Data Polygamy: The Many-Many Relationships among Urban Spatio-Temporal Data Sets

Fernando Chirigati[*], Harish Doraiswamy[*], Theodoros Damoulas[⋆†], Juliana Freire[*]
[*] New York University [⋆] University of Warwick [†] Alan Turing Institute
{fchirigati,harishd,juliana.freire}@nyu.edu damoulas@warwick.ac.uk

ABSTRACT

The increasing ability to collect data from urban environments, coupled with a push towards openness by governments, has resulted in the availability of numerous spatio-temporal data sets covering diverse aspects of a city. Discovering relationships between these data sets can produce new insights by enabling domain experts to not only test but also generate hypotheses. However, discovering these relationships is difficult. First, a relationship between two data sets may occur only at certain locations and/or time periods. Second, the sheer number and size of the data sets, coupled with the diverse spatial and temporal scales at which the data is available, presents computational challenges on all fronts, from indexing and querying to analyzing them. Finally, it is non-trivial to differentiate between meaningful and spurious relationships. To address these challenges, we propose *Data Polygamy*, a scalable topology-based framework that allows users to query for statistically significant relationships between spatio-temporal data sets. We have performed an experimental evaluation using over 300 spatial-temporal urban data sets which shows that our approach is scalable and effective at identifying interesting relationships.

1. INTRODUCTION

Urban environments are the loci of economic activity and innovation. At the same time, most cities face huge challenges around transportation, resource consumption, housing affordability, and inadequate or aging infrastructure. The growing volumes of urban data being collected and made available [3, 19, 26, 27, 32] open up new opportunities for city governments and social scientists to engage in data-driven science to better understand cities, make them more efficient, and improve the lives of their residents.

Urban data is unique in that it captures the behavior of the different components of a city over space and time: its citizens, existing infrastructure (physical and policies), and the environment [20]. The availability of these data makes it possible to not only better understand the individual components but also obtain insights into how they interact. When an expert finds an *unexpected* pattern or feature in a data set, other related data may help explain why and under which conditions the pattern occurs. Consider the top plot in Figure 1, which shows the number of daily taxi trips in New York City (NYC) during 2011 and 2012. While the distribution of trips over time is very similar for the two years, we observe two large drops: one in August 2011 and another in October 2012. A natural question is what might have caused these drastic reductions. By examining wind speed data (bottom plot in Figure 1), we discover that these drops occur on days with unusually high wind speeds; here, the high wind speeds were due to hurricanes Irene and Sandy. This suggests a new hypothesis to be further investigated: high wind speed leads to significant reduction in the number of taxi trips.

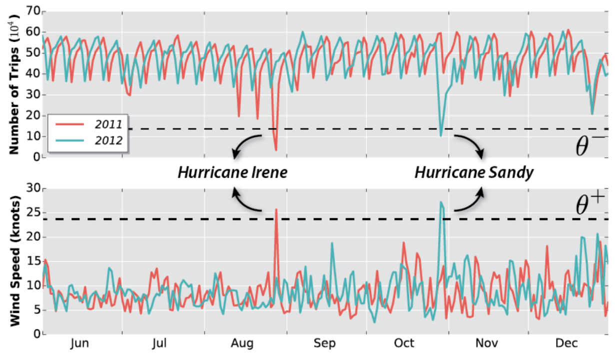


Figure 1: Variation of the number of taxi trips in NYC and its relationship with wind speed.

Besides enabling *hypothesis generation*, studying relationships among data sets can also help with *hypothesis testing*. For instance, the difficulty in finding taxis when it is raining is a notorious problem in Manhattan. One long-standing hypothesis to explain this behavior is that taxi drivers set a daily income goal, and since there is higher demand on rainy days, they reach their goal faster and stop working earlier. Testing for the presence of such a relationship between data sets—in this case, NYC taxi data and weather data—can help experts at the NYC Taxi and Limousine Commission (TLC) frame appropriate policies to counter identified problems.

Relationship Discovery. In this paper, we define and take a first step towards addressing the problem of discovering potential *relationships* between spatio-temporal data sets. We aim to ***guide*** users in the data analysis and exploration process by allowing them to pose ***relationship queries***:

Find all data sets related to a given data set $\mathbb{D}$.

To evaluate the above query, we have to first determine *when and how two data sets are related*. Answering this question in the context of urban data sets gives rise to several challenges. Urban data can be large vertically, containing hundreds of millions to billions of data points, and horizontally, consisting of several attributes [3]. As a point of comparison, five years of taxi data contains 780 million trips [34], and the weather data set has over 200 attributes [35].

Also, there is a large number of urban data sets. NYC alone has published over 1,300 data sets in the past two years [27], and this is just a small fraction of the data collected by the city. Since a data set can be related to zero or more data sets through multiple attributes, there is a combinatorially large number of possible relationships.

This problem is compounded by the fact that these data sets contain both spatial and temporal attributes at different resolutions. For example, values for the weather attributes are collected at hourly intervals (temporal resolution) for the whole city (spatial resolution). In contrast, NYC taxi trips are associated with GPS coordinates with time precision in seconds. Other data sets use spatial resolutions at the level of neighborhoods or zip codes, and temporal resolutions as daily, weekly, and monthly intervals. Since relationships can materialize at any of these resolutions, they should be evaluated at multiple resolutions.

The data complexity coupled with the sheer number of available data sets and the combinatorially large number of possible relationships make it hard for domain experts to comprehend the information and the insights it can potentially offer. Of the several thousand possible relationships between pairs of attributes in different data sets, only a small fraction is actually informative. Unless known a priori, looking for meaningful relationships between these data sets is like, as the cliché goes, "finding a needle in a haystack".

Another challenge in identifying a possible relationship lies in defining the conditions implicating such a relationship. For example, consider the wind speed data and the NYC taxi trip data depicted in Figure 1. There is no apparent relation between the two data sets during the normal course of time: it is only when the wind speed is abnormally high (in that case, due to hurricanes), that we can see a connection with taxi trips. This is a common pattern observed across urban data sets, where relationships become visible only at *spatio-temporal regions* (locations in space and time) that behave differently compared to the regions' neighborhood.

Standard techniques, such as Pearson correlation coefficient [7] or dynamic time warping [21], do not capture these relationships because they ignore the spatio-temporal dependencies inherent in the data and operate globally over the entire data (see Section 6.4). Therefore, we need a method that captures the variation of the data over space and time at different and arbitrary resolutions.

Our Approach. To address these challenges, we propose the ***Data Polygamy*** framework. We introduce the notion of *topology-based relationships*, where two data sets are related if there is a relationship between the *salient features* of the data. A salient feature corresponds to a spatio-temporal region that exhibits an unusual behavior with respect to its neighborhood. To efficiently identify salient features, we use and extend techniques from computational topology. Topology-based techniques are naturally suited for studying properties of data involving spatial and geometric domains (e.g., see [11, 28, 38]). To give some intuition for why and how we apply topology, suppose we model a time step in an urban data set as a terrain, where the height of each point of the terrain represents the data value at that spatial location. In this case, the variation over space is captured by the peaks and valleys of this terrain. This can be extended to include time by modeling the data as a high dimensional terrain. The salient features, which as mentioned earlier correspond to spatio-temporal regions behaving differently from their neighborhood, are inherently represented as tall peaks and deep valleys. Topological methods provide *efficient algorithms to represent and compute such features*. In addition, they can identify features that have an *arbitrary spatial structure* and *straddle multiple time intervals*; they are also *generic*, in the sense that they work on data having different dimensions and resolutions without requiring any modification.

Given two data sets, to determine whether they are related, we assess how similar their corresponding terrains are, i.e., the similarities in the spatio-temporal variation patterns of the data sets. In the *Data Polygamy* framework, this is accomplished in three steps:

1. *Data Set Transformation.* Each attribute of the two data sets is transformed into a *scalar function*. A scalar function provides a mathematical representation of the terrain corresponding to a particular attribute of a data set.
2. *Feature Identification.* A topological data structure is computed for every scalar function, which provides an abstract representation of the peaks and valleys of the scalar function. This structure is used as an *index to efficiently identify salient features* in the data, which are defined based on thresholds that capture the extent of normal behavior of the scalar function. We develop a new method based on the notion of *topological persistence* [8] to automatically compute these thresholds.
3. *Relationship Evaluation.* Possible relationships are then identified based on feature similarity. Our framework filters out relationships that are *not statistically significant*. Since existing Monte Carlo methods assume independence across samples, we develop restricted Monte Carlo permutation tests that respect data dependencies due to spatial and temporal proximity.

Users can then pose relationship queries over the resulting relationships. Hypothesis generation is supported by querying for relationships among all data sets, while a given hypothesis can be tested by querying for relationships between the data sets involved in the hypothesis. Sections 2, 3, and 4 provide the formal definitions and describe the algorithms used in these stages. The end-to-end *Data Polygamy* framework is presented in Section 5.

Because we consider a large number of urban data sets, each containing many attributes, we need to compute thousands of scalar functions and derive millions of relationships. However, both salient feature identification and relationship querying are embarrassingly parallel operations. In Section 5.4, we briefly describe a map-reduce implementation of the *Data Polygamy* framework.

We demonstrate the efficiency and robustness of our framework in Section 6 through an experimental evaluation using over 300 urban data sets of varying spatio-temporal resolutions. We also present use cases demonstrating its effectiveness at identifying informative relationships.

Note that our goal is to support users in the data exploration process by helping them discover data sets that may be relevant for their task—similar to a search engine that returns a set of potentially relevant documents for a given keyword query. Users can then use the identified relationships for further analysis, such as testing for spurious relationships, testing for causality (e.g., to support a hypothesis), and generating new hypotheses.

Contributions. We define and propose a topology-based approach to the problem of identifying relationships across a large number of spatio-temporal data sets. Our main contributions are:

- We introduce the notion of *topology-based relationships* to determine whether data sets are related through salient features.
- We develop a *scalable framework* that identifies salient features based on the topology of the data, and a topology-based index that provides an *output-sensitive* strategy to compute these features: the time taken is linear in the size of the output. We also propose an *algorithm that automatically determines feature thresholds in a data-driven fashion*.
- We define the *relationship operator*, which returns the set of statistically significant relationships between two data sets. To determine whether a relationship is significant, we *develop a strategy that applies Monte Carlo permutation tests and respects data dependencies due to spatial and temporal proximity*.

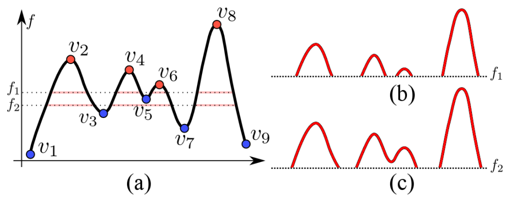


Figure 2: (a) A sample 1D scalar function. The labeled points form the set of maxima (red) and minima (blue). (b) The super-level set at f_1 consists of four components. (c) The super-level set at f_2 consists of three components.

- We describe a scalable, map-reduce implementation of the *Data Polygamy* framework.
- We perform an extensive experimental evaluation, using real and synthetic data, which shows that our framework is robust, efficient, and effective.

2. TOPOLOGY-BASED RELATIONSHIPS

In this section, we provide the required mathematical background and define the terms used in this paper, which are based on concepts from computational topology. We refer the reader to the following textbooks [13, 24] for a comprehensive discussion on these topics. We start by formally defining the concept of a topological feature and topology-based relationships. Then, we propose two measures to evaluate these relationships.

2.1 Topological Features

To discover relationships between two data sets, we first identify the set of topological features of the *scalar functions* that represent the data sets.

Scalar Functions. Let $\mathbb{D}$ be a data set, and A an attribute of $\mathbb{D}$. To identify the set of features with respect to the attribute A, we first represent the attribute as a *time-varying scalar function*.

Definition 1. A *scalar function* $f : \mathbb{S} \to \mathbb{R}$ maps points on a spatial domain $\mathbb{S}$ onto a real value.

