precise, it is 359.

are then re-sampled to generate the designated sequence lengths ranging from 6 to 10,300 samples per every trajectory.

The Complexity of TWEDistance is O($n^{*}m$) where n and m are the lengths of the two input trajectories. Note that in this experiment $m = n$. Fig. 5 shows run time curve as a variable in the sequence length. As shown, the curve shows a quadratic relationship as expected.

5.5. Detecting the transportation mode

In this last experiment, we run a classification task on a real dataset, to assign transportation modes for GPS trajectories. The Geo-Life dataset, used in this experiment, has many transportation modes including car, bus, train, bike, walk. This experiment is applied on 2390 trajectories, all of which were observed in year 2008. These modes were grouped into two classes: driving and non-driving. The driving class included the modes car, bus, train, subway and taxi. The non-driving class included walk and bike.

The classification was done using the leave-one-out method, to each trajectory in turn. That is, the class label of the chosen trajectory is predicted to be the class label of its k-nearest neighbor, based on the given distance (TWEDistance Operator result). The head K-nearest trajectories are fetched and the predicted class label is the dominant class from the k trajectories. If the prediction is correct, then it is a hit; otherwise, it is a miss. The error rate that was measured is the ratio of misses to the total number of trajectories.

The similarity was evaluated based on the movement speed and different runs were made using different *Lambda* values (i.e. [1.0, 5.0, 10.0, 15.0, 20.0, 40.0 and 50.0]) and different k values (i.e. [1–6]). The following query was used with different Lambda values:

```
let speedDistNoTime =
sampleTest feed addcounter[Cnt,l] {a}
sampleTest feed addcounter[Cnt,l] {b}
symmjoin[.Cnt_a < ..Cnt_b]
extend[Dist: TWEDistance(speed(.Trip_a),
speed(.Trip_b), 0.0, 1.0,
fun(x:real,y:real) abs(x - y)),
Speed_a: mreal2Lmreal(speed(.Trip_a)),
Speed_b: mreal2Lmreal(speed(.Trip_b))]
consume;
```

Fig. 6, shows the error rate obtained with different *Lambda* and K values. It shows that as Lambda value increases, the error rate decreases and we get better separation of the curves (i.e. trajectories of a certain mode gets closer to others in the same mode by increasing *Lambda* value).

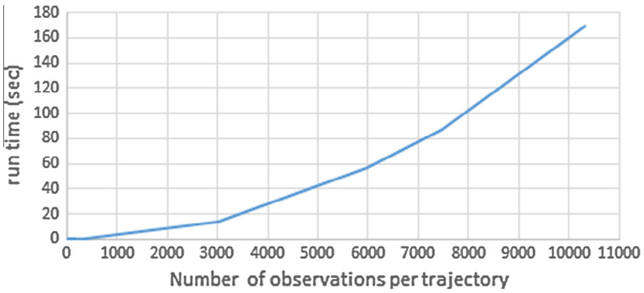


Fig. 5 Scalability results.

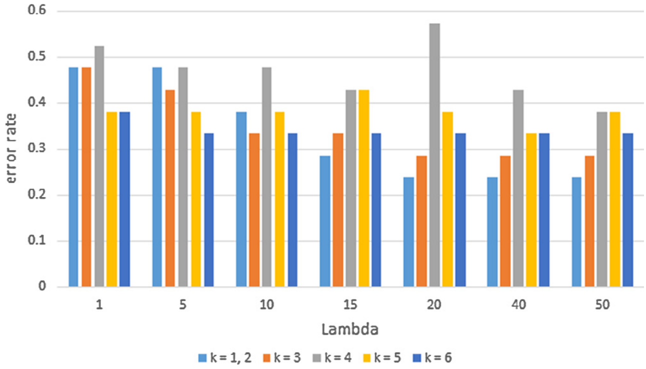


Fig. 6 Classification error rate with respect to k and *Lambda*.

6. Conclusion

As mentioned before, there are variety of similarity measures that define implicitly in their implementation the meaning of similarity regardless of user perspective. That was a research gap in the field of trajectory similarity. So, we proposed a parameterized TWED based similarity operator in secondo (one of the MODs) and named it TWEDistance. It takes a list of parameters which are two moving objects similarity meaning representation, TWED Stiffness, TWED Lambda and a user defined distance function respectively. We proved efficiency of our operator through set of experiments on real and synthetic datasets. Different representations of similarity meanings such as speed, direction and raw route data were passed while considering time once and ignoring another. This operator is based on the discretized form of a trajectory, while the nature of a trajectory is its continuity as mentioned earlier in the paper. So one of the future work is the need for a similarity measure and an operator for evaluating similarity based on the continuous form of a trajectory.

References

[1] Wolfson O, Xu B, Chamberlain S, Jiang L. Moving objects databases: issues and solutions. In: Scientic and statistical database management, 1998. Proceedings. Tenth international conference on, IEEE 1998. p. 111–22.

[2] Güting RH, Böhlen MH, Erwig M, Jensen CS, Lorentzos NA, Schneider M, et al. A foundation for representing and querying moving objects. ACM Trans Database Syst (TODS) 2000;25(1):1–42.

