ID Repair for Trajectories with Transition Graphs

Xingcan Cui[†] Xiaohui Yu[§†] Xiaofang Zhou[‡] Jiong Guo[†]

[†]Shandong University, China [§]York University, Canada [‡]University of Queensland, Australia
xccui@mail.sdu.edu.cn xhyu@yorku.ca zxf@itee.uq.edu.au jguo@sdu.edu.cn

ABSTRACT

In many surveillance applications, capture devices are set on fixed locations to track entities, leading to valuable spatio-temporal trajectories. However, sometimes the IDs of the entities in these trajectories are incorrectly identified due to various reasons (e.g., illumination conditions and partial occlusion). Since very often the movements of the entities are constrained by certain restrictions imposed by the application (e.g., vehicles must move along the given road network), we consider how to repair the erroneous IDs using transition graphs derived from such restrictions. Roughly speaking, the occurrence of erroneous IDs can cause a valid trajectory to be broken into trajectory fragments that violate some movement constraints imposed by the transition graph, and we aim to repair them by rewriting the IDs and merging the fragments. This problem is practically challenging since it is not easy to judge which IDs in the dataset are correct, and also there may be multiple candidates as the correct value for a single error. We formulate the repair process as an optimization problem and propose a two-phase repair paradigm, which includes candidate repair generation and compatible repair selection, to maximize the quality improvement estimated by a designed objective function. Though both phases are intractable, we propose effective algorithms to solve them through exploiting the locality and sparsity of trajectories. We further devise an index structure, as well as a pruning method to make the repair process more efficient. Experiments on both real and synthetic datasets demonstrate the effectiveness and efficiency of the proposed methods.

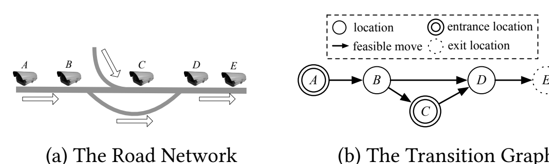
1 INTRODUCTION

Many surveillance related applications require the continuous tracking of entities over time in a specified area. For example, in maritime transport, surveillance devices can be set on ports to track the ships; in traffic surveillance systems, cameras are placed along city streets to capture images of passing vehicles. One of the main tasks for these applications is to identify the unique ID (which may be an atomic value or a composite one consisting of multiple features, such as name, color and shape) of each recorded entity (e.g., a ship or a vehicle) so that tracking records of those entities can be constituted.

For instance, vision-based algorithms are used to identify both the types [3] and names [15] of ships; similarly, optical character recognition (OCR) techniques are used to distill license plate numbers from the captured vehicle images. Due to various reasons (e.g., illumination conditions, partial occlusion or masking), it is not uncommon for the IDs of some entities to be incorrectly identified. Although much effort has been devoted to developing new techniques for improving the recognition accuracy, such errors are still unavoidable, especially when deliberate efforts are made to prevent the entities from being recognized (e.g., in the case of smuggling at sea[1]). According to recent studies [10, 15], the recognition rates of modern approaches are generally over 90% in lab environments, while in real-world settings, the rates may drop (e.g., to about 83% in the real traffic dataset we examined).

In this paper, we take a different perspective and aim to repair erroneous IDs by exploiting the inherent movement constraints, which are formally represented as *transition graphs*, that each entity must follow. Specifically, a transition graph is a directed graph with each vertex corresponding to a location and each edge a feasible move. Furthermore, some vertices are designated as the entrance/exit locations. In general, the transition graphs are the results of geographical restrictions (e.g., road networks), regulations (e.g., shipping routes), etc., and thus can be easily derived or sometimes even explicitly given.

1.1 A Motivating Example



(a) The Road Network (b) The Transition Graph

Figure 1: A Transition Graph Example

Example 1.1. As a running example, consider the transition graph depicted in Figure 1(b) for the road network shown in Figure 1(a) with surveillance cameras installed at locations A, $B, \ldots, E$ (where the hollow arrows indicate driving directions) . The transition graph implies the following two movement constraints: (1) vehicles can only enter this area at locations A or C and leave from location E; and (2) the move of a vehicle must match a directed edge in this graph.

Table 1 shows an example of the tracking records captured by the cameras in Figure 1(a). Without loss of generality, we assume that each tracking record contains at least three fields – the ID, the capture location and the capture timestamp. We also assume that errors occur only in the ID field, as the locations are fixed and the timestamps can be synchronized across cameras and are thus much less error-prone. Records with the same ID can be chronologically sorted and concatenated to form a trajectory. For convenience, we denote a trajectory by the ID followed by the sequence of locations, e.g., $GL21348\langle A \rightarrow B \rightarrow D \rightarrow E\rangle$ represents an entity with ID $GL21348$ moving from A to B to D to E.

Suppose that the dataset is complete, i.e., there are no missing records. Then each trajectory must satisfy both of the aforementioned movement constraints.

Example 1.2. Table 2 shows the composed trajectories from the tracking records in Table 1. Among the three trajectories, only the first one satisfies the movement constraints imposed by

© 2018 Copyright held by the owner/author(s). Published in Proceedings of the 21st International Conference on Extending Database Technology (EDBT), March 26-29, 2018, ISBN 978-3-89318-078-3 on OpenProceedings.org.
Distribution of this paper is permitted under the terms of the Creative Commons license CC-by-nc-nd 4.0.

[1]https://goo.gl/bKGvhb

Table 1: Tracking Records

ID	Loc	Time
$GL21348$	A	08:09:10
$GL21348$	B	08:13:07
$GL03245$	C	08:17:23
$GL21348$	D	08:19:13
$GL83248$	D	08:19:40
$GL21348$	E	08:21:29
$GL83248$	E	08:21:30

Table 2: Trajectories

No.	Trajectory
1	$GL21348\langle A \to B \to D \to E\rangle$
2	$GL03245\langle C\rangle$
3	$GL83248\langle D \to E\rangle$

the transition graph in Figure 1(b) and is thus considered valid; the second and third trajectories are invalid as they fail to satisfy the first movement constraint in Example 1.1.

Note that ID misidentification can cause "fracture" of a valid trajectory and therefore may render the trajectory invalid with respect to the given transition graph. The transition graph is distinct from most existing constraints [9, 25, 27], in that even trajectories with correct IDs may violate the constraints imposed by the graph and thus become invalid.

Example 1.3. Assume that the original trajectory for entity with ID $GL83248$ is $GL83248\langle C \to D \to E\rangle$. Unfortunately, its ID is misidentified as $GL03245$ by the camera at C. The trajectory is thus broken into the second and third trajectories in Table 2, both of which are invalid (though the second trajectory is actually error-free).

Some related approaches [27, 29] try to repair erroneous attributes in temporal events (e.g., logs from manufacturing) by exploiting the structural information or the neighborhood constraints of the activities. They propose efficient methods based on graph structures to detect dirty events and devise heuristic algorithms to repair them based on the *minimum change principle* [4]. However, these approaches fall short in our scenario mainly because (1) they perform isolated label rewritings while in our problem a single repair option may involve multiple ID-rewritings (as an ID may be identified as multiple erroneous values), (2) they do not consider the spatial relationships between the trajectories, which play an important role in our problem, and (3) the minimum change principle they follow may no longer be appropriate in our scenario.

1.2 The Present Work

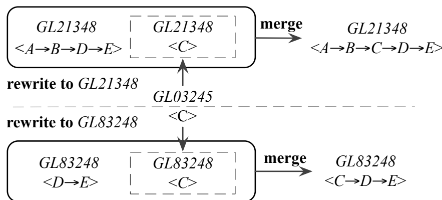


Figure 2: Two Repair Options for Trajectory $GL03245[C]$

We propose to repair the erroneous IDs through rewriting the IDs and merging the trajectory fragments to recover the "original" trajectories that are valid with respect to the transition graph.

Example 1.4. Consider the trajectory $GL03245\langle C\rangle$ in Table 2. As shown in Figure 2, by exploring the inherent location and timestamp relationships, we can rewrite the trajectory's ID to $GL83248$ (or, $GL21348$) and then merge the corresponding tracking records chronologically to form a valid trajectory $GL83248\langle C \to D \to E\rangle$ (or $GL21348\langle A \to B \to C \to D \to E\rangle$).

As shown in Figure 2, there could be multiple options to repair an invalid trajectory, which are mutually exclusive since it is logically inconsistent to repair a single ID to different values at the same time. As such, only one of the options can be used.

Various factors can be considered in evaluating the "goodness" of the two repair options in Example 1.4, e.g., the ID similarity and the number of invalid trajectories eliminated. In this example, the bottom repair option in Figure 2 (i.e., rewriting to $GL83248$) is more likely to be selected roughly because (1) compared with "$GL21348$", the string "$GL83248$" is more similar to "$GL03245$" (in terms of edit distance), and (2) applying this option can eliminate all the invalid trajectories in the dataset, while applying the other one (i.e., rewriting to $GL21348$) will leave a dangling invalid trajectory $GL83238\langle D \to E\rangle$ without other trajectories to merge with.

The examples above represent a simple case where the dataset contains only one misidentified ID. In practice, the problem of trajectory ID repair is much more complex because (1) it is non-trivial to judge which IDs in the dataset are correct, (2) generating the repair options, each of which may involve multiple trajectories, can be time consuming, and (3) there can be a large number of interrelated repair options and we must consider them as a whole to make global decisions.

To address these issues, we propose a repair paradigm that consists of two phases. In the first phase, we generate all potential repair options that meet certain criteria for later consideration; in the second phase, we use an evaluation function to estimate the data quality improvement brought by each repair option, and then search for a set of such options that can be applied in tandem to maximize an overall objective function (which reflects the global quality improvement for the given dataset). Specifically, we extract the core processes in these two phases as a clique generation problem and a weighted independent set problem, which are all NP-hard in general settings. To cope with the first problem, we add some restrictions on the cliques of interest and provide a backtracking algorithm. To solve the second one, we propose an approximate greedy algorithm by exploiting the probability of selecting a correct repair option. Furthermore, we also devise an index structure and a pruning method to make the whole repair approach more efficient.

