Structurally Robust Similarity Search

Anonymous

ABSTRACT

Graph similarity search algorithms usually leverage the structural properties of a database. Hence, these algorithms are effective only on some structural variations of the data and are ineffective on other forms, which makes them hard to use. Ideally, one would like to design a data analytics algorithm that is structurally robust, i.e., it returns essentially the same accurate results over all possible structural variations of a dataset. We propose a novel approach to create a structurally robust similarity search algorithm over graph databases. We leverage the classic insight in the database literature that schematic variations are caused by having constraints in the database. We then present RelSim algorithm which is provably structurally robust under these variations. Our empirical studies show that our proposed algorithms are structurally robust while being efficient and as effective as or more effective than the state-of-the-art similarity search algorithms.

CCS CONCEPTS

• **Computer systems organization** → **Embedded systems**; *Redundancy*; Robotics; • **Networks** → Network reliability.

KEYWORDS

datasets, neural networks, gaze detection, text tagging

ACM Reference Format:

Anonymous. 2018. Structurally Robust Similarity Search. In *Woodstock '18: ACM Symposium on Neural Gaze Detection, June 03–05, 2018, Woodstock, NY*. ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Unsupervised and supervised machine learning (ML) methods are widely used over structured data [29, 43–47, 52, 57]. To deliver effective predictions, these methods usually leverage structural or topological features of the data that quantify relationships between entities. For example, consider the DBLP dataset (*dblp.uni-trier.de*) that stores information about papers, conferences, and research areas in computer sciences whose fragments are shown in Figure 1(a). Assume that a user wants to find the most similar research area to *Data Mining* in this dataset according to their conferences and publications. Similarity search algorithms usually use the structure of the graph to measure the degree of similarity between entities in the data. For example, SimRank [29] is a well-known unsupervised similarity search algorithm over graphs that quantifies the similarity between two entities according to how likely two random surfers will meet each other if they start from the two entities [29, 43, 46, 47, 52, 57]. Thus, it correctly finds *Data Mining* to be more similar to *Databases* than to *Software Engineering* in Figure 1(a). Oversimplifying a bit, since *Data Mining* and *Databases* share relatively more neighbors, i.e., papers, over this dataset, they have a relatively high SimRank score. These similarity measures, such as SimRank, are used as features to build ML models for various data analytics tasks, e.g., pattern query matching or community detection [2, 46, 47, 57].

It is, however, established that different datasets usually represent essentially the same information in different forms and structures [1, 4, 7, 10, 16, 17, 21, 28, 35, 55]. A classic and well-known example of such structural variations is (de-)normalization in relational and XML data where the original and normalized databases represent the same information under different structures [1, 3, 4, 55]. As another example, consider the SIGMOD Record bibliographic dataset (*sigmod.org/publications*) whose fragments are shown in Figure 1(b). Intuitively, this dataset represent essentially the same information about the same set of *paper*, *conference*, and *research area* entities as the one in Figure 1(a). But, each dataset has its own way of representing these entities and their relationships. For example, DBLP connects directly each paper to its research areas and conferences. Given that all papers in a conference share the same set of research areas, one can also choose the structure in Figure 1(b) to represent this information and connect research areas of a paper to its conferences. This dataset represents the relationship between the research area of a paper using a path through the conference of the paper instead of a direct link as in Figure 1(a). We have been working with bioinformatics experts to analyze large graph datasets from various data sources and have observed that they often restructure their graph data to meet certain efficiency-oriented or data quality goals. For instance, if two nodes are far apart in the data and are often queried together in some lookup queries, the experts add new edges that connect these entities directly to rewrite and run lookup queries faster. This modification does *not* add any new information to the database as the new edge can be inferred from the current (long) path between the corresponding entities.

Structural features delivers different values across the aforementioned variations, which may return to inaccurate results on some structures for the same information. For example, SimRank finds *Data Mining* more similar to *Software Engineering* than *Databases* on the data fragment in Figure 1(b), which is clearly not accurate. As explained in the preceding paragraph, Figure 1(a) and 1(b) contain essentially the same information about the relationships between the same set of entities. Thus, the accuracy of SimRank depends on the structure and the form information is presented rather than the content of the underlying data. Due to the constant data evolution [20, 23, 34, 41, 51] and growing need for analyzing different datasets with a great deal of structural variations [5, 6, 10, 19, 21, 22, 35, 42], similarity features and models are often used across datasets with

Woodstock '18, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

(a) Fragments of DBLP (b) Fragments of SIGMOD Record

Figure 1: Example of two bibliography databases whose nodes are papers, conferences and research areas.

different forms and structures. Thus, the accuracy of current similarity features and models is unstable and unpredictable, e.g., they may perform well on the datasets used to develop or train these features but poorly on others.

It is known that variations in the content of the data across different datasets may reduce the efficacy of features and models when applied on a new dataset, e.g., due to their bias toward certain data distributions [9, 31, 33, 36, 39, 54]. Thus, it has been a central problem in ML and data analysis to measure or improve the robustness or generalizability of features and models against variations in the content of datasets [31, 36, 38, 39]. For the same reason, it is also important to ensure that similarity models and algorithms generalize and are effective to *unseen structures*, i.e., the structure of the information other than the ones used to develop or train features and models. For example, consider a developer who is building a system that finds similar nodes in the input graph datasets. To develop and choose similarity features and algorithms, she may test feature on some datasets and select the one that returns the answers that the domain experts deem accurate. Users would like to use this system on datasets other than the ones used to test its effectiveness whose may follow a different style of structure than the datasets used to test the system. It may not be, however, possible for the developer to test the system over sufficiently many datasets that cover all potential structural variations that users may face when using the system. Thus, it is not clear whether the developed features and algorithms for similarity will be effective to the datasets over which the system is used by users. This will significantly limit the off-the-shelf use of such systems.

The robustness and generalizability of features in the face of structural variations across different datasets have not been explored. In this paper, we investigate the robustness of similarity features and algorithms in the face of the structural variations of the data. We focus on the problem of similarity search as it is both a popular type of queries over graphs and an important building block of other data analysis tasks, such as pattern query matching, community detection, and clustering [29, 46, 47, 57]. We have two main goals in our approach to finding robust similarity features and algorithms. First, we aim at building a theoretical framework that captures a sufficiently general notion of structural variations and robustness, which is able capture a wide variety of structural variations. We leverage the rich body of research on structure and schema management and variation in database community to achieve this goal. Second, instead of developing new features and algorithms that can generalize and are robust against structural variations, we would like to extend and enhance current features and algorithms to be robust. Since current methods and features are currently used in data analytics systems, this approach will make it easier to achieve robustness in current systems. Moreover, this effort may lay the foundation and provide insights on how to enhance current features or algorithms used in other ML and data analysis tasks to achieve structural robustness.

In our investigations in this paper, we establish a relationship between the structural robustness of similarity search features and algorithms and the languages used to specify these features in the graph, e.g., the language that expresses the types of paths used for random walks in the graph. That is, oversimplifying a bit, the more expressive the language is, the more robust feature similarity algorithm is. Interestingly, this is in contrast with the robustness and generalizability of models against statistical variations in the content of the data where usually a more expressive model may be harder to generalize to unseen data than a less expressive one [36]. It is very time-consuming to compute values for features that use an overly expressive language [24, 53]. Thus, we propose a feature specification language that is sufficiently expressive to be robust over popular structural variations and its expressed features are computed efficiently over large data. Moreover, as it may be hard for users to work with a complex language to express features, we propose a usable method in which users specify high-level hints for the system to find and compute relevant features. In particular, we make the following contributions.

- We define structural generalizability and robustness of a similarity search algorithm. (Section 3). To find and explore structural variations of a graph database, we leverage results in database literature, which state that database constraints give rise to its structural variations and extend it to graph data [13, 28, 50].
- We show that existing similarity search features and algorithms are *not* robust because of the limited types of relationships they use to measure the degree of similarity. We propose a robust algorithm called *RelSim*, which extends a well-known similarity search algorithm (*PathSim* [40, 43]) by using a sufficiently expressive set of patterns. We also show that features expressed in such language are computed efficiently over large data.
- RelSim requires users to specify the relationship between nodes to compute similarity score between the nodes using the proposed language, which may be relatively hard. We propose an algorithm that leverages simple guidelines from the user to compute structurally robust similarity scores efficiently (Section 5).
- We report the results of our extensive empirical studies over large graph databases, which indicate that our proposed algorithms are structurally robust and improves the effectiveness of its original similarity search algorithm (Section 7). Our empirical studies also show that our algorithms are efficient over large data.

2 GRAPH DATA AND CONSTRAINTS

We fix a countably infinite set of node ids denoted by $\mathcal{V}$. Let $\mathcal{L}$ be a finite set of labels. A *database* D over $\mathcal{L}$ is a directed graph (V, E) in which V is a finite subset of $\mathcal{V}$ and $E \subseteq V \times L \times V$. This definition of graph databases is frequently used in the graph data management

literature [7, 18, 53]. We denote an edge from node u to node v whose label is a as (u, a, v). We say that $(u, a, v) \in D$ whenever $(u, a, v) \in E$. Similarly, we say that $v \in D$ whenever $v \in V$.

Database constraints restrict the instances of a schema. They are usually expressed as logical formulas over schemas [1, 8, 10]. Two widely used type of constraints are *tuple-generating* and *equality-generating* dependencies [8, 10]. A tuple-generating dependency (*tgd* for short) over schema $\mathcal{L}$ is in the form of $\forall \bar{x}(\phi(\bar{x}) \rightarrow \exists \bar{y}\psi(\bar{x}, \bar{y}))$ where $\bar{x}$ and $\bar{y}$ are sets of variables, and ϕ and ψ are logical formulas in a query language over $\mathcal{L}$. A *full tgd* does *not* have any existential variable in its conclusion. An equality-generating dependency (*egd* for short) over schema $\mathcal{L}$ is in the form of $\forall \bar{x}(\phi(\bar{x}) \rightarrow x_1 = x_2)$ where $\bar{x}$ is a set of variables, $x_1, x_2 \in \bar{x}$, and ϕ is logical formulas in a query language over $\mathcal{L}$.

Example 1. *Database shown in Figure 1(a) contains a constraint* $(x_1, \texttt{area}, x_3) \wedge (x_3, \texttt{pub-in}, x_4) \wedge (x_2, \texttt{pub-in}, x_4) \rightarrow (x_1, \texttt{area}, x_2)$.

Tgds and egds are arguably the most popular and frequently used types of database constraints and generalize popular constraints, such as function and multi-valued dependencies [1].