Definition 2. A *time-varying scalar function* $f : [\mathbb{S} \times \mathbb{T}] \to \mathbb{R}$ maps points on a spatial domain across time onto a real value.

The spatial resolution of the data set $\mathbb{D}$ determines the structure of the spatial domain $\mathbb{S}$. For example, the NYC weather data set provides information on different climate attributes, such as temperature, precipitation, and wind speed. The values of these attributes correspond to an hourly time period for the entire city, i.e., all the values correspond to the same spatial point. In this case, the domain $\mathbb{S} \times \mathbb{T}$ of the time-varying scalar function is a simple time series, i.e., a 1D function (0D in space and 1D in time). Figure 2(a) and the two time series shown in Figure 1 are examples of 1D scalar functions. On the other hand, the NYC taxi data consists of a set of taxi trips, each containing the GPS coordinates for pick-up and drop-off locations. From these data, we can obtain a distribution of taxi trips over space and time by partitioning NYC into a set of polygons (e.g., neighborhoods) and counting the trips that start (or end) in each polygon at different time steps. This is a *density function*, where the spatial domain is 2D, and thus the time-varying scalar function is 3D (2D in space and 1D in time). Figure 3 shows the density function at two different spatial resolutions for one time step (i.e., one hour time period). The different scalar functions that can be used to represent a data set are discussed in Section 5.1.

Irrespective of the temporal resolution, time always contributes to one dimension in the time-varying scalar function. Unless otherwise noted, we use the term *scalar function* to refer to a time-varying scalar function corresponding to a (data set, attribute) pair.

Topological Features. Interesting features of a scalar function f are captured by the *critical points* of f.

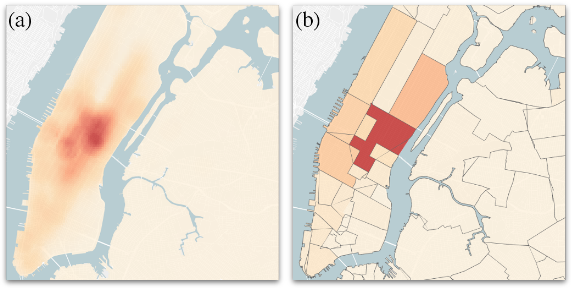


Figure 3: One time step (2D slice) of the 3D function representing the density of taxi trips in NYC at different resolutions. Dark and light regions correspond to high and low trip density, respectively. (a) NYC is represented using a high-resolution grid and the density is provided for each cell of this grid. (b) A lower resolution, at the level of neighborhood, is used.

Definition 3. Given a smooth function f, the *critical points* of f are the points where the gradient becomes zero, i.e., $\nabla f = 0$.

We assume that the scalar function f is a Morse function [24]. A Morse function has the property that (i) no two critical points have the same function value; and (ii) there are no degenerate critical points (i.e., $\nabla^2 f \neq 0$). Any function f can be made Morse via a simulated perturbation of the function by an infinitesimally small value such that no two points have the same function value [12]. We provide a detailed discussion on Morse functions in Appendix B.1.

We are interested in two particular types of critical points, *maximum* and *minimum*, collectively known as *extrema*. Given a Morse function, maximum and minimum points are defined as follows:

Definition 4. A point x is a *maximum* if $f(x) > f(x')$, $\forall x' \in N(x)$, where $N(x)$ defines the local neighborhood of x. Similarly, x is a *minimum* if $f(x) < f(x')$, $\forall x' \in N(x)$.

The red and blue points in Figure 2(a) correspond to the set of maxima and minima, respectively. We use the neighborhood of critical points of a function to represent the topological features of the data. The neighborhoods of the maxima and minima of f are captured by the *super-level* and *sub-level sets* of f, respectively.

Definition 5. Given a scalar function f, the *super-level set* at a real value θ is defined as $f^{-1}([\theta, \infty))$, i.e., the pre-image of the interval $[\theta, \infty)$. The *sub-level set* at θ is defined as $f^{-1}((-\infty, \theta])$.

In other words, the super-level set at a real value θ is the set of all points on the domain of f having function value greater than or equal to θ. For example, the super-level set of the function in Figure 2(a) at function value f_1 consists of 4 components (Figure 2(b)), while the super-level set at f_2 consists of 3 components (Figure 2(c)). Similarly, the sub-level set at θ is the set of all points in the domain of f having function value less than or equal to θ.

We define two types of features—positive and negative—using super-level and sub-level sets, respectively.

Definition 6. Given a feature threshold θ^+, the set of ***positive features*** is defined as the super-level set $f^{-1}([\theta^+, \infty))$.

Definition 7. Given an feature threshold θ^-, the set of ***negative features*** is defined as the sub-level set $f^{-1}((-\infty, \theta^-])$.

Feature Representation. The spatial domain $\mathbb{S}$ of $\mathbb{D}$ is represented as a set of regions $\{s_1, s_2, \ldots, s_n\}$ that partition the spatial extent of $\mathbb{D}$. Each region s_i corresponds to a polygon defined by the resolution used. For instance, the lowest resolution consists of the space represented as a single region or polygon. By using smaller polygons to partition the space, one could obtain a higher resolution representation as shown in Figure 3(a). The temporal domain $\mathbb{T}$ is represented as a set of time intervals $\{t, t+\delta, t+2\delta, \ldots\}$. The temporal resolution is defined by the value of δ. For example, when $\delta =$ 1 hour, the function is specified for hourly time steps.

Definition 8. A *spatio-temporal point* is represented by a (spatial region, time interval) pair.

Topological features of f correspond to a set of spatio-temporal points over the domain of f. Intuitively, they represent spatio-temporal points where attribute A of data set $\mathbb{D}$ deviates from its normal behavior, and capture the variation of A over both space and time. Here, the thresholds θ^+ and θ^- define the extent of normal behavior of A. Salient features of the function can be identified by appropriately setting the values of these thresholds. For example, using the indicated values of θ^+ and θ^- in Figure 1, features corresponding to the hurricanes are obtained. We describe an algorithm to identify the appropriate values for θ^+ and θ^- in Section 3.3.

2.2 Feature Relatedness

Consider two scalar functions: $f_1(\mathbb{D}_1, A_1)$ corresponding to attribute A_1 of data set $\mathbb{D}_1$, and $f_2(\mathbb{D}_2, B_1)$ corresponding to attribute B_1 of $\mathbb{D}_2$. Without loss of generality, we assume that the two functions have the same spatial and temporal resolution. Let Σ_1 and Σ_2 be the set of features of f_1 and f_2, respectively. Let $\Sigma = \Sigma_1 \bigcap \Sigma_2$ be the set of all spatio-temporal points that are features in both f_1 and f_2. The possible relationships between two functions are defined based on the relationship between their features.

Definition 9. Two functions f_1 and f_2 are *feature-related* at a spatio-temporal point $x = (s,t)$ if $x \in \Sigma$.

At points not in Σ, the two functions are not feature-related. Let $\Sigma_i^+ \subset \Sigma_i$ be a set such that $\forall x \in \Sigma_i^+, x$ is a positive feature. Similarly, let $\Sigma_i^- \subset \Sigma_i$ be the set of negative features.

Definition 10. f_1 and f_2 are *positively related* at a spatio-temporal point $x \in \Sigma$ if ($x \in \Sigma_1^+$ and $x \in \Sigma_2^+$) or ($x \in \Sigma_1^-$ and $x \in \Sigma_2^-$).

Definition 11. f_1 and f_2 are *negatively related* at a spatio-temporal point $x \in \Sigma$ if ($x \in \Sigma_1^+$ and $x \in \Sigma_2^-$) or ($x \in \Sigma_1^-$ and $x \in \Sigma_2^+$).

For instance, consider the features from Figure 1 corresponding to the indicated thresholds. At the spatio-temporal point corresponding to hurricane Sandy, there is a negative feature in the taxi density function and a positive feature in the wind speed function. The functions are therefore negatively related at that point.

2.3 Relationship Score and Strength

To assess the relationship between a given pair of functions, we define the following two measures.

Relationship Score τ. We are interested in evaluating the overall nature of the relationship between two functions, i.e., whether it is always positive, always negative, or somewhere in between. To do so, we define the *relationship score* τ between two functions f_1 ($\mathbb{D}_1, A_1$) and f_2 ($\mathbb{D}_2, B_1$). Let Σ_1 and Σ_2 denote the set of features of f_1 and f_2, respectively. As defined earlier, the set $\Sigma = \Sigma_1 \bigcap \Sigma_2$ denotes the feature-relations between the two functions. Let $\#p$ and $\#n$ be number of positive and negative feature relations in Σ. The relationship score is defined as

$$\tau = \frac{\#p - \#n}{|\Sigma|} \tag{1}$$

A value of τ closer to $+1$ indicates that the two functions are positively related, while a value closer to -1 indicates that the functions are negatively related.

Relationship Strength ρ. This measure is used to capture how frequently features in two functions are related: the more frequently the features are related, the stronger the relationship is. We model the set of features as binary classifiers. Consider any spatio-temporal point $x \in \Sigma_1$. If x is also present in Σ, it is considered as a true positive. A point x is a false positive when $x \in \Sigma_1$ and $x \notin \Sigma$. Similarly,

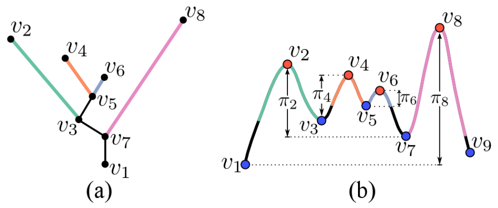


Figure 4: (a) Join tree of the function shown in Figure 2. The edges are colored based on the descending path traversed from the corresponding maxima illustrated in (b). π_i denotes the persistence of maximum v_i.

x is false negative when $x \notin \Sigma_1$ and $x \in \Sigma$. We then use the F1 score to measure the relationship strength as

$$\rho = F_1(f_1, f_2) = 2 \times \frac{precision \times recall}{precision + recall} \tag{2}$$

Note that *precision* gives a measure of how often features in f_1 are related with features in f_2, and *recall* gives a measure of how often features in f_2 are related with features in f_1. Thus, a value of ρ closer to 1 indicates a strong relationship between the two functions, since a feature in one function almost always indicates a feature in the other function as well. Similarly, a value of ρ close to zero indicates a weak relationship.

3. MERGE TREE INDEX

We use a topological data structure called *merge tree* to efficiently identify salient features corresponding to a scalar function. In what follows, we give an overview of merge trees and introduce a new algorithm to compute the features thresholds θ^+ and θ^-, a crucial step in this process.

3.1 Index Creation

Recall that a super-level set (or sub-level set) of a scalar function f at a given function value consists of multiple connected components. Therefore, decreasing or increasing the function values changes the topology, i.e., the number of connected components. A *merge tree* tracks the evolution of super-level sets or sub-level sets of f with changing function value. Formally, there are two types of merge trees.

Definition 12. The *join tree* of f tracks the connected components of the super-level sets of f with decreasing function value.

Definition 13. The *split tree* of f tracks the connected components of the sub-level sets of f with increasing function value.

Consider the 1D function shown in Figure 2(a). At the highest function value, a single super-level set component is created at v_8. As we decrease the function value, the number of components remain at one until the function value is equal to that of the v_2. As we keep decreasing the function value, two more components are created at v_4 followed by v_6 (Figure 2(b)). However, when the function value reaches v_5, the components created at v_4 and v_6 merge into one component, reducing the number of components from 4 to 3 (Figure 2(c)). We stop this process when the function value goes below v_1 (the global minimum). At this point, there is a single super-level set component composed of the entire domain. The join tree tracks this evolution as a graph. Figure 4(a) shows the join tree of the 1D function from Figure 2(a). The nodes of the graph correspond to critical points where the number of components change, while an edge represents the connected super-level set component between its end points. For example, the edge (v_2, v_3) corresponds to the green connected super-level set component in Figure 4(b). The root node of a join tree is the global minimum of f, while the non-root leaf nodes correspond to the set of maxima of f. Similarly, the root node of a split tree corresponds to the global maximum and its non-root leaf nodes correspond to the set of minima of f. In order to compute merge trees, we first have to obtain a discrete representation of a scalar function, which we describe next.

