Minimal Spatio-Temporal Database Repairs

Markus Mauder[1], Markus Reisinger[1], Tobias Emrich[1], Andreas Züfle[1], Matthias Renz[1], Goce Trajcevski[2], and Roberto Tamassia[3]

[1] Ludwig-Maximilians-Universität München
reisinger, mauder, mauder, zuefle, renz@dbs.ifi.lmu.de,
[2] Northwestern University
goce@eecs.northwestern.edu
[3] Brown University
rt@cs.brown.edu

Abstract. This work addresses the problem of efficient detection and fixing of inconsistencies in Spatio-Temporal Databases. In contrast to traditional database settings where integrity constraints pertain to explicitly stored values (and, possibly, values defined via views and aggregates), we observe that spatio-temporal data may exhibit a specific types of violations to constraints that cannot be explicitly tied with the stored values. The main reason is that spatio-temporal phenomena are continuous, but their database representations are discrete. Thus, the constraints are semantic in nature, as opposed to being firmly dependent on the actual stored data. We give a general definition of semantic constraints of a trajectory database, and define rules to repair violations of these constraints. We aim at minimizing the changes needed for repairing violations of such semantic constraints, in order do minimize the distortion of the state of the database induced by the repairs. Towards this, we define dissimilarity between the initial database and its repaired state and, to minimize this dissimilarity, we propose several rules of space- and time-distortion, which shift inconsistent observations in space and time to remove inconsistencies. Our evaluation shows that these simple rules often run into local minima, and thus may not be able to repair a database. To remedy this problem, we propose a hybrid approach which may choose between possible space and time distortions. We show that a greedy approach which always chooses the locally best repair may still run into local minima and propose a Simulated-Annealing approach, which combines greedy and random repairs to avoid these local minima.

1 Introduction

By the end of 2014, there were nearly 7 billion mobile subscriptions worldwide [1]. This fact, along with miniaturization of computing and sensing devices and GPS and RFID technologies, has provided a foundation for generating extremely large volumes of location-in-time data: petabytes of location-based (i.e., spatio-temporal) data are generated every day [11]. The management of (*location*, *time*) information about mobile entities is essential for a variety of application domains, ranging from navigation and efficient traffic management to emergency/disaster rescue management, environmental monitoring, fly-through visualization, and various military applications (e.g., radar data, troops tracking) [14]. Essentially, every application requiring some form of Location Based Services (LBS) [16] needs efficient techniques for storage, retrieval and query processing of spatio-temporal data—topics studied in the field of Moving Objects Databases (MOD) [10].

Physical factors, such as the imprecision of sensing devices and communication links, often cause the location data to be inaccurate and noisy. In addition to this problem—even with perfect sampling accuracy—the data intended to capture a continuous motion can be represented only at discrete time-instances. Moreover, data records can be obsolete as users may update their location infrequently, e.g., due to bad connectivity or to preserve battery power. Thus, one has to cater to the uncertainty as a natural factor when considering the representation of spatio-temporal data (cf. [6]). A complementary observation is that data sources may be various heterogeneous devices: roadside-sensors, weather stations, satellite imagery, (mobile) weather radar, crowd sourced observations, ground and aerial LIDAR—to name but a few. Having multiple sources may yield not only cause type-mismatch issue, but also generate conflicting location information about the same object and cause problems in reconciling the data [18]. Complementary to uncertainty, the above contexts may cause other types of semantic inconsistencies that have not been addressed so far. Namely, a user posing a continuous k-Nearest Neighbor (k-NN) query, may be presented with an answer containing two (or more) vehicles that "have collided." This is a simple example of violating the following basic semantic constraint: "two objects cannot be at the same place at the same time." Such a violation may be due to imprecise location-samples. Also, it often arises from the use of interpolation (linear, Bezier, etc.) in-between observed samples [9].

The main objective this work is to provide techniques for detection of and, just as importantly, "fixing" such inconsistencies. We note that the focus of this paper is not on removing the inherent uncertainty, which is a consequence of the imprecision of location detections and can be an additional cause for inconsistencies. Rather, we aim at repairing the "symptoms" of the "inevitable uncertainty". As a simple example, the interpolation of GPS signals may lead to the consequence of having a trajectory of a given car going through a lake. Fixing this problem by having the trajectory going around the lake may still yield the wrong trajectory, as the true trajectory may look different. Hence, while we cannot claim to have alleviated the root-causes for errors in spatio-temporal databases, we take a first step towards fixing the symptoms based on semantic constraints. Clearly, a method for database repairs should aim at minimizing the distortion between the original database and the repaired database. The main contribution of this work can be summarized as follows:

– We identify and formalize the problem of semantic inconsistencies in spatio-temporal data. This formalization identifies a wide class of problems, that have been largely neglected in the moving object and trajectory database literature.
– Since the problem of finding an optimal database repair is NP-hard, we propose a number of heuristics to repair a spatio-temporal database, which are organized into three general categories of solution, including time-distortion, space-distortion and hybrid approaches.
– We present experimental observations quantifying different trade-offs among the proposed methods.

The rest of this paper is organized as follows. Section 2 presents a review related work. In Section 3, we formalize the problem of Moving Object Database (MOD) inconsistencies along with metrics to measure the quality of a database repair. Section 5 gives the details of the proposed algorithmic solutions and the experimental observations are presented in Section 6. Finally, in Section 7, we conclude the paper and outline directions for future work.

2 Related Work

We now overview the literature on several different topics related to the problems addressed in this paper. However, as we will argue, although each body of work has yielded interesting and relevant results, none has addressed the specific problems tackled by our work, nor has provided a readily applicable "tool-chain."
Relational Database Repairs: Traditional database approaches *repair* [3, 4, 17] the identified inconsistencies by removing objects or by changing attribute values. Such approaches however, can not be applied directly to spatio-temporal data. Arbitrarily changing a (location, time) pair is likely to yield new inconsistencies, as the changed trajectory may reach an unreachable state, or may have an unrealistically high speed in the repaired version of the database. The main challenge in spatio-temporal data is to incorporate repair rules to span a space of semantically meaningful repairs.
Probabilistic Spatio-Temporal Database Repairs: The recently published approach of [12] aims at repairing probabilistic spatio-temporal databases as defined in [13]. In this setting, each mobile object is assigned a set of spatial regions and a probability interval defining the likelihood to be within this region. In an interpretation of such a database, the probability of a region must be within its interval and the probabilities of all regions of an object must sum up to one. Such a database is inconsistent if no interpretation exists. The approach of [12] shows how to minimally change probability intervals in order to obtain an interpretation. The problem setting in this work can not be extended to trajectory databases.

