SPARCLE: Boosting the Accuracy of Data Cleaning Systems through Spatial Awareness

Yuchuan Huang
huan1531@umn.edu
University of Minnesota, USA

Mohamed F. Mokbel
mokbel@umn.edu
University of Minnesota, USA

ABSTRACT

Though data cleaning systems have earned great success and wide spread in both academia and industry, they fall short when trying to clean spatial data. The main reason is that state-of-the-art data cleaning systems mainly rely on functional dependency rules where there is sufficient co-occurrence of value pairs to learn that a certain value of an attribute leads to a corresponding value of another attribute. However, for spatial attributes that represent locations, there is very little chance that two records would have the same exact coordinates, and hence co-occurrence is unlikely to exist. This paper presents SPARCLE (SPatially-AwaRe CLEaning); a novel framework that injects spatial awareness into the core engine of rule-based data cleaning systems through two main concepts: (1) *Spatial Neighborhood*, where co-occurrence is relaxed to be within a certain spatial proximity rather than same exact value, and (2) *Distance Weighting*, where records are given different weights of whether they satisfy a dependency rule, based on their relative distance. Experimental results using a real deployment of SPARCLE inside a state-of-the-art data cleaning system, and real and synthetic datasets, show that SPARCLE significantly boosts the accuracy of data cleaning systems when dealing with spatial data.

PVLDB Reference Format:
Yuchuan Huang and Mohamed F. Mokbel. SPARCLE: Boosting the Accuracy of Data Cleaning Systems through Spatial Awareness. PVLDB, 17(9): 2349-2362, 2024.
doi:10.14778/3665844.3665862

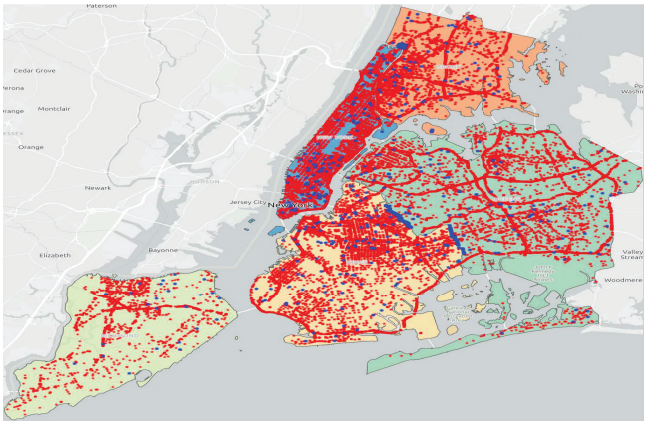
PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/yhuang-db/holoclean-sparcle.

1 INTRODUCTION

Motivated by the imperfection of real data sets, along with the huge efforts carried by data scientists to manually clean their data, efforts have been dedicated to develop various approaches and systems for automated data cleaning. The large majority of such approaches (e.g., see [6, 11, 13, 21, 32, 43]) and systems (e.g., see [14, 19, 20, 28, 34, 44, 45]) are rule-based, where functional dependencies between various attributes guide the data cleaning process. The success and immense need of such data cleaning systems made it widely adopted by industry [2, 18, 23, 36] and commercial startups [27, 50, 51]. Unfortunately, with all its success and wide spread, state-of-the-art data cleaning systems fall short when trying to clean data with spatial attributes dependency. As an example, we investigated the NYC Motor Vehicle Collision data [40], which includes 1,751,624 collision records that took place in the New York City since 2014. A snapshot of this dataset is in Figure 1(a) for five collision records and only four attributes of each collision (*ID*, *Latitude*, *Longitude*, *Borough*). The snapshot shows two kinds of errors: (1) the second record is missing the *Borough* information, and (2) the fourth record has the wrong *Borough* information. To get an idea of the scale of the problem, Figure 1(b) plots all the erroneous records over NYC map (421,013 records), where 418,896 records have a missing borough (plotted in red) and 2,117 records have incorrect borough (plotted in blue). We fed this data along with the functional dependency: (*Latitude*, *Longitude*) → *Borough* to HoloClean [44] system as a state-of-the-art rule-based data cleaning framework. HoloClean only repaired 58.7% of the errors, which is a pretty low accuracy compared to its ability in cleaning non-spatial data with more than 95% accuracy [44]. To understand such poor

ID	Latitude	Longitude	Borough	...
4486519	40.751441	-73.973974	Manhattan	...
4486555	40.690183	-73.956825		...
4486604	40.681582	-73.974638	Brooklyn	...
4486609	40.683304	-73.917274	Queens	...
4486660	40.868165	-73.831487	Bronx	...
...	...	...	...	...

(a) Part of Table of NYC Motor Vehicle Collision Data



(b) Map of NYC Motor Vehicle Collision Data

Figure 1: NYC Motor Vehicle Collision Data

This work is supported by NSF under grants IIS-2203553 and OAC-2118285.

Proceedings of the VLDB Endowment, Vol. 17, No. 9 ISSN 2150-8097.
doi:10.14778/3665844.3665862

Table 1: Error Repairing of NYC Motor Vehicle Collision Data

	HoloClean	Sparcle
Total	58.7%	**99.4%**
Errors at duplicated location	99.6%	**99.7%**
Errors at new location	30.3%	**99.1%**

accuracy, we distinguish between: (a) erroneous records that took place in the *same exact* location of at least one other correct record, and (b) erroneous records that took place in *new* locations where there is no other correct records. As depicted in Table 1, HoloClean was able to correct 99.6% of the former, but only 30.3% of the latter.

The main reason behind such poor performance of HoloClean, as a representative of rule-based data cleaning systems, is twofold: (1) Cleaning with functional dependencies relies on sufficient co-occurrence of value pairs to learn that a certain value of an attribute leads to a corresponding value of another attribute. However, for spatial attributes, there is very little chance that two records have the exact same coordinates, mainly due to the inherent inaccuracy of location-detection devices. Hence, a rule-based system will not be able to find sufficient *spatial* co-occurrence to be used to detect and repair erroneous entries. (2) The outcome of whether a certain record satisfies a rule is binary (*True* or *False*). However, in spatial rules, such outcome needs to be fuzzy, as a certain record may satisfy the rule in stronger terms than other records. It is important to note that in this particular example, we do not rely on any external knowledge of borough boundaries.

This paper presents Sparcle (SPatially-AwaRe CLEaning); a novel framework that injects spatial awareness into the core engine of rule-based data cleaning systems as a means of boosting their accuracy. Our goal in Sparcle is *not* to come up with a new data cleaning system. Instead, it is to *boost* the accuracy of current systems. In particular, Sparcle lives inside a *host* data cleaning system, making it spatially-aware. A key idea behind Sparcle is that it goes beyond the traditional functional dependency rules of the form: *"Two records with the **same** location **should** have the same borough"* to support the more relaxed functional dependency form: *"Two records with **more similar** locations are **more likely** to have the same borough"*. To do so, Sparcle injects two main spatial concepts into its host data cleaning system: (1) *Spatial Neighborhood*. To support going from the *"same"* predicate to the *"similar"* predicate, records with spatial attributes satisfying some spatial neighborhood (similarity) criteria should be considered as relatively equivalent with respect to the spatial functional dependencies; (2) *Distance Weighting*. To support going from *"should"* to *"likely"* and to have the keyword *"more"* in the relaxed functional dependency, records will be given a weight of how much they satisfy each rule, where the weight will be based on the distance between records satisfying the functional dependency. With this, the last column of Table 1 shows that, for NYC collision data, Sparcle was able to correct 99.4% of all errors and 99.1% of the errors with new locations .

As Sparcle injects spatial awareness into existing data cleaning systems, its architecture follows the same common architecture of most rule-based data cleaning systems. In particular, such systems (e.g., [14, 19, 28, 34, 44, 53]) are typically composed of four back-to-back components, *error detector*, *candidate generator*, *input formulator*, and *error corrector*. The first three modules are mainly for error detection and preparing the data in some format that can be repaired using a statistical method in the last module, which is a very system-specific module. Hence, Sparcle focuses on modifying the first three modules to be: *spatial error detector*, *spatial candidate generator*, and *spatial input formulator*, while leaving the fourth module intact as it is system-specific. These modified modules will mainly support spatial dependencies. Non-spatial dependencies are still supported through the host data cleaning system. The outcome of the third module (*spatial input formulator*) is forwarded as is to the downstream host cleaning system to be combined with other non-spatial constraints to find out the repaired value.

Sparcle is novel in its general approach of injecting spatial awareness in the core engine of data cleaning systems. The challenge is that to have such spatial awareness, we need to dig deep into the core engine of each single component and figure out a way to make it spatially aware. Sparcle is also novel in its specific approaches of injecting spatial awareness in each the three main components of data cleaning systems. The closest to Sparcle would be the set of relaxed functional dependencies [9], including matching dependency [16], metric dependency [29], differential dependency [47], and ontology dependency [5]. However, such dependencies fall short in achieving the objectives of Sparcle for four main reasons: (a) They do not consider the *Distance Weighting* concept, which is a cornerstone in Sparcle and its applications, (b) They are designed to catch those *few* occasions of erroneous data, while in spatial data, *all* records can be "erroneous", as it is very rare to have two records with same coordinates, (c) They are designed to catch two entries of the same record, but having slightly different instances (e.g., "Ave" and "Avenue"), while in spatial data, two records with nearby coordinates are truly two different records, and (d) They are not deployed in open-source data cleaning systems [34, 44, 52], while Sparcle main objective is to be injected inside the core engine of such systems.

Extensive experiments based on real implementation of Sparcle inside data cleaning systems and real datasets show that Sparcle significantly boosts the accuracy of its host systems. The rest of the paper is organized as follows: Section 2 presents Sparcle architecture. The three modules of Sparcle are presented in Sections 3 to 5. Experimental results are presented in Section 6. Section 7 highlights related works. Section 8 concludes the paper.