Scalar Function Representation. Consider the spatial domain of a data set $\mathbb{D}$ consisting of regions $\{s_1, s_2, \dots, s_n\}$. Let the temporal domain of $\mathbb{D}$ consist of m time steps $\{t_1, t_2, \dots, t_m\}$. We create a graph $G = (V, E)$ to represent the spatio-temporal domain of $\mathbb{D}$ as follows. Vertex $v_{x,z} \in V$ represents the spatio-temporal point corresponding to region s_x at time t_z. Thus, $|V| = n \times m$. The edges $E = E_S \bigcup E_T$ are divided into two categories:

- *spatial edges*: $E_S = \{(v_{x,z}, v_{y,z}) \mid s_x \text{ adjacent to } s_y, \forall z \in [1,m]\}$
- *temporal edges*: $E_T = \{(v_{x,z}, v_{x,z+1}), \forall x \in [1,n],\ z \in [1,m)\}$

Edges in E_S connect adjacent regions of the space for each time step, and edges in E_T connect a region across adjacent time steps.

We use a piecewise linear (PL) function defined on G to represent the scalar function f: the function is defined on the vertices of G and linearly interpolated within each edge. The graph allows a single representation to be used irrespective of the dimension of the spatio-temporal domain, thus supporting different resolutions and dimensions of the data.

Merge Tree Computation. The merge tree of a PL function can be efficiently computed in $O(N \log N + M\alpha(M))$ time using the union-find data structure, where N and M are the number of vertices and edges, respectively, in G. Since the spatial domains considered in this work correspond to cities, the graph G representing these domains is planar. Thus, $M = O(N)$. The algorithm to compute join trees is given in Procedure ComputeJoinTree (for more details, see Appendix B.2). The split tree is computed analogously by using the function $f' = -f$ in this algorithm.

3.2 Querying Features

We use the join and split trees as indices to efficiently compute the set of features, i.e., the super-level sets and sub-level sets, respectively. Let θ be the feature threshold. The algorithm to compute the super-level set $f^{-1}([\theta, \infty))$ using the join tree J_T is as follows:

1. Identify the set $V^+ = \{v \mid v \text{ is a maximum and } f(v) \geq \theta\}$. This is accomplished by going over the non-root leaf nodes of J_T.
2. Set $\Sigma^+ = \emptyset$.
3. While $V^+ \neq \emptyset$
   (a) Remove v from V^+ and add to Σ^+.
   (b) Let $L^- = \{u | f(u) \leq f(v) \text{ and } u \text{ is adjacent to } v\}$.
   (c) Add $u \in L^-$ to V^+ if $\theta \leq f(u)$.
4. The set Σ^+ contains the vertices of G that belong to the super-level set at θ.

The algorithm performs a descending path traversal of adjacent vertices from the set of valid maxima (having function value greater than θ) until the required threshold is reached. The colored regions in Figure 4(b) indicate the descending paths followed by the algorithm starting from the different maxima of the function shown in Figure 2(a). This is analogous to traversing down the edges of the join tree. The sub-level set at θ is computed similarly, through an ascending path traversal starting from the minima of the function.

Time Complexity. Since the vertices of G are sorted when computing the join (or split) tree, the critical points of the function are also stored in sorted order. Thus, the number of comparisons required to identify V^+ (or V^-) is $|V^+|$ (or $|V^-|$). Each descending (or ascending) path traversal stops as soon as it reaches a vertex u that is not a feature. Thus, the number of vertices touched during querying is $O(\Sigma^+)$ (or $O(\Sigma^-)$). In other words, given the join and split trees, feature identification for a given threshold is ***output-sensitive***.

Procedure ComputeJoinTree

```
Require: Graph $G(V,E)$, Function $f$
 1: Sort $V$ in descending order of $f$
 2: for each $v \in V$ do
 3:     $L^+ = \{u|(v,u) \in E$ and $f(v) < f(u)\}$
 4:     $C = \{\text{Component}(u)|u \in L^+\}$
 5:     if $|C| = 0$ then          /* $v$ is a maximum and creator */
 6:         Create a new join component $C_J$
 7:         Set Head$(C_J) = v$, Creator$(C_J) = v$
 8:     else if $|C| = 1$ then               /* $v$ is not critical */
 9:         Add $v$ to $C$
10:     else      /* $v$ is a destroyer, $|C| = 2$ for Morse functions */
11:         Let $C = \{C_1, C_2\}$, $f(\text{Creator}(C_1)) < f(\text{Creator}(C_2))$
12:         Merge Components $C_J = C_1 \bigcup C_2$
13:         Let $u_1 =$ Head$(C_1)$, $u_2 =$ Head$(C_2)$
14:         Add edges $(u_1, v)$ and $(u_2, v)$ to Join Tree $J_T$
15:         Set Creator$(C_J)$ = Creator$(C_1)$, Head$(C_J) = v$
16:         Pair Creator$(C_2)$ with destroyer $v$
17:     end if
18: end for
19: return Join Tree $J_T$
```

3.3 Feature Threshold Computation

Intuitively, our goal is to classify topological features not adhering to normal behavior as *salient features*. We are also interested in identifying *extreme features*, which correspond to outliers among salient features. For instance, the extremely high wind speeds during a hurricane correspond to extreme features

While users with domain knowledge can provide thresholds for computing features, this might not be feasible over all data sets. Thus, we devise a data-driven approach to identify the required thresholds, θ^+ and θ^-. Our approach is inspired by the *persistence diagram* [8], which is commonly used in scientific visualization applications to visually identify meaningful thresholds [11]. However, instead of relying on users to visually select thresholds, we develop an algorithm to automatically identify them.

Topological Persistence. Consider a sweep of the function f in decreasing order of function value. As mentioned earlier, the topology of the super-level set changes at critical points during this sweep. In particular, at a critical point, either a new super-level set component is created (maximum) or an existing super-level set component is destroyed. A critical point is a *creator* if a new component is created, and a *destroyer* otherwise. Again, consider the example in Figure 4. At critical point v_5, the components created at v_4 and v_6 are merged into one. In this case, the component created last, at v_6, is considered to be destroyed at v_5. Similarly, one can pair up each creator c_1 uniquely with a destroyer d_1 that destroys the topology created at c_1. Note that this pairing can be accomplished while computing the merge tree itself (Line 16 in Procedure ComputeJoinTree). The *persistence value* of c_1 and d_1 is defined as $|f(d_1) - f(c_1)|$, which indicates the lifetime of the feature created at c_1. Figure 4(b) illustrates the persistence values of the different critical points (as π_i) together with the creator-destroyer pairs. Intuitively, the persistence of a maximum (minimum) is equal to the height (depth) of the corresponding peak (valley).

Thresholds for Salient Features. The *persistence diagram* [8, 14] plots the extrema (maxima or minima) of the input function as a set of points on a 2D plane, where the x and y coordinates of a feature correspond to its creation and destruction value, respectively. Figure 5(a) shows the persistence diagram of the minima of the taxi-density function from Figure 1 corresponding to one month in the data. Note that the minima are clearly split into *two groups*: those with high persistence (enclosed by the circle) and those with low persistence. This split becomes even more prominent when we plot just the persistence values (Figure 5(b)). A minimum (maximum) with a higher persistence value is considered important since the sub- (or super-) level set component created at that extremum has a longer lifetime. Our goal is to select a threshold θ^- such that all the high persistence minima are identified as salient. To do this automatically, we perform k-means clustering with $k = 2$ and use the highest value over all minima in the high persistence cluster as the threshold θ^-. This ensures that all the high-persistence minima will have function value less than or equal to θ^-, and will hence be identified as salient features. θ^+ is identified in a similar manner using the persistence of the set of maxima.

Thresholds for Extreme Features. Typically, a minimum (maximum) corresponding to an extreme feature will have function value that is significantly smaller (larger) than those corresponding to salient features. For example, in Figure 5(c), the function value of minima corresponding to the extremely low number of taxi trips in NYC between 2009 and 2013 is significantly different from the function value of other minima (corresponding to salient features). In order to identify the appropriate thresholds, we first compute the minima (or maxima) across all time steps that correspond to salient features. Next, we identify the outlier threshold from this distribution. We use the standard box plot thresholds, i.e., $Q_1 - 1.5 \times IQR$ for minima ($Q_3 + 1.5 \times IQR$ for maxima), as the required thresholds, where Q_1 and Q_3 are the first and third quartile, and IQR is the inter-quartile range. The box plot (and the corresponding outlier threshold) for the extreme negative features corresponding to the taxi density function is illustrated in Figure 5(c).

Adjusting for Seasonal Variations. Processes in a city are typically dependent on the time of year. For example, zero depth of snow during summer is normal, while this could indicate an important phenomena during the winter. Thus, it is important to take into account seasonal variations when computing features. Depending on the temporal resolution, the time range of a data set is divided into smaller intervals, and the threshold for a given interval is computed based on the persistence of the extrema present in that interval. For example, we could use monthly or quarter-yearly intervals.

4. RELATIONSHIP OPERATOR

In the previous sections, we discussed how relationships between functions are identified and measured. We now define the relationship operator, RELATION($\mathbb{D}_1$,$\mathbb{D}_2$), used to compute the relationship between data sets $\mathbb{D}_1$ and $\mathbb{D}_2$. Let $\mathbb{D}_1$ be represented by n functions $\{f_1, f_2, \ldots, f_n\}$, and $\mathbb{D}_2$ by m functions $\{g_1, g_2, \ldots, g_m\}$. There are $n \times m$ possible relationships between the two data sets. Since many of these relationships could be due to random chance, the relationship operator returns the set of statistically significant relationship pairs (f_i, g_j) together with their corresponding relationship score and strength.

To assess the statistical significance of a potential relationship pair (f_i, g_j), we design Generalized Monte Carlo significance tests [5, 15, 18]. Let Σ_1 and Σ_2 be the features corresponding to f_i and g_j, respectively. The null and alternative hypothesis are:

H_0**:** The two functions f_i, g_j are independent in their features Σ_1, Σ_2.

H_1**:** The two functions are dependent in their features.

We examine if we can reject the null hypothesis and accept H_1 for any pair of functions based on the identified features and their corresponding relationship score. The p-value from the Monte Carlo randomization test with test statistic x^* is given by:

$$p = P(X \leq x^* | H_0) = \frac{\sum_i^N \mathrm{I}(x_i \leq x^*)}{N} \quad \text{as} \quad N \to \infty \tag{3}$$

where $\mathrm{I}(\cdot)$ is the indicator function and N the number of permutations on the input. Given a significance level α, the p-value is then used to define a statistically significant relationship as follows:

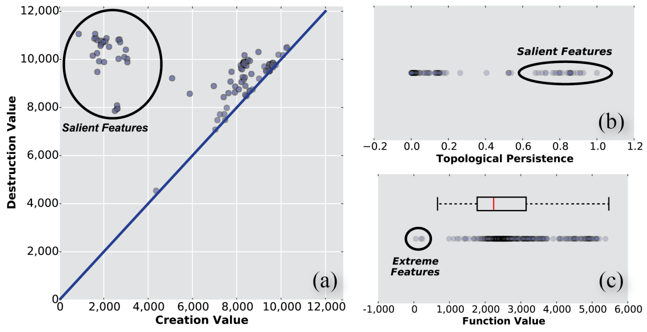


Figure 5: (a) The persistence of a minima in a persistence diagram is the height above the $x = y$ line. (b) A scatter plot of the persistence of the minima. (c) When considering only negative features across all time intervals, note that function values corresponding to extreme features (e.g., during hurricanes) are outliers of the distribution.