A commonly studied query language over graph databases is *conjunctive regular path queries* (conjunctive RPQ), which is used to express tgd and egd constraints over graph databases [7, 12, 21, 53]. The *RPQ* p over schema $\mathcal{L}$ is defined by the following grammar:

$$p := \epsilon \mid a\ (a \in \mathcal{L}) \mid a^{-}\ (a \in \mathcal{L}) \mid p \cdot p \mid p + p \mid p^{*}$$

in which ϵ is an empty label, $^{-}$ is a reverse traversal of an edge, $\cdot$ is a concatenation, $+$ is a disjunction, and $*$ is a Kleene star. To avoid parentheses and ambiguity, it is assumed that the reverse traversal has the highest priority, then Kleene star, then concatenation and then disjunction. Example of an RPQ over a schema of a database shown in Figure 1(b) is $\texttt{field} \cdot \texttt{published-in}^{-}$. The RPQ p defines a binary relation over database nodes. More precisely, the result of evaluating p on database D is a set of pairs of nodes in D such that there is a path defined by p between the two nodes. We denote the result of evaluating p over D as $[[p]]_D$. For example, given label a in the schema of D, the result of $[[a]]_D$ is a set of pairs of nodes $\{(u, v)\}$ where there is an edge with label a from u to v. Let $\bar{x} = (x_1, \ldots, x_n)$ and $\bar{y} = (y_1, \ldots, y_m)$ be tuples of distinct variables. A conjunctive RPQ is a formula $\phi(\bar{x})$ of the form $\exists \bar{y}((z_1, p_1, z_1') \wedge \ldots \wedge (z_k, p_k, z_k'))$ where p_i is an RPQ and $z_i, z_i' \in \{x_1, \ldots, x_n, y_1, \ldots, y_m\}$, $1 \leq i \leq k$ [7, 53]. We call (z_i, p_i, z_i') an *atom* of $\phi(\bar{x})$.

A schema S is a pair $(\mathcal{L}, \Gamma_S)$ in which $\mathcal{L}$ is a (finite) set of labels and Γ_S is a finite set of constraints. By the abuse of notation, we say that a label $l \in S$ if $l \in \mathcal{L}$ and a constraint $\gamma \in S$ if $\gamma \in \Gamma_S$. Each *database* of schema $S = (\mathcal{L}, \Sigma_S)$ is a database over $\mathcal{L}$ such that all constraints in Σ_S holds. We denote the set of all databases of schema S as $\texttt{Inst}(S)$. A similarity query q over database $I(V_I, E_I) \in \texttt{Inst}(S)$ is a node id in V_I [2, 25, 29, 30, 40, 43, 48, 56, 58]. The answers to a similarity query q over I is a ranked list of node ids in I that are not equal to q.

3 STRUCTURAL ROBUSTNESS AND VARIATIONS

3.1 Structural Robustness

Intuitively, a structurally-robust query answering algorithm should return essentially the same (list of) answers for the same query across databases that contain the same information content. Researchers have leveraged the concept of invertible transformation to formalize the equivalence of information stored in different databases [17, 28]. A *transformation* from schema S to schema T is a function from $\texttt{Inst}(S)$ to $\texttt{Inst}(T)$, which maps each database of S to a database of T [17, 28]. We denote a transformation from S to T as Σ_{ST}. For example, a transformation $\Sigma_{1a,1b}$ from schema of the database in Figure 1(a) to the schema of the one in Figure 1(b) changes the structure of the database in Figure 1(a) such that the research areas associated with a paper become connected to the paper via the conference of the paper and produces the database in Figure 1(b).

This definition of transformations may *not* be sufficiently powerful to capture structural variations. Assume that the schema of the database in Figure 1(b) contains an additional type of relationship called *keyword-of* that connects each paper published in a conference to nodes, which store keywords of the paper. Consider an updated version of the database in Figure 1(b) in which each paper is connected to some additional nodes via relationship *keyword-of*. Intuitively, the updated database has more information than the one in Figure 1(a). Let us define transformation $\Sigma_{1a,1c}$ between the schema in Figure 1(b) and the updated schema such that it modifies relationships between research areas, conferences, and papers similar to $\Sigma_{1a,1b}$ and adds some keywords to each paper. This transformation maps each database in the original schema to multiple databases under the transformed schema where each database may have different (number of) keywords for the same paper. Thus, we use a definition of transformation between schemas S and T in which the transformation Σ_{ST} establishes a relation between $\texttt{Inst}(S)$ and $\texttt{Inst}(T)$ to cover the aforementioned cases [7, 16]. It maps each database $I \in \texttt{Inst}(S)$ to at least one database $J \in \texttt{Inst}(T)$ and it is *not* a function. One denotes the fact that J is a transformation I under Σ_{ST} as $(I, J) \models \Sigma_{ST}$ [7, 16]. For brevity and by the abuse of notation, we show the transformed databases of I under Σ_{ST} as $\Sigma_{ST}(I)$.

The transformation Σ_{ST} is *invertible* if there is a transformation Σ_{TS} from T to S such that, for each database $I \in \texttt{Inst}(S)$, Σ_{TS} maps every database $\Sigma_{ST}(I)$ to I and only I, i.e., for each database in $\Sigma_{ST}(I)$, $\Sigma_{TS}(\Sigma_{ST}(I))$ is exactly I. In other words, the composition of $\Sigma_{ST}(I)$ and $\Sigma_{TS}(I)$, shown as $\Sigma_{ST}(I) \circ \Sigma_{TS}$ is equivalent to the identity transformation *id* that maps each database only to itself. In this case, we call Σ_{TS} the *inverse* of Σ_{TS} and denote it as Σ_{ST}^{-1}. If there is an invertible transformation from schema S to T, one can reconstruct exactly the information in each database I from the information available in $\Sigma_{ST}(I)$. In other words, each database in $\Sigma_{ST}(I)$ has at least the same amount of information as I. We say that S has at least as much information as T and denote their relationship as $S \overset{\Sigma_{ST}}{\preceq} T$ or simply $S \preceq T$ if Σ_{ST} is clear from the context.

Example 2. *Consider transformation $\Sigma_{1a,1b}$ from schema of the database in Figure 1(a) to the schema of one in Figure 1(b). Because of the constraint described in Example 1, this transformation is invertible. Intuitively, this constraint over Figure 1(a) implies that papers published in the same conference are related to the same set of research areas. Hence, one may change the structure shown in Figure 1(a) such that the research areas associated with a paper is instead connected to the paper via the conference of the paper and get the database fragment shown in Figure 1(b). One can recover the information in the original database using the information in Figure 1(b). One can find the exact set of research areas directly connected to each paper in Figure 1(a) by checking the research areas directly connected to the conference of that paper. Similarly, $\Sigma_{1a,1c}$ is also invertible as one can follow the same approach to recover the information of the database in Figure 1(a) from the transformed databases.*

Our definition of inverse extends the notion of Fagin-inverse used in the context of relational data exchange [13]. In a Fagin-inverse, a transformation may map multiple databases from schema S to multiples ones under schema T and its inverse maps multiple databases under T back to multiple ones under S. The answers to a similarity query in different databases of S may be different. Hence, we use a more strict version of inverse because our goal is to compare the results of an algorithm over a single database and all its structural variations that contain the same information. Therefore, we are interested in the transformations that map a single database under the original schema to one or more databases under the transformed schema. Consequently, the inverse transformation must map those databases under the transformed schema to the original database and only the original database. Therefore, for the rest of this paper, we assume that each transformation maps a single database to multiple ones and its inverse maps multiple databases back only to the original database.

Next, we present the definition of a *structurally robust* (robust for short) algorithm. Roughly speaking, a robust algorithm must return the same results for the same input query over a database and databases under invertible transformations. Two (ranked) lists of node ids are *equivalent* if they contain exactly the same node ids at the same positions. Two empty lists of answers are equivalent.

Definition 1. *Given schemas S and T such that $S \overset{\Sigma_{ST}}{\preceq} T$, an algorithm is robust under Σ_{ST} if it returns equivalent answers for every input query q over every database $I \in \texttt{Inst}(S)$ and every database in $\Sigma_{ST}(I)$.*

An algorithm is robust under a set of transformations if it is robust under all members of the set.

3.2 Structural Variations

3.2.1 Expressing Transformations. To characterize the structural variations of a schema, one needs to express invertible transformations. Researchers usually use *declarative (schema) mappings* to express schematic variations in graph and relational databases [7, 13, 16]. Roughly speaking, a transformation between schemas S and T is expressed as a set of logical formulas $\phi_S(\bar{x}) \rightarrow \psi_T(\bar{y})$ where $\phi_S(\bar{x})$ and $\psi_T(\bar{y})$ are queries over schemas S and T, respectively. More precisely, transformation Σ_{ST} between graph schemas S and T is a finite set of rules $\phi_S(\bar{x}) \rightarrow \psi_T(\bar{y})$ such that $\bar{x} \subseteq \bar{y}$ and $\phi_S(\bar{x})$, i.e., *premise*, and $\psi_T(\bar{y})$, i.e., *conclusion*, are conjunctive RPQs over S and T, respectively [7]. In each rule $\phi_S(\bar{x}) \rightarrow \psi_T(\bar{y})$, every variable in $\bar{x}$ is universally quantified and every one in the set of $\bar{y}$ either belongs to $\bar{x}$ or is existentially quantified.