A recent approach presented in [8] models the motion of a spatio-temporal object by a stochastic process, such that each possible world is indeed associated with a probability. Constraints such as "Object x must not be in state s at time t" can be incorporated into this model by adapting the corresponding probabilities. More complex constraints, such as inter-object constraints that prohibit objects from being at in same state at the same time, can not be incorporated into such models as easily.
Linear Temporal Logic: Our problem of removing inconsistencies from a trajectory database can be cast in the realm of temporal logic. For instance, using Propositional Linear Temporal Logic (LTL) [7], a trajectory $T = s_1, s_2, ..., s_{|T|}$ can be described using the *eventually* operator $\diamond$ by $\diamond s_1 \diamond s_2 ... \diamond s_{|T|}$. Semantically, this LTL formulation induces a trajectory where eventually state s_1 must be visited after any number of intermediate states, then s_2 must eventually be visited after possible more intermediate states and so on. Further constraints can be formulated, e.g. to constrain the database such that no two objects may be at the same location at the same time, by applying the always operator $\Box$ to express the the rule $\forall T_1, T_2 \in \mathcal{D}, t \in T : \Box T_1(t) \neq T_2(t)$. Using

logical solvers for LTL [15], we can efficiently find an interpretation[4] for each trajectory such that all constraints are satisfied, if any such interpretation exists. While LTL allows to formulate any semantic constraint, the main problem of LTL is that, being a logic rather than a function, it does not allow to find any optimal solution. Thus, LTL allows to check if there exists a model that satisfies all given constraints, which any way of formulating a cost function that can be optimized. In most applications, the problem of finding such a model is trivia. For example, the solution of using a *serial schedule*, which avoids any inconsistency between objects by simple removing any temporal overlap between trajectories, does always work.

While the solution based on serially scheduling each trajectory is valid, it is prohibitively expensive, since "repaired" trajectories may be extensively distorted in time. We are looking for a solution which minimizes the changes to the database performed by the repair.

3 Problem Definition

This section presents the details of the novel types of inconsistencies and desirable properties of (methodologies for) enforcing the semantic constraints in a given MOD.

A spatio-temporal database $\mathcal{D}^{ST}$ stores triples (*oid*, *location*, *time*), where $oid \in \{o_1, ..., o_N\}$ is a unique object identifier, *location* $\in \mathcal{S}$ is a spatial position in space and *time* $\in \mathcal{T}$ is a point in time. Semantically, each such triple corresponds to the location of object o_i at some time. In $\mathcal{D}$, an object can be described by a function $tr_{o_i} : \mathcal{T} \to \mathcal{S}$ that maps each point in time to a location in space[5] $\mathcal{S}$; this function is called *trajectory*. The corresponding trajectory database is denoted as $\mathcal{D} = \{tr_{o_1}, ..., tr_{o_N}\}$.

Assuming that the location of an object o_i is known for any point in time is unrealistic as the location of object o_i can only be determined at discrete time-instants. The frequency of location-samplings is also bounded by physical constraints, such as the availability of a GPS signal. Between discrete observations, the position of a moving object has to be estimated via some type of a *dead reckoning*. These estimations are based on incomplete information, and thus, may be imprecise.

3.1 Spatio-Temporal Constraints

The violation of a constraint in a trajectory database $\mathcal{D}$ indicates that $\mathcal{D}$ contains erroneous trajectories, possibly incurred due to faulty dead reckoning, or due to deficiency of measuring devices used to capture trajectories. Since we are considering historical data, we lack the option of improving the available information, e.g., by requesting the objects to give a more accurate position update. Since the cause for the inconsistency cannot be removed, the only viable approach is to repair the trajectories in order to mitigate the symptoms of this lack of information.

[4] In LTL, an interpretation is a Kripke structure which, in our case, maps each trajectory and each point in time to a state.

[5] Most often the Euclidian 2D space is considered, however, extension to 3 (or higher) dimensions as well as road-network constraints have been commonly considered in the literature.

Definition 1. *Let $\mathcal{C}$ be a set of constraints. A database $\mathcal{D}$ is said to satisfy $\mathcal{C}$, noted as $\mathcal{D} \vDash \mathcal{C}$, if all constraints are satisfied in $\mathcal{D}$. If $\mathcal{D} \nvDash \mathcal{C}$, then $\mathcal{D}$ is said to be inconsistent.*

Loosely speaking, a *spatio-temporal constraint* can be thought of as any rule describing some semantic constraint related to the trajectories in $\mathcal{D}$. A constraint $c \in \mathcal{C}$ may pertain to an individual object. An example of such an *Object Constraint* is the constraint "An object must not enter a specified area R on Sunday between 2am and 5am." This constraint can be formally expressed as

$$\forall(tr_o \in \mathcal{D}), \forall(t \in [\text{Sunday 2am}, \text{Sunday 5am}]) : tr_o(t) \notin R.$$

In contrast, some constraints may be defined between trajectories, such as "two objects must not be in the same place at the same time" which can be expressed as

$$\forall(tr_{o_i}, tr_{o_j}, i \neq j), \forall t : tr_{o_i}(t) \neq tr_{o_j}(t).$$

In practice, constraints involving more than one object lead to hard optimization problems, as a single repair of one trajectory may have a large number of consequences on the constraints involving other objects. Section 4 will show that such constraints lead to NP-hard optimization problems. Since we are considering the general case, we will be considering such hard inter-object constraints in our experimental evaluation in Section 6.

3.2 Database Repair Rules

In this work, we will propose a number of trajectory database repair rules. These rules define for a given trajectory $T \in \mathcal{D}$ the set of possible repairs $\overline{\mathcal{T}^{\mathcal{R}}}$. Before we propose these rules in Section 5, we first formally define the purpose of a repair rule.