[3] Andrienko N, Andrienko G, Pelekis N, Spaccapietra S. Basic concepts of movement data. In: Mobility, data mining and privacy. Springer; 2008. p. 15–38.

[4] Ranacher P, Tzavella K. How to compare movement? A review of physical movement similarity measures in geographic information science and beyond. Cartogr Geogr Inform Sci 2014;41(3):286–307.

[5] Gunopulos D, Trajcevski G. Similarity in (spatial, temporal and) spatio-temporal datasets. In: Proceedings of the 15th international conference on extending database technology. ACM; 2012. p. 554–7.

[6] Wang H, Su H, Zheng K, Sadiq S, Zhou X. An effectiveness study on trajectory similarity measures. Proceedings of the twenty-fourth Australasian database conference, vol. 137. Australian Computer Society, Inc.; 2013. p. 13–22.

[7] Faloutsos C, Ranganathan M, Manolopoulos Y. Fast subsequence matching in time-series databases, vol. 23. ACM; 1994.

Please cite this article in press as: Magdy N et al., A generic trajectory similarity operator in moving object databases, Egyptian Informatics J (2016), http://dx.doi.org/10.1016/j.eij.2016.07.001

[8] Chen Y, Nascimento MA, Ooi BC, Tung AK. Spade: on shape-based pattern detection in streaming time series. In: 2007 IEEE 23rd international conference on data engineering, IEEE 2007. p. 786–95.

[9] Nakamura T, Taki K, Nomiya H, Seki K, Uehara K. A shape-based similarity measure for time series data with ensemble learning. Pattern Anal Appl 2013;16(4):535–48.

[10] Alt H. The computational geometry of comparing shapes. In: Efficient algorithms. Springer; 2009. p. 235–48.

[11] Alt H, Behrends B, Blömer J. Approximate matching of polygonal shapes. Ann Math Artif Intell 1995;13(3–4):251–65.

[12] Chen L, Özsu MT, Oria V. Symbolic representation and retrieval of moving object trajectories. In: Proceedings of the 6th ACM SIGMM international workshop on Multimedia information retrieval. ACM; 2004. p. 227–34.

[13] Li JZ, Özsu TM, Szafron D. Modeling of moving objects in a video database. In: Proceedings IEEE international conference on multimedia computing and systems' 97. IEEE; 1997. p. 336–43.

[14] Keogh E, Ratanamahatana CA. Exact indexing of dynamic time warping. Knowl Inform Syst 2005;7(3):358–86.

[15] Chen L, Ng R. On the marriage of lp-norms and edit distance. Proceedings of the thirtieth international conference on very large data bases, vol. 30. VLDB Endowment; 2004. p. 792–803.

[16] Marteau P-F. Time warp edit distance with stiffness adjustment for time series matching. IEEE Trans Pattern Anal Mach Intell 2009;31(2):306–18.

[17] Vlachos M, Kollios G, Gunopulos D. Discovering similar multidimensional trajectories. In: Data engineering, 2002. Proceedings. 18th international conference on, IEEE 2002. p. 673–84.

[18] Chen L, Ozsu MT, Oria V. Robust and fast similarity search for moving object trajectories. In: Proceedings of the 2005 ACM SIGMOD international conference on management of data, ACM 2005. p. 491–502.

[19] Little JJ, Gu Z. Video retrieval by spatial and temporal structure of trajectories. In: Photonics west 2001-electronic imaging, international society for optics and photonics. p. 545–52.

[20] Hodgson S, Quinn TP, Hilborn R, Francis RC, Rogers DE. Marine and freshwater climatic factors affecting interannual variation in the timing of return migration to fresh water of sockeye salmon (oncorhynchus nerka). Fish Oceanogr 2006;15(1):1–24.

[21] Kjellén N. Differential timing of autumn migration between sex and age groups in raptors at Falsterbo, Sweden. Ornis Scand 1992:420–34.

[22] Allen JF. Maintaining knowledge about temporal intervals. Commun ACM 1983;26(11):832–43.

[23] Ueta M, Sato F, Nakagawa H, Mita N. Migration routes and differences of migration schedule between adult and young Steller's Sea Eagles Haliaeetus Pelagicus. Ibis 2000;142(1):35–9.

[24] Güting RH, Behr T, Düntgen C. Secondo: a platform for moving objects database research and for publishing and integrating research implementations. Fernuniv., Fak. für Mathematik u. Informatik; 2010.

[25] Forlizzi L, Güting RH, Nardelli E, Schneider M. A data model and data structures for moving objects databases, vol. 29. ACM; 2000.

[26] Zheng Y, Zhang L, Xie X, Ma W-Y. Mining interesting locations and travel sequences from GPS trajectories. In: Proceedings of the 18th international conference on World wide web. ACM; 2009. p. 791–800.

[27] Zheng Y, Li Q, Chen Y, Xie X, Ma W-Y. Understanding mobility based on GPS data. In: Proceedings of the 10th international conference on Ubiquitous computing. ACM; 2008. p. 312–21.

[28] Zheng Y, Xie X, Ma W-Y. Geolife: a collaborative social networking service among user. IEEE Data Eng Bull 2010;33(2):32–9.