Definition 14. The relationship between two functions f_i and g_j is statistically significant if $p \leq \alpha$.

Urban data sets have spatial and temporal dependencies (e.g., due to neighborhood and seasonal effects) that need to be accounted for when designing randomization tests. It is well-known in the statistics literature that, if we ignore these dependencies, a simple Monte Carlo procedure for assessing statistical significance would fail and lead to erroneous claims [5, 23]. To account for the spatio-temporal correlation, a plethora of Monte Carlo and Bootstrap techniques have been developed over the last decades ranging from the block-bootstrap [22] to general restricted Monte Carlo techniques [18, 23] such as the one we propose in this paper.

Restricted Monte Carlo Tests for Spatial Correlation. We develop restricted permutation tests that respect the degree of spatial correlation of our data sources. This is typically achieved by designing toroidal shifts, where a function f is wrapped around a two-dimensional torus by connecting the margins, or spatial extents, of the data. Then, a linear map m—that maps the torus onto a rotation of itself—will yield a new randomization that still respects any horizontal interactions [18, 23].

However, given the irregular structure of a city, which is an arbitrary non-convex polygon, wrapping the spatial region over a torus is not straightforward. If we consider the spatial domain as a graph, a toroidal shift basically ensures that the adjacency of the non-boundary vertices are maintained. We make use of this observation to devise a toroidal shifting strategy that is applicable to arbitrary graphs. Given a graph G representing the spatial domain, we define the map $m_i : G \to G$ as follows. We start with a random mapping $m_i(u) = v$. The adjacent vertices of u are then assigned the vertices adjacent to v where applicable. This process is repeated in a breadth-first fashion. This process ensures that, in most cases, the distance between two vertices in G is the same as the distance between them in $m_i(G)$. Using the above mapping, the restricted Monte Carlo test now becomes:

$$p = \frac{\sum_k^{|m|} \mathrm{I}(\tau(f_i, g_j)_k \leq \tau(f_i, g_j)^*)}{|m|} \tag{4}$$

where $\tau(f_i, g_j)_k$ is the relationship score between the two functions f_i, g_j in toroidal shift k, and $|m|$ is the total number of toroidal shifts, which affects the power of the statistical test (we use $|m| = 1{,}000$).

Restricted Monte Carlo Tests for Temporal Correlation. For 1D functions that are purely temporal and have no spatial domain, we wrap time to a one-dimensional torus while rotating the resulting circle to obtain randomizations that respect temporal correlations [18]. We then proceed similarly to Equation 4. Unless otherwise mentioned, a relationship implies a statistically significant relationship for the remainder of the paper.

5. DATA POLYGAMY FRAMEWORK

In this section, we describe the data polygamy framework. We start by presenting the *scalar functions* that are derived from a given data set and how the framework handles different resolutions. Then, we discuss how the data is indexed and queries are evaluated. Finally, we briefly describe a map-reduce implementation of the framework.

5.1 Types of Scalar Function

Consider a data set $\mathbb{D}$ having attributes $\{K, S, T, A_1, A_2, \ldots, A_k\}$. Let K be an optional unique identifier; S and T be the spatial and temporal attributes, respectively; and $A_i, 1 \leq i \leq k$ be numerical attributes. We are interested in identifying scalar functions that not only capture the activity of the objects represented by the data sets, but that also that capture the different properties corresponding to the attributes. For instance, when considering the taxi data, the number of taxis in different locations over time captures the activity of the taxis, while the attribute corresponding to the fare captures fare patterns over time and space. We therefore derive two types of scalar functions to represent $\mathbb{D}$: *count functions* and *attribute functions*. Possible extensions to other types of scalar functions are discussed in Section 8.

Count Functions. *Count functions* are used to capture the activity of the entity represented by the data set. More formally, consider a spatio-temporal point (s, t). Let Γ be the set of tuples in $\mathbb{D}$ such that $S(r) = s$ and $T(r) = t$, $\forall r \in \Gamma$. Here, S and T represent the spatial and temporal attributes of a tuple r. We define two types of count functions: *density* and *unique*. The *density function* assigns the value $|\Gamma|$ to the spatio-temporal point (s, t). For example, the density function of the taxi data assigns the number of trips originating at s during the time period t to the point (s, t). The *unique function* assigns a value equal to the number of unique identifiers of Γ to the spatio-temporal point. For instance, each tuple in the taxi data consists of an identifier corresponding to the medallion of the taxi. Thus, the number of unique medallions in Γ is essentially the number of unique taxis that are present at s during time t. Note that there is one unique function corresponding to each identifier attribute of the data set.

Attribute Functions. For a given attribute A, the *attribute function* assigns the average value of $A(r)$ over all tuples $r \in \Gamma$ to the corresponding spatio-temporal point (s, t); the function represents the variation in the properties of a given attribute over space and time.

Handling Different Data Resolutions. It is important that our framework identifies relationships that occur at different resolutions. As illustrated in Figure 6, these resolutions are represented as a directed acyclic graph (DAG), where the edges are directed from a higher resolution to a compatible lower resolution. The compatibility indicates the ability to convert the data from a higher resolution to a coarser resolution. For example, GPS resolution can be transformed into all of the other resolutions. On the other hand, neighborhood and zip-code resolutions, being incompatible, can be converted only into the city resolution. To evaluate the relationship between two functions having different resolutions, we first transform both functions into the same compatible resolution, and then evaluate the two functions at this resolution.

5.2 Indexing and Feature Identification

Given a data set $\mathbb{D}$, we first compute all possible scalar functions (i.e., count and attribute functions) of $\mathbb{D}$ that cover every viable spatio-temporal resolution. For example, if $\mathbb{D}$ is available at a spatial resolution of GPS locations and temporal resolution of second, then each attribute can be aggregated into 3 spatial resolutions (i.e., zip code, neighborhood, and city) and 4 temporal resolutions (i.e., hour, day, week, and month), thus resulting in a total of 12 spatio-temporal resolutions for which the scalar functions are computed. The merge tree index is then built for each scalar function. This ensures that all resolutions are considered when executing a relationship query. Recall that the computation of feature thresholds takes seasonal variations into account (Section 3.3). In particular, we use monthly and quarter-yearly intervals when the temporal resolution is hourly and daily, respectively. Since thresholds are fixed for a given function, to speed up query evaluation, we pre-compute and store the features (salient and extreme).

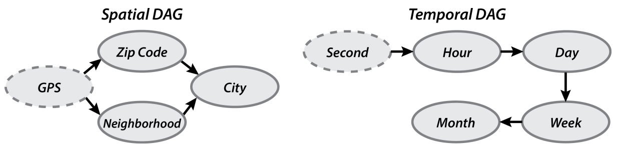


Figure 6: Hierarchical relationship for spatio-temporal resolutions represented by DAGs. Resolutions depicted using solid lines are used for evaluating relationships.

5.3 Query Evaluation

Let $\mathcal{D} = \{\mathbb{D}_1, ..., \mathbb{D}_n\}$ be the corpus of data sets that have been indexed. We support the general form of the *relationship query*:

$$\boxed{\textit{Find relationships between } \mathcal{D}_1 \textit{ and } \mathcal{D}_2 \textit{ satisfying } \textsc{clause}}$$

In this query, $\mathcal{D}_1$ and $\mathcal{D}_2$ are collections of data sets such that $\mathcal{D}_1 \subseteq \mathcal{D}$ and $\mathcal{D}_2 \subseteq \mathcal{D}$. If $\mathcal{D}_2 = \varnothing$, it is assumed that $\mathcal{D}_2 = \mathcal{D}$. When a relationship query is issued, the RELATION operator is applied to all pairs $(\mathbb{D}_i, \mathbb{D}_j)$ of data sets, such that $\mathbb{D}_i \in \mathcal{D}_1$, $\mathbb{D}_j \in \mathcal{D}_2$, and $\mathbb{D}_i \neq \mathbb{D}_j$. The operator uses the pre-computed set of features to assess the relationship between the data sets. Note that, when considering a pair of functions, the relationship between them is evaluated for all possible resolutions starting with the *highest common* resolution. For example, if the spatial resolutions of two functions are neighborhood and zip code, then their relationship is evaluated at the city scale for different possible temporal resolutions. This evaluation is performed for both salient and extreme features.

Computing relationships at different resolutions is important as scalar functions may relate differently depending on how they are aggregated. For example, an hourly resolution might capture variations within a day, but could miss significant variations across different days, which can be captured using a daily resolution (see Section 6.3 for an example).

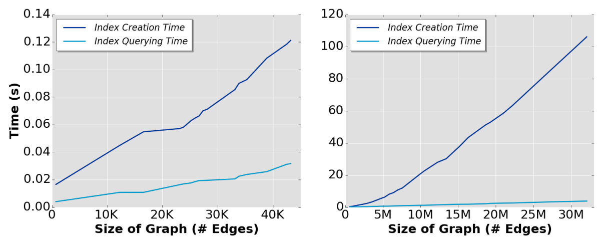
The query returns related scalar function pairs that are statistically significant, together with the resolutions for which the relationships hold. The significance level α is set at the commonly used value of 5% [15]. In the CLAUSE for a query, optional condition parameters can be specified to filter relationships satisfying a minimum score τ and/or strength ρ. Feature thresholds for computing salient and extreme features can also be optionally specified as part of the CLAUSE if the user is familiar with any of the data sets. When these thresholds are specified, features are first identified using the merge tree index before evaluating the relationship.

5.4 Implementation

Given a large number of data sets, such as the urban data sets with which we experimented (Section 6), thousands of scalar functions have to be computed, and the number of relationships to be evaluated during querying is in the order of millions. However, the indexing and querying operations can be run independently for each scalar function and scalar function pair. To leverage the *embarrassingly parallel* nature of these computations, we imple-

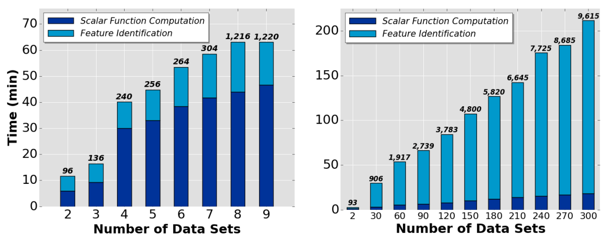
Table 1: Properties of the data sets in the *NYC Urban* collection.

Data Set	Data Size	# Records	Time Range	# Scalar Functions	Spatial Resolution	Temporal Resolution	Description
Gas Prices	13 KB	749	2000–2014	2	City	Week	Average gasoline price in dollars per gallon for NYC
Vehicle Collisions	47 MB	330 K	2012–2014	11	GPS	Second	Traffic collision data provided by NYPD
311 Complaints	574 MB	7.40 M	2003–2014	1	GPS	Second	Records from 311, a telephone number that provides non-emergency services to the city
911 Calls	2.2 GB	6.75 M	2012–2013	1	GPS	Second	Records from 911, a telephone number that provides emergency services to the city
Citi Bike Data	1.6 GB	10.40 M	2013–2014	5	GPS	Second	Trip data from NYC's bike sharing system
NCEI Weather Data	304 MB	64 K	2010–2014	228	City	Hour	Comprehensive weather data from NCEI
Traffic Speed	17 GB	395 M	2009–2012	2	GPS	Hour	Average speed in the streets of Manhattan
Taxi Data	108 GB	868 M	2009–2013	13	GPS	Second	Trip data from taxicabs provided by NYC TLC
Twitter	656 GB	1.10 B	2012–2014	5	GPS	Second	Data obtained from Twitter's public streams



(a) City (b) Neighborhood

Figure 7: Merge tree index creation and feature querying time.