Definition 2 (Trajectory Database Repair Rule). *Let $\overline{\mathcal{D}}$ denote the set of all possible trajectory databases. A trajectory database repair rule $R : \overline{\mathcal{D}} \mapsto \overline{\mathcal{D}}^*$ is a function, which maps a trajectory database $\mathcal{D}$ to a set of possible repairs.*

As an example, a repair rule R may allow to simply remove a trajectory from $\mathcal{D}$. This repair rule can be be specified by

$$R(\mathcal{D}) = \{\mathcal{D}' | \mathcal{D}' \subset \mathcal{D}\}.$$

Definition 3 (Database Repair). *Let $\mathcal{D}$ be a trajectory database inconsistent with respect to a set of semantic constraints $\mathcal{C}$ and let $\mathcal{R}$ be a set of repair rules. Let $\mathcal{D}^R \in \mathcal{R}^*(\mathcal{D})$ be a trajectory database derived by iteratively applying repair rules $R \in \mathcal{R}$ to $\mathcal{D}$. If $\mathcal{D}^R \models \mathcal{C}$ holds, then the trajectory database $\mathcal{D}^R$ is called a* database repair *of $\mathcal{D}$.*

In many cases, such as the aforementioned exemplary repair rule that allows to discard trajectories, one trivial way of obtaining a database repair $\mathcal{D}^R$ which satisfies all given constraints $c \in C$ is, for example, the empty database $\mathcal{D}^R = \{\}$. Given the lack of any actual trajectory, it trivially satisfies many constraints. Hence, strictly speaking, the challenge is not only to find just any database repair, but to find a database repair having the *minimal difference* from the initial database $\mathcal{D}$.

Definition 4 (Minimal Database Repair). *Let $\mathcal{D}$ be a trajectory database inconsistent with respect to a set of semantic constraints C. Let $dist(\mathcal{D}, \mathcal{D}^R)$ be a dissimilarity function between databases. A minimal repair $\mathcal{D}^R_{min}$ is defined as*

$$\mathcal{D}^R_{min} = argMin_{\mathcal{D}^R \in \overline{\mathcal{D}^{\mathcal{R}}}, \mathcal{D} \models C} dist(\mathcal{D}, \mathcal{D}^R),$$

where $\overline{\mathcal{D}^{\mathcal{R}}}$ represents the set of all possible repairs of $\mathcal{D}$.

The goal of this work is to efficiently compute, for a given trajectory database $\mathcal{D}$ and a set of semantic constraints C, a minimal repair $\mathcal{D}^R_{min}$ of $\mathcal{D}$. This problem falls into the class of constraint satisfaction problems and we show in Section 4 that it is NP-hard. We will relax the problem to find heuristic solutions that yield a database repair having sufficiently low dissimilarity to the initial database.

3.3 Quality of a Repair

To measure the quality of a repair, a dissimilarity function $dist(\mathcal{D}, \mathcal{D}^R)$ is needed which, in accordance with Definition 4, will be minimized. Thus, this function defines semantic of a "good" database repair. Semantically, a good database repair is expected to minimize the total number of changes of the database $\mathcal{D}$, and should guarantee that changes are divided fairly over all trajectories. To measure the total dissimilarity between $\mathcal{D}$ and $\mathcal{D}^R$, we can simply aggregate the dissimilarity of individual trajectories:

$$dist(\mathcal{D}, \mathcal{D}^R) = \sum_{T \in \mathcal{D}} dist(T, T^R),$$

where $dist(T, T^R)$ is a dissimilarity function defined on trajectories such as average Euclidean-distance or edit distance. In addition, changes in $\mathcal{D}^R$ should be divided fairly among trajectories, in order to avoid starvation of single trajectories in the repaired database. Such fairness can be enforced as follows

$$dist(\mathcal{D}, \mathcal{D}^R) = \sum_{T \in \mathcal{D}} g(dist(T, T^R)),$$

where $g(x)$ is a function that monotonically increases in $\mathbb{R}^+$, such as the square function, to weight the distances of individual trajectories.

In the remainder of this work, we propose solutions to remove inconsistencies from a trajectory database $\mathcal{D}$. For this purpose, we first provide a formal proof that the NP-hardness of fixing inconsistencies in a trajectory database in Section 4. Thus, heuristic solutions are presented in Section 5. In Section 6, we perform an experimental analysis of the quality of these solutions on real data sets, evaluating both run-time and quality of the resulting repair.

4 Complexity Analysis

In the following, we show that the problem of finding the optimal repair $\mathcal{D}^R$ of a trajectory database $\mathcal{D}$ is generally hard. For this purpose, we show that the simpler problem of finding *any* repair is already NP-complete.

Lemma 1. *Given a trajectory database $\mathcal{D}$, a set of constraints C and a set of repair actions A, the problem of deciding whether there exists a repair $\mathcal{D}^R$ which is derived from $\mathcal{D}$ using rules in A, such that $\mathcal{D}^R \models C$ is NP-complete.*

Proof. Let $\mathcal{D}$ be a database of arbitrary trajectories, and let A be repair action such that for each trajectory $T_i \in \mathcal{D}$ there exists exactly one possible repair. For each $T_i \in \mathcal{D}$, let p_i denote the unrepaired trajectory T_i, and let $\not p_i$ denote the repaired trajectory which is derived by applying the only possible repair in A to T_i. Furthermore, let C be a set of inter-object constraints such that each constraint $c_{s,t} \in C$ requires that at least one object must be in state s at time t. Let $c_{s,t}(\mathcal{D}, A) \subseteq \bigcap_{1 \leq i \leq N}\{p_i, \not p_i\}$ denote the set of all possible trajectories that satisfy constraint $c_{s,t}$, i.e., all possible trajectories that are located in state s at time t. Since each constraint $s_{s,t}$ requires at least one trajectory to be in state s at time t, the constraint $s_{s,t}$ can be rewritten as the disjunction of all trajectories satisfying this constraint:

$$c_{s,t} = \bigvee_{p \in c_{s,t}(\mathcal{D},A)} p.$$

This boolean formula returns true if and only if the constraint $c_{s,t}$ is satisfied. For all constraints to be satisfied, the conjunction of all these disjunctions yields the following boolean formula:

$$\bigwedge_{c_{s,t} \in C} \bigvee_{p \in c_{s,t}(\mathcal{D},A)} p.$$

This formula returns true, if and only if, for a given database repair $\mathcal{D}^R \in \{p_1, \not p_1\} \times \{p_N, \not p_N\}$ satisfies all constraints in C. Consequently, the problem of finding a valid repair of $\mathcal{D}$ is equivalent to the satisfiability problem of the above boolean formula. This satisfiability problem, known as *k-SAT*, is known to be NP-complete. □

Due to the hard nature of the problem, will omit an exact algorithm to find an optimal database repair, i.e., a repair that minimizes the amount of database distortion. It should be noted that such an algorithm can be specified using integer linear programming, yet such a solution may have unbearable run-times even for toy databases. Instead, in the next section, we will propose approximate algorithms, which return a database repair $\mathcal{D}^R$ which may not be minimal in terms of distortion of the original database $\mathcal{D}$, or which may fail to satisfy some constraints.