In repairing the IDs, we do not invent new values and assume the correct IDs can always be found from the dataset, which is in line with most previous work on data repair or data matching in other settings [4, 22, 32]. While the datasets often exhibit another type of error, namely missing records, our focus in this paper is on repairing erroneous IDs, which constitute the main source of error in the datasets we have examined. The reason is that the problem of missing records are often mitigated through the deployment of other supporting or complementary technologies. For example, inductive-loop traffic detectors [2] installed at the same locations as the traffic cameras can detect almost all passing vehicles and trigger shots by the corresponding cameras. Therefore, it is rare for a vehicle not to be captured by the camera and its plate (either correctly or incorrectly) recognized. In general, dealing with missing records is a separate issue worthy of investigation in our future work; in fact, most existing methods on missing value recovery [30, 31] focus on the issue of missing records itself without tackling other data quality problems such as indistinguishable objects. Nonetheless, we conduct experiments in Section 6.3.3 to empirically evaluate the impact of missing records on the performance of the proposed methods.

1.3 Contributions

To the best of our knowledge, we are the first to study the problem of trajectory ID repair facilitated by the moving rules. In summary, we make the following contributions in this paper.

(1) We propose a novel transition graph based trajectory ID repair problem, as well as a two-phase repair paradigm to solve it.
(2) By exploiting the locality and sparsity of the spatio-temporal trajectories, we provide practical algorithms that can solve the problem effectively.
(3) We further devise some optimization methods, which make our approach more efficient.
(4) Extensive experiments are conducted on both real and synthetic datasets, which demonstrate the effectiveness and efficiency of the proposed methods.

The rest of the paper is organized as follows. In Section 2, we provide the preliminaries and formally define the problem. We present the two-phase repair paradigm and the detailed algorithms in Section 3 and Section 4, respectively. We further propose some optimization methods in Section 5, and show the experimental results in Section 6. We provide an overview of the related work in Section 7, and conclude this paper in Section 8.

2 PROBLEM DESCRIPTION

2.1 Preliminaries

We first define several terms that will be used throughout the paper.

Definition 2.1. **Transition Graph**, **Entrance Location** and **Exit Location**. A *transition graph* $\mathcal{G}_t = (\mathbf{V}, \mathbf{E})$ is a directed graph that represents the set of movement constraints that the entities must follow. Each vertex $loc \in \mathbf{V}$ represents a location (e.g., where a surveillance device is installed), and an edge $(loc_i, loc_j) \in \mathbf{E}$ indicates that an entity can directly move from location loc_i to location loc_j. Among these vertices (locations), there are some special ones from which the entities can enter or leave the area of interest. We call them the *entrace locations* (the set of which is denoted by $\mathbf{I}$) and *exit locations* (the set of which is denoted by $\mathbf{O}$).

For the transition graph shown in Example 1.1, $\mathbf{V} = \{A, B, C, D, E\}$, $\mathbf{E} = \{(A, B), (B, C), (B, D), (C, D), (D, E)\}$, $\mathbf{I} = \{A, C\}$ and $\mathbf{O} = \{E\}$.

Definition 2.2. **Valid Path**. Given a transition graph $\mathcal{G}_t = (\mathbf{V}, \mathbf{E})$ with the entrance location set $\mathbf{I}$ and the exit location set $\mathbf{O}$, we call a location sequence $loc_1 \to loc_2 \cdots \to loc_q$ a *valid path* if it satisfies the following three conditions: (1) $loc_1 \in \mathbf{I}$, (2) $(loc_i, loc_{i+1}) \in \mathbf{E} (1 \leq i < q)$, and (3) $loc_q \in \mathbf{O}$.

Definition 2.3. **Tracking Record.** A *tracking record* $\mathbf{r}$ is a triple (id, loc, ts), where id is the entity's unique identifier (which may be erroneous and is the subject of our study), loc is the location, and ts is the timestamp.

Definition 2.4. **Trajectory**, **Valid Trajectory**, and **Invalid Trajectory**. A *trajectory* $\mathbf{T}$ is a chronologically ordered sequence of tracking records with the same ID (denoted by $\mathbf{T}.id$), i.e., $\mathbf{T} = \mathbf{r}_1 \to \mathbf{r}_2 \cdots \to \mathbf{r}_q$ with $\mathbf{T}.id = \mathbf{r}_1.id = \mathbf{r}_2.id = \cdots = \mathbf{r}_q.id$ and $\mathbf{r}_1.ts < \mathbf{r}_2.ts < \cdots < \mathbf{r}_q.ts$. We can represent a trajectory with the ID followed by the location sequence, e.g., $GL12345\langle A \to B \to C\rangle$. Given a transition graph $\mathcal{G}_t$ and a trajectory $\mathbf{T}$, we call $\mathbf{T}$ a *valid trajectory* (or VT for short) if the location sequence $\mathbf{r}_1.loc \to \mathbf{r}_2.loc \cdots \to \mathbf{r}_q.loc$ of $\mathbf{T}$ is a valid path w.r.t. $\mathcal{G}_t$. Otherwise we call $\mathbf{T}$ an *invalid trajectory* (or IVT for short).

In an ideal setting, each trajectory in a dataset is error-free and contains all the tracking records of an entity (e.g., $GL21348\langle A \to B \to D \to E\rangle$ in Example 1.2). Due to ID errors, however, a trajectory may be broken into multiple fragments, each containing tracking records with a different ID (e.g., $GL03245\langle C\rangle$ and $GL83248\langle D \to E\rangle$ in Example 1.2). Certainly, at most one of the IDs is correct, and the the rest are all erroneous. Most of the time, those fragments are invalid trajectories (including the one with the correct ID), but there does exist a slight possibility that each of the fragments coincidentally corresponds to a valid path in the transition graph and is thus deemed valid. Considering its rarity, we ignore this case in our subsequent discussion.

We view the ID repair problem as one of "restoring" the original true trajectory through ID rewriting. That is, we seek to find a subset of trajectories and rewrite their IDs to (hopefully) the correct one, and then merge those trajectories to form a valid trajectory. As such, we introduce the following definitions.

Definition 2.5. **Join**, **Joinable Subset** and **Target ID**. Given a transition graph $\mathcal{G}_t$, we define *join* on a trajectory set $\mathcal{T}'$ and an existing ID r from $\mathcal{T}'$ as first rewriting the ID for each $\mathbf{T} \in \mathcal{T}'$ to r and then merging all tracking records in $\mathcal{T}'$ *chronologically* to constitute a new trajectory $\widehat{\mathbf{T}}_r$, i.e., $\widehat{\mathbf{T}}_r = join(\mathcal{T}', r)$. The join is valid iff the newly formed trajectory $\widehat{\mathbf{T}}_r$ is a VT w.r.t. $\mathcal{G}_t$, and we call r the *target ID* and $\mathcal{T}'$ a *joinable subset* iff such a valid join exists.

Definition 2.6. **Candidate Repair**. A *candidate repair* (or *repair* for short) R is a pair $(\mathcal{T}', r)$ consisting of a joinable subset $\mathcal{T}'$ and a target ID r. For a repair R, the corresponding joinable subset and the invalid trajectories contained therein are denoted by $jns(R)$ and $ivt(R)$. With R defined, the join operation can be rephrased as $\widehat{\mathbf{T}}_r = join(R)$. If r is the true ID for all trajectories contained in $\mathcal{T}'$ (which implies that they actually come from the same entity), and $\widehat{\mathbf{T}}_r$ is the true trajectory for the entity with ID r, we call R a *correct candidate repair* (or *correct repair* for short). For two candidate repairs R_i and R_j, if their joinable subsets are mutually exclusive, i.e., $jns(R_i) \cap jns(R_j) = \emptyset$, we say that R_i and R_j are *compatible*; otherwise *incompatible.* If all pairs of repairs in a set of repairs are compatible, then we call this set a *compatible repair set.*

Figure 2 shows two candidate repairs whose correctness is unknown. They are incompatible due to the sharing of trajectory $GL03245\langle C\rangle$.

Given a trajectory set $\mathcal{T}$ and a compatible repair set $\mathcal{R}'$, a new trajectory set $\widehat{\mathcal{T}}$ can be produced by joining the trajectories indicated by each $R \in \mathcal{R}'$. As illustrated before, the reason why the repairs must be compatible is that it does not make sense to rewrite a trajectory's ID to more than one target ID at the same time.

2.2 Problem Statement

Given a transition graph $\mathcal{G}_t$, a set of trajectories $\mathcal{T}$ with ID errors, we use $\mathcal{R}$ to represent the set which contains all the potential candidate repairs. The ID repair problem can be regarded as searching a compatible repair set $\mathcal{R}' \subseteq \mathcal{R}$, and joining trajectories designated by the compatible candidate repairs in $\mathcal{R}'$ to obtain a new trajectory set $\widehat{\mathcal{T}}$.

Note that there may exist various (incompatible) candidate repairs in $\mathcal{R}$ for a single trajectory, and our goal is to find the

most promising one. We thus need a function $\omega(R)$ to evaluate the effectiveness of R, which serves as an estimate of how much quality improvement can be gained by R.

2.2.1 The Evaluation Function for Repair Effectiveness. We first discuss the factors that we have considered in devising a suitable evaluation function for the effectiveness of a repair.

- **The individual fitness of a repair**. In real applications, the erroneous IDs often bear some similarities with their correct values, i.e., the more similar two IDs are, the more likely they correspond to the same entity. In the case of composite IDs, even if attempts are made to camouflage the entities with a fake name, the remaining components of the IDs, such as color and type, are more difficult to conceal and thus the erroneous IDs would still be similar to the true IDs. For that reason, we use ID similarity to evaluate the individual fitness of a repair. This similarity can be measured by the distance between the strings or feature vectors representing the IDs, and there have been dozens of metrics (e.g., edit distance, overlap coefficient, and cosine similarity) proposed in the literature for this purpose [24]. In this paper, we choose edit distance as the similarity metric, but this can be replaced by other metrics for different applications. Specifically, for a repair $R = (\mathcal{T}', r)$, we use the *similarity function* $sim(R)$, which is based on the minimum similarity from an ID in $\mathcal{T}'$ to the target ID r, to evaluate the individual fitness of R:

$$sim(R) = \min_{\mathbf{T} \in \mathcal{T}'} \left(1 - \frac{dist(r, \mathbf{T}.id)}{max(|r|, |\mathbf{T}.id|)}\right) \tag{1}$$

  where $dist(r, \mathbf{T}_i.id)$ is the edit distance between r and $\mathbf{T}_i.id$, and $|\mathbf{T}_i.id|$ is the length of $\mathbf{T}_i.id$. Apparently, the range of the similarity function is $[0, 1]$.
- **The potency of a repair.** As illustrated before, ID errors will cause invalid trajectories. Candidate repairs that fix more IVTs are considered more "powerful" and are expected to bring greater quality improvement to a dataset.
- **The rarity of a repair.** For a set of trajectories $\mathcal{T}$ and a set of corresponding potential candidate repairs $\mathcal{R}$, each IVT $\mathbf{T}' \in \mathcal{T}$ may be covered by multiple repairs in $\mathcal{R}$. We call the number of those repairs the *degree of* $\mathbf{T}'$ (denoted by $d(\mathbf{T}')$). A smaller degree implies that the trajectory is more "endangered", i.e., there are fewer candidate repairs that are able to fix it. On the other hand, each such candidate is considered more precious or rarer and thus should be preferred. We define the *rarity* of a repair R as

$$ra(R) = \min_{\mathbf{T} \in ivt(R)} d(\mathbf{T}) \tag{2}$$

  The value of $ra(R)$ ranges from 1 to $|\mathcal{R}|$. When $ra(R)$ takes the minimum value of 1, it means that only R can fix a certain IVT in the dataset.