2 ARCHITECTURE

Figure 2 depicts Sparcle architecture, deployed inside a host data cleaning system. Sparcle takes two types of inputs, the raw data to be cleaned and the constraints that define functional dependencies. The output of Sparcle is the detected erroneous *cells*, where a *cell* is a certain attribute of a certain record, along with a weighted list of suggested correct values for each cell. Internally, Sparcle follows similar architecture to that of rule-based data cleaning systems, mainly composed of three modules, *spatial error detector*, *spatial candidate generator*, and *spatial input formulator*. A brief description of Sparcle input, modules, and output is below:

Sparcle **Input: Constraints.** As Sparcle is injected into a host rule-based data cleaning system (e.g., [19, 34, 44, 52]), it is only triggered when there are spatial constraints, and it only takes care

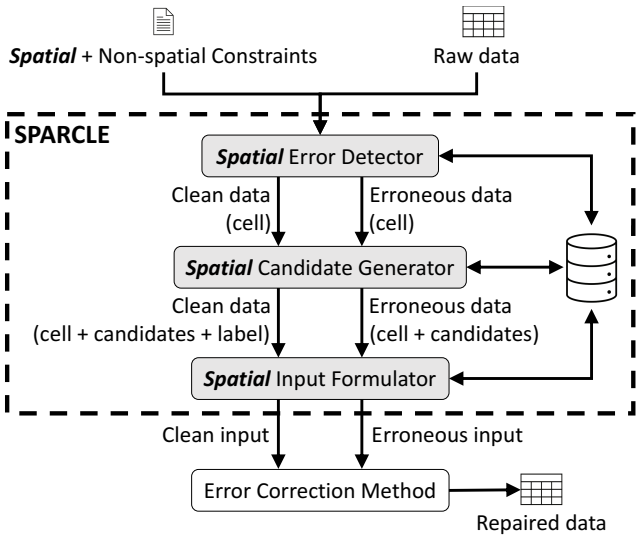


Figure 2: Sparcle Architecture

of such constraints. Non-spatial constraints over the same input data will still be supported by the host data cleaning system without any interference from Sparcle.

Spatial Error Detector. The input to this module is the input to Sparcle. The output is two sets of cells, erroneous and clean cells. It injects spatial-awareness into existing error detection modules. Hence, instead of detecting errors based on exact co-occurrence, it relaxes the co-occurrence criteria to consider records within spatial proximity. It also assigns weights to all detected errors based on the distance between co-occurred records.

Spatial Candidate Generator. The input to this module is the two sets of erroneous and clean cells coming out of the *spatial error detector*. The output is two similar sets of cells, considering the following: (1) There will be more cells in the clean set, as some erroneous cells will be cleaned, (2) Each cell will have a set of weighted candidate values, where Sparcle believes that one of these values in the correct value, (3) Clean cells will be labeled with the value that Sparcle believes it is the correct one.

Spatial Input Formulator. This module mainly injects spatial-awareness into existing input formulator modules, where the input is the output of the *spatial candidate generator*, while the output is the output of Sparcle. As these modules are very specific to the host cleaning systems, Sparcle has to have various versions of such module to match its host system. The goal is to score each possible candidate value and prepare the output in a certain format to match the requirements of the host error correction module.

Sparcle **Output: Interaction with Host System.** The output of Sparcle is the output of the *Spatial Input Formulator*, which is the detected erroneous *cells* and their weighted suggested values. If we only have spatial constraints, then the output of Sparcle is the completely corrected input data. When having non-spatial constraints, the output of Sparcle is sent to the *error correction* module of its host data cleaning system. Then, it will be integrated with other suggested values from the non-spatial constraints to statistically come up with the final correct value. As the correction module is very system-specific, the *Spatial Input Formulator* has to customize the output of Sparcle based on its host system.

3 SPATIAL ERROR DETECTOR

The *spatial error detector* module is the first module in Sparcle, where its input is the input to Sparcle, which is the dataset to be cleaned, along with a set of spatial constraints that need to be enforced for error detection and correction. Then, it deals with multiple spatial constraints separately and sequentially. For each spatial constraint, the output is two sets of *cells*, where a *cell* is a certain attribute of a certain record. The first set is the set of *cells* that are deemed erroneous, while the second set is all other *cells* that are considered clean for now. The main challenge in this module is twofold: (a) current denial constraints in data cleaning systems [13, 19, 44, 52] do not support spatial constraints like range and kNN queries, with their own weight functions, and (b) spatial denial constraints are pretty expensive to compute. The novelty of Sparcle in this module includes: (1) providing the ability to incorporate both spatial neighborhood and distance weighting concepts in the language defining spatial constraints (Section 3.1), (2) building spatial data infrastructure that boosts Sparcle performance to support spatial constraints (Section 3.2), and (3) employing spatial neighborhood and distance weighting concepts to detect erroneous *cells* and weight their spatial constraint violation (Section 3.3).

3.1 Spatial Denial Constraints

Denial Constraints. Most rule-based data cleaning systems (e.g., [13, 19, 44]) use denial constraints to express the cases that should be denied for any dataset. A denial constraint of the form $\forall r_1, r_2, \cdots \in \mathcal{R} : \neg(p_1 \wedge p_2 \wedge \cdots \wedge p_m)$ means that the set of predicates p_1 to p_m cannot be *True* together for any combination of records, where the result of a predicate evaluation is either *True* or *False*. For the example of Figure 1(a), a denial constraint would be:

$$\forall r_1, r_2 \in R : \neg(r_1.Latitude = r_2.Latitude \wedge r_1.Longitude = r_2.Longitude \wedge r_1.Borough \neq r_2.Borough)$$

, which indicates that two records r_1 and r_2 cannot have the same values for both latitude and longitude, but have different borough values. In other words, if r_1 and r_2 have the same latitude and longitude values, they should have the same borough value.

Spatial Denial Constraints. The current form of denial constraints is not suitable for spatial data as it is rare to have two records with the same exact latitude and longitude. Hence, Sparcle extends the constraint language to support spatial denial constraints. This is done as merely a language extension, not as a new class of constraints. This makes Sparcle inherit all system support of denial constraints, including constraint satisfaction, validation, and implication. For NYC data in Figure 1(a), Sparcle supports the following construct for a *SpatialRange* denial constraint:

$$\forall r_1, r_2 \in R : \neg(SpatialRange(r_1.Latitude, r_1.Longitude, r_2.Latitude, r_2.Longitude, d, \mathcal{F}, \mathcal{W}) \wedge r_1.Borough \neq r_2.Borough)$$

, which indicates that if two records r_1 and r_2 are within distance range d from each other, according to distance function $\mathcal{F}$, then they are *likely*, according to weight function $\mathcal{W}$, to have the same borough. $\mathcal{F}$, responsible on enforcing the *spatial neighborhood* concept, may employ either Euclidean or road network distance.

$\mathcal{W}(r_1, r_2)$, responsible on enforcing the *distance weighting* concept, employs an arbitrary function (e.g., linear or exponential) that returns a decreasing value from 1 to 0 as the distance between r_1 and r_2 increases from 0 to d. *SpatialRange* returns *True* as a non-zero value (based on $\mathcal{W}$) if $\mathcal{F}(r_1, r_2) < d$, otherwise it returns *False*.

Sparcle also supports *k-nearest-neighbor* (kNN) denial constrains with the following language construct:

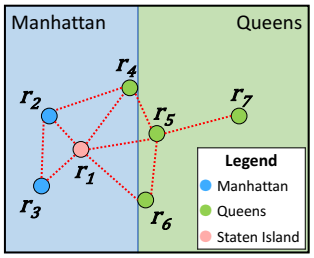
$$\forall r_1, r_2 \in R : \neg(SpatialkNN(r_1.Latitude, r_1.Longitude, r_2.Latitude, r_2.Longitude, k, \mathcal{F}, \mathcal{W}) \wedge r_1.Borough \neq r_2.Borough)$$

, which indicates that if record r_2 is among the k nearest records to r_1, according to distance function $\mathcal{F}$, then they are *likely*, according to weight function $\mathcal{W}$, to have the same borough. *SpatialkNN* returns *True* as a non-zero value (based on $\mathcal{W}$) if r_2 is among the k nearest records to r_1, otherwise it returns *False*.

3.2 Building Data Infrastructure

Current data cleaning systems check their constraint violations by simply doing a self-join for the input dataset based on equality for one of the predicates, then, scanning the result for other predicates. Further operations in the data cleaning process are basically employing inexpensive equality search. Unfortunately, this is not the case for evaluating spatial constraints violation along with downstream operations that will be needed in Sparcle. In particular, the most needed operations in Sparcle are spatial range and k-nearest-neighbor queries per the underlying spatial constraints. Since these are pretty expensive operations, compared to the equality search, Sparcle: (a) employs a spatial database system [42], where the input raw data is spatially indexed based on their *Latitude* and *Longitude* coordinates, and (b) for each spatial constraint C, Sparcle uses the spatial index to efficiently perform a self-join of the raw input data based on either range or k-nearest-neighbor query with the parameters defined in C. The result of the join is then materialized and stored in a table, termed *DistanceMatrix*, with the schema: ($\underline{R_1}$, $\underline{R_2}$, v_1, v_2, D, W), where R_1 and R_2 are two record identifiers such that R_2 is either within distance d from R_1 or is one of the k nearest neighbors of R_1, according to a distance function $\mathcal{F}$ as defined by range or k-nearest neighbor spatial constraints in Section 3.1. v_1 and v_2 are the corresponding values for R_1 and R_2 for the attribute mentioned in C that we aim to clean. D is the distance between R_1 and R_2 according to the function $\mathcal{F}$ and W is the weight for the distance between R_1 and R_2 according to the weight function $\mathcal{W}$.

Figure 3 gives an example for the *DistanceMatrix* computations. In particular, Figure 3a shows a set of seven records on part of the map that includes areas from two NYC boroughs, Manhattan and Queens, plotted in light blue and green, respectively. It is important to note that the borough boundaries are not known to the data cleaning system, and they are just depicted here for illustration of the ground truth, but this information is not used at all in any of Sparcle computations. Meanwhile, records are colored based on the borough information they have in their raw records, which could be right or wrong. Figure 3b gives statistics about the whole dataset, in terms of the number of records in the dataset for each borough value. These statistics, collected in this module, will be used by later modules for their computations. Figure 3c gives the



(a) Dataset

Candidate Value	# Appearance
Bronx	100
Brooklyn	200
Man.	300
Queens	300
S. Island	100

(b) Value Frequency

R_1	R_2	v_1	v_2	D	W
r_1	r_2	S. Island	Man.	200	0.64
r_1	r_3	S. Island	Man.	500	0.25
r_1	r_4	S. Island	Queens	800	0.04
r_1	r_5	S. Island	Queens	800	0.04
r_1	r_6	S. Island	Queens	800	0.04
r_2	r_1	Man.	S. Island	200	0.64
r_2	r_3	Man.	Man.	600	0.16
r_2	r_4	Man.	Queens	900	0.01
r_3	r_1	Man.	S. Island	500	0.25
r_3	r_2	Man.	Man.	600	0.16
r_4	r_1	Queens	S. Island	800	0.04
r_4	r_2	Queens	Man.	900	0.01
r_4	r_5	Queens	Queens	600	0.16
r_5	r_1	Queens	S. Island	800	0.04
r_5	r_4	Queens	Queens	600	0.16
r_5	r_6	Queens	Queens	600	0.16
r_5	r_7	Queens	Queens	900	0.01
r_6	r_1	Queens	S. Island	800	0.04
r_6	r_5	Queens	Queens	600	0.16
r_7	r_5	Queens	Queens	900	0.01

(c) Distance Matrix

Figure 3: DistanceMatrix Example

DistanceMatrix for the seven records (r_1 to r_7) of Figure 3a, based on a *SpatialRange* denial constraint with distance d=1km (plotted as red dashed lines between records), and a weight function $W(r_1, r_2) = (1 - \frac{\mathcal{F}(r_1, r_2)}{d})^2$, where $\mathcal{F}(r_1, r_2)$ is the distance between R_1 and R_2. For r_1, with borough value Staten Island, there are five records (r_2 to r_6) within distance d. Hence, there are five corresponding records in the *DistanceMatrix*. Two of these records, r_2 and r_3 have borough value Manhattan, with distances 200m and 500m, which result in weight values 0.64 and 0.25, respectively. Three of these records (r_4 to r_6) have distance 800m from r_1, and hence they would have weight 0.04. In a similar way, r_2, r_3, r_4, r_5, r_6, and r_7 would have three, two, three, four, two, and one records in the *DistanceMatrix*.