5 Algorithms

Before discussing our algorithmic solutions for spatio-temporal database repairs in Section 5.2, we specify the following components in Section 5.1:

1. Spatio-temporal constraints and techniques for their detection
2. Allowed repair rules
3. Dissimilarity function to measure the quality of a database repair

5.1 Component Specifications

Spatio-Temporal Constraints There are several alternatives for spatio-temporal constraints. In this paper, we consider the following very general constraint: “Two objects must not be within a threshold of ε meters within each other at any time.” This constraint is formally expressed as follows:

$$\forall(tr_{o_i}, tr_{o_j}, i \neq j), \forall t : dist(tr_{o_i}(t), tr_{o_j}(t)) > \varepsilon.$$

This constraint is able to ensure that objects with a spatial extent of ε never occupy the same space at the same time, or that objects do not get too close to each other.

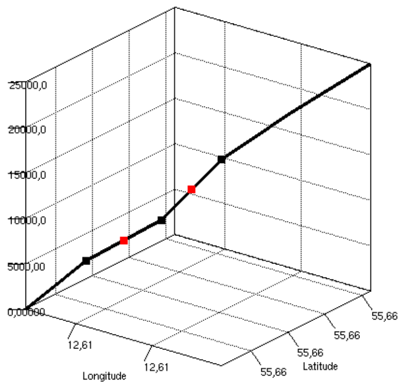
As the next step, it is important to be able to quickly find violations of the above constraint in the database. To detect these violations, we use a spatio-temporal R^*-tree to index the set $\mathcal{S}$ of all trajectory segments defined by two successive GPS signals of the same object, using time as a third dimension. Each trajectory segment s is minimally bounded by a rectangle $\Box(s)$ and added to the tree. Thus, each leaf of this R^*-tree is a single rectangle pointing to the exact representation of the approximated trajectory segment. To find all initial collisions, we perform a similarity-join [5] joining the indexed database with itself (ignoring identity) and using ε as the similarity threshold that is only applied on the spatial dimensions (and not on the time). The result is a set of intersection pairs (s, c) where s and c are segments of two different trajectories.

Once the initial collisions have been found, future collisions caused by database repairs can be found very efficiently, by querying against the tree only segments that have been changed by a repair.

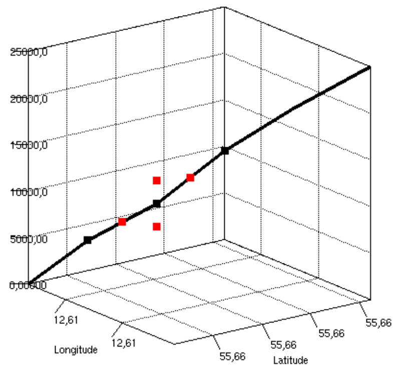
Repair Rules In our problem setting (Section 3), a database repair is still unspecified. In the following, we focus on the manipulation of the vertices of the trajectories in order to obtain a countable number of possible repairs. To identify the vertex to be repaired to remedy a constraint violation, we always consider the vertex closest to the violation point of both corresponding trajectories. The vertex can then be manipulated in one of the following ways:

- *Time domain:* The manipulation of a vertex v back in time implies that the movement from the previous vertex to v is slowed down and the movement from v to its subsequent vertex is sped up. The manipulation of v forward in time has the opposite effect. Note that the time manipulation of a vertex is constrained by its predecessor and its successor. Manipulating the time of v beyond the times of its predecessor or its successor yields anomalous movement in the spatial domain.
- *Spatial domain:* Manipulating the spatial position of a trajectory vertex has also impact on the speed of the movement.
- *Time and spatial domains*: Obviously, the spatial and temporal manipulation can be combined. A special case of spatio-temporal manipulation is the manipulation of v along the spatio-temporal path to its predecessor or its successor.

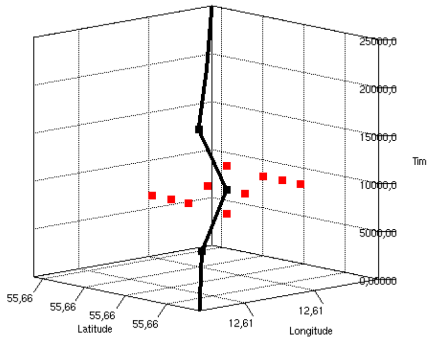
Based on these observations, we define the following three rules named after the cardinality of the set provided. Throughout this section, the input to a rule is the repair triple v_p, v, v_f, where v is the vertex to repair, v_p is the predecessor of v, and v_f is the



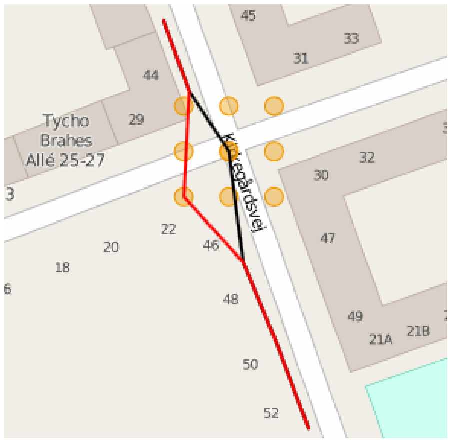
(a) Two-Rule



(b) Four-Rule



(c) Ten-Rule



(d) Ten-Rule

Fig. 1. Repair Rules

successor of v in the trajectory. Furthermore, a vertex v is characterized by the triple $(v.t, v.x, v.y)$ representing the time, the x position and the y position of v, respectively.

Definition 5 (Two-Rule). *Given repair triple v_p, v, v_f, the Two-Rule returns two vertices v'_1 and v'_2, where*

$$\begin{aligned} v'_1 &= \frac{v_p + v}{2} \\ v'_2 &= \frac{v_f + v}{2} \end{aligned} \tag{1}$$

Note that the two vertices returned by the Two-Rule are located in time and space half the way forward and backward around vertex v.