Example 3. *The transformation $\Sigma_{1a,1b}$ from schema of the database in Figure 1(a) to the one in Figure 1(b) is expressed using a mapping with two rules: $(x_1, \texttt{pub-in}, x_2) \rightarrow (x_1, \texttt{pub-in}, x_2)$ and $(x_1, \texttt{area} \cdot \texttt{pub-in}, x_2) \rightarrow (x_1, \texttt{field}, x_2)$. The inverse of $\Sigma_{1a,1b}$ can also be expressed using following rules: $(x_1, \texttt{field} \cdot \texttt{pub-in}^-, x_2) \rightarrow (x_1, \texttt{area}, x_2)$ and $(x_1, \texttt{pub-in}, x_2) \rightarrow (x_1, \texttt{pub-in}, x_2)$. The transformation $\Sigma_{1a,1c}$ that maps databases under the schema in Figure 1(a) to the one shown in Figure 1(b) with some keyword nodes connected to each paper published in a conference can be expressed using the aforementioned rules plus rule $(x_1, \texttt{pub-in}, x_2) \rightarrow (y_1, \texttt{keyword-of}, x_1)$, where y_1 is an existential variable that represents the nodes that contain keywords of the paper x_1. This rule can map a database to multiple one each of which has different number of keyword nodes. The inverse of $\Sigma_{1a,1c}$ is the same as the inverse of $\Sigma_{1a,1b}$ as it does not need the keywords to reconstructs the data.*

Transformation Σ_{ST} maps each database $I \in \texttt{Inst}(S)$ to $J \in \texttt{Inst}(T)$ if for each rule $\phi_S(\bar{x}) \rightarrow \psi_T(\bar{y})$ in Σ_{ST}, we have $\bar{u} \in [[\psi_T(\bar{x})]]_J$ if $\bar{u} \in [[\phi_S(\bar{y})]]_I$ [7]. We use the closed world semantic for schema mappings [26]. That is, transformation Σ_{ST} maps $I \in \texttt{Inst}(S)$ to only databases whose nodes and edges are constructed using the mapping rules. The alternative semantic is the open world semantic in which the transformations of $I \in \texttt{Inst}(S)$ under Σ_{ST} may contain nodes and edges that are *not* created as the result of the schema mapping rules [7, 13, 16]. A consequence of this semantic is that the inverse of transformation Σ_{ST} will map each database $\Sigma_{ST}(I)$ to other databases in addition to I, e.g., all databases in S that include at least the same amount of information as I [13]. Our goal, however, is to compare the results of similarity queries over the original database I and its variations. Thus, we use the closed world semantic for schema mappings.

3.2.2 Information Preservation. Next, we present the full characterization of invertible transformations of a schema. It helps us to identify the set of structural variations of a schema, which we use to design robust algorithms. Consider transformation Σ_{ST} from schemas S to T and Σ_{TR} from schemas T to R. The *composition* of Σ_{ST} and Σ_{TR}, denoted as $\Sigma_{TR} \circ \Sigma_{ST}$, is a transformation from S to R such that if $J \in \Sigma_{ST}(I)$ and $K \in \Sigma_{TR}(J)$, then $K \in (\Sigma_{TR} \circ \Sigma_{ST})(I)$.

Given transformations Σ_{ST} from schema S to T, its inverse Σ_{ST}^{-1} is a transformation from T back to S. Thus, the composition $\Sigma_{ST}^{-1} \circ \Sigma_{ST}$ will map the database $I \in \texttt{Inst}(S)$ to (only) itself. In other words, the composition of $\Sigma_{ST} \circ \Sigma_{ST}^{-1}$ are a set of rules, i.e., constraints, over S. We have the following proposition by applying the definitions of the inverse and composition of transformations.

Proposition 1. *Given schemas S and T such that $S \overset{\Sigma_{ST}}{\preceq} T$, for all databases $I \in \texttt{Inst}(S)$ we have $I \models \Sigma_{ST}^{-1} \circ \Sigma_{ST}$.*

Proposition 1 extends the results on the lossless decomposition of a relational schema [50] and the ones on the inverse of a relational schema mapping in [13]. According to Proposition 1, the constraints on schema S determine its structural variations. Following this proposition, the constraints on S that give rise to invertible structural variations are in tgds.

Example 4. *The composition* $\Sigma_{1a,1b} \circ \Sigma^{-1}_{1a,1b}$ *in Example 3 results in a constraint* $(x_1, \texttt{area}, x_4) \wedge (x_4, \texttt{published-in}, x_3) \wedge (x_2, \texttt{published-in}, x_3) \rightarrow (x_1, \texttt{area}, x_2)$ *which is equivalent to the constraint of the database shown in Figure 1(a) as described in Example 1.*

We should note that the composition of two transformations may *not* be expressible using first order schema mapping formulas [7, 14]. Roughly speaking, each rule in $\Sigma_{TR} \circ \Sigma_{ST}$ is created by replacing an atom (x, exp, y) in the premise of a rule in Σ_{TR} by the premise of a rule in Σ_{ST} whose conclusion matches (x, exp, y) [7, 10, 14]. Assume that the conclusion of a rule in Σ_{ST} contains an atom (x, exp, y) with an existentially quantified variable, which may be stated explicitly in the rule or introduced by using concatenation, Kleene star, or disjunction, i.e., +, in exp. Also, assume that there is an atom in the premise of Σ^{-1}_{ST} that matches (x, exp, y). In this case, one needs second-order logic to express this composition [14]. If this happens to a transformation and its inverse, then the set of constraints on the original schema will be tgds in second order logic. To the best of our knowledge, there has *not* been any work on database constraints over graph (or relational) databases that are in languages more expressive than the first order logic, e.g., second order logic, and the database constraints in the first order logic are by far more widely used than the ones expressed in higher order logics [1, 10]. Thus, we focus our attention to the first order constraints as explained in Section 2. If the atoms in the premise of Σ^{-1}_{ST} match the atoms with only universally quantified variables in the conclusion of Σ_{ST}, their compositions can be expressed using the (first order) tgds as defined in Section 2 [7, 14]. Therefore, we consider only transformations whose composition with their inverses can be expressed in first order logic.

As shown in Example 4, the composition of transformation $\Sigma_{1a,1b}$ and its inverse and transformation $\Sigma_{1a,1c}$ and its inverse can be expressed as such tgds. In this case the shared atoms between the premises of rules in Σ^{-1}_{ST} and the conclusions of rules in Σ_{ST} will be in form of (x, l, y) or (x, l^{-1}, y) where l is a label in schema T. Thus, each rule in $\Sigma^{-1}_{ST} \circ \Sigma_{ST}$ is created by replacing an atom (x, l, y) in the premise of a rule in Σ^{-1}_{ST} by the premise of each rule in Σ_{ST} whose conclusion matches (x, l, y) (or (x, l^{-1}, y) by exchanging the positions of x and y). This method naturally extends to the atoms of the form (x, l^{-1}, y) in the premise of rules in Σ^{-1}_{ST}.

Consider a transformation Σ_{ST} from S to T where the conclusions of a rule α in Σ_{ST} contain an existentially quantified variable z. Given $I \in \texttt{Inst}(S)$, the nodes in databases $\Sigma_{ST}(I)$ created as the result of applying α to I and correspond to z do *not* have any fixed ID (or value if applicable) as they do *not* correspond to any node in I. Thus, $\Sigma_{ST}(I)$ will contain more than a single database. Since a transformation maps a database to multiple ones, its inverse must map multiple databases to a single one. Thus, the inverse can be expressed using a set of rules without any existentially quantified variable in their conclusions. Similar results have been shown for the structure of similar types of inverse of schema mappings over relational databases [13]. Thus, the composition of Σ_{ST} and Σ^{-1}_{ST} is a set of rules where the premise of each rule is a conjunctive RPQ and its conclusion is a single RPQ atom in form of (x, exp, y) where exp is either l or l^- where l is a label in S. Hence, $\Sigma^{-1}_{ST} \circ \Sigma_{ST}$ is a set of full tgds over S.

The set of tgds introduced by Proposition 1 is necessary to have invertible transformations for schema S, but it is *not* sufficient. We show that S must satisfy an additional group of tgds to have invertible variations. Let σ denote the set of tgd constraints in $\Sigma^{-1}_{ST} \circ \Sigma_{ST}$. Given σ, we create another group of tgd constraints over S, denoted as σ^*, as follows. For each tgd constraint in σ whose conclusion is in the form of $\chi_1(x, y) \rightarrow (x, l^-, y)$, we replace it with constrain $\chi_1(y, x) \rightarrow (y, l, x)$. Then, for all tgds with the same atom in their conclusions, i.e., $\chi_1(x, y) \rightarrow (x, l, y)$, ..., $\chi_m(x, y) \rightarrow (x, l, y)$ in σ, we construct the constraint $(x, l, y) \rightarrow \chi_1(x, y) \vee ... \vee \chi_m(x, y)$. For each label l' in $\mathcal{L}_S$ that does *not* appear in a conclusion of any constraint in σ, we create the constraint $(x, l', y) \rightarrow \texttt{FALSE}$, which means that there is *not* any database in $\texttt{Inst}(S)$ with any edge whose label is l'.

Proposition 2. *Given transformations* Σ_{ST} *from* S *to* T *and* Σ_{TS} *from* T *to* S*, let* σ *denote* $\Sigma_{ST} \circ \Sigma_{TS}$*.* Σ_{ST} *is invertible with inverse* Σ_{TS} *if and only if, for every database* $I \in \texttt{Inst}(S)$*, we have* $I \models \sigma \wedge \sigma^*$*.*

In the rest of this paper, we refer to an invertible transformation simply as a transformation.

3.2.3 Computing Inverses. There have been numerous works on computing the inverses of a schema mapping over relational databases [5, 13, 15]. In this paper, we focus on analyzing the robustness of algorithms over transformations and designing algorithms that are robust against them. The theoretical results presented in the preceding section enable us to characterize these transformations and design robust algorithms over them. We do not need to compute the inverse of a transformation in this paper. Thus, the problem of computing an inverse of an input schema mapping is not the subject of this paper and is an interesting future work. To provide the reader with some insight on the problem of computing an inverse of a graph schema mapping, we extend the following result from the relational schema mappings. The proof uses the results of and is similar to the one of Theorem 14.9 in [13].

Theorem 1. *Given transformation* Σ_{ST} *from* S *to* T*, the decision problem of whether* Σ_{ST} *is invertible is coNP-hard.*

4 ROBUST SIMILARITY SEARCH

4.1 Robustness of Current Methods

To the best of our knowledge, frequently used structural-based similarity search features and algorithms are based on random walks, e.g., RWR [47], pairwise random walk, e.g., SimRank [29] and P-Rank [58], or path-constrained framework, e.g., PathSim [43] and HeteSim [40]. There are also other similarity search algorithms that extend the aforementioned algorithms such as common neighbors, $Katz_\beta$ measure, commute time, and sampled set of random paths/walks between nodes [32, 57]. The algorithms that are not path-constrained, e.g., SimRank and RWR, are mainly developed to measure similarity over graphs where all edges have the same label [29, 47]. However, these algorithm are sometimes used over data graphs with multiple edge labels [43]. They have also been used as a basis for methods developed for data graphs with multiple labels [40, 43]. We consider both the original and the extended versions of SimRank and RWR, for the sake of completeness.

However, some people have modified and used these two algorithms over databases with multiple types of entities and relationship types [43]. In this paper, we consider the extended version of SimRank and RWR.