3.3 Detecting Spatial Constraint Violation

Algorithm 1 gives the pseudo code of Sparcle spatial error detector. With the *DistanceMatrix*, computed by a self-join as described in Section 3.2, the spatial error detection module becomes straightforward and pretty inexpensive. It is basically one scan over the *DistanceMatrix*, where for each record (R_1, R_2, v_1, v_2, D, W), if $v_1 \neq v_2$, we consider that both R_1.*Borough* and R_2.*Borough* are erroneous cells. Hence we move these two cells from the set of all (clean) cells to the set of erroneous ones. Once we finish a full scan, we output both sets of cells. The rationale behind this is that each row in the *DistanceMatrix* refers to two records that satisfy the spatial predicate (i.e., within range distance or k-nearest neighbor). Hence, they are expected to have the same borough value (i.e., v_1 is likely to be the same as v_2). If not, then at least one of these two records might have wrong borough value. Since we are not sure which one is wrong, we put both cells in the erroneous set.

For the example in Figure 3, all the *Borough* cells (attributes) of records r_1 to r_6 will be added to the set of erroneous cells as they appear in the first five rows of the *DistanceMatrix*, where $v_1 \neq v_2$. r_7.*Borough* will be considered a clean cell as all its rows in the *DistanceMatrix* have $v_1 = v_2$.

Algorithm 1 Error Detection Pseudo Code

Procedure SpatialErrorDetection(Data $\mathcal{I}$, DistMatrix M)

```
ErroneousCells ← φ
CleanCells ← All cells in Input Data I
for each row (R1, R2, v1, v2, D, W) in M do
    if v1 ≠ v2 then
        Move R1.Borough from CleanCells to ErroneousCells
        Move R2.Borough from CleanCells to ErroneousCells
    end if
end for
Return CleanCells and ErroneousCells
```

4 SPATIAL CANDIDATE GENERATOR

Though this module takes its input from the *Spatial Error Detector* module (Section 3) as two sets of cells, *clean* and *erroneous*, for each spatial constraint, it mainly operates on the *erroneous* list of each constraint separately, aiming to: (1) generate a list of candidate values for each erroneous cell, along with the probability that each candidate is the correct one, (2) use the probabilities of the generated candidates to decide if any of the erroneous cells can be safely moved to the list of clean cells. The output would be another two sets of cells, *clean* and *erroneous*, along with the generated candidate values for each cell. The main challenge in this module is that standard candidate generation procedures in rule-based data cleaning systems [19, 34, 44, 52] cannot accommodate both spatial neighborhood and distance weighting concepts, needed to identify possible candidate values for each erroneous result. The novelty of Sparcle in this module is that it modifies all the equations and logic to come up with the set of candidate values and their probability, by taking into account the spatial weight of each possible candidate value, based on its spatial neighborhood. In particular, this process goes through three phases as outlined in Algorithm 2. The first phase (Section 4.1) generates an initial list of possible candidate values for each erroneous cell. The second phase (Section 4.2) estimates a probability of correctness for each candidate value. The third phase (Section 4.3) finds if there is one clear dominant candidate value. If so, it is considered as the correct value of its cell, and the cell is moved to the clean list.

4.1 Phase 1: Initial Candidate Generation

Data cleaning systems mainly generate the candidate values based on *counting* the *co-occurrence* between record attributes. For example, the value of r_i.*Borough* would be a likely candidate of r_j.*Borough* if r_i and r_j share the same *Latitude* value (i.e., *co-occurrence* of latitudes). The likeliness of the candidacy will be based on the *counting* of how many times such co-occurrence took place. Apparently, this is not applicable to spatial data as it is rare to have two records with the same *Latitude* and/or *Longitude* values. In fact, applying this to the example in Figure 3 yields zero co-occurrence and hence no candidates are generated for any of the erroneous cells.

Sparcle enriches existing candidate generators with spatial awareness. In particular, for any record r_i where its cell/attribute *Borough* is marked erroneous, Sparcle modifies the candidate generation process in two ways: (1) The *co-occurrence* of record values is relaxed from *exact* value co-occurrence to be *nearby* co-occurrence.

Algorithm 2 Candidate Generation Pseudo Code

Procedure SpatialCandGeneration(Cells $\mathcal{C}$, Cells $\mathcal{E}$, *MaxProb*)

```
for each erroneous cell E in E do
    E.CandList ← InitCandidates(E)                    ▷ (Phase 1)
    for i=1 to |E.CandList| do                        ▷ (Phase 2)
        E.CandList[i].Prob ← ProbEval(E.CandList[i].value)
    end for
    ProbNormalization(E.CandList)                     ▷ (Phase 3)
    if |E.CandList| = 1 or TopProb(E.Candlist) > MaxProb then
        E.Label ← TopProbCandidate(E.Candlist)
        Move E from E to C
    end if
end for
Return E and C
```

Hence, the candidate values for r_i.*Borough* would include r_i.*Borough* itself, along with all *Borough* values for any record r_j that lies within (range or kNN) proximity from r_i according to the spatial denial constraint. This is done through a lookup search over the *DistanceMatrix* for all rows where R_1 is r_i. (2) The *counting* of the co-occurrence is relaxed from being an *absolute* count to be a *weighted* count based on how far the co-occurred records from each other. This can be done by computing the sum of the weights in the *DistanceMatrix* for all co-occurred records, i.e., all rows where R_1 is r_i. If none of the nearby records share the same value of r_i.*Borough*, we would still have the value of r_i.*Borough* in our candidate list, yet with a default minimal weight value of 0.01. The weights of all candidate values will be used in the second phase (Section 4.2) to estimate the probability of each candidate value.

Example. The second and third columns of Table 2 give the list of candidate values, along with their weights for the *Borough* attribute for record r_1 in Figure 3. There are three candidate values, Manhattan and Queens as they appear two and three times, respectively, in the *DistanceMatrix* with nearby records (i.e., nearby co-occurrence). The third candidate value is Staten Island, even though no nearby record has this value, but it is the raw *Borough* value of r_1, and hence we need to consider it. The weights for Manhattan and Queens are set to 0.89 and 0.12, respectively, computed as the sum of weights of their corresponding records in the *DistanceMatrix*. The weight for Staten Island is set to the default minimal value of 0.01.

4.2 Phase 2: Candidate Probabilities Estimation

This phase aims to estimate the probability of each candidate value to be the correct one. Current data cleaning systems do so by adopting various statistical methods. One typical method is the *Naive-Bayes* [38], where the probability that a cell C of record R has a certain candidate value v (*Prob* $(C = v)$) is computed as *Prob* $(v \in D)$ $\times \prod_A$ *Prob* $((v \rightarrow R.A) \in D)$, which is the probability of having v in the whole dataset D, multiplied by, for each attribute A other than C, the ratio of records in D where having v in C implies the value in attribute A. Apparently this is not applicable to spatial data as the co-occurrence of v and the spatial attributes in A is pretty rare, which will make the probability of each candidate zero.

As it is the case for Phase 1, Sparcle enriches existing candidate probability estimators with spatial awareness. For any record r_i

Table 2: Candidate Generation State of r_1.*Borough*

Cell	Candidate Value	Sum Weights	$\frac{\lvert Spatial(v,R)\rvert}{\lvert D\rvert}$	$\frac{Count((v,R.ID),D)}{Count(v,D)}$	Probability	Normalized Prob.
r_1.Borough	Manhattan	0.89	0.00089	0.1/300	89/300000000	0.68
	Queens	0.12	0.00012	0.1/300	1/25000000	0.09
	S. Island	0.01	0.00001	1/100	1/10000000	0.23

with erroneous *Borough* cell, Sparcle modifies the candidate probability estimation in two ways: (1) In the calculation of $Prob\,(C = v)$, Sparcle replaces the term $Prob\,((v \rightarrow R.A) \in D)$ by the term $Prob\,((v \rightarrow \text{location near } R) \in D)$ for the spatial attributes in A. This means that instead of considering the exact co-occurrence between v and each single location attribute, Sparcle employs its *spatial neighborhood* concept to consider the nearby co-occurrence between v and R according to the spatial proximity defined in the denial constraint. (2) When calculating the co-occurrence probability, Sparcle does not count the nearby co-occurrences. Instead, it sums the co-occurrence weights as closer ones are weighted higher than further ones, per the *distance weighting* concept. Both nearby co-occurrences and their weights are directly obtained from the *DistanceMatrix*. Formally, in Sparcle, the probability estimation for any candidate value v of cell C of record R is:

$$Prob(C = v) = Prob(v \in D) \times \prod_{A'} Prob((v \rightarrow R.A') \in D) \times Prob((v \rightarrow location\ \ near \quad R) \in D) \tag{1}$$

, where A' is the set of attributes in R excluding C and the spatial attributes. The first probability factor, $Prob\,(v \in D)$, is basically $\frac{Count(v,D)}{|D|}$, which is the number of times that v has appeared in the dataset D divided by the number of records in D. The second probability factor for each attribute in A', $Prob\,((v \rightarrow R.A') \in D)$, is $\frac{Count((v,R.A'),D)}{Count(v,D)}$, which is the number of times that v and the value of $R.A'$ have appeared together in D divided by the number of times that v has appeared in D. The third probability factor, $Prob\,((v \rightarrow \text{location near } R) \in D)$ is $\frac{|Spatial(v,R)|}{Count(v,D)}$, which is the sum of the weights of those records where v appears within a spatial proximity of R, divided by the number of times that v has appeared in D. With this, Equation 1 can be rewritten as:

$$Prob(C = v) = \frac{Count(v,D)}{|D|} \times \prod_{A'} \frac{Count((v,R.A'),D)}{Count(v,D)} \times \frac{|Spatial(v,R)|}{Count(v,D)} = \frac{|Spatial(v,R)|}{|D|} \times \prod_{A'} \frac{Count((v,R.A'),D)}{Count(v,D)}$$

, where |*Spatial(v,R)*| is basically the third column of Table 2, computed at Phase 1 (Section 4.1). *Count*(*v*,*D*) is obtained directly from the value frequency table computed in Figure 3b. For *Count*((*v*,*R.A'*),*D*), we query the original input table to get the *count* of the number of $R.A'$ when the value of cell $R.C$ is v. In case $R.A'$ is the record identifier, such *count* would be set as either (a) 1, if the candidate value v is the original value of $R.C$, as the record identifier would naturally appear only once in the dataset, combined with the record original value, regardless of whether it is right or not, or (b) 0.1, if v is not the original value of $R.C$. Even though there is a zero co-occurrence between the record identifier and any value of v that is not the original, we follow the same practice used by state-of-the-art data cleaning systems [19, 44, 52], known as the principle of minimality, where we put 0.1 co-occurrence value for any non co-occurrence. This gives ten times more bias towards the original record value, which again, follows existing data cleaning systems that favor the original value.