The Four-Rule adds temporal repairs.

Definition 6 (Four-Rule). *Given repair triple v_p, v, v_f and time distortion Δt, the Four-Rule returns the two vertices returned by the Two-Rule plus the two vertices v'_3 and v'_4, where*

$$\begin{aligned} v'_3 &= (v.t - \Delta t, v.x, v.y) \\ v'_4 &= (v.t + \Delta t, v.x, v.y) \end{aligned} \tag{2}$$

ensuring that $v.t - \Delta t \geq v_p.t$ and $v.t + \Delta t \leq v_p.t$

The Ten-Rule adds eight absolute spatial distortions.

Definition 7 (Ten-Rule). *Given repair triple v_p, v, v_f, time distortion Δt, and space distortion Δs, the Ten-Rule returns the following ten vertices:*

$$\begin{aligned} v'_3 = (v.t - \Delta t, v.x, v.y), v'_4 = (v.t + \Delta t, v.x, v.y), \\ v'_5 = (v.t, v.x - \Delta s, v.y), v'_6 = (v.t, v.x + \Delta s, v.y), \\ v'_7 = (v.t, v.x, v.y - \Delta s), v'_8 = (v.t, v.x, v.y + \Delta s), \\ v'_9 = (v.t, v.x - \Delta s, v.y - \Delta s), v'_{10} = (v.t, v.x + \Delta s, v.y + \Delta s), \\ v'_{11} = (v.t, v.x - \Delta s, v.y + \Delta s), v'_{12} = (v.t, v.x + \Delta s, v.y - \Delta s) \end{aligned} \tag{3}$$

ensuring that $v.t - \Delta t \geq v_p.t$ and $v.t + \Delta t \leq v_p.t$

Figure 1 gives an overview of these three rules, where we show in 1(a) the shift on the segment and we show in 1(b) the additional time shift. Figures 1(c) and 1(d) show all ten options, while the Southwest option was chosen in Figure 1(d). The effectiveness of these rules will be evaluated later on, but obviously the Ten-Rule should perform best, as it offers the most possibilities and so the algorithms will typically choose one of the ten vertices output by the Ten-Rule as the best repair. Besides that, the Two-Rule and Four-Rule are again working relative to the surrounding vertices which brings the already discussed disadvantages.

Quality Measure As the cause of a constraint violation is unknown, the only sensible approach is to limit the changes to the database as much as possible. Accordingly, the quality of a repair is assessed by the magnitude of its effect on $\mathcal{D}$. A heuristic solution will generally generate a number of possible solutions, one of which will be chosen as the best solution after a finite amount of processing time. For this purpose, a quality-measurement function $dist(\mathcal{D}, \mathcal{D}^R)$ for repairs is required upon which a ranking can be based.

$$dist(\mathcal{D}, \mathcal{D}^R) = \sum_{i \in [1, |\mathcal{D}|]} dist(tr_i, tr_i^R). \tag{4}$$

For the purpose of measuring the distance between the original trajectories and the repaired trajectories, we propose the following three dissimilariy functions.

The first function, $dist_{euclid}$, intuitively yields the spatial difference of two trajectories.

$$dist_{euclid}(tr, tr^R) = \sum_{i \in [1,|tr|]} \left((v_{i_\phi} - v^R_{i_\phi})^2 + (v_{i_\lambda} - v^R_{i_\lambda})^2 + (v_{i_t} - v^R_{i_t})^2 \right)^{\frac{1}{2}} . \quad (5)$$

Utilizing weights for every dimension (w_ϕ, w_λ, w_t) it is easy to provide a Weighted Euclidean Distance function that takes into account the weighting of the time in contrast to the spatial dimensions.

$$dist_{weighted}(tr, tr^R) = \sum_{i \in [1,|tr|]} \left(w_\phi(v_{i_\phi} - v^R_{i_\phi})^2 + w_\lambda(v_{i_\lambda} - v^R_{i_\lambda})^2 + w_t(v_{i_t} - v^R_{i_t})^2 \right)^{\frac{1}{2}} . \quad (6)$$

Finally, in order to provide an alternative to the Euclidean Distance, the third function is based on the Maximum Distance:

$$dist_{max}(tr, tr^R) = \sum_{i \in [1,|tr|]} max\{(v_{i_\phi} - v^R_{i_\phi}), (v_{i_\lambda} - v^R_{i_\lambda}), (v_{i_t} - v^R_{i_t})\}. \quad (7)$$

In order to improve efficiency, the implementation of the above dissimilarity functions does not compute the complete $dist(\mathcal{D}, \mathcal{D}^R)$ after every repair step, but rather compares only trajectories that have been changed during the repair step and sums up the differences for every repair step.

5.2 Generate database repairs

The components outlined above can be combined to create an algorithm to generate a database repair. As finding a minimal database repair is NP-hard (Section 4), any resulting algorithm should employ heuristics to find a good (but not necessarily optimal) repair.

We have identified the following paradigms as possible approaches: Random, Greedy, and Simulated Annealing.

In our description of these algorithms we use the following functions:

- $c : \mathcal{D} \rightarrow \mathcal{V}$ returns the set of vertices that are part of any conflict in $\mathcal{D}$.
- $R_v : \mathcal{D} \rightarrow \mathcal{D}$ is the repair function R (as defined in Section 3), but limited to manipulations of the conflicting vertex v.

Random The simplest approach does not try to choose a good repair function at all. Instead it applies a random instance of a set of possible repair functions to a random conflicting vertex in the database. See Algorithm 1 for a detailed description.

Applying a random repair function does not necessarily reduce the number of conflicts. As a consequence, the algorithm might not converge on a solution.

Algorithm 1 Random($\mathcal{D}$, $\mathcal{R}$)

```
1: while $c(\mathcal{D}) \neq \emptyset$ do
2:     $V \leftarrow c(\mathcal{D})$
3:     $v \leftarrow rnd(V)$
4:     $R \leftarrow rnd(\mathcal{R})$
5:     $\mathcal{D} \leftarrow R_v(\mathcal{D})$
6: end while
```

Greedy The more sophisticated *Greedy* algorithm uses the number of remaining constraints after applying each function to make a better choice. The Random algorithm's weak spot is its unguided choice of repair function. The Greedy algorithm considers only the local improvement of each repair. The repair yielding the lowest number of remaining constraint violations is picked and applied to $\mathcal{D}$. See Algorithm 2 for details.