Different effectiveness evaluation functions can be designed based on the factors above. We find the following one performs well in our scenarios:

$$\omega(R) = \begin{cases} 0 & |ivt(R)| = 0 \\ sim(R) + \lambda \log_{ra(R)+1} |ivt(R)| & |ivt(R)| \geq 1 \end{cases} \tag{3}$$

where $sim(R)$ and $ra(R)$ are the similarity function and the rarity function respectively, $|ivt(R)|$ represents the number of invalid trajectories in R, and $\lambda \in (0, 1]$ is a coefficient controlling the trade-off between the two terms.

In Equation (3), the first term $sim(R)$ acts as assurance on the matching fitness and makes it unlikely for an ID to be rewritten to another arbitrary ID. The second term $\log_{ra(R)+1} |ivt(R)|$ represents the potency impact scaled by the rarity factor. According to this term, repairs holding more invalid trajectories and being more rare will be more effective. In general, $ra(R) + 1 \gg |ivt(R)|$ and thus the range for $\log_{ra(R)+1} |ivt(R)|$ is also $[0, 1]$.

Note that due to the introduction of $ra(R)$, the effectiveness of a repair cannot be evaluated unless a full candidate repair set $\mathcal{R}$ is provided.

2.2.2 The ID Repair Problem. Given a trajectory set $\mathcal{T}$ (with ID errors), a transition graph $\mathcal{G}_t$, and an evaluation function w for repair effectiveness, the ID repair problem is to search for a compatible repair set $\mathcal{R}'$ that can maximize the sum of the repair effectiveness. Formally, this can be described as

$$\begin{aligned} \underset{\mathcal{R}'}{\text{maximize}} \quad & \Omega(\mathcal{R}') = \sum_{R \in \mathcal{R}'} \omega(R) \\ \text{subject to} \quad & jns(R_i) \cap jns(R_j) = \emptyset, \forall R_i, R_j \in \mathcal{R}'. \end{aligned} \tag{4}$$

2.3 Applicability and Assumptions

The ID repair approach we adopt exploits the locality and sparsity properties of the real world spatio-temporal trajectories:

- *Locality of movement* – a given entity is more likely to move within a local geospatial neighborhood than "jumping" between far-away locations in a relatively short time span. This implies that it is more likely to find the true value of an erroneous ID based on entities that are close in time and space.
- *Sparsity of IDs* – two entities with different but very similar IDs are highly unlikely to appear in the same local neighborhood during a relatively short period of time.

Based on these properties, we make the following assumptions. First, we assume that identical IDs in the data, whether correct or not, belong to the same entity. In other words, we would not break a trajectory into smaller pieces. In exceptional cases, it may so happen that a tracking record with an erroneous ID becomes part of a valid trajectory in place of a missing record with the true ID; but such cases are so rare due to the sparsity of IDs that we could consider them negligible.

Second, we consider all records with the same ID constitute a trajectory with bounded length and time span. The rationale is that despite some entities intending to wander around in the area of interest, the majority should be passing traffic and thus their trajectories should not be too long. Based on this assumption, we set two bounds θ and η, where θ is the maximum possible length of a VT (i.e., the maximum number of tracking records in it) in a dataset, and η is the maximum time span for a VT.

Finally, we assume that the error rate for ID identification is not too high (which is consistent with what we observe from real data), i.e., the number of trajectory fragments caused by erroneous IDs in a trajectory is limited. We use a bound ζ to represent the maximum possible number of trajectories in a joinable subset. Although there may be extraordinary cases where these assumed bounds do not hold, their establishment grants us the opportunity to significantly improve the efficiency of the proposed algorithms, as will be shown in Section 6.2.

3 A TWO-PHASE REPAIR PARADIGM

We now describe a two-phase repair paradigm that serves as the framework for solving the ID repair problem. The detailed

algorithms involved in this paradigm will be presented in Section 4.

3.1 Overview of the Paradigm

The basic idea of the proposed paradigm is to first generate all possible candidate repairs and then select a set of repairs that can maximize the objective function (Equation (4)). Figure 3 depicts the repair paradigm consisting of two phases: *candidate repair generation* and *compatible repair selection*.

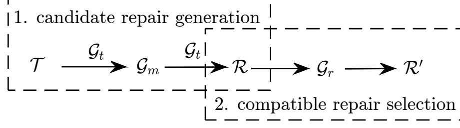


Figure 3: The Two-Phase Repair Paradigm

In the candidate repair generation phase, we generate all potential candidate repairs $\mathcal{R}$ (which are not necessarily compatible with each other) from the input trajectory set $\mathcal{T}$ according to the transition graph $\mathcal{G}_t$. This phase can be accomplished using an undirected graph with each vertex representing a different trajectory and each edge indicating that the two trajectories corresponding to its connected vertices can appear in the same joinable subset. We call this graph the *trajectory graph* (denoted by $\mathcal{G}_m$).

In the compatible repair selection phase, we perform the actual ID repair by selecting a set of compatible repairs $\mathcal{R}' \subseteq \mathcal{R}$ that have the maximum total effectiveness in terms of Equation (4). This task can be carried out by introducing another undirected graph that reflects the incompatible relationships between different repairs in $\mathcal{R}$. We call this graph the *repair graph* (denoted by $\mathcal{G}_r$).

3.2 Candidate Repair Generation

This phase performs two core tasks, namely *joinable subset determination* and *target ID assignment*.

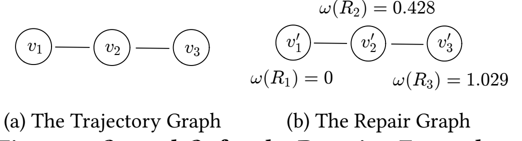
3.2.1 Joinable subset determination. Before delving into the details of joinable subset determination, we first introduce two predicates.

(1) **The** *cex* **predicate.** Given a transition graph $\mathcal{G}_t$, the *cex* predicate works by checking whether two trajectories can coexist in a joinable subset w.r.t. $\mathcal{G}_t$, i.e., $\{\mathbf{T}_x, \mathbf{T}_y | cex(\mathbf{T}_x, \mathbf{T}_y)\} = \{\mathbf{T}_x, \mathbf{T}_y | \exists \mathcal{T}'(\mathbf{T}_x, \mathbf{T}_y \in \mathcal{T}'),$ and $\mathcal{T}'$ is a joinable subset$\}$. Apparently, only if the location sequence for the chronologically merged records of the two trajectories is a subsequence (which does not have to be continuous) of a path in $\mathcal{G}_t$, can this predicate evaluate to true.

(2) **The** *jnb* **predicate.** Given a transition graph $\mathcal{G}_t$, the *jnb* predicate is used to determine whether a set of trajectories is a joinable subset w.r.t. $\mathcal{G}_t$, i.e., $\{\mathcal{T}_x | jnb(\mathcal{T}_x)\} = \{\mathcal{T}_x | \exists \mathcal{T}'(\mathcal{T}_x = \mathcal{T}'),$ and $\mathcal{T}'$ is a joinable subset$\}$. This can be performed by checking whether the location sequence for the chronologically merged records is a valid path in $\mathcal{G}_t$. As a special case, for a trajectory set with only one element, the *jnb* predicate evaluates to *true* if that element is a valid trajectory.

We first use the *cex* predicate to construct the trajectory graph $\mathcal{G}_m$. Specifically, each vertex v_i in $\mathcal{G}_m$ corresponds to a $\mathbf{T}_i \in \mathcal{T}$, and each undirected edge (v_i, v_j) corresponds to a trajectory pair $(\mathbf{T}_i, \mathbf{T}_j)$ such that $cex(\mathbf{T}_i, \mathbf{T}_j)$ evaluates to true.

Example 3.1. Let us use $\mathbf{T}_1$, $\mathbf{T}_2$, and $\mathbf{T}_3$ to represent the three trajectories shown in Table 2. To construct $\mathcal{G}_m$, we first create three vertices v_1, v_2, and v_3, corresponding to $\mathbf{T}_1$, $\mathbf{T}_2$, and $\mathbf{T}_3$ respectively. Since both $cex(\mathbf{T}_1, \mathbf{T}_2)$ and $cex(\mathbf{T}_2, \mathbf{T}_3)$ evaluate to true, two edges (v_1, v_2) and (v_2, v_3) are added to $\mathcal{G}_m$. Figure 4(a) shows the constructed trajectory graph for the dataset.



(a) The Trajectory Graph (b) The Repair Graph

Figure 4: $\mathcal{G}_m$ and $\mathcal{G}_r$ for the Running Example

Such construction allows the problem of searching for joinable subsets to be transformed to operations on $\mathcal{G}_m$, as shown in the theorem below.

THEOREM 3.2. *A necessary but not sufficient condition for a trajectory set to be a joinable subset is that its corresponding vertex set in $\mathcal{G}_m$ is a clique.*

To compute the joinable subsets, we first generate all cliques in $\mathcal{G}_m$ and then use the *jnb* predicate to check whether their corresponding trajectory sets can really be joinable subsets. Actually, the clique generation process can be regarded as a preprocessing step which saves us from enumerating all the trajectory combinations. In general settings, listing all cliques in a graph is NP-hard [19]. Fortunately, since both the length of a VT and the number of trajectories contained in a candidate repair are bounded according to our aforementioned assumptions, the problem can be solved in polynomial time in our case. We present the detailed algorithm for generating the qualified-cliques in Section 4.1.2.