Example. The fourth and fifth columns of Table 2 give the two terms used to compute the probability of each candidate value for r_1. The sixth column in the table presents the probability of each candidate to be the correct value for the record, which is basically the multiplication of the fourth and fifth columns. The first probability term (fourth column) is basically the sum of weights (third column) divided by 1,000, which is the total number of input records. Since our toy example has only one non-spatial attribute, namely, the record identifier, the probability of the value Manhattan and Queens for r_1 would need to be multiplied by 0.1/300. The 0.1 is the default minimal value and 300 is the number of times that the values Manhattan and Queens appear in the input dataset. Meanwhile, as Staten Island is the original value of r_1, we multiply the probability by 1/100, where 1 for the original value and 100 is the number of times of Staten Island in the input dataset. The output of this phase is both the second and sixth columns of Table 2 as the candidate values with their probabilities.

4.3 Phase 3: Candidate Labeling and Cutoffs

The goal of this phase is to identify: (a) If there are candidate values that have marginal probability to the extent that there is no need to consider them further, and (b) If there is a certain candidate value that is clearly dominant and we can safely identify that this is the correct value for its corresponding erroneous cell. To do so, as outlined in Algorithm 2, Sparcle first normalizes the candidate probabilities to have a sum of 1. Then, it employs two parameters, *MinProb* and *MaxProb*. Any candidate value that has a probability less than *MinProb* will be considered marginal, removed, and not considered further. Then, if there is only one remaining candidate value or if there is one candidate value that has a significantly high probability more than *MaxProb*, we consider that this value is the correct one and move the corresponding cell from the erroneous list to the clean list. The output of this phase is the module output, which includes both the erroneous and clean cells, along with a list of remaining candidate values per each cell.

Example. The last column of Table 2 gives the normalized probability of the sixth column. Assuming *MinProb*=0.1, we exclude the candidate value Staten Island from r_1. Assuming *MaxProb*=0.9, no candidate of r_1.*Borough* will be marked as clean. Clean and erroneous cells with their remaining candidate values are passed to the next module.

5 SPATIAL INPUT FORMULATOR

The input to this module is two sets of clean and erroneous cells, each with its own candidate list, identified from the *spatial candidate generation*, per each spatial constraint. Then, for each spatial constraint, the module mainly operates on the erroneous cells, and aims to identify the correct value for each erroneous cell, with a score based on how much each candidate value satisfies (or violates) the spatial denial constraint. The output of this module is the detected erroneous *cells* and their weighted candidate values, per each spatial constraint. If we only have spatial constraints, this will be the final output of the data cleaning process as the repaired value for each cell can be inferred from its score, e.g., the value with highest score. With non-spatial constraints, the output of this module will be sent to the underlying error repair module of the host data cleaning system to be integrated with the outcome of the non-spatial constraints that are processed by the host system.

The main challenge in this module is that standard input formulators are not equipped to support spatial data, as they deal with error detection and candidate generation modules that do not support such data. The novelty of Sparcle in this module is twofold: (a) It shows how input formulators can acknowledge spatial data properties with its two main concepts of spatial neighborhood and distance weighting, and (b) It does so for three families of open-source data cleaning systems. Unlike the previous two modules that are generic for any data cleaning systems, each system has its own system specific *Input Formulator* that puts the results of the two other modules in the form needed by the underlying statistical repairing method. Hence, Sparcle*Spatial Input Formulator* has to be developed for each *family* of data cleaning systems that use the same output form, which is the input to the underlying data repair. We show how Sparcle injects *Spatial Neighborhood* and *Distance Weighting* concepts in the input formulator module of three data cleaning systems, namely, AimNet [52] (Section 5.1), Baran [34] (Section 5.2), and HoloClean/MLNClean [19, 44] (Section 5.3).

5.1 Violation-based Feature Vectors

AimNet [52], the error correction method of the HoloClean's open source distribution, requires a feature vector V per cell per constraint, where $v[i]$ represents the score of how the ith candidate of the cell violates the denial constraint. To construct its feature vector, AimNet [52] counts the number of violations of the constraint that are caused by the cell taking its ith candidate. The first column of Figure 4a shows AimNet feature vector for r_1 in our running example of Figure 3. Setting r_1.*Borough* to Manhattan will cause three instances of denial constraint violations with r_4, r_5, and r_6 (the 3rd to 5th rows in *DistanceMatrix*). Hence, the score is set to 3. Similarly, the scores for Queens and Staten Island are set to 2 and 5.

Apparently, this is not suitable for spatial constraints as it equally weights the constraint violations between r_1 and r_2 with the constraint violation between r_1 and r_5. Spatially speaking, the constraint violation (r_1, r_5) should be *weaker* than (r_1, r_2), as the distance between r_1 and r_5 is much more than the distance between r_1 and r_2. To inject the spatial awareness in the input formulator of AimNet, Sparcle fills in the feature value $V[i]$ by summing up the weights of violations that are caused by the cell taking its ith candidate. The second column of Figure 4a gives such vector for r_1.

Candidate Value	Aim Net	Sparcle
Man.	3	**0.12**
Queens	2	**0.89**
S. Island	5	**1.01**

(a) Violation-based Feature Vector

Baran	Sparcle
(0, 0)	**(0.88)**
(0, 0)	**(0.12)**
(0, 0)	**(0)**

(b) Probability-based Feature Vector

Holo Clean	Sparcle
-1	**+0.77**
+1	**-0.77**
-5	**-1.01**

(c) Factor Graph

Figure 4: Input of r_1.*Borough* by Original Method VS Sparcle

If r_1.*Borough* is set to Manhattan, then three constraint violations would take place with r_4, r_5, and r_6, each with a weight 0.04, and hence the total score of the constraint violation would be 0.12. Similar, if r_1.*Borough* is set to Queens, then two constraint violations would take place with r_2 and r_3, with weights 0.64 and 0.25, respectively. Hence, the total score is 0.89. Finally, if r_1.*Borough* is set to Staten Island, then five constraint violations would take place with r_2 to r_6 with a total weight of 1.01.

It is important to note here that the lower the score the more likely the value would be considered correct. The fact that Manhattan has the lowest score of 0.12, i.e., the lowest violation score, is an indication that, per the spatial constraint, it is the most favored value for r_1.*Borough*. The final feature vector from Sparcle will be passed to the repair module of AimNet [52] to consider it along with other non-spatial constraints for the final repaired dataset.

5.2 Probability-based Feature Vectors

Unlike AimNet [52], the Baran [34] system requires the input to its error correction method as a feature vector per cell per candidate, where each vector value represents the probability of the candidate according to a specific dependency. Meanwhile, Baran does not ask for user-input constraints, instead, it assumes all possible dependencies from all other attributes to the cell. For the example of Figure 3, Baran assumes the dependency from *Latitude* and *Longitude* separately. Then, for each dependency, e.g., *Latitude* $\rightarrow$ *Borough*, it estimates the probability based on co-occurrence. The first column in Figure 4b gives the feature vector values for r_1 for each possible *Borough* value. They are all zero vectors as there is zero co-occurrence between the *Latitude* and *Longitude* values with any *Borough* value. This makes the error correction module of Baran fail to identify the correct answer.

To inject the spatial awareness in the input formulator of Baran, Sparcle uses the weights and spatial neighborhood records that were evaluated in Section 3 along with the candidate values computed in Section 4 to calculate the probability of a combined dependency on the form *(Latitude, Longitude)* $\rightarrow$ *Borough*. The second column of Figure 4b gives such vector for r_1. Since there is only two possible values among the candidate ones that have proximity co-occurrence with r_1, we set their vector values as their normalized probability 0.88 for Manhattan and 0.12 for Queens. The last row for Staten Island is set to 0 as there is no proximity co-occurrence.

Unlike the case of AimNet, the higher the values here the more likely the candidate value is the correct one. This is mainly because these values represent a probability rather than a violation. Finally, such form of output vector of Sparcle will be sent to the repair module of Baran [34] to consider it along with other non-spatial constraints for the final repaired dataset.

5.3 Factor Graph

HoloClean [44] and MLNClean [19] include error correction methods that are based on Markov Logic Network [46], which requires its input to be in a form of a factor graph. To construct the factor graph, each functional dependency instance needs a factor function that returns a value reflecting how the instance satisfies the dependency. In particular, for HoloClean, the factor function returns 1 if the instance satisfies the dependency, otherwise it returns −1. Then, the data cleaning process aims to find the values that maximize the sum of all factor functions in the dataset. The first column in Figure 4c shows the sum of factor functions related with r_1 in Figure 3. For Manhattan, the sum of factor functions would be -1 as the factor function would return -1 three times (with r_4, r_5 and r_6) and 1 two times (with r_2 and r_3) when r_1.*Borough* is set to Manhattan. For Staten Island, the sum would be -5 as the factor function would return -1 five times.