(a) *NYC Urban* (b) *NYC Open*

Figure 8: Performance of feature indexing and identification.

mented the framework using *map-reduce*. We use three *map-reduce* jobs: 1. *Scalar Function Computation* generates all possible scalar functions at different resolutions; 2. *Feature Identification* creates the merge tree indexes and identifies the set of features of the different functions; and 3. *Relationship Computation* evaluates the relationships between the pairs of functions corresponding to a given query. Implementation details can be found in Appendix C and the released code.[1]

Space Overhead. The space required to store a scalar function for a given resolution is equal to the number of vertices in the graph G representing the domain. For a resolution at the city level and hourly intervals, this corresponds to the number of time steps, which is approximately 35 KB (365×24 float values) per year. For higher spatial resolutions, the space required to store the function is approximately $n \times 35$ KB, where n is the number of polygons used to partition the domain. For example, in NYC, for zip code and neighborhood resolutions, $n \approx 300$. Typically, the space overhead to store scalar functions over all resolutions is significantly less than the original data itself. As a point of reference, the 5 years of the raw taxi data takes up 108 GB of space. In contrast, the 13 possible scalar functions over 8 resolutions uses only 417 MB. The size of the merge tree index is proportional to the number of critical points of the function. While the number of critical points is bounded by the size of the input graph in the worst case, in practice it is significantly smaller. Similarly, even the number of features, while being bounded by the size of the input graph, is usually much smaller. For example, storing all features (salient and extreme) for the taxi data over all different resolutions takes only 8 MB.

6. EXPERIMENTAL EVALUATION

We have performed an extensive evaluation to assess different aspects of the *Data Polygamy* framework. We carried out a controlled experiment to quantitatively evaluate the correctness and robustness of the relationship operator, and we also used real-world data sets to study efficiency and effectiveness characteristics. Efficiency was measured to show the feasibility of computing relationships over a large number of data sets. Effectiveness was evaluated in two different ways: to demonstrate that the framework is able to prune spurious relationships, thus reducing the exploration space presented to users, and to show that our approach uncovers *interesting*, non-trivial relationships. We also analyzed the relationships obtained from standard correlation techniques and discuss their shortcomings in identifying interesting relationships.

Experimental Setup. We used Apache Hadoop 2.2.0 and Java 1.7.0. The *map-reduce* jobs were executed on a cluster with 20 compute nodes, each node having an AMD Opteron(TM) Processor 6272 (4x16 cores) running at 2.1GHz, and 256GB of RAM.

Data Sets. We used two collections of data sets in our experiments. The *NYC Urban* collection consists of nine urban data sets obtained from different NYC agencies or gathered through publicly-available APIs. These data sets have been used by various domain experts (mostly in isolation) for different analyses (see, e.g., [6, 17]) and are thus useful to evaluate the effectiveness of our framework at identifying meaningful relationships. Table 1 describes these data sets and their properties. They vary in size from a few KBs to hundreds of GBs, and have different temporal and spatial resolutions.

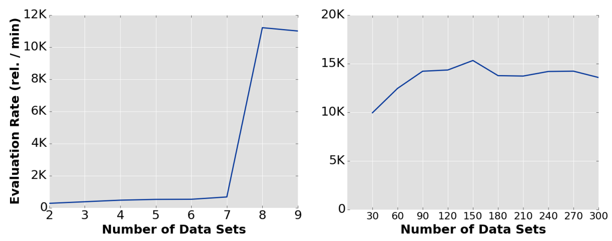
The second collection, referred to as *NYC Open*, was primarily used to test the performance of our framework. It consists of 300 spatio-temporal data sets from NYC Open Data [27]. Even though most of these data sets are relatively small in size (less than 1 GB), the sheer number of data sets and the number of attributes they contain (on average, 8 attributes per data set) results in over 2.4 million possible relationships for a single resolution.

Each data set in these collections consists of a set of tuples having metadata about the spatial, temporal, and numerical attributes as well as keys. We use this metadata to perform an additional preprocessing step that selects data corresponding to these attributes and feeds it to the scalar function computation module.

6.1 Performance Evaluation

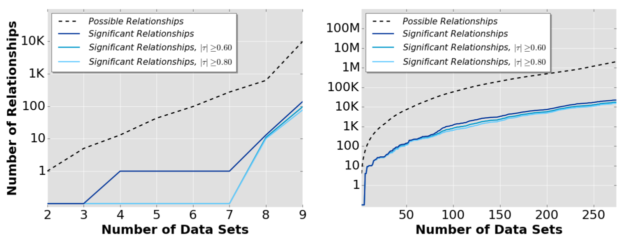
Indexing and Feature Identification. To assess the efficiency of feature identification and indexing, we studied the performance of the merge tree index for a single data set as well its behavior for an increasing number of data sets. Figure 7 plots the running times to

[1]URL of the code is withheld due to the double blind requirement.



(a) *NYC Urban* (b) *NYC Open*

Figure 9: Query performance.



(a) *NYC Urban* (b) *NYC Open*

Figure 11: Relationship pruning.

create the index and query for features for the Taxi data (using its density function), for both city (1D) and neighborhood (3D) resolutions. Here, we used a single node in the cluster. The plots indicate that the time for creating the merge tree and identifying features is almost linear in the size of the function (i.e., number of edges in the spatial domain graph). Note that the indexing time includes the creation of both join and split trees, and the querying time includes the computation of thresholds as well as the identification of negative and positive features. Even for an input having more than 30 million edges, the operations took less than 2 minutes. This shows that our approach is scalable and able to handle large data sets.

The indexing component also performs well as the number of data sets increases. This is shown in Figure 8. The numbers on the bars indicate the total number of computations performed. Recall that scalar functions are computed for all attributes at all spatio-temporal resolutions. When using *NYC Urban* (Figure 8(a)), the large increase in time when moving from 3 to 4 data sets was due to the 4th data set, the Taxi data, which is not only large but also contains many attributes. It also has the highest resolution both in space and time, requiring each scalar function to be computed over all resolutions. There was also a significant increase in the number of computations when the Weather data set was introduced (the 8th data set): this data set has 228 numerical attributes. However, since it is relatively small compared to other data sets in *NYC Urban*, the running time was not significantly affected.

For *NYC Open* (Figure 8(b)), the time taken to identify the features was significantly larger than that for computing the scalar functions. This behavior differs from what we observed for *NYC Urban* due to two reasons: (1) the data sets in this collection are much smaller; (2) most of the 3D data sets in *NYC Open* are already in zip code resolution, making it faster to compute the scalar functions; in contrast, tuples in the 3D *NYC Urban* data sets are GPS points, which required additional computations to aggregate into the neighborhood and/or zip code resolutions. The total time taken to compute the indexes and features for all data sets in *NYC Urban* and *NYC Open* was about 1 hour and 4 hours, respectively.

Query Performance. To test the efficiency of the querying component, we executed a series of queries that identify the relationships between a fixed number of data sets. Figure 9 plots the relationship evaluation rate with increasing number of data sets. Using both collections, *NYC Urban* and *NYC Open*, we were able to consistently evaluate relationships at a rate greater than 10^4 relationships per minute. The evaluation rate stabilized once the number of relationships increased above this number, e.g., with the addition of the Weather data (data set 8) in Figure 9(a). A total of 290 thousand relationships were evaluated when the query used all data sets from *NYC Urban*; for the *NYC Open*, this number was 17.4 million. The constant rate, irrespective of the data set pairs, indicates that relationship evaluation is independent of size and resolution of the original data. This can be attributed to the abstraction of the data as functions. Note that over 90% of the querying time is spent on the statistical significance tests, which involve re-evaluating each relationship for 1,000 random spatial or temporal permutations. Also, the queries executed did not include any filtering clause. When using a clause C, the query evaluation step skips the significance test when C is not satisfied, which further improves the performance.

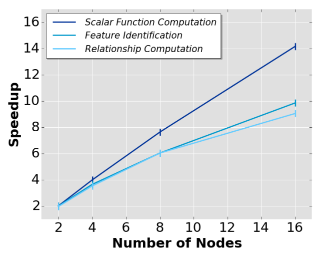


Figure 10: Speedup

Scalability. To test the scalability of our framework, we computed the speedup attained by the different components with increasing number of nodes in the cluster. This experiment was performed on Amazon Web Services (AWS) using the *NYC Urban* collection. We used AWS because it allows the configurations of clusters of different sizes. Each node in the cluster had an Intel Xeon E5-2670 v2 processor (8-core) and 61GB of RAM. Figure 10 shows the speedup for the three components of the framework, which was computed against the the time taken by a single node. A relatively lower speedup was attained for identifying features and evaluating relationships than for the scalar function computation. This is primarily due to the presence of straggler reducers that deal with higher spatio-temporal resolutions, thus increasing the computation time for the randomization tests.

Relationship Pruning. Figure 11 plots the the number of identified relationships (in log scale), when considering the (week, city) resolution, with increasing number of data sets. For the data sets in *NYC Urban* (Figure 11(a)), there was a significant decrease in the number of relationships—from 9,745 to 137, a decrease of about 98.60%. If we filter relationships having $\tau \geq 0.6$ and $\tau \geq 0.8$, the reduction further increased to 99% and 99.20%, respectively. When handling a larger number of data sets, such as *NYC Open*, the advantages of our framework become even more evident, as Figure 11(b) shows. Given the over 2 million possible relationships for the (week, city) resolution, our framework identified 22,327 of them to be statistically significant, which corresponds to a decrease of about 98.90%. Although the number of identified relationships is still large, this is significantly better than trying to make sense of over 2 million relations. In addition, we envision that users will explore these relationships by searching, querying, and filtering them based on different attributes (e.g., τ, ρ, α, space, and time).

6.2 Correctness and Robustness

Correctness. Most urban data sets have become available only recently and work on integrating them is still incipient [3]. Since *there are no ground-truth benchmarks* that can be used to evaluate the correctness of the identified relationships, we used prior knowledge about the data sets and designed a controlled experiment to test if our technique can uncover strong relationships that are expected to occur. Consider the Taxi data in Figure 1. The density functions for taxi trips in 2011 and 2012 have a similar behavior: the number of trips in the city over time follows a similar pattern over the two years, except in specific situations, such as during extreme weather conditions. Thus, if each year of data is modeled as a function (starting at the same day and time), a strong positive relationship should be observed for the two functions. This observation was used to test our technique, which indeed identified the two functions to be strongly and significantly related across different resolutions. The relationship score and strength for the (hour, city) and the (hour, neighborhood) resolutions were ($\tau = 0.99$, $\rho = 0.85$) and ($\tau = 1$, $\rho = 0.87$), respectively.

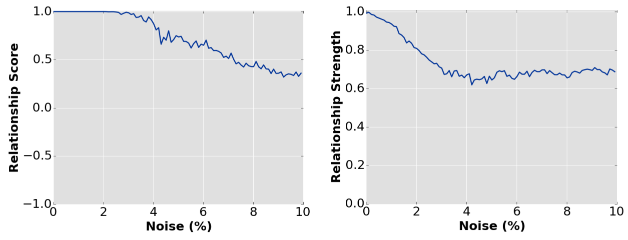
Robustness. To assess the robustness of our technique in the presence of noise, we fixed a scalar function f, and by artificially introducing noise to f, we created a new (noisy) function f^*. We used a random Gaussian noise where the amount of noise was bounded by a fraction of the inter-quartile range of the function. Note that noise was added to every spatio-temporal point of the function domain. We then evaluated the relationship between f and f^*. Figure 12 plots the relationship scores and strengths with increasing levels of noise added to the taxi density function. Note that even when the added noise was as large as 10% of the normal function range, we were still able to obtain a strong positive relationship (which was statistically significant) between the two functions. Furthermore, the relationship score remained 1 even when the noise level was greater than 2%. This behavior can be attributed to the fact that topological persistence, which is used to identify thresholds for the salient features, is robust to noise [8]; small local maxima and minima, which are created due to the addition of noise, do not significantly affect the feature threshold.