Algorithm 2 Greedy($\mathcal{D}$, $\mathcal{R}$)

```
1: while $c(\mathcal{D}) \neq \emptyset$ do
2:     $V \leftarrow c(\mathcal{D})$
3:     $v \leftarrow V[0]$
4:     $R_{opt} \leftarrow argmin_{R \in \mathcal{R}} \|R(\mathcal{D})\|$
5:     $\mathcal{D} \leftarrow R_{opt}(\mathcal{D})$
6: end while
```

Compared to the Random algorithm, Greedy's locally optimal repairs yield a much faster convergence on a (possibly local) optimum. However, the increase of complexity leads to an increase in runtime. To find a repair that is closer to the minimal database repair, an algorithm must avoid the local minimum Greedy is prone to converge on. The following algorithm addresses this problem by combining random and greedy elements.

Simulated Annealing The deterministic nature of greedy algorithms makes them prone to local minima. For this reason, algorithm Greedy presented above is likely to return a valid database quickly, but this result is unlikely to be minimal (or close to minimal). To increase the likelihood of finding a global minimum, we now describe an algorithm based on simulated annealing. See Algorithm 3 for a detailed description.

Algorithm 3 SA($\mathcal{D}$, $\mathcal{R}$)

```
1: $\delta = 1$
2: while $c(\mathcal{D}) \neq \emptyset$ do
3:     if $random(]0;1]) < \delta$ then
4:         $\mathcal{D} \leftarrow Random(\mathcal{D}, \mathcal{R})$
5:     else
6:         $\mathcal{D} \leftarrow Greedy(\mathcal{D}, \mathcal{R})$
7:     end if
8:     $\delta \leftarrow \delta - \Delta_\delta$
9: end while
```

By consolidating the Random and Greedy algorithms we counter the overly deterministic nature of greedy algorithms by introducing some randomness in a directed way. This algorithm avoids local minima by initially choosing random repairs, then trying to improve on the best random result using more and more greedy approaches. In each iteration, this algorithm first decides to either perform a Random repair or a Greedy repair with increasing bias toward greediness. In the first iteration, the probability δ of performing a Greedy repair is zero. In each subsequent iteration, this probability increases by a parameter $\varDelta_\delta \in [0, 1]$.

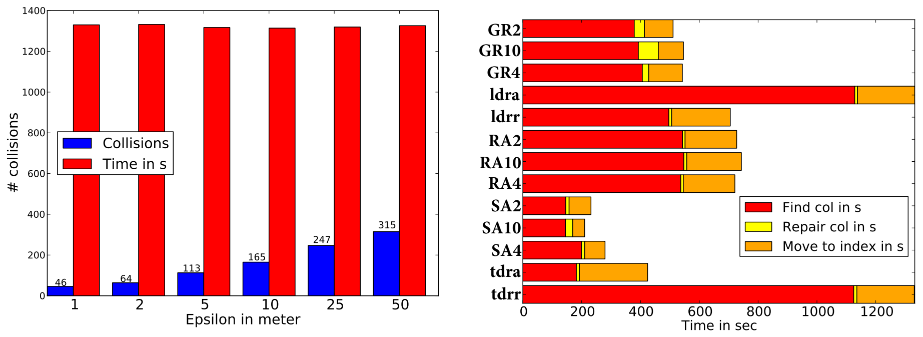
The Simulated Annealing algorithm is expected to be slower than the Greedy algorithm, but more flexible and able to find a more global minimum. Compared to the Random algorithm, Simulated Annealing is faster and more directed. This claim will be evaluated in the following.

6 Experiments

The experimental evaluation presented in this section was conducted using a desktop computer having an Intel i7-870 CPU at 2.93 GHz and 8GB of RAM. The spatio-temporal dataset that we are using consists of workout GPS data, i.e., running and hiking GPS-traces obtained from Endomondo (https://www.endomondo.com). For each GPS-position of a workout a trajectory is stored in $\mathcal{D}$ using linear interpolation, which is the main source of inconsistencies. The service is most popular in Scandinavia, so most workouts are located in cities there. The dataset we used was from the area of Copenhagen, which has a number of vertices between 2567 and 652854. In a data cleaning step, we removed: (1) trajectories that do not have an absolute time-stamp, to avoid having a huge number of runners at the beginning of time; (2) outlier GPS signals yielding a run-speed of more than 50 kilometers per hour. The constraint is that two objects must not be closer than ϵ to each other, where ϵ is a parameter that we can vary in order to alter the number of inconsistencies, called *collisions*. Unless otherwise specified, the default value is $\epsilon = 3$ meter. In this evaluation, we use four straightforward algorithms as a baseline. These four algorithms randomly pick a conflicting GPS-signal p that is adjacent to a conflicting trajectory segment. The, p is distorted by (i) moving its time-stamp one second towards the time of the next GPS-signal (*Absolute Time-Distortion*), or (ii) by moving its time-stamp half-way to the time of the next GPS-signal, or (iii) moving its location one meter towards the location of the next GPS-signal, or (iv) by moving its location half-way to the location of the next GPS-signal. Table 1 lists the various repair heuristics that we presented in Section 5 and shows the respective abbreviations that we will use in the following evaluation.

`tdra`	Absolute Time-Distortion Repair	`tdrr`	Relative Time-Distortion Repair
`ldra`	Absolute Location-Distortion Repair	`ldrr`	Relative Location-Distortion Repair
`ra`	Random	`gr`	Greedy
`sa`	Simulated Annealing	`2`	Two-Repair-Rules
`4`	Four-Repair-Rules	`10`	Ten-Repair-Rules

Table 1. Abbreviations for experiments



(a) Number of collisions (b) Overall Run-Times

Fig. 2. Runtime Experiments

6.1 Collision Detection

In Section 5.1 we describe how we can find collisions in a trajectory database. This is performed by querying individual trajectory segments at an R^*-tree. For our experiments, we use the R^*-tree implementation of the ELKI-framework [2]. The average time required for a single intersection query, depends on the capacity of the R^*-tree. For leaf capacities of 10, 100, and 1000, we measured an average query time of 0.1283, 0.3390, and 7.2444, respectively. We are using a leaf capacity of 10 in the following. Figure Figure 2(a) shows the total time required to find all initial collisions, which requires a large number of intersection queries. The number of collisions is also influenced by the intersection pipe radius ϵ, and Figure 2(a) illustrates the effect on the Endomondo dataset. It is notable that the time required to find collisions seems independent of ϵ. This is attributed to the fact that even for a ϵ of 50m, the number of collision candidates that have to be evaluated is too small to significantly impact the run-time. Thus, the vast majority of time is lost in the collision candidate generation step.