Apparently, this is not suitable for spatial constraints as the factor function would only return either 1 or -1 regardless of how strong a certain instance satisfies the spatial constraint. To inject the spatial awareness in the input formulator of HoloClean [44], Sparcle modifies the factor graph construction by multiplying the factor function output (1 or -1) by the weight of the instance, computed in the *DistanceMatrix* of Figure 3. For example, the second column in Figure 4c gives Sparcle output for factor graphs for r_1. For Manhattan, the sum of factor functions would be 0.77, computed as -1*(0.04+0.04+0.04)+1*(0.64+0.85). For Queens, the sum of factor functions would be 1*(0.04+0.04+0.04)-1*(0.64+0.85)=-0.77. For Staten Island, the sum will be -1.01. The higher the value of the sum the more likely the candidate value is the correct one. This shows how spatial awareness changed the favored value from Queens to Manhattan. Such form of factor graph of Sparcle will be sent to the repair module of HoloClean [44] to consider it along with other non-spatial constraints for the final repaired dataset.

6 EXPERIMENTAL RESULTS

This section compares different implementations of Sparcle showing the impact of the injected spatial awareness on both accuracy and efficiency. In particular, we pick two state-of-the-art rule-based data cleaning systems, HoloClean [22] and Baran [3], as host systems. HoloClean+Sparcle (referred by Sparcle for short) is a full-fledged real system deployment and it is the main open-source release of Sparcle [49]. Baran+Sparcle, which is released as a secondary open-source release of Sparcle [4], is a proof-of-concept implementation that emulates the behaviour of the first two modules of Sparcle within Baran, then produces spatial-aware feature vectors through a real deployment of the spatial input formulator inside Baran. We also introduce a third implementation of Sparcle (n=0), where n is the exponential factor of the weight function $\mathcal{W}(r_i,r_j)= (1-\frac{\mathcal{F}(r_i,r_j)}{d})^n$. Having n=0 cancels the *distance weighting* concept as the weight between any pair of records would be always 1 regardless of how far they are from each other. This would mimic the behavior of relaxed functional dependencies that can be utilized to present the spatial neighborhood concept. For Sparcle and Baran+Sparcle, we set n=2 as a default value.

Competitors. We compare the three implementations of Sparcle with (1) HoloClean [44], with its open-source distribution [22], is a data repairing system that unifies integrity constraints with other signals such as statistics and external knowledge. We mute other signals and only keep the integrity constraints to limit the comparison to constraint-based data cleaning. HoloClean uses a conservative error detection process that marks erroneous for all cells that involve in constraint violations. (2) Baran [34] is an error correction system that takes signals from functional dependencies as well as statistical perspectives. We run Baran as is with all its signals, human labeling budget set to 20, and use its sister system Raha [35] for error detection. (3) SMFL [17], which is a matrix factorization technique that leverages the locality in spatial data to predict spatial-correlated values. It only performs missing value imputation where it employs Raha [35] to detect errors, then erases the erroneous entries, and attempts to impute them.

Table 3: Experiment Datasets

Dataset	Dependency	Records	Errors	Dup.	Dis.
Austin-Code	(*Lat*, *Lon*) → *zipcode*	93,414	13,968	0.00	50
	(*Lat*, *Lon*) → *city*		12,224	0.00	9
Boston-311	(*Lat*, *Lon*) → *public_district*	10,000	100	0.00	14
	(*Lat*, *Lon*) → *police_district*		122	0.00	12
	(*Lat*, *Lon*) → *ward*		120	0.00	22
	(*Lat*, *Lon*) → *zipcode*		2,578	0.01	30
Chicago-Building	(*Lat*, *Lon*) → *community*	731,734	105,240	0.64	77
	(*Lat*, *Lon*) → *census_tract*		138,953	0.64	980
	(*Lat*, *Lon*) → *ward*		181,119	0.58	50
NYC-Crash	(*Lat*, *Lon*) → *borough*	1,751,624	421,013	0.44	5
	(*Lat*, *Lon*) → *zipcode*		528,565	0.30	230
Chicago-Synthetic	(*Lat*, *Lon*) → *police_district*				23
	(*Lat*, *Lon*) → *ward*				50
	(*Lat*, *Lon*) → *zipcode*				59
	(*Lat*, *Lon*) → *beat*				275
	(*Lat*, *Lon*) → *census*				801

Datasets. Table 3 shows the properties of the four real and one synthetic datasets we are using in our experiments. For each real dataset, we only keep the spatial attributes *Latitude* and *Longitude* and the attributes dependent on them: (1) Austin-Code [1]. 93+K records of Austin Code Department complaint cases. Two spatial functional dependencies for *zipcode* and *city* need to be kept, with around 14K and 12K errors (either missing or incorrect), respectively. The second last column in Table 3 presents the error duplication ratio, which is the ratio of erroneous records that took place on the same location of other correct records. The last column indicates the data has 50 distinct zipcode and 9 city values. (2) Boston-311 [7]. Randomly sampled 10K records for the 311 service requests in Boston, with four spatial dependencies. We had to sample this dataset as some competitors cannot scale for large datasets. (3) Chicago-Building [12]. 731+K records for the building permits issued by the City of Chicago, with three spatial dependencies. (4) NYC-Crash [40]. The dataset shown in Figure 1 with 1.7+M records for the location of vehicle crashes in NYC since 2014, with two spatial dependencies. (5) Chicago-Synthetic. A synthetic data in the spatial extent of Chicago, in which each record is a random location with five functional dependencies. Number of records, errors, and duplication ratios are not shown here as would vary these parameters in our experiments.

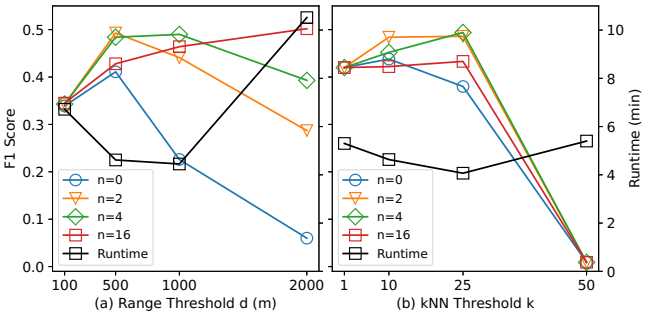


Figure 5: SPARCLE Parameter Tuning

Evaluation Metrics and Experiment Design. We evaluate SPARCLE and competitors based on four metrics: (1) *Precision*: The fraction of number of correct repairs over total number of repairs made by the system, (2) *Recall*: The fraction of number of correct repairs over total number of errors, (3) *F1 score*: The harmonic mean of precision and recall, i.e., $\frac{2*(Prec.*Rec.)}{Prec.+Rec.}$, and (4) the system run time. All experiments are performed on a Linux server with 20 CPU@2.2GHz, 96GB memory and 1TB SSD. In this section, we first perform a parameter study of SPARCLE to set on its optimal parameters (Section 6.1). Then, we compare SPARCLE against competitors in terms of accuracy (Sections 6.2 to 6.3) and efficiency (Section 6.4).

6.1 SPARCLE Parameter Tuning

This section studies the impact and trade-offs of SPARCLE parameters on its accuracy and efficiency, namely, the spatial range d, the number of nearest neighbors k, and the exponential weight factor n. To do so, we create a Chicago-Synthetic dataset of 20,000 records and focus on the functional dependency (*Lat*, *Lon*) → *census*.

Spatial range parameter d. Figure 5(a) shows the impact of increasing the spatial range d from 100 to 2000, on both the accuracy (*F1 score*) and efficiency. For accuracy, we plot SPARCLE with different values of n as it impacts the system accuracy. No need to do the same for efficiency as n has no impact on efficiency. For d from 100 to 2000, the highest *F1 score* at each distance increases, and finally, achieves its highest when d=2000 with n=16. This suggests that a larger neighborhood has a potential to achieve higher accuracy. Meanwhile for n, a larger neighborhood requires a larger n to achieve that high accuracy. For example, for d of 500, 1000 and 2000, the best n is 2, 4, 16, respectively. From efficiency perspective, large value of d (e.g., 2000) encounters very high overhead to search a large neighborhood area and build the *DistanceMatrix*.

Nearest-neighbor parameter k. Figure 5(b) shows the impact of increasing k from 1 to 50. Similar to spatial range, a large k with a large n achieve high accuracy. However, the neighborhood cannot be too large as the accuracy almost drops to 0 when k=50. The reason is that the dataset has 20K records but 801 distinct *census* values. So, a pretty large neighborhood (e.g., k=50) would involve too many neighbors from different census_tracts, which may not guide the cleaning logic. Hence, we come up with a recommendation ceiling value that k should not be greater than $\frac{|D|}{|A|}$ where $|D|$ is the dataset size and $|A|$ is the number of distinct values of attribute A. For efficiency, also similar to spatial range, large values of k suffer from the *DistanceMatrix* computation.

6.2 System Overall Accuracy

We compare the SPARCLE and Baran+SPARCLE with their host systems HoloClean and Baran, respectively, on cleaning real data to show the effectiveness of the spatial awareness injected by SPARCLE. We also include SMFL for comparison and SPARCLE (n=0) for ablation study. Table 4 gives the cleaning accuracy in terms of precision, recall, and F1 score for each functional dependency, and then for the overall accuracy for the whole dataset with all functional dependencies combined. For Baran and SMFL, we only show the results for the Austin and Boston datasets, and for Baran+SPARCLE we only show the results for Boston as these approaches did not scale to larger datasets. More about scalability is in Section 6.4.

SPARCLE VS HoloClean. For overall accuracy in terms of F1, SPARCLE clearly outperforms HoloClean for every single dependency as well as the whole dataset. In particular, SPARCLE constantly achieves higher recall than HoloClean, owing to (1) the spatial neighborhood concept that lets SPARCLE detect more errors than HoloClean, and (2) the spatial awareness injected into the feature vectors that leads to more error corrected. For precision, SPARCLE also does better on most dependencies, with some high-precision/low-recall exceptions by HoloClean, such as the city attribute of Austin dataset and public_district and police_district attribute of Boston dataset. The reason is that HoloClean only makes a few repairs on such dependencies, as it cannot generate a candidate due to the lack of duplication in those datasets. However, SPARCLE is able to draw correct candidates from the spatial neighborhood so that does not have such problem. In fact, HoloClean gets extremely low recall on all dependencies for Austin and Boston dataset, and performs better on Chicago and NYC dataset. The main reason is that, as depicted in Table 3, Austin and Boston data have no duplicates, which makes it very hard to clean by current data cleaning systems. We study the impact of duplication ratio in more details in Section **??**. Despite of duplication ratio, the number of distinct values also affects the cleaning accuracy: both SPARCLE and HoloClean perform better on city attribute than zipcode in Austin data, although both attributes have 0 duplication ratio. The main reason is that the *city* dependency has only 9 distinct values, which is much less than the distinct values for *zipcode*, which is 50 (Table 3). Naturally, it is much harder to clean data with more distinct values. We study the impact of distinct values in more detail in Section 6.3.