6.3 Effectiveness: Interesting Relationships

We carried out a detailed study using *NYC Urban* to assess the effectiveness of our approach at finding interesting relationships and pruning uninformative relationships that are not statistically significant. In our evaluation, we found Weather to be the most *polygamous* data set, being related through different attributes with all data sets in the collection, except Gas Prices, indicating the impact it has on different aspects of a city. We discuss some of these relationships below (additional relationships are described in Appendix E.2). Unless otherwise noted, all relationships are with respect to salient features. Note that, while some of the results might imply the presence of a causal relationship, further analysis (not in the scope of this work) is required to ascertain causality.

Weather and Taxi. One of the relationships identified for the Taxi and Weather data sets in the (hour, city) resolution is between the number of taxis and the average precipitation, having score $\tau = -0.62$ and strength $\rho = 0.75$. The values indicate a strong negative relationship: the higher the precipitation, the lower the number of taxis in the city. As discussed in Section 1, this confirms the difficulty in finding taxis on a rainy day. When testing the hypothesis that this is due to taxi drivers being target earners, we found a positive relationship between average fare and precipitation ($\tau = 0.73, \rho = 0.7$) implying increased earnings when it rains. Note that Farber [16] refuted this hypothesis since he did not find a correlation (using OLS regression) between the drivers' earnings and rainfall. This is primarily due to two reasons: (i) he did not take into account the amount of rainfall—instead, he used a binary value indicating whether it rained or not; and (ii) more importantly, he considered the entire time period—periods with very sparse rainfall are considered equivalent to those having higher rainfall. Thus, this case study also provides evidence for the importance of looking at salient features, since they can evince relationships that are not visible when the whole data is taken into account.

While examining relationships involving extreme features, we found a negative relationship between the number of trips and the average wind speed. This relationship has a high score ($\tau = -1$) and



(a) Score (b) Strength

Figure 12: Robustness evaluation using the density of taxi trips.

a low strength ($\rho = 0.13$). The low strength is due to the presence of significant drops in the number of taxi trips at other periods, for example, during Thanksgiving, Christmas, and New Year [17] that are unrelated to the wind speed. However, the high score indicates that, whenever there is high wind speed, the number of taxi trips is significantly lower than usual, which is related to the impact of hurricanes in the city. We also found the same impact in the relationship between number of unique taxis and average precipitation (having $\tau = -1.0$) when considering extreme features. It is worth noting that the number of taxi trips and wind speed are not related through salient features alone (also observed from Figure 1). This demonstrates the importance of computing relationships for salient as well as extreme features.

Weather and Citi Bike. We found a positive relationship between the average snow precipitation and the average bike trip duration in the (hour, city) resolution, having $\tau = 0.61$ and $\rho = 0.16$. This implies that bike trips are longer in snowy days (or shorter when there is no snow), which is consistent with what we would expect. For a lower resolution—(day, city)—we found a negative relationship between the average snow precipitation and the active Citi Bike stations ($\tau = -0.88$ and $\rho = 0.65$), i.e., fewer bike stations are used when it snows. We believe that this is related not only to the drop in bike usage under such weather, but also because heavier snow may impact certain stations more than others: the city clears snow at different frequencies depending on the location, and some stations may get cleared faster than others. Note that the latter relationship had a low score ($\tau = 0$) when considered at the higher resolution (hour, city): the impact is usually reflected only after the snow accumulates, and such accumulation is not captured when using an hourly time step (higher resolution). This case illustrates the need for evaluating relationships at multiple resolutions.

Vehicle Collisions and Weather. We found interesting relationships between Vehicle Collisions and Weather, which correspond to the increased danger of accidents when it rains. There was a strong positive relationship between rainfall and number of motorists killed ($\tau = 0.90, \rho = 0.95$) as well as number of injured pedestrians ($\tau = 0.75, \rho = 0.66$). However, we found no significant relationship between number of accidents and rainfall, implying that, even though the number of accidents does not increase when there is heavy rain, their severity does. This leads to a new hypothesis that may explain the lack of taxis during rainfall: taxi drivers, being experienced with the possible danger during high rainfall, might return home during these periods.

Taxi and Traffic Speed. We found a positive relationship between the average taxi fare and the average traffic speed at the (hour, neighborhood) resolution with $\tau = 0.79$ and $\rho = 0.44$, implying that drivers may in fact earn less in the presence of heavy traffic. We also found strong negative relationships between the number of taxi trips and the average traffic speed in the (hour, city) resolution ($\tau = -0.90$ and $\rho = 0.65$). This is expected, especially in a city such as NYC, which has around 13,000 taxis: a larger number of trips increases the traffic, thus slowing down the traffic.

Vehicle Collisions, 311, and Taxi. We identified relationships that involve the Vehicle Collision data set at the (hour, neighborhood) resolution: a strong positive relationship between the number of collisions and the number of 311 complaints ($\tau = 0.99, \rho = 0.86$), and a strong positive relationship between the number of collisions and number of taxi trips ($\tau = 0.99$, $\rho = 0.79$). While the implication of such relationships, especially the latter, is not clear, it provides a starting point for experts, pointing them towards data sets and attributes to be considered for further detailed analysis.

Effectiveness of Statistical Significance Test. Since there is no gold data available, to test the ability of the significance tests to remove potentially uninformative relationships, we evaluated a randomly chosen set of statistically non-significant relationships. For instance, many relationships between the fare tax for taxi trips and different attributes from the Weather, 311, and 911 data sets were found not to be statistically significant. This indicates that, even though some of these relationship have $|\tau| > 0.60$, they are mostly random and coincidental. In fact, the tax charged in taxi trip fares does not have anything to do with different weather conditions, let alone with 311 and 911 complaints. Other examples of spurious relationships pruned by our framework include: mileage of taxi trips (Taxi) and number of injured pedestrians (Vehicle Collisions), having $\tau = 0.90$; number of bike trips (Citi Bike) and number of tweets (Twitter), having $\tau = 0.87$; and number of 311 complaints and average speed (Traffic Speed), having $\tau = 0.76$.

We also computed the statistical significance for relevant relationships using the standard Monte Carlo procedure. Many of these relationships were found to be not significant using this test, including the ones between the average snow precipitation and the average bike trip duration. This underscores the importance of taking the spatial and temporal dependencies into account while assessing statistical significance.

Note that such tests represent a best-effort approach to identify candidate relationships, and thus, they can both return spurious relationships and miss important ones. Gold data are needed to quantitatively study the trade-offs for the different techniques. Nonetheless, our initial experiments indicate that these significance tests are useful and can help guide users in the data discovery process.

6.4 Comparison against Standard Techniques

We used the *NYC Urban* collection to compare our approach against established techniques for identifying dependencies between data: Pearson correlation coefficient (PCC) [7], mutual information (MI) criterion [33], and dynamic time warping (DTW) routines [21, 31]. Some of these techniques are not naturally normalized for inter-dataset comparisons (e.g., DTW and MI) or directly extendable to spatio-temporal data. Thus, for this experiment, we proposed normalizations that provide a meaningful range of relationship score (see Appendix D for details) and focused on data represented as a time series aggregated over the city resolution.

Overall, we observed that standard approaches can identify the basic relationships that are present across the entire data. For example, the relationship between average snow precipitation and Citi Bike trip duration could be detected by PCC as well as by MI. Similarly, the relationship between number of taxi trips and average traffic speed could be found using PCC and DTW. However, these techniques did not find relationships that are only visible under certain conditions, such as the ones between rainfall and number of taxis, or wind speed and number of taxi trips. Also, relationships that take into account space, such as the ones between number of collisions and number of taxi trips, are not identified by any of the above techniques due to their inherent 1D nature.

7. RELATED WORK

In addition to the approaches discussed in Section 6.4, other notions of relationship have also been explored by the data mining and data integration communities. Some methods focused on identifying relationships between data points *within a single* data set. Achtert et al. [1] focused on computing correlation clusters, which are composed of data points that present correlations between different attributes. Yang et al. [37] used subspace clustering, which finds different clusters of points for different subspaces of attributes, to identify relationships. Other methods have been proposed which identify different kinds of relationships. Sarma et al. [9] focused on finding candidate tables that can be unioned and joined, and considered such tables to be related. There has also been work on *data fusion*, where relationships are sought between data sets that overlap or complement each other to resolve conflicts between different sources [10, 29] and to find their derivation history [2]. These relationships are orthogonal to and can be used in conjunction with our technique to enrich the data discovery process. To the best of our knowledge, no existing method addresses the problem of identifying spatio-temporal relationships that take into account salient features in the data.

Recently, there has been a renewed interest in finding explanations for surprising results, or *outliers*, in database queries [30, 36]. Scorpion [36] focused on understanding aggregate queries over a single data set. Roy and Suciu [30] handled more complex database schemas and proposed techniques that can be used to explain relationships between different data sets. However, the thresholds for outliers must be specified by the user for each query, which becomes impractical when handling hundreds to thousands attributes and without proper knowledge about the data sets. Since our *Data Polygamy* framework generates an overview of the relationships among different data sets, the approach proposed by Roy and Suciu [30] could be used to further explore and understand the most eye-catching relationships. Thus, the techniques complement each other in the data exploration process.

Methods for comparing scalar functions use topological abstractions directly for this comparison (e.g., [4, 25]), due to which two functions are considered to be similar even if some affine transformation of the functions are similar, i.e., spatio-temporal locations of the topological features are not considered. Unlike such methods, we are interested in comparing scalar functions based on the spatio-temporal locations of their topological features.

8. DISCUSSION AND FUTURE WORK

Scalar Functions. In this work, we mainly considered data whose spatial domain has dimension up to two. However, our framework is general and can handle higher dimensions as well. For instance, data corresponding to noise in buildings can be obtained in 3D, where in addition to geo-location, the noise level varies with height. By constructing an appropriate graph to represent this spatial domain, the framework can be used as is.

While we have focused on numerical attributes, non-numerical attributes can be taken into account if they are mapped into numerical values (e.g., categorical values can be mapped to unique numbers). In addition, while we chose to use the average to represent functions, it is straightforward to extend our framework to support other functions such as *sum*, *median*, *min*, or *max*. Alternatively, users can define custom functions as well.

Types of Features. Our current feature identification approach can miss unusual patterns in the data due to its use of a single threshold. For example, a sudden increase in taxi trips in a relatively calm area and time will not be identified if the density of taxi trips does not exceed the computed threshold. Instead, we could consider the *gradient* of this density over space and time. High values of this function correspond to regions that have sudden increase/decrease of function value: the increase during a non-busy hour will show as a high-gradient region, and can thus be identified as a feature.

In future work, we plan to do a comprehensive study of the different types of scalar functions that can be derived from urban data sets, and use them to classify the types of features they identify. We will use this classification to create a taxonomy of scalar functions which can then be used by domain experts to appropriately choose the types of features and relationships in which they are interested. We will also investigate the use of a ROC curve which considers multiple thresholds. This would help users pick the operating point of interest depending on the desired sensitivity-specificity trade-off.

Spatio-Temporal Resolution. We currently support conversion from one resolution to another only when they are compatible. We plan to explore methods to support conversion between resolutions that do not have a direct translation (e.g., neighborhood and zip code) in order to help evaluate relationships directly at a higher resolution rather than moving to a lower resolution (the latter may result in loss of information).

Future Extensions. In this paper, we used salient, topological features as the basis for identifying spatio-temporal relationships across disparate data sets. One direction we would like to explore is the use of event detection techniques as an alternative to the topological features. While the topology-based approach identifies local features corresponding to maxima and minima, event detection techniques must first create a model for what constitutes normal behavior to detect events that do not follow this behavior. Even though this can be expensive computationally, especially for spatio-temporal data, it would be interesting to study the performance trade-offs of the two approaches as well as compare the quality of the relationships they derive.

With the help of domain experts, we intend to use the available open data to create a benchmark for evaluating existing and future techniques. While we are able to prune a significant number of relationships, a large number of them might still need to be explored. We plan to design a visual interface to help this exploration process. We also plan to explore techniques to identify relationships that are causal. Finally, we intend to extend the statistical significance test to use a 3-torus to incorporate both space and time together.