Example 3.3. For the trajectory graph shown in Figure 4(a), there are five cliques: $\{v_1\}$, $\{v_2\}$, $\{v_3\}$, $\{v_1, v_2\}$, and $\{v_2, v_3\}$. However, evaluating *jnb* on their corresponding trajectory sets reveals that there are only three joinable subsets: $\{\mathbf{T}_1\}$, $\{\mathbf{T}_1, \mathbf{T}_2\}$, and $\{\mathbf{T}_2, \mathbf{T}_3\}$.

3.2.2 Target ID Assignment. After generating all the joinable subsets, we assign target IDs to them. Given a joinable subset $\mathcal{T}'$, the target ID is decided by selecting a trajectory $\mathbf{T}_c$ that maximizes the following equation.

$$\mathbf{T}_c = \underset{\mathbf{T}_i \in \mathcal{T}'}{argmax} \left(\sum_{\mathbf{T}_j \in \mathcal{T}'} \frac{|\mathbf{T}_i|}{|\mathbf{T}_j|} \left(1 - \frac{dist(\mathbf{T}_i.id, \mathbf{T}_k.id)}{max(|\mathbf{T}_i.id|, |\mathbf{T}_j.id|)} \right) \right) \quad (5)$$

In this equation, $|\mathbf{T}_i|$ ($|\mathbf{T}_j|$) is the length of trajectory $\mathbf{T}_i$ ($\mathbf{T}_j$), $|\mathbf{T}_j.id|$ ($|\mathbf{T}_j.id|$) is the length of ID $\mathbf{T}_i.id$ ($\mathbf{T}_j$.id), and $dist(\mathbf{T_i}.id, \mathbf{T}_j.id)$ is the edit distance between them. The rationale behind the choice of this equation is that when all trajectories have the same length, our goal is to choose a target ID that can maximize the sum of similarities of all IDs with the target ID. We also give preference to longer trajectories, as the error rate for ID identification is usually low and it is unlikely for the same error to be made at consecutive locations in a trajectory. Note that we choose to use edit distance here to measure the (dis-)similarity between IDs, but other distance measures can also be used where appropriate.

Example 3.4. Based on Equation (5), we assign $GL21348$, $GL21348$, and $GL83248$ as the target IDs for the three joinable subsets $\{\mathbf{T}_1\}$, $\{\mathbf{T}_1, \mathbf{T}_2\}$, and $\{\mathbf{T}_2, \mathbf{T}_3\}$ generated in Example 3.3 and combine them respectively to generate three candidate repairs: $R_1 = (\{\mathbf{T}_1\}, GL21348)$, $R_2 = (\{\mathbf{T}_1, \mathbf{T}_2\}, GL21348)$, and $R_3 = (\{\mathbf{T}_2, \mathbf{T}_3\}, GL83248)$. Then we calculate their effectiveness values according to Equation (3), which are $\omega(R_1) = 0$, $\omega(R_2) = 0.428$, and $\omega(R_3) = 1.029$.

3.3 Compatible Repair Selection

In the previous phase, we have generated all candidate repairs $\mathcal{R}$. Next, we select compatible repairs from $\mathcal{R}$ to maximize the objective function in Equation (4). The selection process can be mapped to operations on another undirected graph, namely the repair graph, $\mathcal{G}_r$, which can be constructed as follows: (1) for each candidate repair $R_i \in \mathcal{R}$, add a corresponding vertex v'_i to $\mathcal{G}_r$; (2) if $R_i, R_j \in \mathcal{R}$ share an identical trajectory, add an undirected edge (v'_i, v'_j) to $\mathcal{G}_r$.

Example 3.5. To construct $\mathcal{G}_r$ for the three candidate repairs generated in Example 3.4, we first add three corresponding vertices v'_1, v'_2, and v'_3. Since $jns(R_1) \cap jns(R_2) = \{\mathbf{T}_1\}$ and $jns(R_2) \cap jns(R_3) = \{\mathbf{T}_2\}$, two edges (v'_1, v'_2) and (v'_2, v'_3) are added. Finally, we get the repair graph shown in Figure 4(b).

Such construction allows us to view the problem of selecting compatible repairs as packing vertices from $\mathcal{G}_r$ where no pairs are adjacent. This translates to the well known *weighted independent set problem*, which is NP-hard in common settings [23]. Considering the inherent relationships between repairs, we present a greedy algorithm to approximately solve this problem in Section 4.2.

4 ALGORITHMS

In this section, we present the core algorithms for the two-phase repair paradigm.

4.1 Algorithms for Repair Generation

4.1.1 Evaluating the cexnd jnbredicates. Given a transition graph $\mathcal{G}_t$, the *cex* predicate determines whether two trajectories $\mathbf{T}_1$ and $\mathbf{T}_2$ can coexist in a joinable subset. The key idea in evaluating this predicate is to check whether the location sequence for the chronologically merged records is a subsequence of a path in $\mathcal{G}_t$. This can be considered a reachability problem, i.e., for the merged location sequence $loc_1 \rightarrow loc_2 \rightarrow \cdots \rightarrow loc_q$ ($q = |\mathbf{T}_1| + |\mathbf{T}_2|$), if loc_i and loc_{i+1} belong to different trajectories, we check whether loc_{i+1} is reachable from loc_i.

A straightforward solution for this problem using breadth-first (or depth-first) search takes linear time, but we can do some preprocessing to have it done in constant time. Specifically, the Floyd Warshall algorithm [16] can be employed to calculate the shortest-path matrix $\mathcal{M}$ for $\mathcal{G}_t$, where $\mathcal{M}[i][j]$ indicates the number of edges in the shortest path from loc_i to loc_j. After this preprocessing step, the reachability queries can be answered instantly by consulting the elements in $\mathcal{M}$.

Moreover, recall that we have two user-defined bounds, the maximum length θ and the maximum time span η for each VT. Thus we should also check whether $|\mathbf{T}_1| + |\mathbf{T}_2| \leq \theta$ and whether the time span for the merged sequence exceeds η. Putting things together, we show the algorithm for evaluating the *cex* predicate in Algorithm 1.

Compared with the *cex* predicate, the *jnb* predicate is more strict in that it evaluates to *true* only if the given trajectories can perfectly make up a joinable subset. The algorithm for this predicate is similar to that for the *cex* predicate with the following two additional restrictions: (1) the location attributes in the earliest and the latest records of the input trajectories must be an entrance location and an exit location in $\mathcal{G}_t$, respectively, and (2) no matter whether two adjacent records (in the merged sequence) $\mathbf{r}_i$ and $\mathbf{r}_{i+1}$ belong to the same trajectory or not, there must be an edge from $\mathbf{r}_i.loc$ to $\mathbf{r}_{i+1}.loc$. Due to space limitation, the detailed algorithm for evaluating the *jnb* predicate is omitted.

Algorithm 1 Algorithm for the *cex* predicate

Input: The reachability matrix $\mathcal{M}$ for $\mathcal{G}_t$; the maximum length θ; the maximum time span η; two trajectories $\mathbf{T}_1$ and $\mathbf{T}_2$.
Output: *true* if $\mathbf{T}_1$ and $\mathbf{T}_2$ can coexist in a joinable subset; *false* otherwise.

```
1:  if |T1| + |T2| > θ then
2:      return false
3:  merge records in T1 and T2 by their timestamps to get a
    sequence r1 → r2 → ··· → rq;
4:  if rq.ts − r1.ts > η then
5:      return false
6:  for all ri and ri+1 in the merged sequence do
7:      if ri.id ≠ ri+1.id then
8:          if M[i][i+1] ≥ θ then
9:              return false
10: return true
```

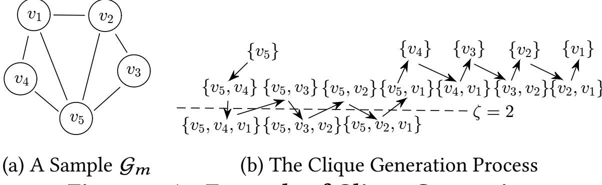
4.1.2 Generating Qualified Cliques. We now introduce the algorithm for generating the qualified cliques in $\mathcal{G}_m$. In general, enumerating all cliques in an undirected graph requires exponential time, and the running time is output-sensitive (i.e., the running time depends on the size of the output). However, when the trajectory length and clique size are bounded (by θ and ζ respectively), the problem can be much simplified. We show how to generate the qualified cliques in Algorithm 2, which runs in $O(|\mathbf{V}_m|^{\zeta})$ time.

Algorithm 2 Algorithm for generating qualified cliques

Input: The adjacency matrix representation of trajectory graph $\mathcal{G}_m = \{\mathbf{V}_m, \mathbf{E}_m\}$; a set C to store vertices in the current clique; an index list L for vertices; the maximum trajectory length θ; the maximum clique size ζ.
Output: a set of generated cliques *results*.

```
1:  fill L with 1, 2, ··· , |Vm|
2:  function Clique(C, L)
3:      for i ← L.size(), 1 do
4:          v ← L.get(i)
5:          C ← C ∪ {Tv}
6:          create a new list Lnew
7:          for j ← 0, i do
8:              w ← L.get(j)
9:              if (v, w) ∈ Em then
10:                 Lnew.add(w)
11:         results ← results ∪ {C}
12:         if ¬Lnew.empty() and C.RecordNumber < θ and
    |C| < ζ then
13:             Clique(C, Lnew)
14:         C ← C \ {Tv}
15:         L.remove(i)
16:     return results
```

The main idea of the generation algorithm is backtracking. It iteratively adds to a temporary vertex set C a vertex that is adjacent to all existing ones in C from an input vertex list, outputs the result, starts a recursion with a new list of vertices that are adjacent to the newly added vertex, and finally removes the vertex added in this round. As shown in Line 12, unnecessary recursions are eliminated using the bounds θ and ζ.



(a) A Sample $\mathcal{G}_m$ (b) The Clique Generation Process

Figure 5: An Example of Clique Generation

Example 4.1. Figure 5 shows an example of the clique generation process. Given a trajectory graph with 5 vertices, the 15 cliques are generated in turn with Algorithm 2. As shown in Figure 5(b), if $\zeta = 2$, the algorithm will automatically skip generating cliques beneath the dashed line.