Similarity scores computed by algorithms that use random walks and pairwise random walks are largely influenced by the topology of the graph. Because some transformations may modify the topological structure of a database, structural-based similarity search algorithms such as RWR and SimRank are not robust under these variations as shown in our empirical studies in Section 7. Similar to these algorithms, there are algorithms that leverage the idea of random walks by randomly picking some walks or paths between two nodes and randomly traversing certain number steps on each walk [57]. The more similar two nodes are, the more likely it is for one to reach to one of them by starting from another one using the aforementioned method. As they use similar core ideas to RWR and SimRank to measure the similarity between two nodes, their results are similarly influenced by transformations that modify the database topology. An invertible transformation may change the length and the number of paths and walks between two nodes. For example, the length of the path between nodes *VLDB* and *Pattern Mining* is one in Figure 1(a) and two in its variation (Figure 1(b)). One may significantly reduce or increase the length of paths between two entities in a database and its variations under similar transformation. Thus, these methods may deliver different similarity scores for the same pairs of nodes over a a database and the database under an invertible transformation.

Two entities may be similar based on the paths or patterns that separate them in a database where those paths or patterns represent a certain type of relationship. The degree of similarity between two entities may largely depend on the type of relationship between them. For instance, consider a database with researchers, conferences in which they publish, and their affiliations. Some users may want to find similar researchers from the point of view of their affiliations. Other users may like to find similar researchers based on the conferences that they publish their papers in. Thus, one often has to consider the type of relationship between two entities to define an effective similarity metric with a clear semantic. Path-constrained similarity search algorithms, such as PathSim and HeteSim, follow this approach [40, 43]. They allow users to supply a path template, or *pattern*, that specifies the type of relationship between entities in their queries. Example of a pattern in Figure 1 is $m_1 : \texttt{pub-in} \cdot \texttt{pub-in}^{-}$, which reflects the relationship between two papers through a conference in which they are both published in. Each instance of a pattern in database D is a path in D whose sequence of edge labels match the sequence of labels in the pattern. For example, *Similarity Mining*·pub-in·*VLDB*·pub-in^{-}·*Pattern Mining* is an instance of m_1 in Figure 1(b). The PathSim score of entities u and v in database D given a pattern p is

$$\texttt{sim}_p(u, v, D) = \frac{2 \times |u \rightsquigarrow_p v|}{|u \rightsquigarrow_p u| + |v \rightsquigarrow_p v|} \quad (1)$$

where $|u \rightsquigarrow_p v|$, $|u \rightsquigarrow_p u|$ and $|v \rightsquigarrow_p v|$ denote the numbers of (u, p, v), (u, p, u) and (v, p, v) path instances in D, respectively. The robustness of PathSim or other path-constrained similarity search methods largely depend on the representation of the underlying relationships.

Example 5. *Consider two representations of bibliographic data in Figure 1. Suppose a user wants to find similar research areas to* Data Mining *based on their shared conferences. In Figure 1(a), the user uses the pattern* $p_1 : \texttt{area} \cdot \texttt{published-in} \cdot \texttt{published-in}^{-} \cdot \texttt{area}^{-}$ *to represent the relationship and compute similarity scores between research areas. PathSim then finds* Data Mining *more similar to* Databases *than to* Software Engineering. *However, in Figure 1(b), the same user may use the pattern* $p_2 : \texttt{field} \cdot \texttt{field}^{-}$ *to compute similarity scores between research areas. This pattern, however, finds that both* Software Engineering *and* Databases *are equally similar to* Data Mining.

4.2 Achieving Robustness

Example 5 illustrates that there may *not* be any (obvious) pattern over some structure or schema of a dataset to express the desired relationship between entities. If a user wants to find the similarity of two research areas based on their shared conferences, she can use pattern p_2 over the structural representation in Figure 1(b), but she cannot find the same pattern in Figure 1(a). One may propose a candidate pattern p_1 over Figure 1(b). However, it includes more information than p_2, i.e., information about the set of papers published in the conferences can be captured by p_1, but not by p_2.

On the other hand, the user may like to measure the similarity of research areas based on their shared conferences and also the papers published in those conferences. Over Figure 1(a), she can use pattern p_1 to help measure the similarity. However, there is no pattern that expresses that relationship in Figure 1(b).

One may solve this problem by using a language that is more expressive than the sequence of relationship labels to express relationship types between entities in a database.

Example 6. *Following Example 5, one can create an equivalent relationship to the one expressed by* p_2 *in Figure 1(a) by modifying* p_1 *to treat the set of all paths through some papers from a conference to a research area as a single path, i.e, skip details of entities visited along those paths.*

The resulting pattern from Example 6 considers only the existence of a connection between a research area and a conference in the database as opposed to p_1 that takes into account all papers that connect a research area to a conference. This pattern intuitively represents an equivalent relationship over Figure 1(a) to the one conveyed by p_2 over Figure 1(b). Hence, one has to define and add an operation that implements the aforementioned skipping behavior to the language that describes relationships between entities.

Example 7. *Following Example 5, one can modify pattern* p_2 *such that the a that follows the pattern can visit the publications of conference while visiting a conference. This way, we will get a relationship between two research areas that takes into consideration both the conferences and publications shared between them in Figure 1(b). The resulting pattern expresses an equivalent relationship to what* p_1 *represents over Figure 1(a).*

However, even following this approach, one should be careful not to increase the expressivity of the relationship language too much as it takes a long time to find all instances of a complex pattern and compute its similarity score in a large database.

We present a relationship expressing language that is expressive enough to represent equivalent relationships across various representations of the same datasets. We also show that using this relationship language, there is a similarity algorithm that returns equal similarity scores between every pair of corresponding entities over different representations of the same information. More precisely, an algorithm that computes a similarity score using Equation 1 where p is written in our proposed language is structurally robust.

To implement the solution presented in Example 7, one may use the idea of nested operation in the nested regular expression (NRE) language [7]. Let $[p]$ denote a nested path of p where a path $(u, [p], u)$ exists if and only if there exists a node v such that a path (u, p, v) exists. To achieve the same results as p_1 over Figure 1(a), the user should use the pattern p_4 : field$\cdot$ [published-in$^-$] $\cdot$ [published-in$^-$] $\cdot$ field$^-$ to compute a similarity score between research areas. That is, similar research areas are based on shared conferences, and the strength of this relationship is based on the number of publications published in those conferences.

Next, We define the extension to NRE namely *rich-relationship expression* (RRE), over schema S as

$$p := \epsilon \mid a\ (a \in S) \mid p^- \mid p^* \mid p \cdot p \mid p + p \mid [p] \mid \lceil\lceil p \rfloor\rfloor$$

where [ ] denotes a nested operation and $\lceil\lceil\ \rfloor\rfloor$ denotes a skip operation.

Since Equation 1 used the number of instances of a specified pattern when calculating the similarity score, we define an *instance* of an RRE as follows. An instance of some RRE in a graph database D is a ternary relation (u, v, s) representing a traversal over D from node u to node v whose actual traversal are recorded in a sequence s. Each entry in the recorded sequence s is either a node id, an edge label or a string of pattern. Equivalence between two RRE instances is defined naturally by entry-wise comparison.

Given a sequence $s = \langle s_1, ..., s_m\rangle$ and $t = \langle t_1, ..., t_n\rangle$ of m and n entries, respectively, let $s \bullet t = \langle s_1, ..., s_m, t_2, ..., t_n\rangle$ which is defined only if $s_m = t_1$; and let $\bar{s} = \langle \grave{s_m}, ..., \grave{s_1}\rangle$ where, for each $i = 1...m$, $\grave{s_i} = s_i$ if s_i represents a node and $\grave{s_i} = s_i^-$ otherwise. A set of instances of an RRE p in a database D in schema S, denoted by $\mathcal{I}_D(p)$, is defined as follows. For a given label $a \in S$, arbitrary RREs p, p_1 and p_2 over S, we have

$$\begin{aligned}
\mathcal{I}_D(\epsilon) &= \{(u, u, \langle u\rangle) \mid u \text{ is a node in } D\}\\
\mathcal{I}_D(a) &= \{(u, v, \langle u, a, v\rangle) \mid (u, a, v) \in D\}\\
\mathcal{I}_D(p^-) &= \{(v, u, \bar{s}) \mid (u, v, s) \in \mathcal{I}_D(p)\}\\
\mathcal{I}_D(p_1 \cdot p_2) &= \{(u, v, s_1 \bullet s_2) \mid \forall w, (u, w, s_1) \in \mathcal{I}_D(p_1)\\
&\qquad \text{and } (w, v, s_2) \in \mathcal{I}_D(p_2)\}\\
\mathcal{I}_D(p_1 + p_2) &= \{(u, v, s) \mid (u, v, s) \in \mathcal{I}_D(p_1) \cup \mathcal{I}_D(p_2)\}\\
\mathcal{I}_D(p^*) &= \{(u, v, s) \mid (u, v, s) \in \mathcal{I}_D(\epsilon) \cup \mathcal{I}_D(p) \cup \mathcal{I}_D(p^2) \cup ...\}\\
\mathcal{I}_D(\lceil\lceil p \rfloor\rfloor) &= \{(u, v, \langle u, \widetilde{p}, v\rangle) \mid \exists s, (u, v, s) \in \mathcal{I}_D(p)\}\\
\mathcal{I}_D([p]) &= \{(u, u, s \bullet \langle v, u\rangle) \mid \forall v, (u, v, s) \in \mathcal{I}_D(p)\}
\end{aligned}$$

where $\widetilde{p}$ is a string p with all $\lceil\lceil\ \rfloor\rfloor$ removed, e.g., $\widetilde{\lceil\lceil a \cdot b \rfloor\rfloor} = a \cdot b$. We define the definition of instances of an RRE for a particular pair of nodes u and v in database D such that

$$\mathcal{I}_D^{u,v}(p) = \{(u, v, s) \mid \forall s, (u, v, s) \in \mathcal{I}_D(p)\}.$$

If database D is clear from the context, we may write $\mathcal{I}_D^{u,v}(p)$ and $\mathcal{I}_D(p)$ simply as $\mathcal{I}^{u,v}(p)$ and $\mathcal{I}(p)$, respectively. For the remaining of this paper, we assume all relationship patterns are RREs.