9. REFERENCES

[1] E. Achtert, C. BÅąhm, H.-P. Kriegel, P. KrÅąger, and A. Zimek. Robust, Complete, and Efficient Correlation Clustering. In *SDM*, 2007.

[2] A. Alawini, D. Maier, K. Tufte, and B. Howe. Helping Scientists Reconnect Their Datasets. In *SSDBM*, pages 29:1–29:12, 2014.

[3] L. Barbosa, K. Pham, C. Silva, M. Vieira, and J. Freire. Structured Open Urban Data: Understanding the Landscape. *Big Data*, 2(3), 2014.

[4] U. Bauer, X. Ge, and Y. Wang. Measuring distance between reeb graphs. In *Proc. Symp. Comp. Geom.*, pages 464:464–464:473, 2014.

[5] J. Besag and P. Clifford. Generalized Monte Carlo Significance Tests. *Biometrika*, 76(4):633–642, 1989.

[6] A. Chohlas-Wood, A. Merali, W. Reed, and T. Damoulas. Mining 911 Calls in New York City: Temporal Patterns, Detection and Forecasting. In *AAAI Workshop On AI For Cities*, 2015.

[7] V. A. Clark, O. J. Dunn, and R. M. Mickey. *Applied Statistics: Analysis of Variance and Regression*. John Wiley, 2004.

[8] D. Cohen-Steiner, H. Edelsbrunner, and J. Harer. Stability of persistence diagrams. *Disc. Comput. Geom.*, 37(1):103–120, 2007.

[9] A. Das Sarma, L. Fang, N. Gupta, A. Halevy, H. Lee, F. Wu, R. Xin, and C. Yu. Finding Related Tables. In *SIGMOD*, pages 817–828, 2012.

[10] X. L. Dong, B. Saha, and D. Srivastava. Less is More: Selecting Sources Wisely for Integration. *PVLDB*, 6(2):37–48, 2012.

[11] H. Doraiswamy, V. Natarajan, and R. S. Nanjundiah. An Exploration Framework to Identify and Track Movement of Cloud Systems. *TVCG*, 19(12):2896–2905, 2013.

[12] H. Edelsbrunner. *Geometry and Topology for Mesh Generation*. Cambridge Univ. Press, England, 2001.

[13] H. Edelsbrunner and J. Harer. *Computational Topology: An Introduction*. Amer. Math. Soc., 2009.

[14] H. Edelsbrunner, D. Letscher, and A. Zomorodian. Topological Persistence and Simplification. *Disc. Comput. Geom.*, 28(4):511–533, 2002.

[15] M. D. Ernst et al. Permutation Methods: A Basis for Exact Inference. *Statistical Science*, 19(4):676–685, 2004.

[16] H. S. Farber. Why You Can' t Find a Taxi in the Rain and Other Labor Supply Lessons from Cab Drivers. Technical Report 20604, National Bureau of Economic Research, 2014.

[17] N. Ferreira, J. Poco, H. T. Vo, J. Freire, and C. T. Silva. Visual Exploration of Big Spatio-Temporal Urban Data: A Study of New York City Taxi Trips. *TVCG*, 19(12):2149–2158, 2013.

[18] M.-J. Fortin and G. M. Jacquez. Randomization Tests and Spatially Auto-Correlated Data. *Bulletin of the Ecological Society of America*, pages 201–205, 2000.

[19] B. Goldstein and L. Dyson. *Beyond Transparency: Open Data and the Future of Civic Innovation*. Code for America Press, 2013.

[20] B. Katz and J. Bradley. *The Metropolitan Revolution: How Cities and Metros Are Fixing Our Broken Politics and Fragile Economy*. Brookings Focus Book. 2013.

[21] E. Keogh and A. Ratanamahatana. Everything You Know About Dynamic Time Warping is Wrong. *Workshop on Mining Temporal and Sequential Data*, 2004.

[22] H. R. Kunsch. The jackknife and the bootstrap for general stationary observations. *The Annals of Statistics*, pages 1217–1241, 1989.

[23] B. F. Manly. *Randomization, Bootstrap and Monte Carlo Methods in Biology*. 1996.

[24] J. Milnor. *Morse Theory*. Princeton Univ. Press, 1963.

[25] V. Narayanan, D. M. Thomas, and V. Natarajan. Distance between extremum graphs. In *IEEE PacificVis*, pages 263–270, 2015.

[26] City of Chicago Data Portal. `https://data.cityofchicago.org`.

[27] NYC Open Data. `https://nycopendata.socrata.com`.

[28] V. Pascucci, X. Tricoche, H. Hagen, and J. Tierny, editors. *Topological Methods in Data Analysis and Visualization*. Springer, 2010.

[29] R. Pochampally, A. Das Sarma, X. L. Dong, A. Meliou, and D. Srivastava. Fusing Data with Correlations. In *SIGMOD*, pages 433–444, 2014.

[30] S. Roy and D. Suciu. A Formal Approach to Finding Explanations for Database Queries. In *SIGMOD*, pages 1579–1590, 2014.

[31] H. Sakoe and S. Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 26(1):43–49, 1978.

[32] N. Shadbolt, K. O'Hara, T. Berners-Lee, N. Gibbins, H. Glaser, H. Wendy, and M. Schraefel. Linked Open Government Data: Lessons from Data.gov.uk. *IEEE Intelligent Systems*, 27(3):16–24, 2012.

[33] Y. Su, G. Agrawal, J. Woodring, A. Biswas, and H.-W. Shen. Supporting Correlation Analysis on Scientific Datasets in Parallel and Distributed Settings. In *HPDC*, pages 191–202, 2014.

[34] TLC Trip Record Data. `http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml`, 2015.

[35] Weather Data. `https://www.ncei.noaa.gov/`, 2015.

[36] E. Wu and S. Madden. Scorpion: Explaining Away Outliers in Aggregate Queries. *PVLDB*, 6(8):553–564, 2013.

[37] J. Yang, W. Wang, H. Wang, and P. S. Yu. delta-Clusters: Capturing Subspace Correlation in a Large Data Set. In *ICDE*, pages 517–528, 2002.

[38] E. Zhang, K. Mischaikow, and G. Turk. Feature-based Surface Parameterization and Texture Mapping. *ACM TOG*, 24(1):1–27, 2005.

APPENDIX

A. TABLE OF SYMBOLS

Symbol	Description
$\mathbb{D}$	Data set
A, B	Data set attribute
f, g	Time-varying scalar function
$\mathbb{S}$	Spatial domain
$\mathbb{T}$	Temporal domain
θ	Feature threshold
θ^+	Threshold for positive features
θ^-	Threshold for negative features
Σ_i	Set of features of a scalar function
Σ	Set of feature-related points
Σ_i^+	Set of positive features of a scalar function
Σ_i^-	Set of negative features of a scalar function
G	Graph representing the spatio-temporal domain of a scalar function
τ	Relationship score
ρ	Relationship strength

B. ADDITIONAL NOTES ON TOPOLOGY

B.1 Critical Points and Morse Functions

Given a smooth, real-valued function $f : \mathbb{R}^d \rightarrow \mathbb{R}$, a critical point c_p of f is *non-degenerate* if the determinant of the *Hessian* matrix at c_p is non-singular. The function f is called a *Morse function* if it satisfies the following conditions [5]:

1. All critical points of f are non-degenerate.
2. All critical values are distinct *i.e.*, $f(p) \neq f(q)$ for all critical points $p \neq q$.

An important property of Morse functions is that critical points of such a function can be classified based on the behavior of the function within a local neighborhood [11]. In case of a PL function defined on a graph, the local neighborhood of a vertex v is defined using the *link* of that vertex.

Definition 15. The *link* of a vertex v is the sub-graph induced by the vertices adjacent on v. The *upper link* of v is the sub-graph induced by adjacent vertices having function value greater than v, while the *lower link* of v is the sub-graph induced by adjacent vertices having function value lower than v.

Banchoff [1] and Edelsbrunner et al. [9] describe a combinatorial characterization for critical points of a PL function, which are always located at vertices of the graph. Critical points are characterized by the number of connected components of the lower and upper links. The vertex is *regular* if it has exactly one lower link component and one upper link component. All other vertices are *critical*. A critical point is a *maximum* if the upper link is empty and a *minimum* if the lower link is empty. It is a *simple saddle* if it has one upper link and two lower link components, or two upper link and one lower link components. All other critical points are degenerate.

Both the conditions for a Morse function typically do not hold in practice for PL functions. In case of a degenerate critical point, it can be split into multiple simple saddles as shown by Edelsbrunner et al. [9] and Carr et al. [2]. A simulated perturbation of the function [8] ensures that no two critical values are equal. This perturbation is accomplished by adding an infinitesimally small value to the vertices such that it imposes a total order on the vertices of the graph domain. This helps in consistently identifying the vertex with the higher function value between a pair of vertices, thus ensuring that Condition 2 holds. Note that in the latter case, when considering a flat local minimum (or maximum), one point in that region is identified as critical. This does not affect the identified features, since the thresholds are computed based on the persistence of the critical points, which does not change because of the location [4].

B.2 Merge Tree Computation

For completeness, we now briefly describe the algorithm to compute merge trees of a PL function. For a more detailed description, we refer the reader to the work by Carr et al. [2]. The algorithm uses the local neighborhood of a vertex in G to classify whether the vertex is a critical point or not. The merge tree is built based on this classification.

The join tree is computed by first sorting the vertices of G in decreasing order of function value. Next, for each vertex v in this sorted list, the algorithm performs the following operations:

- If v is a maximum, create a new component containing v and set v as its *head*. A vertex is a maximum if its upper link is empty [1]. This is a direct extension of Definition 4 to PL functions.
- if the vertices in the upper link belong to one component, then the vertex is not critical. Add v to this component.
- v is a critical point and not a maximum. Given a Morse function, the upper link in this case consists of 2 components [9, 2]. Add an edge between v and the head of each of the 2 components. Next, merge these components and set v as the head of the merged component.

Similarly, the split tree is computed by traversing the vertices of G in increasing order of function values and looking at the lower link of the vertices. The main operations performed in the algorithm are: (i) creating components; (ii) looking up components; and (iii) merging components. This can be efficiently accomplished using the union-find data structure [6].

Time Complexity. Let $N = |V|$ be the number of vertices in G. When computing the join / split trees, the vertices are first sorted, which takes $O(N \log N)$ time. Then, for each vertex v, a *create component* or *merge component* operation is called once, resulting in a total of N such operations. A component look up of a vertex v is executed whenever a vertex in its upper link is processed. So, the total number of look up operations is equal to $|E|/2$ (total degree over all vertices). The spatial domains considered in this work correspond to cities, which are planar. Thus, $|E| = O(N)$. Therefore, there is a total of $O(N)$ union-find operations requiring $O(N\alpha(N))$ time, where $\alpha()$ is the inverse Ackermann function. Combining all these steps, the join and split trees can be computed in $O(N \log N + N\alpha(N))$ time.

C. IMPLEMENTATION

The number of scalar functions to be computed is proportional to the product of the total number of attributes in the data collection and the different resolutions. For the urban data sets we assessed (Section 6), thousands of scalar functions were computed, and the number of relationships evaluated during querying was in the order of millions. However, both the indexing and querying operations can be run independently for each scalar function and scalar function pair, respectively. To leverage the *embarrassingly parallel* nature of these computations, we implemented the framework using *map-reduce*. We use three *map-reduce* jobs:

- *Scalar Function Computation Job.* The map phase of the scalar function computation job maps each data point (or tuple) into the appropriate spatio-temporal points based on the resolution. The reduce phase aggregates data from each spatio-temporal point and generates all the scalar function values for that point.
- *Feature Identification Job.* The map phase splits the different scalar functions based on the spatio-temporal resolution, while the reduce phase creates the merge tree index and identifies the features for a single function. Recall that each spatio-temporal resolution of a (data set, attribute) pair is represented by one function.
- *Relationship Computation Job.* The map phase generates the different possible combinations between data sets based on the input query (including different resolutions and feature types). The reduce phase then evaluates each pair of functions based on the pre-computed set of features. To evaluate a relationship, we need to compute the intersection of feature sets. This is accomplished by representing each set of features as a bit vector, which not only reduces the memory size, but also allows us to use efficient bit operations to compute this intersection.