4.2 Algorithms for Repair Selection

As discussed in Section 3.3, the repair selection problem corresponds to a weighted-independent set problem on $\mathcal{G}_r$, which is NP-hard. In search of efficient solutions, consider the effectiveness evaluation function in Equation (4). It is defined as an indicator of the potential quality improvement due to a repair, but by no means a definitive measure of the true improvement. That is, for two compatible repair sets $\mathcal{R}'_1$ and $\mathcal{R}'_2$, if $\Omega(\mathcal{R}'_1) > \Omega(\mathcal{R}'_2)$, it just indicates that $\mathcal{R}'_1$ is likely better than $\mathcal{R}'_2$, but not definitely, especially when the values of $\Omega(\mathcal{R}'_1)$ and $\Omega(\mathcal{R}'_2)$ are close. This is confirmed by a large number of experiments on different datasets, from which we find that the Ω values of the optimal compatible repair sets (which include all the correct repairs) are randomly distributed in the proximity of, but not exactly the same as, the optimal results from the weighted-independent set problem.

The observation inspires us to seek approximate solutions to the repair selection problem instead. Many heuristic algorithms have been proposed for the weighted-independent set problem (see [5] for a survey). Here we propose a greedy algorithm named *maximum-effectiveness first* (EMAX), which gives superior empirical results in our settings. As shown in Algorithm 3, the EMAX algorithm always selects from $\mathcal{G}_r$ a vertex whose corresponding repair is evaluated to be the most effective according to Equation (3), and then discards its adjacent vertices until there is no vertex left.

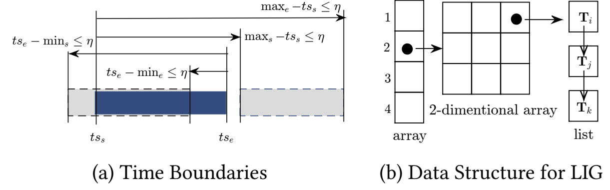
Algorithm 3 The EMAX algorithm

Input: the repair graph $\mathcal{G}_r = (\mathbf{V}_r, \mathbf{E}_r)$.
Output: a set of selected vertices $\mathcal{V}$.

```
1: sort vertices in V_r by the ω values of their corresponding
   repairs in a decreasing order
2: for all v in V_r do
3:     if v.discard = false then
4:         add v to 𝒱
5:         for all v_a adjacent to v do
6:             v_a.discard ← true
7: return 𝒱
```

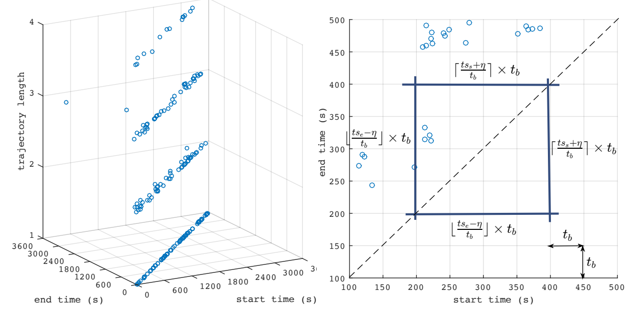
Compared with the exact algorithm that requires exponential running time, the EMAX algorithm runs in only $O(|\mathbf{V}_r| \log |\mathbf{V}_r|)$ time. The rationale behind this heuristic is that, repairs evaluated to be more effective are more likely to be correct, and thus giving them priorities them makes it more likely to select the best result.

Example 4.2. For the repair graph shown in Figure 4(b), the EMAX algorithm first selects the vertex v'_3 corresponding to candidate repair R_3 (since it is the most effective according to Equation (3)) and then discards v'_2 adjacent to v'_3. Since the effectiveness of R_1, the only vertex left, is zero, v'_1 will not be selected.



(a) Time Boundaries (b) Data Structure for LIG

Figure 6: Time Boundaries and the Data Structure for LIG



(a) A Three-Dimensional View (b) A Slice of LIG with Length 3

Figure 7: Overview of the Length-Indexed Grids

5 OPTIMIZATION

In this section, we provide some optimization methods to make the repair approach more efficient.

5.1 An Index for Constructing $\mathcal{G}_m$

The candidate repair generation phase requires evaluating the *cex* predicate on each pair of trajectories in $\mathcal{T}$ for constructing the trajectory graph $\mathcal{G}_m$. Suppose that there are $|\mathbf{V}_m|$ trajectories. This procedure requires $O(|\mathbf{V}_m|^2)$ comparisons, which could be costly in practice. Recall that valid trajectories are upper-bounded in length and time span, and we thus make use of the bounds θ and η to filter out some unnecessary comparisons.

Definition 5.1. **Start Time** and **End Time**. The *start time* and the *end time* of a trajectory $\mathbf{T}$ are defined as the timestamps of the earliest and latest records in $\mathbf{T}$, respectively.

Given a trajectory $\mathbf{T}_k$ with start time ts_s and end time ts_e, another trajectory $\mathbf{T}_u$ that may constitute a joinable subset with $\mathbf{T}_k$ must first meet the length criterion, i.e., $|\mathbf{T}_u| \le \theta - |\mathbf{T}_k|$. Also, for the bound on time span η, the max/min start time (denoted by max_s and min_s) and max/min end time (denoted by max_e and min_e) of $\mathbf{T}_u$ should satisfy the following inequalities (which are demonstrated in Figure 6(a)): $ts_e - min_s \le \eta$, $max_s - ts_s \le \eta$, $ts_e - min_e \le \eta$, and $max_e - ts_s \le \eta$. According to these inequalities, both the start time and end time of $\mathbf{T}_u$ should fall in $[ts_e - \eta, ts_s + \eta]$, and we can transform the criteria into a range query on a three-dimensional index structure on the trajectories called Length-Indexed Grids (LIG).

Overview. As shown in Figure 7(a), the three dimensions of LIG are the length, the start time and the end time of a trajectory. Specifically, we divide the time span of interest along both the start time and end time dimensions into time bins with fixed size t_b, resulting in a two-dimensional time grid shown in Figure 7(b). A separate time grid is created for each trajectory length appearing in the dataset.

The Data Structure. Figure 6(b) illustrates the data structure of LIG. An array is used to store the grids with different trajectory lengths. Each grid is actually a two-dimensional array. Trajectories are distributed to grids according to their start/end times and trajectories in the same grid are linked to be an element of

the two-dimensional array. We construct LIG by successively add trajectories. For each trajectory, we first decide the grid it belongs to according to the trajectory's length. Then we assign a time grid for the trajectory and add it as a new element to the tail of the corresponding list.

Usage. We use the index to answer the range query by first deciding a set of feasible grids according to the trajectory's length and the threshold θ. After that, in each grid, we select trajectory lists that meet the start/end time restrictions from the two-dimensional array. Without loss of generality, suppose that the timestamps of tracking records are represented as offsets to the earliest timestamp in the dataset. Then the target trajectories we are interested in should be contained in elements whose indices are bounded by $[\lfloor \frac{ts_s - \eta}{t_b} \rfloor \times t_b, \lceil \frac{ts_e + \eta}{t_b} \rceil \times t_b]$ in both dimensions.

With the index technique provided above, we can prune many useless trajectory comparisons. As the time grids are static, the index can be constructed efficiently in $\Theta(|\mathbf{V}_m|)$ time. As such, the running time for generating $\mathcal{G}_m$ can be significantly reduced.

5.2 A Pruning Method for Clique Generation

In Section 4.1.2, we show how to generate qualified cliques from the trajectory graph $\mathcal{G}_m$. All the trajectory sets corresponding to the cliques will be further checked by the *jnb* predicate to see if they are really joinable subsets. Considering that Algorithm 2 is output-sensitive, it will be more efficient if we can eliminate some worthless vertex combinations early on during the clique generation process. We propose an optimization method named *minimum cover prefix pruning* for this purpose.

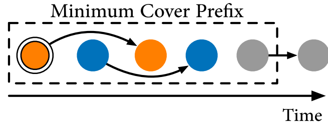


Figure 8: The Minimum Cover Prefix

Definition 5.2. **Minimum Cover Prefix**. As shown in Figure 8, given a trajectory set $\mathcal{T} = \{\mathbf{T}_1, \mathbf{T}_2, \cdots, \mathbf{T}_p\}$, we can merge their tracking records chronologically to get a sequence $\mathbf{r}_1 \to \mathbf{r}_2 \to \cdots \mathbf{r}_q$. The *minimum cover prefix* (abbreviated as MCP) for $\mathcal{T}$ is defined as the minimum prefix of this sequence that contains at least one tracking record from all trajectories in $\mathcal{T}$.

THEOREM 5.3. ***The MCP condition.*** *Given a list of trajectories $[\mathbf{T}_1, \mathbf{T}_2, \cdots, \mathbf{T}_p]$ sorted by their start times in an increasing order (i.e., $\mathbf{T}_i.startTime \le \mathbf{T}_{i+1}.startTime$), a necessary but not sufficient condition for these trajectories to compose a joinable subset is that the location sequence for the MCP of any $\{\mathbf{T}_1, \mathbf{T}_2, \cdots, \mathbf{T}_{p-k}\}$ $(0 \le k < p)$ must be a prefix of a valid path in $\mathcal{G}_t$.*

According to Theorem 5.3, when generating cliques, if the vertices are added to C in an increasing order of the start times of their corresponding trajectories, we can prune some unnecessary vertex combinations according to the MCP condition. The checking is performed with a *pck* predicate.

The *pck* predicate. Similar to the *cex* and *jnb* predicates, given a trajectory graph $\mathcal{G}_t$, the *pck* predicate can be applied on one or more trajectories and evaluates to *true* iff the MCP condition holds, i.e., $\{\mathcal{T}_x | (pck(\mathcal{T}_x))\} = \{\mathcal{T}_x | \exists \mathbf{P}(\mathbf{r}_1.loc \to \cdots \to \mathbf{r}_k.loc = prefix(\mathbf{P})), [\mathbf{r}_1, \cdots, \mathbf{r}_k]$ is the MCP of $\mathcal{T}_x$ and $prefix(\mathbf{P})$ is the prefix of a valid path $\mathbf{P}$ in $\mathcal{G}_t\}$. In terms of "restrictiveness", this predicate falls somewhere between the *cex* and *jnb* predicates. Compared with *cex*, it further requires that the location sequence must be a prefix of a valid path, not just a subsequence; compared with *jnb*, it just ensures that the first location is an entrance location. Due to space limitation, the detailed algorithm for evaluating this predicate is omitted.