Baran+SPARCLE VS Baran. Baran+SPARCLE outperforms Baran for all dependencies in Boston-311 dataset, boosting overall F1 score from 0.07 to 0.78. In particular, the superiority in recall is more significant than precision, which means that Baran+SPARCLE attempts to make more corrections while keeping good correction precision. Given that they both use Raha [35] for error detection, which means they receive the same set of detected errors, the reason Baran+SPARCLE is able to makes more corrections is that the error correction method of Baran works in a way of generalizing human corrections to the rest of the dataset. However, due to the lack of location duplication, Baran itself finds no relation between locations hence cannot generalize a human correction. On the contrary, Baran+SPARCLE recognizes the spatial neighborhood relevance and generalizes the human correction at a location to its neighborhood, leading to high recall. Meanwhile, the distance weighting concept that injected by SPARCLE guides the generalization so that closer

Table 4: Cleaning Accuracy on Real Data

		Austin-Code			Boston-311					Chicago-Building				NYC-Crash		
		zipcode	city	overall	public	police	ward	zipcode	overall	comm.	census	ward	overall	borough	zipcode	overall
Prec.	Sparcle (n=2)	**0.853**	0.992	**0.921**	0.483	0.569	0.266	**0.867**	0.779	**0.990**	**0.829**	**0.746**	**0.836**	**0.998**	**0.821**	**0.909**
	Sparcle (n=0)	0.790	0.992	0.885	0.327	0.352	0.117	0.745	0.593	0.983	0.724	0.727	0.785	0.997	0.803	0.898
	HoloClean	0.001	0.993	0.093	**1.000**	0.800	0.000	0.112	0.115	0.925	0.398	0.717	0.627	0.683	0.384	0.533
	Baran+Sparcle	-#	-#	-#	0.969	**0.975**	**0.836**	0.853	**0.847**	-#	-#	-#	-#	-#	-#	-#
	Baran	0.374	0.995	0.882	0.791	0.943	0.815	0.528	0.688	-*	-*	-*	-*	-#	-#	-#
	SMFL	0.580	**0.997**	0.789	0.889	0.873	0.770	0.462	0.490	-*	-*	-*	-*	-*	-*	-*
Rec.	Sparcle (n=2)	**0.782**	0.992	**0.880**	0.730	0.639	**0.650**	**0.859**	**0.837**	**0.982**	**0.894**	0.685	0.827	**0.994**	0.662	0.809
	Sparcle (n=0)	**0.782**	0.992	**0.880**	**0.840**	**0.754**	0.625	0.828	0.817	0.977	0.893	**0.710**	**0.836**	**0.994**	**0.667**	**0.812**
	HoloClean	0.000	0.024	0.011	0.050	0.033	0.000	0.101	0.092	0.635	0.393	0.437	0.472	0.587	0.264	0.407
	Baran+Sparcle	-#	-#	-#	0.620	0.402	0.425	0.757	0.714	-#	-#	-#	-#	-#	-#	-#
	Baran	0.010	0.674	0.324	0.270	0.209	0.217	0.011	0.037	-*	-*	-*	-*	-#	-#	-#
	SMFL	0.508	**0.997**	0.736	0.560	0.451	0.392	0.428	0.432	-*	-*	-*	-*	-*	-*	-*
F1.	Sparcle (n=2)	**0.816**	0.992	**0.900**	0.582	**0.602**	0.378	**0.863**	**0.807**	**0.986**	**0.860**	0.714	**0.832**	**0.996**	**0.733**	**0.856**
	Sparcle (n=0)	0.786	0.992	0.882	0.471	0.480	0.197	0.784	0.687	0.980	0.800	**0.718**	0.810	0.995	0.729	0.853
	HoloClean	0.000	0.046	0.020	0.095	0.063	0.000	0.106	0.102	0.753	0.396	0.543	0.538	0.632	0.313	0.462
	Baran+Sparcle	-#	-#	-#	**0.756**	0.544	**0.564**	0.802	0.775	-#	-#	-#	-#	-#	-#	-#
	Baran	0.019	0.683	0.441	0.388	0.340	0.334	0.022	0.070	-*	-*	-*	-*	-#	-#	-#
	SMFL	0.542	**0.997**	0.761	0.687	0.595	0.519	0.444	0.459	-*	-*	-*	-*	-*	-*	-*

* cannot finish due to memory error
# cannot finish after 1 day

neighbors learn more from the human input than distant neighbors, resulting in high precision.

Sparcle, Baran+Sparcle VS SMFL. SMFL has better accuracy than HoloClean and Baran but still lower than Sparcle and Baran+Sparcle. On one hand, SMFL captures the spatial information of the datasets, thus outperforms the non-spatial systems. On the other hand, SMFL falls short comparing with Sparcle-facilitated systems for the reasons: (1) SMFL does not have an error detection method adapted for spatial data. When using Raha [35] for error detection, it cannot catch the errors with respect to spatial neighborhood, which eventually limits the cleaning recall. (2) SMFL natively only supports numerical values. To work with the categorical attributes of the experiment datasets, we have to number the categories. For example, Manhattan as 1, Queens as 2 and Staten Island as 5. This brings in an implication that Manhattan is closer to Queens than Staten Island, which is neither necessary nor valid. (3) SMFL natively does not support integrity constraints as it predicts its values from a spatial correlation perspective, which is relatively "softer" and does not apply to constraint-based cleaning.

Sparcle (n=2) VS Sparcle (n=0). We compare Sparcle (n=2) with Sparcle (n=0) as an ablation study that shows the impact of *distance weighting* concept on real data. Sparcle (n=2) and Sparcle (n=0) have similar accuracy on dense datasets like NYC-Crash, but Sparcle (n=2) is superior on smaller datasets like Boston-311. The reason is that given a fixed kNN parameter, in dense datasets, most neighbors belong to the same class, hence distance weighting may not have an impact. Yet, in other datasets, neighbors belong to different classes, and hence distance weighting will have the most impact. This shows that *distance weighting*, as a crucial concept for spatial data, enhances the cleaning robustness regrading to various datasets.

Figure 6 shows the impact of duplication ratio on the accuracy of all systems. To control the duplication ratio, we use the Chicago-Synthetic dataset (20K records with 2K errors) in Figure 6a to 6c, where we measure the precision, recall, and F1 score when having error duplication ratios of 0, 0.33, 0.67, and 1 for the dependency $(Lat, Lon) \rightarrow ward$. A duplication ratio of 0 means that none of the erroneous records happen in a location of other records, while a ratio of 1 means that all erroneous records happened in the same location of some other records. Sparcle with n=2 significantly outperforms HoloClean in low duplicate ratios for precision, recall, and F1 score, and gives similar performance for duplicate ratio of 1. The trend is similar for Baran+Sparcle and Baran. The reason is that having erroneous records with duplicate values gives HoloClean and Baran the chance to learn the correct values from the duplicates, and hence can perform better. However, Sparcle can still perform well even with 0 duplicates, as it employs the spatial neighborhood concept, which somehow considers records with spatial proximity as duplicates. Meanwhile, Sparcle with n=2 performs much better than the case when n=0, and the superiority increases with high duplicate ratios. The main reason is that with more duplicates, it becomes more important to set accurate weights for each record with respect to satisfying the functional dependency. Hence, Sparcle with n=2 takes advantage of its distance weighting concept to accommodate this. SMFL also cannot benefit from duplicates because it does not enforce functional dependencies.

Figure 6d shows the same experiment of Figure 6c, yet for the four real datasets of Austin, Boston, NYC, and Chicago that have duplicate ratios 0, 0.01, 0.36, 0.61. For each dataset, we compute the duplicate ratio as a weighted average of the duplicate ratios of its dependencies. For all cases, Sparcle significantly improves their base systems.

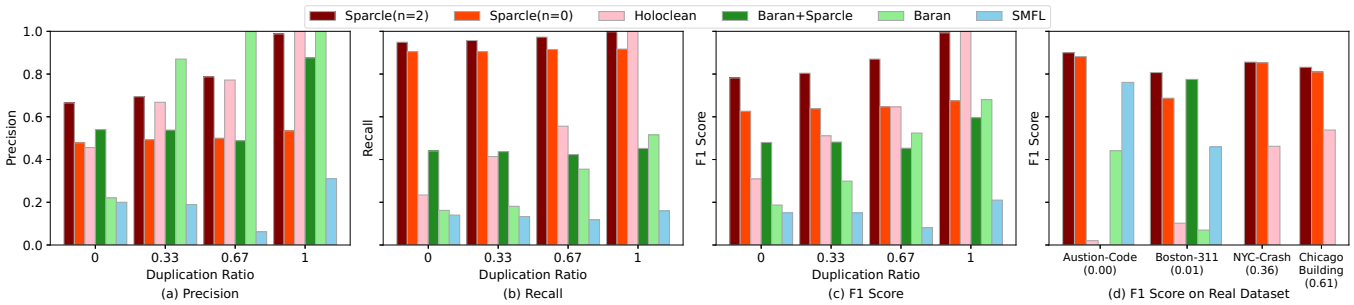


Figure 6: Impact of Duplication Ratio

Table 5: Accuracy per Attribute in Chicago-Synthetic.

		police	ward	zipcode	beat	census
Prec.	Sparcle (n=2)	**0.97**	**0.67**	0.86	0.60	**0.35**
	Sparcle (n=0)	0.95	0.47	0.79	0.49	0.25
	HoloClean	0.61	0.46	0.21	0.43	0.03
	Baran+Sparcle	0.95	0.54	**0.90**	**0.76**	0.33
	Baran	0.46	0.40	0.44	0.60	0.23
	SMFL	0.54	0.20	0.14	0.03	0.01
Rec.	Sparcle (n=2)	**0.99**	**0.95**	**0.98**	**0.93**	**0.84**
	Sparcle (n=0)	**0.99**	0.90	0.97	0.89	0.77
	HoloClean	0.31	0.23	0.11	0.21	0.01
	Baran+Sparcle	0.50	0.44	0.46	0.39	0.26
	Baran	0.13	0.06	0.07	0.01	0.00
	SMFL	0.29	0.14	0.12	0.02	0.01
F1.	Sparcle (n=2)	**0.98**	**0.78**	**0.92**	**0.73**	**0.49**
	Sparcle (n=0)	0.97	0.62	0.87	0.63	0.38
	HoloClean	0.41	0.31	0.14	0.28	0.02
	Baran+Sparcle	0.66	0.48	0.61	0.51	0.29
	Baran	0.18	0.08	0.09	0.20	0.01
	SMFL	0.38	0.15	0.12	0.02	0.01

6.3 Impact of Distinct Values on Accuracy

To better study the impact of the number of distinct values, we use the Chicago-Synthetic dataset (20K records, 2K errors) with five different dependencies, each with different number of distinct values as outlined in Table 3. To visualize the challenges, Figure 7 plots the map of Chicago outlines the possible values of the five dependency attributes, *police_district*, *ward*, *zipcode*, *city*, and *census_tract*, with 23, 50, 59, 275, and 801 distinct values, respectively. Table 5 gives the precision, recall, and F1 score for all systems on the five dependencies. Sparcle and Baran+Sparcle clearly boost accuracy for HoloClean and Baran in all measures and dependencies. A general trend is that the accuracy of all techniques degrade with the increase of the number of distinct values, yet, Sparcle is way more resilient to number of distinct values than its competitors. This is to the extent that, for the census tract, the F1 score for HoloClean, Baran and SMFL is 0.02, 0.01 and 0.01, respectively, while it is 0.49 for Sparcle (n=2). This shows that Sparcle is still able to clean a good ratio of the records, while other systems cannot really clean any record here. The main reason is that, as plotted in Figure 7e for census tract boundaries, with more area distinct values, there are longer boundaries between each two values and more records close to the boundary, which are naturally hard to clean.