We use Hadoop for the *map-reduce* implementation and HDFS as the distributed file system, where the input data sets, intermediate data, and output are stored. We use compression (BZip2 or Snappy codecs) for both map and reduce outputs, and this led to significant speed ups, in particular for large data sets, as fewer bytes need to be read and written. We also use caching to speed-up future queries: when a query is issued, the framework first checks if it can be answered using the cache before invoking the *map-reduce* job. In case CLAUSE provides a threshold with respect to a data set, then features are first identified before evaluating the relationship.

D. STANDARD APPROACHES

In Section 6.4, we compared our approach against three well-known techniques for identifying correlations: Pearson correlation coefficient (PCC) [3], mutual information (MI) [14], and dynamic time warping (DTW) [10, 12]. In what follows, we describe these approaches in detail.

Pearson's Correlation Coefficient (PCC). The Pearson's Correlation Coefficient measures the linear correlation between two variables [3]. Let X and Y be discrete random variables. The PCC score β_{PCC} can be computed as:

$$\beta_{PCC}(X,Y) = \rho_{X,Y} = \frac{\mathrm{cov}(X,Y)}{\sigma_X \sigma_Y}$$

where *cov* is the covariance and σ is the standard deviation. The PCC score is a value between -1 and +1, where -1 indicates total negative correlation, +1 indicates total positive correlation, and 0 indicates no linear correlation.

Mutual Information (MI). In information theory, mutual information quantifies the amount of information obtained about one random variable through another random variable. Formally, the MI between two discrete random variables X and Y is given as:

$$I(X,Y) = \sum_{i \in X} \sum_{j \in Y} P(i,j) \log \frac{P(i,j)}{P(i)P(y)}$$

where $P(i,j)$ is the joint probability distribution function of X and Y, and $P(i)$ and $P(j)$ are marginal distribution functions of X and Y, respectively. Since we wanted to compare different pairs of variables, we use a normalized variant of MI (β_{MI}):

$$\beta_{MI}(X,Y) = \frac{I(X,Y)}{\sqrt{H(X)H(Y)}}$$

where $H(X)$ and $H(Y)$ are the Shannon entropies of X and Y, respectively. The MI score ranges from 0 to 1: while 0 indicates that the variables are independent, 1 specifies that variables are completely dependent.

Dynamic Time Warping (DTW). Dynamic Time Warping is a dynamic programming approach [12, 13] that is used to construct a *minimum edit-distance* between two temporal sequences. Traditionally, DTW has been used to compare different pairs of time series and perform clustering. However, since DTW distances are not naturally normalized, they cannot be directly compared across multiple time series. Therefore, we propose and employ the following normalized DTW score β_{DTW}:

$$\beta_{DTW}(X,Y) = 1 - \frac{\mathrm{DTW}(X,Y)}{\mathrm{DTW}(X,0) + \mathrm{DTW}(0,Y)}$$

where 0 represents a constant line at $y = 0$, and variables X and Y are Z-normalized. This score ranges from 0 (dissimilar/uncorrelated) to 1 (identical/correlated) and allows us to compare across DTW scores and time series distances.

E. ADDITIONAL EXPERIMENTS

E.1 Robustness Evaluation

In addition to the taxi density function, we also tested the robustness of our technique for other scalar functions of the Taxi data set, including the unique function (Figure I), the average miles function (Figure II), and the average fare function (Figure III). Similar to the result reported in Section 6.2 for the density function, we find that our technique is robust with other functions as well.

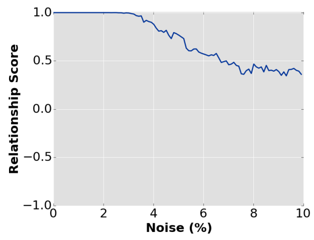
E.2 Interesting Relationships

In this section, we discuss some additional interesting and potentially informative relationships that were identified for the *NYC Urban* collection.

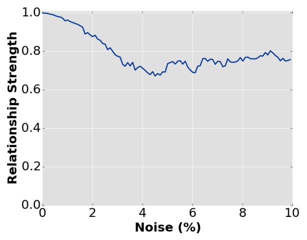
Weather and Taxi. A relationship between the unique number of taxi medallions and the average precipitation could also be identified in the (day, city) resolution with a higher score than in the (hour, city) resolution: $\tau = -0.81$. We also found out that the unique number of taxi medallions is positively related to the average visibility in the (day, city) resolution ($\tau = 0.7, \rho = 0.45$), and negatively related to the average snow depth in the (week, city) resolution ($\tau = -0.95, \rho = 0.95$): this might indicate that some taxi drivers avoid working when the weather conditions are not appropriate (e.g., low visibility or high snow accumulation in streets). There was also a negative relationship between the number of taxi trips and the average snow precipitation in the (week, city) resolution ($\tau = -0.87, \rho = 0.82$) and in the (month, city) resolution ($\tau = -1.0, \rho = 0.07$).

Weather and Citi Bike. We identified negative relationships that involve the unique number of Citi Bikes with the average snow precipitation ($\tau = -1.0, \rho = 0.19$), the average snow depth ($\tau = -0.62, \rho = 0.45$), and rainfall ($\tau = -0.62, \rho = 0.44$), all in the (day, city) resolution. This indicates that less Citi Bikes are used when weather conditions are unpleasant.

Weather and Traffic Speed. For the Weather and Traffic Speed data sets, we found a positive relationship between the average visibility and the average traffic speed in the (week, city) resolution ($\tau = 0.75, \rho = 0.24$) and in the (month, city) resolution ($\tau = 1.0, \rho = 0.14$). Such relationships are expected: when the visibility in the streets is low (e.g., due to a foggy weather), cars tend to drive slower to avoid accidents.

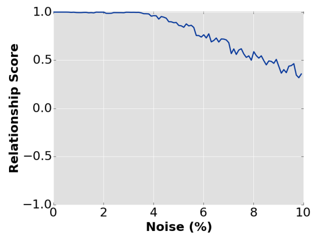


(a) Score

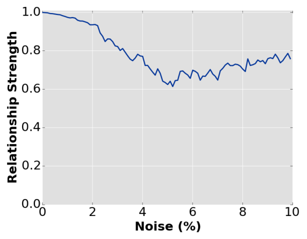


(b) Strength

Figure I: Robustness evaluation (unique number of taxis).

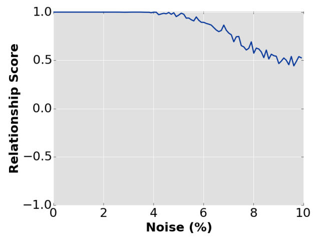


(a) Score

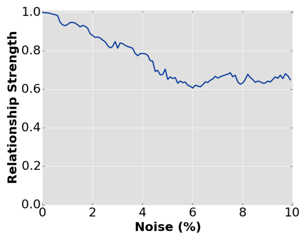


(b) Strength

Figure II: Robustness evaluation (average of traveled miles).



(a) Score



(b) Strength

Figure III: Robustness evaluation (average of total taxi fare).

Taxi and Traffic Speed. Recall that we discovered a negative relationship between the number of taxi trips and the average traffic speed in the (hour, city) resolution. This relationship was also identified in other resolutions, including (day, neighborhood) ($\tau = -0.99, \rho = 0.08$); (day, city) ($\tau = -0.84, \rho = 0.4$); and (week, city) ($\tau = -1.0, \rho = 0.16$). Additionally, there was also a negative relationship between the average taxi trip duration and the average traffic speed in the (day, city) resolution, having $\tau = -0.93$ and $\rho = 0.51$.

Taxi and Gas Prices. We found a positive relationship between the average fare and the average gas price in the (month, city) resolution, with $\tau = 1.0$ and $\rho = 0.5$: in fact, the per-mile metered taxi rate may increase when gas prices also increase [7]. We find this at a monthly resolution because it is difficult to detect the affect of gas prices at a lower resolution. Note that the Gas Prices data is available at a weekly resolution. We also found a negative relationship between the unique number of taxis and the average gas price in (week, city) resolution ($\tau = -1.0, \rho = 0.008$) and (month, city) resolution ($\tau = -1.0, \rho = 0.5$). However, the implication of this relationship is not clear.

311 and 911. There were some curious relationships between the number of 311 complaints and the number of 911 calls, in both (day, neighborhood) resolution ($\tau = 0.92, \rho = 0.27$) and (week, neighborhood) resolution ($\tau = 1.0, \rho = 0.65$). Experts can perform further analysis to better understand these relationships and find out if emergency and non-emergency calls are indeed related.

Vehicle Collisions. We identified some interesting relationships between Vehicle Collisions and other data sets. In the (day, city) resolution, there was a positive relationship between the average number of motorists injured in a vehicle crash and the average traffic speed ($\tau = 0.64, \rho = 0.46$), which might indicate that high speeds influence the occurrence of collisions that are dangerous for drivers. In the (day, neighborhood) resolution, there were relationships between the number of collisions and the number of 311 complaints ($\tau = 0.84, \rho = 0.41$), and also with the number of 911 calls ($\tau = 0.94, \rho = 0.18$). The former also showed up in the (week, neighborhood) resolution, with $\tau = 0.72$ and $\rho = 0.22$. In this resolution, there was also a positive relationship between the number of collisions and the number of taxi trips ($\tau = 0.99, \rho = 0.25$). Again, although these relationships may not have apparent implications, experts can use this information to perform further analysis and identify the reason behind them.

F. REFERENCES

[1] T. F. Banchoff. Critical Points and Curvature for Embedded Polyhedral Surfaces. *Am. Math. Monthly*, 77:475–485, 1970.

[2] H. Carr, J. Snoeyink, and U. Axen. Computing Contour Trees in All Dimensions. *Comput. Geom. Theory Appl.*, 24(2):75–94, 2003.

[3] V. A. Clark, O. J. Dunn, and R. M. Mickey. *Applied Statistics: Analysis of Variance and Regression*. John Wiley, 2004.

[4] D. Cohen-Steiner, H. Edelsbrunner, and J. Harer. Stability of persistence diagrams. *Disc. Comput. Geom.*, 37(1):103–120, 2007.

[5] K. Cole-McLaughlin, H. Edelsbrunner, J. Harer, V. Natarajan, and V. Pascucci. Loops in Reeb graphs of 2-manifolds. *Disc. Comput. Geom.*, 32(2):231–244, 2004.

[6] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, 2001.

[7] P. Donohue. Taxi Drivers Petition NYC for Fare Hike over Soaring Gas Prices. *NY Daily News*, April 2011.

[8] H. Edelsbrunner. *Geometry and Topology for Mesh Generation*. Cambridge Univ. Press, England, 2001.

[9] H. Edelsbrunner, J. Harer, V. Natarajan, and V. Pascucci. Morse-Smale Complexes for Piecewise Linear 3-Manifolds. In *Symp. Comput. Geom.*, pages 361–370, 2003.

[10] E. Keogh and A. Ratanamahatana. Everything You Know About Dynamic Time Warping is Wrong. *Workshop on Mining Temporal and Sequential Data*, 2004.

[11] J. Milnor. *Morse Theory*. Princeton Univ. Press, 1963.

[12] H. Sakoe and S. Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 26(1):43–49, 1978.

[13] S. Salvador and P. Chan. Toward Accurate Dynamic Time Warping in Linear Time and Space. *Intell. Data Anal.*, 11(5):561–580, 2007.

[14] Y. Su, G. Agrawal, J. Woodring, A. Biswas, and H.-W. Shen. Supporting Correlation Analysis on Scientific Datasets in Parallel and Distributed Settings. In *HPDC*, pages 191–202, 2014.