With the *pck* predicate defined, we try to modify the qualified-clique generation algorithm by pruning worthless results and recursions. First of all, we must ensure that the cliques are generated in the order of their trajectories' start times. Fortunately, Algorithm 2 iterates through the vertices with an index list L. Thus, to keep the generation order, we just need to sort the vertices in $\mathcal{G}_m$. Then, each time before outputting a generated clique to the result set, we check its corresponding trajectory set with the *pck* predicate, and only if it evaluates to *true*, can we accept the clique and continue adding more vertices into the result set. The modified algorithm snippet is shown in Algorithm 4.

Algorithm 4 Clique generation with pruning

```
  ...
  sort vertices in G_m by their corresponding trajectories' start
  times in descending order
  function CLIQUE(C, L)
      ...
      if pck(C.trajectories) = true then
          results ← results ∪ {C}
          if ¬L_new.empty() and C.RecordNumber < θ and
  |C| < ζ then
              CLIQUE(C, L_new)
      ...
```

Example 5.4. Suppose that the vertices in Figure 5(a) are already sorted by the start times of their corresponding trajectories, i.e., $\mathbf{T}_5.startTime \le \mathbf{T}_4.startTime \le \cdots \le \mathbf{T}_1.startTime$. If the MCP condition does not hold on $\{\mathbf{T}_5\}$, any cliques containing v_5 (e.g., $\{v_5, v_4\}$ and $\{v_5, v_4, v_1\}$) will be pruned by the modified algorithm. For the same reason, if the MCP condition does not hold on $\{\mathbf{T}_5, \mathbf{T}_2\}$, the cliques $\{v_5, v_2\}$ and $\{v_5, v_2, v_1\}$ will be pruned. Obviously, the modified algorithm is more efficient thanks to the pruning of some cliques and unnecessary calculations.

6 EXPERIMENTS

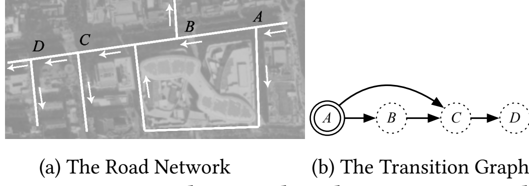
We conduct extensive experiments on both real and synthetic datasets to thoroughly evaluate the properties of the proposed approach and compare it to baseline methods.

6.1 Experimental Settings

All algorithms are implemented in Java and run on a desktop PC with a 2.5GHz Intel i5 CPU and 8GB of memory. Each set of experiments are repeated at least 30 times and the average results are recorded.

A real dataset and a series of synthetic datasets with different characteristics are used in the experiments. According to Section 2.3, the repair approach we proposed in this paper is in the interest of local regions. For such small regions, the transition graphs may seem simple. However, note that even for such seemingly simple graphs, the repair problem is still quite challenging, as revealed in Section 3.

6.1.1 Datasets. **Real Dataset**. The real dataset is obtained from a real traffic surveillance system in a provincial capital in China. We choose a specific region of this city and extract 699 trajectories of vehicles which contain 2,045 tracking records between 8:00 a.m. and 9:00 a.m. on a particular day. Figure 9(a) illustrates the road network and the distribution of surveillance



(a) The Road Network (b) The Transition Graph

Figure 9: Road Network and Transition Graph

cameras in this region. The license plate numbers of the vehicles are captured by cameras located at A, B, C, and D whenever they pass by these sites. Figure 9(b) is the corresponding transition graph we derived. Due to OCR errors and other issues, some of the license plates in the dataset were misidentified. We manually label the plate numbers by examining the *original photos* taken by the cameras, which serves as the ground truth. In this way, we obtain a labeled dataset that contains both the raw and the true values. The default values of θ, η, ζ and λ for the real dataset are empirically set to 4, 600 seconds, 4, and 0.5, respectively, unless otherwise specified.

Synthetic Datasets. To generate a synthetic trajectory set for ID repair, we first choose a transition graph, based on either the real dataset or a sample of the California road network [21]. Then we repeatedly sample random valid paths and generate corresponding trajectories until we have obtained the desired number of trajectories. Without loss of generality, we assume that an ID consists of 7 to 9 lower-case letters only, which are independently and identically generated following a uniform distribution. The time span is sampled from the empirical distribution of travel time between the corresponding locations in the real dataset. After that, using the edit distance distribution for erroneous IDs in the real dataset as a ballpark, we randomly inject ID errors to the tracking records with a specified error rate, and eventually get a synthetic dataset. The default error rate is set to 20%, unless otherwise specified.

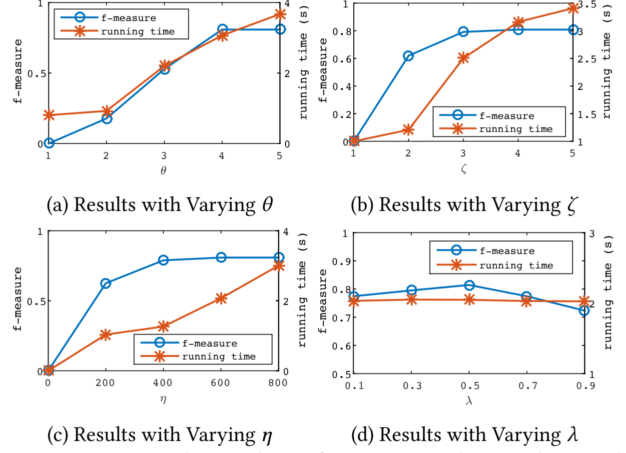
6.1.2 Metrics. We use elapsed time as the metric for efficiency, and adopt *precision*, *recall* and *f-measure* as the general metrics for effectiveness. Using $\mathcal{T}_e$ to represent all the trajectories with ID errors, $\mathcal{T}_r$ to represent those trajectories with ID rewritten by applying candidate repairs, and $\mathcal{T}_c$ to represent all the trajectories whose IDs are correctly repaired, we define recall $= |\mathcal{T}_c|/|\mathcal{T}_e|$, precision $= |\mathcal{T}_c|/|\mathcal{T}_r|$, and f-measure $= \frac{2\cdot(\text{precision}\cdot\text{recall})}{(\text{precision}+\text{recall})}$. There are also some specialized metrics used in certain groups of experiments, which will be introduced later.

6.2 Effects of Parameters

We first evaluate the effects of different parameters through a group of experiments on the real dataset.

The effects of θ, ζ, and η. Figures 10(a), 10(b) and 10(c) show the f-measure and running time with varying values of θ, ζ, and η, respectively, with all other parameter values fixed at their default values. We observe that for each of these parameters, the running time grows with increasing parameter values. For the f-measure, it initially increases as well, but eventually flattens out. This verifies our earlier observation that for a particular dataset, there exist bounds on these parameters, beyond which no further gains in repair effectiveness can be achieved. Thus, by carefully choosing the bounds, we can reduce the running time of the repair process significantly.

The effect of λ. Figure 10(d) shows the effect of λ in Equation (3). With λ varying from 0.1 to 0.9, the running time remains stable, and the f-measure first increases and then decreases after



(a) Results with Varying θ (b) Results with Varying ζ

(c) Results with Varying η (d) Results with Varying λ

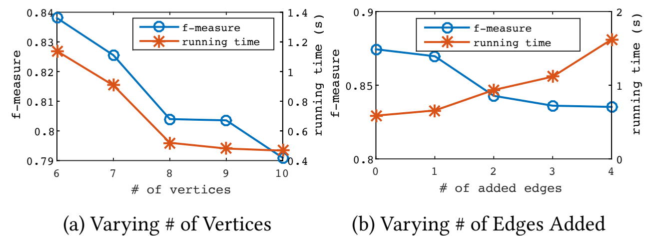
Figure 10: Results with Different Bounds on the Real Dataset

$\lambda = 0.5$. The results imply that (1) there exists an optimal λ value with which the best results can be produced, and (2) the repair results are not sensitive to changes in λ.

6.3 Effects of Data Characteristics

We next conduct a set of experiments using synthetic datasets to investigate the impact of different data characteristics. All the datasets used in this set of experiments are produced based on 500 original trajectories (before injecting errors). The actual number of trajectories in a dataset is affected by different parameters (e.g., error rate and record missing rate), and will be shown for different groups of experiments. The default values of θ, η, ζ and λ for the synthetic datasets used here are set to 8, 600 seconds, 4, and 0.5, respectively.

6.3.1 Size and Density of the Transition Graph. The first data characteristics we explore are the size (number of vertices) and density (number of edges) of the transition graph. The experiments are conducted on synthetic trajectory sets generated from transition graphs with different sizes and different densities. We vary the density of a transition graph with 8 vertices $\mathcal{G}_t = (\mathbf{V}, \mathbf{E}, \mathbf{I}, \mathbf{O})$, where $\mathbf{V} = \{loc_1, loc_2, \cdots, loc_8\}$, $\mathbf{E} = \{(loc_1, loc_2), (loc_2, loc_3), \cdots, (loc_7, loc_8)\}$, $\mathbf{I} = \{loc_1\}$, and $\mathbf{O} = \{loc_8\}$, by randomly adding a specific number of edges (without duplicate) to it.



(a) Varying # of Vertices (b) Varying # of Edges Added

Figure 11: Effect of the Size and Density of Transition Graphs

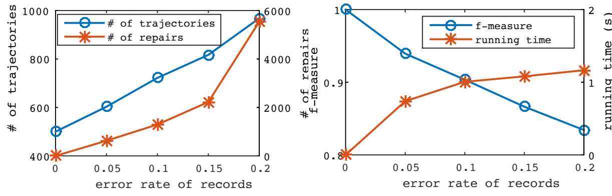
The effect of graph size. Figure 11(a) shows the results on varying transition graph sizes. It is evident that both the f-measure and the running time decrease with the number of vertices increasing. That is because a transition graph with more vertices tends to have longer valid paths, and the longer the valid paths are, the harder it is to form candidate repairs that could "reassemble" the original trajectory;

The effect of graph density. The results for adding varying number of edges to a given transition graph are shown in 11(b).