Table 6: Running Time on Real Data

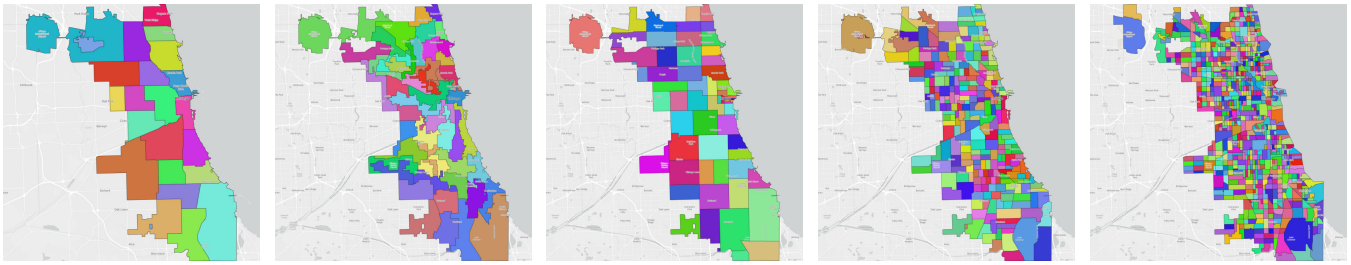
	Austin	Boston	Chicago	NYC
Sparcle	22m10s	2m17s	2h14m29s	3h58m10s
HoloClean	17m39s	2m08s	1h55m28s	3h06m55s
Baran+Sparcle	-#	5h58m34s	-#	-#
Baran	1h03m50s	5m53s	-*	-#
SMFL	38m23s	0m30s	-*	-*

* cannot finish due to memory error
\# cannot finish after 1 day

It is interesting to see that though *zipcode* dependency has slightly more distinct values than *ward* dependency, but Sparcle actually performs better for it. The main reason is that per the boundary maps of Figures 7b and 7c, the *ward* areas have more complex shapes than *zipcode* areas, which means that there are longer boundaries between areas, and hence it is much harder to clean. Overall, the shape of the boundary areas as well as the number of areas (i.e., distinct value) impact the accuracy of all techniques. Table 5 shows the impact of the *distance weighting* concept in Sparcle, where having n=2 makes Sparcle way more resilient to larger numbers of distinct values. With n=0, the accuracy of Sparcle degrades much higher with the increase of distinct values.

6.4 System Efficiency

Table 6 gives the running time of all systems. SMFL and Baran-based systems could not finish for Chicago and NYC datasets as they are proposed as in-memory frameworks and cannot handle out-of-memory large datasets. Baran takes much more time than HoloClean-based systems for Austin and Boston as it assumes dependency between every pair of attributes, hence ends up in processing too many dependencies, which takes significant execution time overhead. SMFL is much faster for Boston and but much slower for Austin, indicating its high time complexity. For all datasets, Sparcle encounters 17% to 29% extra overhead than HoloClean, mainly due to the time taken to build the *DistanceMatrix* as a self-join of the input data. Given that spatial operations are pretty expensive compared to non-spatial operations, having less than 30% overhead is highly acceptable. Baran+Sparcle takes orders of more time than Baran since it is only a proof-of-concept with no efficiency optimization. On one hand, this means that adding spatial awareness into data cleaning systems is non-trivial, requiring system implementation and careful efficiency consideration. On the other hand, this again indicates the importance of efficiency optimizations of Sparcle that lives inside HoloClean.



(a) Police District (23) (b) Ward (50) (c) Zipcode (59) (d) Beat (275) (e) Census Tract (801)

Figure 7: Map of Dependency Attribute of Chicago-Synthetic

7 RELATED WORK

Rule-based Data Cleaning. Motivated by a real need, there has been huge efforts over the last two decades for automated data cleaning [24–26]. By far, the most common of these approaches are the rule-based data cleaning systems (e.g., [14, 19, 20, 28, 34, 44, 45]), where it is deployed in open-source systems [3, 22, 33, 41] and industry [27, 50, 51]. Almost all rule-based approaches share the core idea of trying to minimize or eliminate a set of constraint violations, where a constraint is presented as a dependency rule that needs to be followed by any given dataset. Recent notable examples of such approaches include the HoloClean [44] and Baran [34] systems. HoloClean is a holistic rule-based data cleaning system that was first relying on Markov Logic Network [44], which is later replaced by an attention-based learning network [52] in its open source distribution [22]. Baran [34] is a configuration-free, human-in-the-loop data cleaning system that assumes functional dependencies between each pair of attributes. Unfortunately, rule-based data cleaning systems fall short in supporting spatial data as they mainly rely on the exact co-occurrence of record values, which would rarely happen for spatial data captured by inherently inaccurate devices. Sparcle aims to fix this by injecting spatial awareness into the core engine of rule-based data cleaning systems, boosting their accuracy when dealing with spatial data.

Relaxed Functional Dependencies. Earlier data cleaning approaches have considered the use of the approximation in the functional dependency rules [13, 14, 28, 43, 48, 55], along with a relaxed form of functional dependencies [9], including matching dependency [16], metric dependency [29], differential dependency [47], and ontology dependency [5]. This was mainly proposed to tolerate marginal syntactic difference for entities that are actually considered the same, e.g., the words "Ave." and "Avenue" should mean the same thing. Though one can use such relaxed dependencies to mimic the *spatial neighborhood* concept in Sparcle, still such dependencies fall short in achieving the objectives of Sparcle for four main reasons: (a) None of such relaxed dependencies support the *Distance Weighting* concept, which is a cornerstone in Sparcle and its applications, where two records could match the dependency in more stronger terms than two other records. (b) They are applicable to applications where errors are *few* and occasional, which is not the case for spatial data, where it is pretty rare to have two records with the same coordinates, and hence to that relaxed dependencies, *all* records are erroneous (c) Their goal is to catch two entries of the *same* record, while in spatial data, two records with nearby coordinates are truly two *different* records, and (d) They are actually not deployed in open-source rule-based data cleaning systems [34, 44, 52], as it is pretty challenging to do so and they are designed to solve certain problems of interest, while Sparcle main objective is to be injected inside the core engine of such systems.

Spatial Data Cleaning. Within the spatial computing community, spatial data cleaning approaches mainly focus on improving the quality of spatial attributes [30], but not on following any kind of constraint rules. Examples include correcting erroneous GPS point through map matching [8, 39], employing signal triangulation to enhance the accuracy of indoor locations [31, 54], machine learning approaches for trajectory imputation [15, 37], and data analysis techniques to reduce the uncertainty of location data [10, 56]. None of these approaches apply to our case as our goal is not to correct the spatial data itself, but use the spatial information to enhance the accuracy of data cleaning systems for other attributes. Meanwhile, a recent approach aims to inject spatial locality awareness into matrix factorization [17] as a means of making it suitable for spatial data cleaning. However, matrix factorization does not support functional dependency rules, categorical data, and error detection, as it mainly repairs missing numerical data by imputing them based on other records. As Sparcle lives in a rule-based data cleaning system, it natively supports functional dependencies, categorical data, and error detection, which distinguish it from matrix factorization.

8 CONCLUSION

This paper presented Sparcle; a novel framework built inside the core engine of rule-based data cleaning systems to boost their accuracy when dealing with spatial data. Sparcle system architecture is made similar to rule-based data cleaning systems where it injects spatial awareness in every system component. Sparcle is composed of three main components, *spatial error detector*, *spatial candidate generator*, and *spatial input formulator*. It relaxes the functional dependency that drives data cleaning rules from "records with same location would have same value in a dependent attribute" to "records with more similar locations are more likely to have same value in a dependent attribute", which is more suitable for spatial attributes. Sparcle applies two main spatial concepts, *spatial neighborhood* and *distance weighting*. With spatial neighborhood, records within spatial proximity are considered similar and involved in the dependency constraint. With distance weighting, some records satisfy the dependency constraint more than others. Experimental results with real and synthetic datasets and a real deployment of Sparcle inside HoloClean data cleaning system show that Sparcle significantly boosts the accuracy of its host data cleaning system.

REFERENCES

[1] AustinCode [n.d.]. Austin Open Data. Austin Code Complaint Cases. https://data.austintexas.gov/Public-Safety/Austin-Code-Complaint-Cases/6wtj-zbtb.

[2] AWS [n.d.]. Prepare data for machine learning - Amazon SageMaker Data Wrangler - Amazon Web Services. https://aws.amazon.com/sagemaker/data-wrangler/.

[3] Baran [n.d.]. Baran. https://github.com/BigDaMa/raha.

[4] BaranSparcle [n.d.]. Baran+Sparcle. https://github.com/yhuang-db/baran-sparcle/tree/sparcle.

[5] Sridevi Baskaran, Alexander Keller, Fei Chiang, Lukasz Golab, and Jaroslaw Szlichta. 2017. Efficient Discovery of Ontology Functional Dependencies. In *Proceedings of the International Conference on Information and Knowledge Management, CIKM* (Singapore).

[6] Philip Bohannon, Michael Flaster, Wenfei Fan, and Rajeev Rastogi. 2005. A Cost-Based Model and Effective Heuristic for Repairing Constraints by Value Modification. In *Proceedings of the ACM International Conference on Management of Data, SIGMOD* (Baltimore, Maryland, USA).

[7] Boston311 [n.d.]. Analyze Boston. Boston 311 Service Requests. https://data.boston.gov/dataset/311-service-requests.