Proposition 3. *Given a schema S, $a \in S$, p, p_1 and p_2 are arbitrary RREs over S, and a database $D \in$ Inst(S) where nodes u and v are in D, the following properties hold.*

(1) If $\mathcal{I}_D^{u,v}(p) \neq \emptyset$, then $|\mathcal{I}_D^{u,v}(\lceil\lceil p \rfloor\rfloor)| = 1$. Otherwise, $|\mathcal{I}_D^{u,v}(\lceil\lceil p \rfloor\rfloor)| = 0$.
(2) $\mathcal{I}_D^{u,v}(\lceil\lceil a \rfloor\rfloor) = \mathcal{I}_D^{u,v}(a)$
(3) $|\mathcal{I}_D^{u,v}(p_1 \cdot p_2)| = \sum_{w \in D} |\mathcal{I}_D^{u,w}(p_1)||\mathcal{I}_D^{w,v}(p_2)|$
(4) If $(u, p_1, v) \in D$ iff $(u, p_2, v) \in D$, then $|\mathcal{I}_D^{u,v}(\lceil\lceil p_1 \rfloor\rfloor)| = |\mathcal{I}_D^{u,v}(\lceil\lceil p_2 \rfloor\rfloor)|$.
(5) $|\mathcal{I}_D^{u,u}([p])| = |\mathcal{I}_D^{u,u}(p \cdot \lceil\lceil p^- \rfloor\rfloor)|$

Given a transformation $\gamma : \phi(\bar{x}) \rightarrow (x_1, a, x_2)$, one can construct an undirected graph $G_\gamma = (V, E)$ such that $V = \{\bar{x}\}$ and E is a set of an edge (x_i, p, x_j) where (x_i, p, x_j) is an atom in $\phi(\bar{x})$. We say that γ is acyclic if G_γ contains *no* cycle. In the following theorem, we assume that the premise of every transformations are acyclic. Using Proposition 3, we prove the following theorem by induction on the number of disjunctions and concatenated labels in a given RRE pattern.

Theorem 2. *Given schemas S and T, for every invertible transformation Σ_{ST} and every pattern p over S, there exists a pattern p' over T such that, for every database $D \in$ Inst(S) and $J \in \Sigma_{ST}(D)$ $\forall u, v \in D$, $|\mathcal{I}_D^{u,v}(p)| = |\mathcal{I}_J^{u,v}(p')|$.*

Proof. For (1) and (2), the statements hold directly from definitions of path instances. For (3), proofs are done by counting. For (4), assume $\exists (u, p_1, v) \in D$. We have $(u, p_1, v) \in D$ iff $(u, p_2, v) \in D$, and so $\mathcal{I}_D^{u,v}(p_1) \neq \emptyset$ iff $\mathcal{I}_D^{u,v}(p_2) \neq \emptyset$. That is, $|\mathcal{I}_D^{u,v}(\lceil\lceil p_1 \rfloor\rfloor)| = 1$ iff $|\mathcal{I}_D^{u,v}(\lceil\lceil p_2 \rfloor\rfloor)| = 1$. Otherwise, $\mathcal{I}_D^{u,v}(p_1) = \mathcal{I}_D^{u,v}(p_2) = \emptyset$, and so $|\mathcal{I}_D^{u,v}(\lceil\lceil p_1 \rfloor\rfloor)| = |\mathcal{I}_D^{u,v}(\lceil\lceil p_2 \rfloor\rfloor)| = 0$. For (5), by definitions, $(u, p, v) \in D$ iff $(u, \widetilde{p}, v)$ iff $(v, \widetilde{p^-}, u)$. Hence, $|\mathcal{I}_D^{u,u}([p])| = |\{(u, u, s \bullet \langle v, u\rangle) \mid \forall v, (u, v, s) \in \mathcal{I}_D(p)\}| = |\{(u, u, s \bullet \langle v, \widetilde{p^-}, u\rangle) \mid \forall v, (u, v, s) \in \mathcal{I}_D(p)\}| = |\{(u, u, s \bullet \langle v, \widetilde{p^-}, u\rangle) \mid \forall v, (u, v, s) \in \mathcal{I}_D(p)$ and $(v, u, \langle v, \widetilde{p^-}, u\rangle) \in \mathcal{I}_D(\lceil\lceil p^- \rfloor\rfloor)\}| = |\mathcal{I}_D^{u,u}(p \cdot \lceil\lceil p^- \rfloor\rfloor)|$. □

Then, we prove the theorem.

Proof. Assume each node in a graph database has node ids; and those same ids always exist in any solution under any invertible schema mapping.

If every label in the pattern p exists in both schemas S and T, we have that, for each label $a \in S$ appearing in p, $\forall u', v' \in D$, $(u', a, v') \in D$ iff $(u', a, v') \in \Sigma_{ST}(D)$. Clearly, $|\mathcal{I}_D^{u,v}(p)| = |\mathcal{I}_{\Sigma_{ST}(D)}^{u,v}(p)|$.

Suppose $p = \lceil\lceil r \rfloor\rfloor$ for some pattern r over S. By Proposition 3(4), if there exists a pattern r' over T s.t. $|\mathcal{I}_D^{u,v}(r)| > 0$ iff $|\mathcal{I}_{\Sigma_{ST}(D)}^{u,v}(r')| > 0$, we have $|\mathcal{I}_D^{u,v}(p)| = |\mathcal{I}_{\Sigma_{ST}(D)}^{u,v}(\lceil\lceil r' \rfloor\rfloor)|$. Also, by Proposition 3(5), one may write $p \cdot \lceil\lceil p^- \rfloor\rfloor$ instead of $[p]$. Hence, we may consider a pattern p without any use of $\lceil\lceil\ \rfloor\rfloor$ or [ ]. Further, since $\mathcal{I}(p^*) = \mathcal{I}(\epsilon) \cup \mathcal{I}(p) \cup \mathcal{I}(p^2) \cup ...$, if there exists p' such that $|\mathcal{I}_D^{u,v}(p)| = |\mathcal{I}_{\Sigma_{ST}(D)}^{u,v}(p')|$, then $|\mathcal{I}_D^{u,v}(p^*)| = |\mathcal{I}_{\Sigma_{ST}(D)}^{u,v}(p'^*)|$. Hence, we may also consider a pattern p without the use of $*$.

Assume $p = p_1 + ... + p_m$ where $p_1, ..., p_m$ are distinct and contain no '+'. That is $\mathcal{I}_D(p_i) \cap \mathcal{I}_D(p_j) = \emptyset$ for any $i \neq j$. Clearly, $|\mathcal{I}_D(p)| = |\mathcal{I}_D(p_1)| + ... + |\mathcal{I}_D(p_m)|$.

We first show that, for each $i = 1...m$, there exists a pattern p_i' over T s.t. $\forall u, v \in D$, $|\mathcal{I}_D^{u,v}(p_i)| = |\mathcal{I}_{\Sigma_{ST}(D)}^{u,v}(p_i')|$ using strong induction over the number of concatenations in p_i.

Clearly, if $p_i = a$ or $p_i = a^-$ where $a \in S$ and $a \in T$, then the statement holds. Otherwise, since Σ_{ST} is information preserving, there exists a transformation rule in its inverse s.t. $\phi(x_1, x_2, \bar{x}) \rightarrow (x_1, a, x_2)$. One can then construct a pattern p_i' that traverses $\phi(\bar{x})$ from x_1 to x_2. We have that, $\forall u, v \in D$, $(u, p_i, v) = (u, a, v) \in D$ iff $\phi(u, v, \bar{x})$ iff $(u, p_i', v) \in \Sigma_{ST}(D)$. Let $p_i'' = \lceil \lceil p_i' \rfloor \rfloor$. By Proposition 3(4), $|\mathcal{I}_D^{u,v}(p_i)| = |\mathcal{I}_{\Sigma_{ST}(D)}^{u,v}(p_i'')|$. The proof extends for the case where $p = a^-$.

Suppose the statement holds for any p_i that contains up to k concatenations. Without losing generality, let $p_i = p_{i,1} \cdot p_{i,2}$, for some $p_{i,1}, p_{i,2} \neq \epsilon$, containing $k+1$ concatenations. Hence, $p_{i,1}$ and $p_{i,2}$ contain at most k concatenations. Consider that, $\forall u, v, w \in D$, there exists r_{i1} and r_{i2} in $T(D)$ s.t. $|\mathcal{I}_D^{u,w}(p_{i1})| = |\mathcal{I}_{\Sigma_{ST}(D)}^{u,w}(r_{i1})|$ and $|\mathcal{I}_D^{w,v}(p_{i2})| = |\mathcal{I}_{\Sigma_{ST}(D)}^{w,v}(r_{i2})|$.
Thus $|\mathcal{I}_D^{u,v}(p)| = \sum_{w \in D} |\mathcal{I}_D^{u,w}(p_{i1})||\mathcal{I}_D^{w,v}(p_{i2})|$
$= \sum_{w \in \Sigma_{ST}(D)} |\mathcal{I}_{T(D)}^{u,w}(r_{i1})||\mathcal{I}_{\Sigma_{ST}(D)}^{w,v}(r_{i2})| = |\mathcal{I}_{\Sigma_{ST}(D)}^{u,v}(r_{i1} \cdot r_{i2})|$.
That is, $p_i'' = r_{i1} \cdot r_{i2}$ satisfies the claim.

Using an induction over the number of disjunction over p, we have that there exists a pattern $p' = p_1' + ... + p_k'$ s.t. the theorem holds. □

We restrict our attention to transformations with acyclic premises in order to reduce the expressivity of the relationship language and keep the computation of similarity scores efficient. A cyclic premise allows multiple traversals from one variable to another in the premise, and requires an indicator in the relationship language whether two variables along the traversal are the same, e.g., starting and ending nodes in a cycle are the same. In this case, it is *not* possible to rewrite this pattern over the premise to an equivalent one without a conjunction ($\wedge$). For instance, consider the pattern representing the premise of a cyclic tgd $(x_1, \mathsf{a}, x_2) \wedge (x_2, \mathsf{b}, x_3) \wedge (x_3, \mathsf{c}, x_4) \wedge (x_1, \mathsf{d}, x_3) \wedge (x_2, \mathsf{e}, x_4) \rightarrow (x_1, \mathsf{f}, x_4)$. Assume we want to rewrite the pattern over this premise similar to (x_1, exp, x_4) for some RRE exp. To remove the conjunction in $(x_1, \mathsf{a}, x_2) \wedge (x_2, \mathsf{b}, x_3)$, one may write $(x_1, \mathsf{a} \cdot \mathsf{b}, x_3)$. However, because x_2 is specified in (x_2, e, x_4), x_2 cannot be removed, and so this conjunction is necessary. Hence, the language to properly express this relationship pattern should be a conjunctive RRE. Since conjunctive RRE is more complex, it will take longer to compute the similarity scores between nodes. Regardless, the result of Theorem 2 extends for general tgd constraints if a conjunction is added to our proposed relationship expression language.