The f-measure decreases while the running time increases with more edges added, due to the following reasons: (1) adding edges to the transition graph will increase the number of valid paths and thus there will be more candidate repairs; (2) with the number of candidate repairs growing, there may be more false positive repairs (vertices) being selected and that will cause the f-measure to deteriorate; and (3) having more candidate repairs also leads to longer candidate generation and selection time, resulting in an increase in the total running time.

The results above imply that our ID repair approach is more suitable for sparse transition graphs with limited number of vertices, which is actually the case in many, if not most, application scenarios. This is also consistent with our assumptions and analysis made in Section 2.3.

6.3.2 ID Error Rate. To evaluate the effect of the ID error rate, we create a cohort of synthetic datasets by randomly injecting ID errors, each time with a different error rate, into an identical original trajectory set.

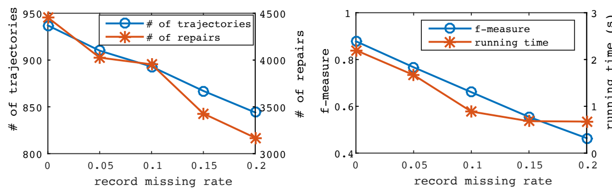


(a) # of Trajectories/Repairs (b) F-Measure and Running Time

Figure 12: Effect of ID Error Rate

The experiment results are reported in Figures 12(a) and 12(b), from which we can observe that with the error rate increasing, (1) the number of trajectories for the input dataset increases linearly; (2) both the number of candidate repairs and the running time increase polynomially; and (3) the f-measure drops near linearly. The reason is as follows. Since ID errors can cause a trajectory to break into multiple pieces, the input number of trajectories grows linearly with respect to the error rate. Both the number of candidate repairs and the running time also increase accordingly. The f-measure drops mainly because intuitively it is more difficult to "reassemble" the original trajectory with more IDs misidentified. Also, recall that our repair approach assumes that all the correct IDs exist in the dataset, which may no longer hold if the error rate gets high. In summary, the lower the ID error rate is, the better our repair approach works.

6.3.3 Record Missing Rate. As mentioned in Section 1, in this work we only consider errors caused by ID misidentification, ignoring the effect of missing records. In practice, however, there may be a slight chance of record missing from the dataset. We thus conduct experiments to evaluate whether this has a significant impact on the effectiveness of the proposed approach. To this end, we first generate a synthetic dataset and then randomly remove records from it with varying record missing rates.



(a) # of Trajectories/Repairs (b) F-Measure and Running Time

Figure 13: Effect of Record Missing

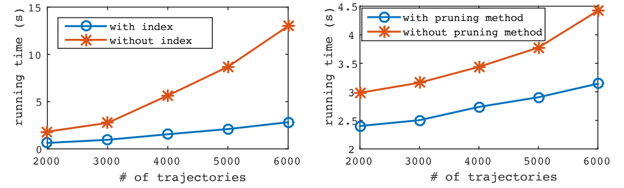
As illustrated in Figures 13(a) and 13(b), with the missing rate increasing from 0% to 20%, all the metrics decrease. The reason is that (1) record missing will make some joinable subsets incomplete and thus cannot compose the corresponding candidate repairs (this is verified by the decrease of candidate repairs shown in Figure 13(a)); (2) trajectories belonging to different entities may be joined due to the absence of some trajectories; and (3) records containing the true ID for an entity may have all been removed, which makes some errors irreparable.

According to the experiment result, although having missing records has a notable impact on the effectiveness of the proposed ID repair approach, it is still applicable for datasets with relatively low record missing rates.

6.4 Effectiveness of the Optimization Methods

The main purpose of the next group of experiments is to explore the performance improvements brought by the Length-Indexed Grids (in Section 5.1), as well as the pruning method (in Section 5.2).

We conduct the experiments on synthetic datasets with the number of trajectories varying from 2,000 to 6,000 and the corresponding number of records varying from 5,189 to 15,795. All the datasets are generated using the same transition graph as that for the real dataset.



(a) Construction Time for $\mathbf{G}_m$ with Different Data Sizes (b) Running Time with Different Data Sizes

Figure 14: Effectiveness of Optimization Methods

Figure 14(a) shows the running time of the trajectory graph construction process with different number of trajectories. From this figure we can observe that without indexing, the construction time of $\mathcal{G}_m$ grows superlinearly with the number of trajectories, whereas the trend becomes almost linear with the Length-Indexed Grids. This observation indicates that the Length-Indexed Grids can help eliminate a large number of unnecessary trajectory comparisons.

Figure 14(b) reports the running time of the whole repair process with the number of trajectories varying from 2,000 to 6,000. We can see that the time increases polynomially with the number of trajectories. Besides, compared with the basic clique generation algorithm, algorithm with the pruning method can reduce about 30% running time.

6.5 Comparison with Competing Approaches

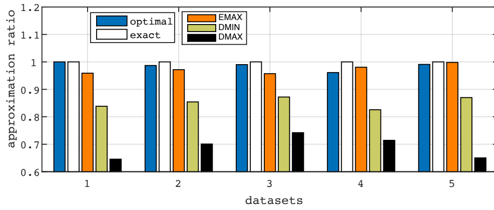
To evaluate the effectiveness of our proposed method, we compare it with other approaches that use different repair selection algorithms or exploit different constraints.

6.5.1 Alternative Repair Selection Algorithms. In this set of experiments, we aim to investigate the performance of different algorithms for the repair selection phase. In addition to EMAX and the exact algorithms introduced in Section 4.2, we also implement three other algorithms for comparison. The first algorithm,

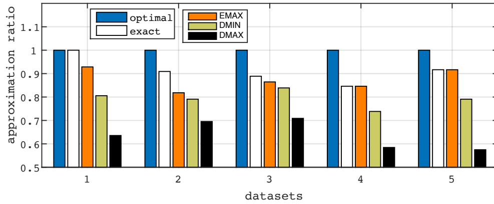
named *optimal selection*, is an oracle machine based algorithm that always selects and applies correct candidate repairs regardless of their ω values. Theoretically, this algorithm can achieve the highest quality improvement. The second and the third algorithms are minimum degree first (DMIN) and maximum degree first (DMAX). As their names suggest, they select the vertex with the minimum/maximum degree from $\mathcal{G}_r$ in each step and discard adjacent vertices until there is no vertex left.

As the exact algorithm for weighted-independent set is time consuming, the experiments are conducted on 5 small synthetic datasets whose sizes do not exceed 100. Even so, the average running time for the exact algorithm is still thousands times longer than the other algorithms. Thus we only report on their effectiveness rather than the performance.

To measure the real quality of a dataset, we employ the metric *trajectory accuracy*, which is defined as the ratio of trajectories with correct IDs. Thereby, the real quality improvement after repairing can be measured by the increment in this metric. As trajectory merging can change the data size, we will only perform ID rewritings. Using ΔE (ΔA) and ΔE_{max} (ΔA_{opt}) to represent the selected Ω value (trajectory accuracy improvement) and the maximum selected Ω value (maximum trajectory accuracy improvement), the approximation ratio for maximum Ω value selection and data quality improvement can be calculated by $\Delta E/\Delta E_{max}$ and $\Delta A/\Delta A_{opt}$.



(a) Approximation Ratios for Repairs Selection



(b) Approximation Ratios for Quality Improvement

Figure 15: Approximation Ratios for Different Selection Algorithms on Synthetic Datasets

Figures 15(a) and 15(b) report the experiment results of maximum Ω value selection and data quality improvement, from which we can observe that (1) the selected Ω value can reflect the data quality improvement well, (2) the total selected Ω value for the optimal selection algorithm is randomly distributed around, rather than always coincides with, the maximum value, and (3) remarkably, the proposed EMAX algorithm can achieve an average approximation ratio of more than 0.95 and 0.85 for repair selection and real data quality improvement respectively, which significantly beats the other two heuristic algorithms. In summary, as the optimal selection algorithm is evasive in practice and the exact algorithm is time-consuming, the proposed EMAX algorithm seems highly promising.

6.5.2 Comparison with Other Repair Approaches. To evaluate our proposed ID repair approach, we implement a baseline approach based on ID similarity = 3, i.e., trajectories with ID similarity $\leq$ 3 are considered to come from the same entity and thus will be merged. Also, we implement another greedy heuristic method based on neighborhood constraints proposed in [27]. We take the transition graph $\mathcal{G}_t$ as the constraint graph and the trajectory graph $\mathcal{G}_m$ as the instance graph. The cost function is set to be the edit distance of two ID strings. To make sure the algorithm terminates, we add a variation to the approach that edges are allowed to be removed from $\mathcal{G}_m$ during relabeling.

The repair results of the three approaches are shown in Figure 16, from which we can observe that (1) while the precision of the other competing approaches is somewhat close to our proposed approach, their recall is significantly lower; and (2) the neighborhood constraint based method performs even worse than the baseline method for our problem. The recall of the ID similarity based approach is better than that of the neighborhood constraint based approach because it supports "partial recovery" of the original trajectories. Actually, both the ID similarity based approach and the neighborhood constraint based approach are binary constraints that only consider the relationship between trajectories pairs. In contrast, our transition graph based approach considers the relationships between multiple trajectories, which is why it can cover more correct repairs.

7 RELATED WORK

There has been a sizable body of work in the areas of data repair and data matching that can be considered related to our work, which we summarize below.

7.1 Data Repair

Most previous work on data repair has focused on relational data by exploiting the different types of dependencies [1], e.g., matching dependencies [12],differential dependencies [26], and order dependencies [28]. Fan et al. extend the functional and inclusion dependencies with conditions [6, 13] and also extend their data inconsistency detection method to distributed environments [14]. Although highly successful, most of the work has not considered spatial and temporal factors.