[8] Sotiris Brakatsoulas, Dieter Pfoser, Randall Salas, and Carola Wenk. 2005. On Map-Matching Vehicle Tracking Data. In *Proceedings of the International Conference on Very Large Data Bases, PVLDB* (Trondheim, Norway). ACM.

[9] Loredana Caruccio, Vincenzo Deufemia, and Giuseppe Polese. 2016. Relaxed Functional Dependencies - A Survey of Approaches. *IEEE Transactions on Knowledge and Data Engineering, TKDE* 28, 1 (2016).

[10] Reynold Cheng, Dmitri V. Kalashnikov, and Sunil Prabhakar. 2003. Evaluating Probabilistic Queries over Imprecise Data. In *Proceedings of the ACM International Conference on Management of Data, SIGMOD* (San Diego, California, USA).

[11] Fei Chiang and Renée J. Miller. 2011. A unified model for data and constraint repair. In *Proceedings of the International Conference on Data Engineering, ICDE* (Hannover, Germany).

[12] ChicagoBuilding [n.d.]. Chicago Data Portal. Building Permits. https://data.cityofchicago.org/Buildings/Building-Permits/ydr8-5enu.

[13] Xu Chu, Ihab F. Ilyas, and Paolo Papotti. 2013. Holistic data cleaning: Putting violations into context. In *Proceedings of the International Conference on Data Engineering, ICDE* (Brisbane, Australia).

[14] Michele Dallachiesa, Amr Ebaid, Ahmed Eldawy, Ahmed K. Elmagarmid, Ihab F. Ilyas, Mourad Ouzzani, and Nan Tang. 2013. NADEEF: a commodity data cleaning system. In *Proceedings of the ACM International Conference on Management of Data, SIGMOD* (New York, NY, USA).

[15] Mohamed M. Elshrif, Keivin Isufaj, and Mohamed F. Mokbel. 2022. Network-less trajectory imputation. In *Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, ACM SIGSPATIAL GIS* (Seattle, Washington).

[16] Wenfei Fan. 2008. Dependencies revisited for improving data quality. In *Proceedings of the ACM Symposium on Principles of Database Systems, PODS* (Vancouver, BC, Canada).

[17] Chenguang Fang, Yinan Mei, and Shaoxu Song. 2023. Matrix Factorization with Landmarks for Spatial Data. In *Proceedings of the International Conference on Data Engineering, ICDE* (Anaheim, CA, USA).

[18] GCP [n.d.]. Dataprep by Trifacta, Google Cloud. https://cloud.google.com/dataprep.

[19] Congcong Ge, Yunjun Gao, Xiaoye Miao, Bin Yao, and Haobo Wang. 2022. A Hybrid Data Cleaning Framework Using Markov Logic Networks. *IEEE Transactions on Knowledge and Data Engineering, TKDE* 34, 5 (2022), 2048–2062.

[20] Floris Geerts, Giansalvatore Mecca, Paolo Papotti, and Donatello Santoro. 2020. Cleaning data with Llunatic. *VLDB Journal* 29, 4 (2020), 867–892.

[21] Stella Giannakopoulou, Manos Karpathiotakis, and Anastasia Ailamaki. 2020. Cleaning Denial Constraint Violations through Relaxation. In *Proceedings of the ACM International Conference on Management of Data, SIGMOD* (Portland, OR, USA).

[22] HoloClean [n.d.]. HoloClean. http://www.holoclean.io/.

[23] IBM [n.d.]. Data Quality Tools and Solutions, IBM. https://www.ibm.com/data-quality.

[24] Ihab F. Ilyas and Xu Chu. 2015. Trends in Cleaning Relational Data: Consistency and Deduplication. *Found. Trends Databases* 5, 4 (2015), 281–393.

[25] Ihab F. Ilyas and Xu Chu. 2019. *Data Cleaning*. ACM.

[26] Ihab F. Ilyas and Felix Naumann. 2022. Data Errors: Symptoms, Causes and Origins. *IEEE Data Engineering Bulletin* 45, 1 (2022), 4–9.

[27] Inductiv [n.d.]. Inductiv. https://cs.uwaterloo.ca/news/waterloo-based-ai-start-up-inductiv-acquired-apple.

[28] 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 *Proceedings of the ACM International Conference on Management of Data, SIGMOD* (Melbourne, Australia).

[29] Nick Koudas, Avishek Saha, Divesh Srivastava, and Suresh Venkatasubramanian. 2009. Metric Functional Dependencies. In *Proceedings of the International Conference on Data Engineering, ICDE* (Shanghai, China).

[30] Huan Li, Hua Lu, Christian S. Jensen, Bo Tang, and Muhammad Aamir Cheema. 2023. Spatial Data Quality in the Internet of Things: Management, Exploitation, and Prospects. *ACM Comput. Surv.* 55, 3 (2023), 57:1–57:41.

[31] Xiao Li, Huan Li, Harry Kai-Ho Chan, Hua Lu, and Christian S. Jensen. 2023. Data Imputation for Sparse Radio Maps in Indoor Positioning. In *Proceedings of the International Conference on Data Engineering, ICDE* (Anaheim, CA, USA).

[32] Ester Livshits, Benny Kimelfeld, and Sudeepa Roy. 2020. Computing Optimal Repairs for Functional Dependencies. *ACM Transactions on Database Systems, TODS* 45, 1 (2020), 4:1–4:46.

[33] Llunatic [n.d.]. Llunatic. https://github.com/donatellosantoro/Llunatic.

[34] Mohammad Mahdavi and Ziawasch Abedjan. 2020. Baran: Effective Error Correction via a Unified Context Representation and Transfer Learning. *Proceedings of the International Conference on Very Large Data Bases, PVLDB* 13, 11 (2020), 1948–1961.

[35] Mohammad Mahdavi, Ziawasch Abedjan, Raul Castro Fernandez, Samuel Madden, Mourad Ouzzani, Michael Stonebraker, and Nan Tang. 2019. Raha: A Configuration-Free Error Detection System. In *Proceedings of the ACM International Conference on Management of Data, SIGMOD* (Amsterdam, The Netherlands). ACM.

[36] MS [n.d.]. Data Cleansing - Data Quality Services (DQS) in Microsoft SQL Server. https://learn.microsoft.com/en-us/sql/data-quality-services/data-cleansing?view=sql-server-ver16.

[37] Mashaal Musleh and Mohamed F. Mokbel. 2023. A Demonstration of KAMEL: A Scalable BERT-based System for Trajectory Imputation. In *Proceedings of the ACM International Conference on Management of Data, SIGMOD* (Seattle, WA, USA).

[38] NaiveBayes [n.d.]. NaiveBayes. https://en.wikipedia.org/wiki/Naive_Bayes_classifier.

[39] Paul Newson and John Krumm. 2009. Hidden Markov map matching through noise and sparseness. In *Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, ACM SIGSPATIAL GIS* (Seattle, Washington, USA).

[40] NYCOpenData [n.d.]. NYC Open Data. Motor Vehicle Collisions - Crashes. https://data.cityofnewyork.us/Public-Safety/Motor-Vehicle-Collisions-Crashes/h9gi-nx95.

[41] openclean [n.d.]. openclean. https://github.com/VIDA-NYU/openclean.

[42] PostGIS [n.d.]. PostGIS. https://postgis.net/.

[43] Nataliya Prokoshyna, Jaroslaw Szlichta, Fei Chiang, Renée J. Miller, and Divesh Srivastava. 2015. Combining Quantitative and Logical Data Cleaning. *Proceedings of the International Conference on Very Large Data Bases, PVLDB* 9, 4 (2015), 300–311.

[44] Theodoros Rekatsinas, Xu Chu, Ihab F. Ilyas, and Christopher Ré. 2017. HoloClean: Holistic Data Repairs with Probabilistic Inference. *Proceedings of the International Conference on Very Large Data Bases, PVLDB* 10, 11 (2017), 1190–1201.

[45] El Kindi Rezig, Mourad Ouzzani, Walid G. Aref, Ahmed K. Elmagarmid, Ahmed R. Mahmood, and Michael Stonebraker. 2021. Horizon: Scalable Dependency-driven Data Cleaning. *Proceedings of the International Conference on Very Large Data Bases, PVLDB* 14, 11 (2021), 2546–2554.

[46] Matthew Richardson and Pedro M. Domingos. 2006. Markov logic networks. *Mach. Learn.* 62, 1-2 (2006).

[47] Shaoxu Song and Lei Chen. 2011. Differential dependencies: Reasoning and discovery. *ACM Transactions on Database Systems, TODS* 36, 3 (2011), 16:1–16:41.

[48] Shaoxu Song, Aoqian Zhang, Lei Chen, and Jianmin Wang. 2015. Enriching Data Imputation with Extensive Similarity Neighbors. *Proceedings of the International Conference on Very Large Data Bases, PVLDB* 8, 11 (2015), 1286–1297.

[49] Sparcle [n.d.]. Sparcle. https://github.com/yhuang-db/holoclean-sparcle/tree/latest-aimnet-310-sparcle.

[50] Tamr [n.d.]. Tamr. https://www.tamr.com/.

[51] Trifacta [n.d.]. Trifacta. https://www.trifacta.com/.

[52] Richard Wu, Aoqian Zhang, Ihab F. Ilyas, and Theodoros Rekatsinas. 2020. Attention-based Learning for Missing Data Imputation in HoloClean. In *Proceedings of Machine Learning and Systems, MLSys* (Austin, TX, USA).

[53] Mohamed Yakout, Laure Berti-Équille, and Ahmed K. Elmagarmid. 2013. Don't be SCAREd: use SCalable Automatic REpairing with maximal likelihood and bounded changes. In *Proceedings of the ACM International Conference on Management of Data, SIGMOD* (New York, NY, USA).

[54] Faheem Zafari, Athanasios Gkelias, and Kin K. Leung. 2019. A Survey of Indoor Localization Systems and Technologies. *IEEE Commun. Surv. Tutorials* 21, 3 (2019), 2568–2599.

[55] Zheng Zheng, Longtao Zheng, Morteza Alipour Langouri, Fei Chiang, Lukasz Golab, Jaroslaw Szlichta, and Sridevi Baskaran. 2022. Contextual Data Cleaning with Ontology Functional Dependencies. *ACM J. Data Inf. Qual.* 14, 3 (2022), 20:1–20:26.

[56] Andreas Züfle, Goce Trajcevski, Dieter Pfoser, Matthias Renz, Matthew T. Rice, Timothy Leslie, Paul L. Delamater, and Tobias Emrich. 2017. Handling Uncertainty in Geo-Spatial Data. In *Proceedings of the International Conference on Data Engineering, ICDE* (San Diego, CA, USA).