The following is an immediate result of Theorem 2.

Corollary 1. *Given a database D of a schema S, for every transformation Σ_{ST} for some schema T, there is a mapping M between the set of patterns over S and the set of patterns over T such that, for a given pattern p over S, we have that* $\forall D \in \mathtt{Inst}(S)$, $\forall u, v \in D$, $\mathtt{sim}_p(u, v, D) = \mathtt{sim}_{M(p)}(u, v, \Sigma_{ST}(D))$.

Corollary 1 guarantees that, for each pair of entities u and v and pattern p between them over a dataset, one can always find a equivalent pattern with equal similarity score to p between u and v on other variations of the database. Hence, the returned ranked list of answers to a similarity query across databases under this transformation are always the same. We call the algorithm that uses Equation 1 to compute similarity on RRE patterns *Relationship-Similarity* (RelSim).

One may extend RWR or SimRank so that the similarity measurement is based on a particular relationship pattern between entities [43]. RWR computes a similarity score between nodes u and v in a dataset using the steady-state probability that a random walk from u will stay at v. SimRank, on the other hand, computes the score based on the probability that two random walks from u and v are met at a vertex in the data graph. Technically, the probability of a random walk from u to v computes the chance that a walk from u *hop*s from a node to its neighbor repeatedly until reaching v. Each hop, hence, is intuitively defined as a single edge between two nodes. In this extended RWR or SimRank, given a relationship pattern, a hop is defined only if a walk follows the given pattern from one node to another node. Following this idea, we can use the same measurement as SimRank and RWR to compute similarity scores over a relationship pattern as similarly specified in RelSim. Using a similar proof to Theorem 2, we prove the following proposition. Let $\mathtt{RWR}_p(u, v, D)$ and $\mathtt{SimRank}_p(u, v, D)$ denote a similarity score between nodes u and v computed using RWR and SimRank scoring function that only considers walks that follows RRE p.

Proposition 4. *Given a database instance D of a schema S, for every transformation Σ_{ST} for some schema T, there is a mapping M between a set of patterns over S and a set of patterns over T such that, for a given pattern p over S, we have* $\forall D \in \mathtt{Inst}(S)$, $\forall u, v \in D$, $\mathtt{RWR}_p(u, v, D) = \mathtt{RWR}_{M(p)}(u, v, \Sigma_{ST}(D))$ *and* $\mathtt{SimRank}_p(u, v, D) = \mathtt{SimRank}_{M(p)}(u, v, \Sigma_{ST}(D))$.

Proof. The proof is similar to the one of Theorem 2. Specifically, RWR and SimRank assume that the weight of connectivity between two nodes in a data graph depends on the number of instances of the relationship pattern in the database. The weight matrices are then used to compute similarity score between two nodes. □

PathSim is shown to be more effective than RWR and SimRank [43]. Thus, we focus on our extension of PathSim.

4.3 Computing Similarity Scores

For a pattern with only concatenations, the number of RRE instances can be computed using *commuting matrix* [43]. Given labels $l_1, ..., l_m$ in a schema S, a commuting matrix of pattern $p = l_1 \cdot ... \cdot l_m$ over database D is $\mathbf{M}_p = \mathbf{A}_{l_1}\mathbf{A}_{l_2}...\mathbf{A}_{l_m}$ where $\mathbf{A}_{l_i}$ is an adjacency matrix that represents a number of edges of label l_i between pairs of nodes in D. Each entry $\mathbf{M}_p(u, v)$ represents the number of instances of p from node u to node v in D. Given a commuting matrix, we can compute a similarity score $\mathtt{sim}_p(u, v, D)$ as $\frac{2\mathbf{M}_p(u,v)}{\mathbf{M}_p(u,u)+\mathbf{M}_p(v,v)}$ [43].

We extend the computation of commuting matrix for RREs as follows. Given matrices $\mathbf{X}$ and $\mathbf{Y}$, let $>$ be a boolean operation such that each entry (i, j) of $\mathbf{X} > \mathbf{Y}$ is 1 if $\mathbf{X}(i, j) > \mathbf{Y}(i, j)$ or 0 otherwise, and $\mathbf{diag}\{\mathbf{X}\}$ denote a diagonal matrix of $\mathbf{X}$. Given a label a and

Algorithm 1: PatternGenerator

Input: schema $S = (\mathcal{L}, \Gamma)$, simple pattern $p = l_1 ... l_n$ over S
Output: subset $\mathcal{E}_p$ of RREs over S

1 $done \leftarrow \{\}$;
2 $processing \leftarrow \{(\epsilon, 0)\}$; // For a pair $(r, i) \in processing$, r is an RRE r processed up to the position of label l_i in p
3 **foreach** $(r, i) \in processing$ **do**
4 Remove (r, i) from $processing$;
5 **if** $i \geq n$ **then**
6 Add r to $done$; **continue**;
7 Add $(r \cdot l_{i+1}, i+1)$ to $processing$; // Use original pattern
8 // Modify each sub-pattern of $s = l_{i+1} \cdot l_{i+2} \cdot ... l_n$
9 **foreach** $\gamma \in \Gamma$ **do**
10 $\mathcal{R} \leftarrow \mathcal{R} \cup$ ModPatternRefsPerConstraint(γ, s);
11 **foreach** $j \geq i+1$ **do**
12 **foreach** $(l_{i+1} \cdot l_{i+2} \cdot ... \cdot l_j, e') \in \mathcal{R}$ **do**
13 Add $(r \cdot e', j)$ to $processing$;
14 **return** $\mathcal{E}_p \leftarrow done$;

Algorithm 2: ModPatternRefsPerConstraint

Input: constraint γ, simple pattern $s = l'_1 \cdot ... \cdot l'_m$
Output: set $\mathcal{R} = \{(e, e')\}$ where e' is a corresponding RRE to e which is a sub-pattern of s

1 $G_\gamma \leftarrow$ the premise graph representing γ;
2 **foreach** $i > 0, j \geq i, j \leq m$ **do**
3 $e \leftarrow l'_i \cdot l'_{i+1} \cdot ... \cdot l'_j$;
4 **if** a path e from some v_g to v_h exists in $G_{pre}(\gamma)$ **then**
5 **foreach** connected subgraph H of $G_{pre}(\gamma)$ **do**
6 Find all RREs $e' : v_g \hookrightarrow_H v_h$ that traverse H from v_g to v_h and visit each edge of H once;
7 Add (e, e') and (e^-, e'^-) to $\mathcal{R}$;
8 **return** $\mathcal{R}$

arbitrary patterns p, p_1 and p_2 over database D in schema S, we have $\mathbf{M}_a = \mathbf{A}_a$, $\mathbf{M}_{p^-} = \mathbf{M}_p^T$, $\mathbf{M}_{p_1 \cdot p_2} = \mathbf{A}_{p_1}\mathbf{A}_{p_2}$, $\mathbf{M}_{p_1+p_2} = \mathbf{A}_{p_1} + \mathbf{A}_{p_2}$ if $p_1 \neq p_2$, $\mathbf{M}_{p_1+p_2} = \mathbf{A}_{p_1} = \mathbf{A}_{p_2}$ if $p_1 = p_2$, $\mathbf{M}_{\lceil\lceil p \rfloor\rfloor} = \mathbf{M}_p > \mathbf{0}$, and $\mathbf{M}_{[p]} = \mathbf{diag}\{\mathbf{M}_p(\mathbf{M}_p^T > \mathbf{0})\}$ where $\mathbf{0}$ denotes a matrix whose entries are zero.

Since computing a commuting matrix for RRE p over database D follows standard matrix operations, the complexity is bounded by $O(m|V|^3 + n|V|^2)$ where m denotes the number of matrix multiplications, e.g., the number of concatenations and nested operations in p, n denotes the number of other operations, and $|V|$ denotes the number of nodes in D. Therefore, RelSim still has the same complexity as that of PathSim.

5 SIMPLIFYING RELSIM

The relationship expression language presented in Section 4 may be too complicated for average users. For instance, p_4 : field · [pubslihed-in$^-$]· [pubslihed-in$^-$]· field$^-$, which involves nested operations, is less intuitive than p_2 : field·field$^-$ over Figure 1(b), which uses only concatenations. To improve the usability of a similarity search algorithm, we would like to enable users to submit their patterns using a relatively intuitive set of operations, such as concatenations and reverse traversals. Additionally, users may like to measure the similarity between entities using a set of different types of relationships to get a more holistic and robust view of the similarity between entities [43]. For example, users may want to use both p_2 and p_4 to compute the similarity between research areas in Figure 1(b).

Thus, we propose a robust and effective algorithm whose input is a pattern that may contain only concatenation and reverse traversal operations, i.e., *simple pattern*. Given a simple input pattern, our algorithm leverages the constraints in the database to generate a set of RRE patterns *related* to the input pattern. The algorithm aggregates the similarity scores of these patterns to compute the similarity between entities.

Algorithm 1 finds a set $\mathcal{E}_p$ of RREs by minimally modifying the input simple pattern p such that the results of Corollary 1 and Proposition 4 hold for the aggregated scores of all patterns in $\mathcal{E}_p$. For example, given an input p_2 : field· field$^-$ over Figure 1(b), the algorithm returns a set of RREs whose members are including both p_2 and p_4 : field· [pub-in$^-$] · [pub-in$^-$] · field$^-$. Our method computes and aggregates the similarity scores of pattern in the set of RREs returned by Algorithm 1 using Equation 1. If $p_5 : p_5 :$ area · pub_in · pub_in$^-$ · area$^-$ is the input of Algorithm 1 over Figure 1(a), the algorithm returns a set of RREs whose members are p_5 and p_6 : $\lceil\lceil$area · pub_in$\rfloor\rfloor$ · $\lceil\lceil$pub_in$^-$ · area$^-\rfloor\rfloor$. According to the mapping defined in Corollary 1 between two data fragments in Figures 1(b) and Figure 1(a), p_2 and p_4 map to p_6 and p_5, respectively. Thus, there is a one-to-one mapping between the sets of RRE returned by the algorithm over data fragments. By computing the scores through counting, one finds that the similarity results for p_2 over Figure 1(b) and p_5 over Figure 1(a) are the equal.