Figure 2(b) shows the time required to repair the found collisions. In each iteration of each algorithm, three steps are required: (i) Repairing a collision, (ii) then updating the index with the new distorted trajectory, and (iii) finding new collisions involving the distorted trajectory. The times required for these three steps are shown in Figure 2(b). We note that despite the use of an efficient index structure, the time needed to repair two colliding trajectories lasts only a fraction of the time needed to find the collision and update/move the trajectory.

6.2 Run Time

The time to repair a collision is further shown in Table 2, along with the number of repair iterations – which varies depending on the heuristic used (likely that the collision has not been fixed, or new collisions may have been incurred). In Table 2, stars next to run-times imply that in at least one case, the repair algorithm did not terminate. Non-terminating cases are ignored for the computation of run-times in this experiments. We can make the following observations: We see that purely time distorting heuristics (tdra

Algorithm	Time to repair	# Repairs	t/#rep
GR4	16.294	342	0.047643
GR10	51.181	330	0.155094
GR2	10.522	429	0.024527
ldra	0.198^*	341	0.000581
ldrr	20.92^*	1341	0.015600
RA4	0.557	545	0.001022
RA10	0.503	519	0.000969
RA2	0.898	684	0.001313
SA4	16.046	343	0.046781
SA10	47.673	332	0.143593
SA2	11.708	464	0.025233
tdra	18.02^*	5506	0.003273
tdrr	17.823^*	1340	0.013301

Table 2. Runtime of all algorithms

and tdrr) and purely location distorting heuristics (ldra and ldrr) are able to repair a database quickly. However, due to the simple rules that these approaches follow, they are unable to handle some cases which may occur in trajectory databases.

– In the case of relative repairs (tdrr and ldrr) this is caused be the fact that if two trajectory segments completely falls into their ϵ-range, then no distortion on these segments can yield a successful repair.
– In the case of absolute repairs (tdra and ldra), some special cases can not be handled. For instance, in the case where two trajectories remain at the same location for multiple GPS-signals: in this case GPS-signals are shifted, but the likelihood of reaching a state where all signals are collision free becomes minimal.

When omitting the cases where these approaches do not terminate, we note that the fastest repair is achieved by the ldra heuristics, which distort observed GPS-signals in space. Furthermore, we can see that among the heuristics to choose a possible repair, the Random-heuristics (RA2, RA4, RA10) perform best, which is expected as these heuristics are not required to make any expensive greedy probing steps. The Greedy (GR2, GR4, GR10) and the Simulated Annealing (SA2, SA4, SA10) require approximately the same time to apply their repairs, but require significantly more time than the pure random approach. Finally, we can see that an increase of the number of repair rules does not affect the random approach, since the time to pick a rule at random can be neglected. For the Simulated Annealing and Greedy approaches, the run-time increases sub-linearly in the number of repair rules: Firstly, each greedy-step requires probing all possible repair rules to pick the most promising one. Yet, this greedy choice is rewarded by reducing the number of total repair iterations that are required to fix the database, thus lowering the run-time.

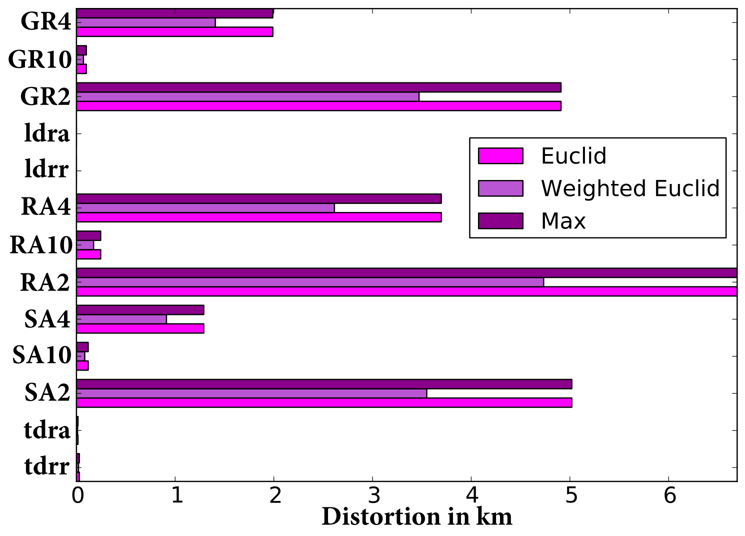


Fig. 3. Quality of Repairs.

6.3 Experiments on Quality of Repair

In Section 5.1 we established three different dissimilarity functions. The results of the experiments are shown in Figure 3. The larger the dissimilarity, the lower the quality of the corresponding repair. The Euclidean and Maximum Distances almost always return the same values, as in most cases, a single trajectory is distorted at one segment only. For the Euclidean distance, we set the weights to $(1, 1, 0.5)$ in order to weight the location stronger, because the domain of latitude and longitude is smaller than the time domain of a workout in seconds. At first glance, it appears that the purely time distorting heuristics (`tdra` and `tdrr`) and purely location distorting heuristics (`ldra` and `ldrr`) seem to yield a nearly perfect repair quality. However, this conclusion is misleading, since for this experiment, we were not able to consider the cases where `ldra`, `ldrr`, `tdra` and `tdrr` do not terminate. These cases however, are the interesting and hard cases, where the most distortion is required to repair the database. Despite this bias, which arises from the fact that `ldra`, `ldrr`, `tdra` and `tdrr` can not repair some collisions, we decided to keep the quality experiments for completeness. Another important observation that we can make in Figure 3, is that for the repair quality of approaches utilizing several repair rules (Random, Greedy and Simulated Annealing), the repair quality improves significantly as the number of possible repair rules increases. In particular, the approach that allows to dodge collisions by distorting space in one of eight directions or by distorting time in one of two directions (the ten-repair-rule case) achieves an extremely high repair quality. When we compare the three heuristics to choose a repair rule, we see that the random heuristic performs by far the worst, thus leading to a large number of needless distortions. The greedy heuristic and the simulated annealing heuristic show comparable results. In fact, the simulated annealing approach yields a better quality in some cases. This is possible, as our greedy approach only selects the locally best next repair rule, which may lead to the global best repair. In contrast, the simulated annealing allows to initially do quick random decisions to get rid of the majority of collisions, and then fix the remaining ones by using greedy decisions.