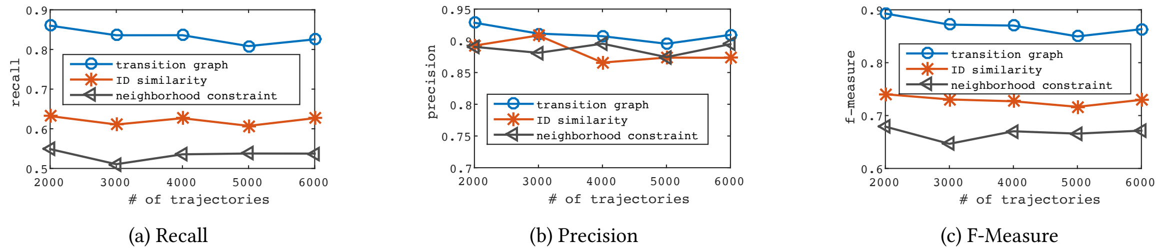
Moreover, sequential dependencies [18] are developed to constrain attributes' transitions. Song et al. use neighborhood constraints to repair vertex labels in graphs [27]. Wang et al. employ the Petri Net to repair the names of event logs [29]. Similar to our study that focuses on repairing the IDs, Song et al. propose a method for cleaning timestamps facilitated by temporal constraints [25].

For unified approaches, Ilyas et al. propose a novel holistic repairing algorithm [8], as well as a general system [11] that puts multiple constraints into consideration and repairs them all at once. Similarly, Geerts et al. develop a uniform data-cleaning framework with a cell group and partial order based cleaning semantics [17]. However, those methods cannot be trivially adopted, since it is difficult to transform the constraints posed by transition graphs in our setting into the denial constraints or equality-generating dependencies required by those methods.

7.2 Data Matching

In the field of data matching, Yakout et al. try to identify the same entity in different transactions by detecting regularity patterns from merged behavior logs [32]. Similarly, Zhu et al. perform heterogeneous event matching by finding an optimal mapping that can maximize the frequency similarity of patterns [33].

When patterns are not explicitly given, Li et al. propose a temporal model, as well as an algorithm to perform temporal



(a) Recall (b) Precision (c) F-Measure

Figure 16: Comparisons with Other Repair Approaches

records clustering [22]. They utilize both the usual similarity metrics and the temporal model with collected evidences to make the decision. Chiang et al. extend their work and develop a two-phase method called "static first, dynamic second" to reduce the complexity of the temporal model [7]. Also, they use signatures to improve the computing efficiency. Note that both their work and ours are to identify entities by exploring their transitions. The main difference is that while their work mainly focuses on when the state attributes of entities should change, we focus on how (through which paths) the entities pass through the area of interest.

8 CONCLUSIONS AND FUTURE WORK

We have studied a novel problem of repairing erroneous IDs in spatio-temporal trajectories with transition graphs. A two-phase repair paradigm, which includes candidate repair generation and compatible repair selection, is proposed to address this problem. Since both phases are intractable in general, we exploit the locality and sparsity properties of trajectories and present efficient solutions in restricted but practical scenarios. For the candidate repair generation phase, we propose a backtracking algorithm, as well as a pruning method to speed it up. For the candidate repair selection phase, we present a practical greedy algorithm. Extensive experiments are conducted on both real and synthetic data to study the effects of various parameters and data characteristics. In addition, we compare our proposed approach with some baseline methods and the results have confirmed its effectiveness.

One possible direction for future work would be to deploy our algorithms on some distributed repair systems with UDF support [20]. It would also be interesting to study solutions that could perform ID repair as the tracking records stream in.

ACKNOWLEDGMENTS

This work was supported in part by the National Basic Research 973 Program of China under Grant No. 2015CB352502, the National Natural Science Foundation of China under Grant Nos. 61272092 and 61572289, the Natural Science Foundation of Shandong Province of China under Grant No ZR2015FM002, and the NSERC Discovery Grants.

REFERENCES

[1] Serge Abiteboul, Richard Hull, and Victor Vianu. 1995. *Foundations of databases*. Vol. 8.
[2] S Sheik Mohammed Ali, Boby George, Lelitha Vanajakshi, and Jayashankar Venkatraman. 2012. A multiple inductive loop vehicle detection system for heterogeneous and lane-less traffic. *IEEE Transactions on Instrumentation and Measurement* 61, 5 (2012), 1353–1360.
[3] Jorge A Alves. 2001. *Recognition of ship types from an infrared image using moment invariants and neural networks.* Technical Report. NAVAL POSTGRADUATE SCHOOL MONTEREY CA.
[4] Philip Bohannon, Michael Flaster, Wenfei Fan, and Rajeev Rastogi. 2005. A Cost-Based Model and Effective Heuristic for Repairing Constraints by Value Modification. In *SIGMOD*. 143–154.
[5] Immanuel M Bomze, Marco Budinich, Panos M Pardalos, and Marcello Pelillo. 1999. The maximum clique problem. In *Handbook of combinatorial optimization*. 1–74.
[6] Loreto Bravo, Wenfei Fan, and Shuai Ma. 2007. Extending Dependencies with Conditions. In *VLDB*. 243–254.
[7] Yueh-Hsuan Chiang, AnHai Doan, and Jeffrey F. Naughton. 2014. Tracking Entities in the Dynamic World: A Fast Algorithm for Matching Temporal Records. *PVLDB* 7, 6 (2014), 469–480. http://www.vldb.org/pvldb/vol7/p469-chiang.pdf
[8] Xu Chu, Ihab F. Ilyas, and Paolo Papotti. 2013. Holistic data cleaning: Putting violations into context. In *ICDE*. 458–469.
[9] Dong Deng, Guoliang Li, Jianhua Feng, and Wen-Syan Li. 2013. Top-k string similarity search with edit-distance constraints. In *ICDE*. 925–936.
[10] Shan Du, Mahmoud Ibrahim, Mohamed Shehata, and Wael Badawy. 2013. Automatic license plate recognition (ALPR): A state-of-the-art review. *IEEE Transactions on circuits and systems for video technology* 23, 2 (2013), 311–325.
[11] Amr Ebaid, Ahmed Elmagarmid, Ihab F Ilyas, Mourad Ouzzani, Jorge-Arnulfo Quiane-Ruiz, Nan Tang, and Si Yin. 2013. NADEEF: A generalized data cleaning system. *PVLDB* 6, 12 (2013), 1218–1221.
[12] Wenfei Fan. 2008. Dependencies revisited for improving data quality. In *PODS*. 159–170.
[13] Wenfei Fan, Floris Geerts, Xibei Jia, and Anastasios Kementsietsidis. 2008. Conditional functional dependencies for capturing data inconsistencies. *ACM Trans. Database Syst.* 33 (2008).
[14] Wenfei Fan, Floris Geerts, Shuai Ma, and Heiko Müller. 2010. Detecting inconsistencies in distributed data. In *ICDE*. 64–75.
[15] Joao C Ferreira, Jorge Branquinho, Paulo Chaves Ferreira, and Fernando Piedade. 2017. Computer Vision Algorithms Fishing Vessel MonitoringâĂŤIdentification of Vessel Plate Number. In *International Symposium on Ambient Intelligence*. Springer, 9–17.
[16] Robert W Floyd. 1962. Algorithm 97: shortest path. *Commun. ACM* 5, 6 (1962), 345.
[17] Floris Geerts, Giansalvatore Mecca, Paolo Papotti, and Donatello Santoro. 2013. The LLUNATIC data-cleaning framework. *PVLDB* 6, 9 (2013), 625–636.
[18] Lukasz Golab, Howard J. Karloff, Flip Korn, Avishek Saha, and Divesh Srivastava. 2009. Sequential Dependencies. *PVLDB* 2, 1 (2009), 574–585.
[19] Richard M. Karp. 1972. Reducibility Among Combinatorial Problems. In *Proceedings of a Symposium on the Complexity of Computer Computations*. 85–103. http://www.cs.berkeley.edu/~luca/cs172/karp.pdf
[20] Zuhair Khayyat, Ihab F Ilyas, Alekh Jindal, Samuel Madden, Mourad Ouzzani, Paolo Papotti, Jorge-Arnulfo Quiané-Ruiz, Nan Tang, and Si Yin. 2015. Bigdansing: A system for big data cleansing. In *SIGMOD*. ACM, 1215–1230.
[21] Jure Leskovec and Andrej Krevl. 2014. SNAP Datasets: Stanford Large Network Dataset Collection. http://snap.stanford.edu/data. (June 2014).
[22] Pei Li, Xin Luna Dong, Andrea Maurino, and Divesh Srivastava. 2011. Linking Temporal Records. *PVLDB* 4, 11 (2011), 956–967.
[23] R Garey Michael and S Johnson David. 1979. Computers and intractability: a guide to the theory of NP-completeness. *WH Free. Co., San Fr* (1979), 90–91.
[24] Sunita Sarawagi and Alok Kirpal. 2004. Efficient set joins on similarity predicates. In *SIGMOD*. 743–754.
[25] Shaoxu Song, Yue Cao, and Jianmin Wang. 2016. Cleaning timestamps with temporal constraints. *PVLDB* 9, 10 (2016), 708–719.
[26] Shaoxu Song and Lei Chen. 2011. Differential dependencies: Reasoning and discovery. *ACM Trans. Database Syst.* 36, 3 (2011), 16.
[27] Shaoxu Song, Hong Cheng, Jeffrey Xu Yu, and Lei Chen. 2014. Repairing Vertex Labels under Neighborhood Constraints. *PVLDB* 7, 11 (2014), 987–998.
[28] Jaroslaw Szlichta, Parke Godfrey, Jarek Gryz, and Calisto Zuzarte. 2013. Expressiveness and Complexity of Order Dependencies. *PVLDB* 6, 14 (2013), 1858–1869.
[29] Jianmin Wang, Shaoxu Song, Xuemin Lin, Xiaochen Zhu, and Jian Pei. 2015. Cleaning structured event logs: A graph repair approach. In *ICDE*. 30–41.
[30] Jianmin Wang, Shaoxu Song, Xiaochen Zhu, and Xuemin Lin. 2013. Efficient Recovery of Missing Events. *PVLDB* 6, 10 (2013), 841–852.
[31] Andreas Wombacher. 2011. A-posteriori detection of sensor infrastructure errors in correlated sensor data and business workflows. *Business Process Management* (2011), 329–344.
[32] Mohamed Yakout, Ahmed K. Elmagarmid, Hazem Elmeleegy, Mourad Ouzzani, and Alan Qi. 2010. Behavior Based Record Linkage. *PVLDB* 3, 1 (2010), 439–448.
[33] Xiaochen Zhu, Shaoxu Song, Jianmin Wang, Philip S. Yu, and Jiaguang Sun. 2014. Matching heterogeneous events with patterns. In *ICDE*. 376–387.