More precisely, let Algorithm 1 take a simple pattern $p = l_1 \cdot ... \cdot l_n$ over a schema $S = (\mathcal{L}, \Gamma)$. Let (r, i) denote a generated RRE r by Algorithm 1 from p, which is processed up to label l_i in p. Given (r, i), let s be the the remaining unprocessed sub-pattern of p, e.g., $s = l_{i+1} \cdot ... \cdot l_n$. Algorithm 1 examines each sub-pattern $e : l_{i+1} \cdot l_j$ of s for some $i + 1 \leq j \leq n$. Then, it uses Algorithm 2 to find a set $\mathcal{R}$ of RREs for e according to each constraint in S (Line 10). If Algorithm 2 generates RRE e' for e, Algorithm 1 replaces e in s with e' and marks that all labels up to l_j as processed, e.g., $(r \cdot e', j)$ (Line 13). It also includes the input pattern p in the set (Line 7).

Before we explain Algorithm 2, we define the *premise graph* of a constraint. Consider a constraint $\gamma : \phi_\gamma(\bar{x}) \rightarrow \psi_\gamma(\bar{x})$. We assume that for each atom (x_i, e, x_j) in ϕ_γ, e is *not* written as $e_1 \cdot e_2$ for some non-empty RPQs e_1 and e_2. If such atom exists, we rewrite it as $(x_i, e_1, x') \wedge (x', e_2, x_j)$ for some fresh variable x'. The *premise graph* of γ, denoted by $G_{pre}(\gamma)$, is a directed graph (V, E) whose nodes are variables in ϕ_γ and edges are labeled by the RPQ patterns between each pair of those variables in ϕ_γ. More precisely, a node $v_{x_i} \in V$ if and only if x_i is a variable in ϕ_γ, and an edge $(v_{x_i}, e, v_{x_j}) \in E$ if and only if (x_i, e, x_j) is in ϕ_γ. The premise graph of constraint $\gamma_1 : (x_1, \text{area}, x_3) \wedge (x_3, \text{pub-in}, x_4) \wedge (x_2, \text{pub-in}, x_4) \rightarrow (x_1, \text{area}, x_2)$ over Figure 1(a) is $v_1 \xrightarrow{\text{area}} v_3 \xrightarrow{\text{pub-in}} v_4 \xleftarrow{\text{pub-in}} v_2$.

The underlying idea of the Algorithm 2 is that each constraint γ implies information-preserving transformations that may add or remove an edge from the current schema. These transformations modify only the (simple) patterns in the underlying database that match the premise of γ. Intuitively, using the results of Section 4.2, these variations are confined within RREs. Thus, for each input simple pattern s and constraint γ, generally speaking, Algorithm 2 finds all common (sub-)paths between s and the premise graph of γ. Let e be such a common path that starts from v_g and ends at v_h in the premise graph of γ. Algorithm 2 generates all patterns in the premise graph of γ that start from v_g and end at v_h and are expressed as RREs, i.e., all traverses from v_g and end at v_h in the premise graph of γ that are expressed as RREs.

We briefly describe the recursive procedure to compute an RRE $v_s \hookrightarrow_G v_t$ that traverses a premise graph G from node v_s to v_t in Algorithm 2. We adopt a breath-first search algorithm to find all paths, i.e. simple patterns, from node v_i to node v_j in G. An RRE pattern of all n paths that traverses a pair of nodes v_i and v_j is $p_1^{i,j} + ... + p_n^{i,j}$. Since we assume the constraints and consequently their premise graph to be acyclic, n is exactly 1. Let us denote this pattern as $p^{i,j}$. At each node v_i which connects to some leaf node v_k in G, we construct a pattern $[p^{i,k}]$. Then, we concatenate $[p^{i,k}]$ at the front of any pattern from v_i or at the end of any pattern to v_i. We mark each edge as visited when each pattern $p^{i,j}$ or $[p^{i,k}]$ is constructed. The base case is to first construct a pattern $p^{s,t}$. The procedure ends when all edges in G are visited. We must note that each constructed $p^{i,j}$ can also be written as $\lceil\lceil p^{i,j} \rfloor\rfloor$, which results in multiple patterns of this traversal.

As an example, consider the premise graph of constraint γ_1, $G_{pre}(\gamma_1) : v_1 \xrightarrow{\texttt{area}} v_3 \xrightarrow{\texttt{pub-in}} v_4 \xleftarrow{\texttt{pub-in}} v_2$, and a simple pattern $\texttt{area} \cdot \texttt{pub-in}$. Possible RRE patterns that traverse this graph from v_1 to v_4, i.e., $v_1 \hookrightarrow_{H \subseteq G_{pre}(\gamma)} v_3$, are $\texttt{a} \cdot \texttt{p}$, $\lceil\lceil \texttt{a} \cdot \texttt{p} \rfloor\rfloor$, $\texttt{a} \cdot \texttt{p} \cdot [\texttt{p}^-]$ and $\lceil\lceil \texttt{a} \cdot \texttt{p} \rfloor\rfloor \cdot [\texttt{p}^-]$. Algorithm 2 adds all $(\texttt{a} \cdot \texttt{b}, e')$ to the resulting set $\mathcal{R}$ where e' is one of the above patterns except the first one.

The complexity of Algorithm 1 largely depends on the number of patterns generated in Algorithm 2. Since each simple pattern p found in G_γ can be either p or $\lceil\lceil p \rfloor\rfloor$, this algorithm is exponential in the number of p's. However, since G_γ is acyclic, each p is a path. Hence, the number of simple pattern p's is $1 + \sum_{v \in V_{\deg>2}} (\deg(v) - 1)$ where $V_{\deg>2}$ is a set of nodes in G whose degrees are greater than 2. Since $\sum_{v \in V(G)} \deg(v) = 2|E(G)|$, the procedure is $O(\exp(|E(G)|))$. There is also a linear-time algorithm $O(V(G)$ that finds all connected subgraphs G of G_γ [27]. Since the number of iterations over all sub-patterns (Line 2) is polynomial in the length of p, the total complexity of Algorithm 2 is $O((n^2)(\exp(|E(G)|) + |V(G)|))$ where n is the number of nodes in the input simple pattern. Constraints of a schema are usually have a very small number of terms compared to the size of databases. We may view the exponent factor as a (small) constant. Hence, Algorithm 2 is $O(n^2)$ where n is usually small [43]. Let $O(M)$ denote the number of patterns generated from Algorithm 2. Consider that there are possible $O(\exp(k))$ possible sub-patterns to the user input pattern with k nodes. Also, algorithm 1 may replace each of those sub-patterns by the generated patterns from Algorithm 2. Hence, the complexity of Algorithm 1 is $O(M^{\exp(k)})$. The value of k, i.e., the number of nodes in the submitted simple pattern is also significantly smaller than the size of the database. We will discuss the optimizations to improve the running time of Algorithm 2 in Section 6.

Consider each label l in the input simple pattern. Algorithm 2 finds all possible patterns according to the set of constraints that is mapped to each label l in the input pattern and follows Theorem 2. Using similar arguments to Theorem 2 and Corollary 1, we have the following proposition.

Proposition 5. *Given a database D of schema S and a equivalent schema T under transformation Σ, for every simple pattern p_S over S, there exists a simple pattern p_T over T such that, $\forall u, v \in D$, $\sum_{p \in \mathcal{E}_{p_S}} \texttt{sim}_p(u, v, D) = \sum_{p' \in \mathcal{E}_{p_T}} \texttt{sim}_{p'}(u, v, \Sigma(D))$*

Proof. Let $f_{crpq}(v_g \hookrightarrow_G v_h)$ denote a CRPQ representing an RRE $exp' : v_g \hookrightarrow_G v_h$ in Algorithm 2. Consider a transformation $\gamma : f_{crpq}(v_g \hookrightarrow_G v_h) \rightarrow (x_g, l, x_h)$ for some variables x_g and x_h corresponding to v_g and v_h, respectively, and some label $l \notin S$. Clearly, γ is information preserving, and $\lceil\lceil v_g \hookrightarrow_G v_h \rfloor\rfloor$ is mapped to l according to Theorem 2. Similarly, there exists an RRE exp'' over T that maps to l for some Σ'. By transitivity, exp' is mapped to exp'' according to Theorem 2. We must note that γ always exists, and Algorithm 2 yields all possible such Σ''s. Hence, using similar arguments to Theorem 2 and Corollary 1, we have that our proposition holds. □

Thus, a similarity algorithm that computes aggregate scores over the set of RREs returned by Algorithm 1 is structurally robust.

6 OPTIMIZING PATTERN GENERATION

We have simplified the use of our framework in Section 5 so that users can take advantage of it by submitting only simple patterns. Using database constraints, our algorithm finds a set of relationship patterns related to the input pattern and use them to compute an aggregated similarity score. If the set of such RREs patterns is too large, our system has to compute the similarity scores for many patterns, therefore, it may *not* be efficient on a large database. In this section, we show that the algorithms proposed in Section 5 can avoid generating and computing the similarity scores of many such patterns without losing their robustness or effectiveness.

According to our discussion in Section 3, to have an invertible variation of a database I, I must satisfy some tgd constraints. However, these constraints may be *trivial* , e.g., $(x, a, y) \rightarrow (x, a, y)$. Obviously, it is *not* efficient to consider all trivial constraints over a database in the algorithms proposed to simplify our system. We show that a structural variation actually needs I to have non-trivial constrains. Thus, as far as our proposed algorithms use non-trivial constraints, it is structurally robust. We also show that a simple pattern will be restructured based on a tgd only if at least one of its labels appear in both left- and right hand-side of the tgd. This enables Algorithm 1 to ignore many tgds for a given input pattern. These optimizations reduce the number of patterns need to be generated by Algorithm 1 and significantly improve its running time and the running time of its similarity search.

6.1 Avoiding Trivial Constraints

Our proposed Algorithms 1 and 2 indeed require a non-empty set of a schema constraints. According to Proposition 2, structural variations require database constraints. However, the proposition

does not exclude the use of trivial constraints. Intuitively, a trivial constraint is a constraint such that its premise and its conclusion are logically equivalent, e.g., $\phi(\bar{x}) \rightarrow \phi(\bar{x})$. Clearly, every database in a schema S satisfies all trivial constraints for every label $a \in S$. In order to simplify our discussion, since trivial constraints do *not* put any restriction on a database, if a schema (database) satisfies only trivial constraints, we say that it does *not* satisfy any constraint.