To summarize, our initial proposed repair rules using only spatial distortion (`ldra` and `ldrr`) and our proposed repair rules using only time distortion (`tdra` and `tdrr`) are not able to repair complex inconsistencies. Nevertheless, these approaches are easily implemented and have low run-times, such that these approaches might find applications in cases where a few remaining inconsistencies can be tolerated. Regarding our proposed repair rules, we saw that the random heuristic is able to achieve the fastest run-time, but incurs a repair-error that may not be tolerable in practice. The greedy approach has the worst run-time, which is attributed to the fact that in every iteration all possible repair rules are tested to choose the locally best. The simulated annealing approach yields a good trade-off, achieving a repair quality comparable to the quality of the greedy approach, while being much faster. Furthermore, we saw a trade-off between run-time and repair quality in the number of repair rules: a larger number of repair rules leads to a (sub-linear) increase in run-times but also to a (drastic) improvement of repair quality. Clearly, a proper choice of repair rules is highly domain specific, depending on the types of inconsistencies that are repaired, and depends on the time-constraints given to the algorithm.

7 Conclusions

In this work, we have formalized a category of problems that has been largely neglected in moving object literature – repairing inconsistencies in historical trajectory databases. This is an important problem since such databases are inherently uncertain for a number of reasons and, in addition, attempt to capture continuous phenomena via discrete values. We have shown that this problem is NP-hard, such that we aim at finding heuristics that find a good repair rather than finding the optimal repair. For this purpose, we presented a number of initial solutions, including a time-distortion algorithm, a space-distortion algorithm, as well as a set of generic algorithms that apply pre-defined repair rules, including a random algorithm, a greedy algorithm and a simulated annealing algorithm. Our experimental setting is aimed at one specific type of inconsistency, namely collisions. The results show that the simple approaches fail to find any repair at all. In contrast, our proposed repair-rule based solutions are able to find a good repair in acceptable time. We believe that this work will spur many challenges in identifying different domain-properties and corresponding heuristics to speed up the "fixings" for different constraints. While finding an optimal repair is a hard problem, we feel that a combination of the techniques presented in this work, as well as the consideration of new ideas, may yield a new solution that combines the best these worlds.

The problem of fixing inconsistencies in moving objects database becomes even more challenging when moving regions are involved. For instance, objects may correspond to hurricanes, consisting of “eye” and “tail”, each approximated by regions, passing over same spatial area. The removal of inconsistencies in such setting may have the potential of existing prediction models that are used in geo-sciences. Addressing the context of mobile regions is one part of our future work. We are also planning to investigate the trade-offs between fixing the inconsistencies in the data vs. fixing inconsistencies in the (answers to) pending queries – which can be challenging in the context of streaming (*location,time*) data. Another challenge that we plan to address is to investigate the impact – and efficient removal – of the inconsistencies in various spatio-temporal data mining tasks.

References

1. Mobile subscribers 2014. Source: ITU World Telecommunication/ICT Indicators database. (`http://www.itu.int/en/ITU-D/Statistics/Documents/facts/ICTFactsFigures2014-e.pdf`).
2. E. Achtert, H.-P. Kriegel, E. Schubert, and A. Zimek. Interactive data mining with 3d-parallel-coordinate-trees. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pages 1009–1012. ACM, 2013.
3. M. Arenas, L. Bertossi, and J. Chomicki. Consistent query answers in inconsistent databases. In *Proceedings of the eighteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '99, pages 68–79, 1999.
4. P. Bohannon, W. Fan, M. Flaster, and R. Rastogi. A cost-based model and effective heuristic for repairing constraints by value modification. In *Proc. SIGMOD*, pages 143–154, 2005.
5. T. Brinkhoff, H. Kriegel, and B. Seeger. Efficient processing of spatial joins using r-trees. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, Washington, D.C., May 26-28, 1993.*, pages 237–246, 1993.
6. R. Cheng, T. Emrich, H. Kriegel, N. Mamoulis, M. Renz, G. Trajcevski, and A. Züfle. Managing uncertainty in spatial and spatio-temporal data. In *IEEE 30th International Conference on Data Engineering, Chicago, ICDE 2014, IL, USA, March 31 - April 4, 2014*, pages 1302–1305, 2014.
7. E. Emerson. Temporal and modal logic. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*, 1990.
8. T. Emrich, H.-P. Kriegel, N. Mamoulis, M. Renz, and A. Züfle. Indexing uncertain spatio-temporal data. In *Proc. CIKM*, pages 395–404, 2012.
9. T. Gindele, S. Brechtel, and R. Dillmann. Learning driver behavior models from traffic observations for decision making and planning. *IEEE Intell. Transport. Syst. Mag.*, 7(1):69–79, 2015.
10. R. H. Güting and M. Schneider. *Moving Objects Databases*. Morgan Kaufmann, 2005.
11. J. Manyika, M. Chui, B. Brown, J. Bughin, R. Dobbs, C. Roxburgh, and A. H. Byers. Big data: The next frontier for innovation, competition, and productivity. *McKinsey & Company Report*, May 2009.
12. F. Parisi and J. Grant. Repairs and consistent answers for inconsistent probabilistic spatio-temporal databases. In *Scalable Uncertainty Management*, pages 265–279. Springer, 2014.
13. A. Parker, V. Subrahmanian, and J. Grant. A logical formulation of probabilistic spatial databases. *Knowledge and Data Engineering, IEEE Transactions on*, 19(11):1541–1556, 2007.
14. E. Pitoura and G. Samaras. Locating objects in mobile computing. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 13(4), 2001.
15. K. Rozier and M. Vardi. Ltl satisfiability checking. In *Automated Technology for Verification and Analysis*, 2011.
16. J. Schiller and A. Voisard. *Location-Based Services*. The Morgan Kaufmann Series in Data Management Systems, 2004.
17. J. Wijsen. Database repairing using updates. *ACM Trans. Database Syst.*, 30(3), 2005.
18. B. Zhang and G. Trajcevski. The tale of (fusing) two uncertainties. In *Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, Dallas/Fort Worth, TX, USA, November 4-7, 2014*, pages 521–524, 2014.