In relational database, it has been proved by Hull that there is *not* any variation of a relational schema with the same information without any constraint beyond simple renaming of the scheme elements [28]. We show that this result also holds for graph databases. Given transformation Σ_{ST} and $I \in \texttt{Inst}(S)$, let $img_{\Sigma_{ST}}(I)$ be the set of databases where each is created by removing all nodes and edges constructed using existentially quantified variables in Σ_{ST} in each $J \in \Sigma_{ST}$. We denote the set of $img_{\Sigma_{ST}}(I)$ for all $I \in \texttt{Inst}(S)$ as $img_{\Sigma_{ST}}(S)$

Theorem 3. *Given two schemas $S = (\mathcal{L}_S, \Gamma_S)$ and $T = (\mathcal{L}_T, \Gamma_T)$ where Γ_S and Γ_T are empty, if there exists an invertible Σ_{ST} between S and T there is a bijection between $\mathcal{L}_S$ and the labels used in $img_{\Sigma_{ST}}(S)$.*

Proof. Suppose $S \neq T$ and there is no bijection between $\mathcal{L}_S$ and $\mathcal{L}_T$. Hence, $|S| \neq |T|$. Without losing generality, assume $|S| > |T|$. For a given set of nodes V, we have that the set $V \times \mathcal{L}_S \times V$ has the order of $|V| \times |\mathcal{L}_S| \times |V| > |V| \times |\mathcal{L}_T| \times |V|$ in which the latter is the order of $V \times \mathcal{L}_T \times V$. Let $\texttt{Inst}_V(S)$ denote a set of database instances of S whose vertex set is V. By the definition of a database, for each $D_S \in \texttt{Inst}_V(S)$, $E_{D_S} \subseteq V \times \mathcal{L}_S \times V$. Similarly, for each $D \in \texttt{Inst}_V(T)$, $E_{D_T} \subseteq V \times \mathcal{L}_T \times V$. Without any restriction in the set E_{D_S} and E_{D_T}, we have that $|\texttt{Inst}_V(S)| > |\texttt{Inst}_V(T)|$. That is, for any surjective mapping function from $\texttt{Inst}_V(S)$ to $\texttt{Inst}_V(T)$, there exists an instance of $\texttt{Inst}_V(S)$ that maps to multiple instances of $\texttt{Inst}_V(T)$. Since the statement holds for any $V \subseteq \mathcal{V}$, we have that there is no surjective mapping function from $\texttt{Inst}_V(S)$ to $\texttt{Inst}_V(T)$ that is also injective. Hence, there exists no bijection between $\texttt{Inst}(S)$ and $\texttt{Inst}(T)$. Therefore, no such Σ_{ST} exists. □

Thus, we have that for a representation variations beyond renaming, either the source schema or the target schema must contain some non-trivial constraints.

Assume that all associated constraints with a schema S are trivial. It is possible to have a non-identity transformation Σ_{ST} from S to a target schema T. For instance, consider a schema $S = \{a, b\}$ without any non-trival constraint. A transformation Σ_{ST} from S to a target schema $T = \{a, b, c\}$ described as $\{(x_1, a, x_2) \wedge (x_2, b, x_3) \rightarrow (x_1, c, x_3)$ and $(x_1, l, x_2) \rightarrow (x_1, l, x_2), l \in S\}$ is invertible. In this case, T consists of a constraint $(x_1, a, x_2) \wedge (x_2, b, x_3) \rightarrow (x_1, c, x_3)$. However, with only trivial constraints available in S, Algorithm 2 does *not* produce an RRE $\lceil\lceil a \cdot b \rfloor\rfloor$ over S which is mapped to c over T because the pattern involves two separate constraints. In fact, with only a trivial constraint, the algorithms does not modify an input pattern.

Using the result of Proposition 2, we have the following theorem regarding constraints of T.

Theorem 4. *Given two schemas $S = (\mathcal{L}_S, \Gamma_S)$ and $T = (\mathcal{L}_T, \Gamma_T)$ where $\Gamma_T = \emptyset$, if there is an invertible transformation Σ_{ST}, then there is a bijection between $L' \subseteq \mathcal{L}_S$ and $\mathcal{L}_T$ where there is no constraint in S whose conclusion contains label in L', and for each $l \in \mathcal{L}_S \setminus L'$, there exists a constraint $\lambda(\bar{x}) \rightarrow (x_1, l, x_2)$ in Γ_S where l does not appear in any atom in λ.*

Proof. It is implied by Theorem 3 that $|\mathcal{L}_S| \leq |\mathcal{L}_T|$. Further, using similar argument in the proof of Theorem 3, we have that, if there exists no $L' \subseteq \mathcal{L}_T$ such that a bijection between $\mathcal{L}_S$ and L' exists, then there is no bijection between $\texttt{Inst}(S)$ and $\texttt{Inst}(T)$.

Consider a set of instances $V \subseteq \mathcal{V}$. We have that, since there is no constraints whose conclusion contains a label in L', for each $I \in V \times L' \times V$, there exists $I' \in \texttt{Inst}(T)V$ and $I \subseteq I'$. Suppose there is no constraint $\lambda(\bar{x}) \rightarrow (x_1, l, x_2)$, for some $l \in \mathcal{L}_T \setminus L'$. For some subset $V \subseteq \mathcal{V}$, there must exist a database J containing edges of label $\mathcal{L}_T \setminus L'$ and there is no $K \in V \times L' \times V$ s.t. $K \subseteq J$. Therefore, $|\texttt{Inst}_V(T)| > |V \times L' \times V| = |\texttt{Inst}_V(S)|$. That is, a surjective mapping between $\texttt{Inst}(S)V$ and $\texttt{Inst}_V(T)$, there exists an instance in $\texttt{Inst}_V(S)$ that maps to multiple instances in $\texttt{Inst}_V(T)$. In addition, since the bijection between $\mathcal{L}_S$ and L' exists, for any $V' \neq V \subseteq \mathcal{V}$, $|\texttt{Inst}_{V'}(S)| \leq |\texttt{Inst}_{V'}(T)|$. Using similar argument to Theorem 3, we show that there is no bijective mapping between $\texttt{Inst}(S)$ and $\texttt{Inst}(T)$. Therefore, S and T are not information equivalent. □

Following Theorem 4, we have that if a schema S contains no constraint, every invertible transformation from S preserves all edges (or up-to renaming of those edges). We call this type of transformations *easy*.

Based on Theorem 4, our proposed algorithms can ignore producing any pattern over a non-trivial constraint in the form of $\phi(\bar{x}) \rightarrow (x_1, l, x_2)$ where l does *not* appear in a CRPQ ϕ. Consider a schema S whose constraint is $\phi(\bar{x}) \rightarrow (x_1, l, x_2)$, where l does not appear in ϕ. There is an invertible transformation from S to a schema $T = S \setminus \{l\}$. We have that $x_1 \hookrightarrow_{G_\phi} x_2$ exists in both S and T. Also, following the proof of Theorem 2, we have that pattern l over S is mapped to a pattern $r : \lceil\lceil x_1 \hookrightarrow_{G_\phi} x_2 \rfloor\rfloor$. However, r is not a simple pattern and might not be easily discovered by a user. If we would like to ensure the robustness of RelSim via the use of Algorithm 1, then either label l should be ignored or every l should be replaced with $x_1 \hookrightarrow_{G_\phi} x_2$ that does *not* contain a skip-operation.

6.2 Filtering Constraints Based on Their Conclusions

Intuitively, in order to modify a database structure, a transformation may add or remove edges of certain labels. Also, we have shown that a database constraint, either in source or target schema, is necessary for structural variations beyond renaming. However, not every label appearing in the constraint can be removed. Following Proposition 2, let us define a transformation induced by a constraint γ over schema S, denoted by Σ^{γ}_{ST}, as an invertible transformation Σ_{ST} whose inverse Σ^{-1}_{ST} satisfies $\Sigma^{-1}_{ST} \circ \Sigma_{ST} \equiv \gamma$. We show in Proposition 6 that, given a constraint γ, an invertible transformation induced by γ may remove only edges of a label that appears in the conclusion of γ.

Proposition 6. *Given a schema S with a constraint $\gamma : \phi_\gamma(\bar{x}) \rightarrow (x_1, a, x_2)$, for every invertible transformation Σ^{γ}_{ST} from S to a target schema T, there exists a mapping $M : S \rightarrow T$ such that $(x, l, y) \rightarrow (x, M(l), y)$, for all $l \neq a \in S$, in Σ^{γ}_{ST}.*

Proof. Let $r \neq a$ be a label in schema S. Consider $D_1, D_2 \in \texttt{Inst}(S)$ in which D_1 consists of nodes u, v and an edge (u, r, v) and D_2 consists of nodes u and v without (u, r, v). Suppose $(x, r, y) \rightarrow (x, M_l(r), y)$ does not exists in Σ^{γ}_{ST}. Then, we have that there is no edge between u and v in $\Sigma^{\gamma}_{ST}(D_1)$. That is, $\Sigma^{\gamma}_{ST}(D_1) = \Sigma^{\gamma}_{ST}(D_2)$. Since Σ^{γ}_{ST} is information preserving, then $D_1 = D_2$ which is contradiction. Hence, the proposition holds. □

Consider that each transformation rule $(x, l, y) \rightarrow (x, M(l), y)$ simply renames each edge label l to a new edge label $M(l)$ in the target schema. For simplicity of our model and analysis, we refer to $M(l)$ in the target schema simply as l and assume that this transformation rule always exists. `published-in` is an example of such label in the transformation between the two databases presented in Figure 1. We also call a transformation that preserves all edges that appears in the premise of a constraint an *easy* transformation.

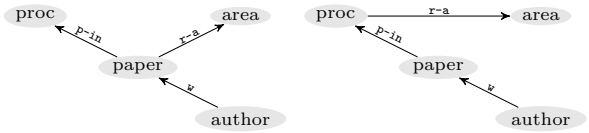
Some constraint may induce an invertible transformation that is *not* easy. For instance, a transformation between structure of databases shown in Figure 1 is *not* easy. However, not every non-easy transformation Σ is invertible unless there exists an inverse Σ^{-1} such that $\Sigma^{-1} \circ \Sigma$ is equivalent to the constraint. In this paper, we do *not* provide a procedure to determine whether a transformation is invertible. Regardless, using Propositions 6, we may conclude that non-easy transformations are induced by some constraint $\phi(\bar{x}) \rightarrow (x_1, l, x_2)$ where l appears in $\phi(\bar{x})$. That is, an RRE that does *not* contain label l is obtained from some easy transformation.

To this end, we should filter out all RREs returned by Algorithm 1 that are induced by any easy transformation. For instance, given a constraint $(x_1, \texttt{area}, x_3) \wedge (x_3, \texttt{published-in}, x_4) \wedge (x_2, \texttt{published-in}, x_4) \rightarrow (x_1, \texttt{area}, x_2)$ in Figure 1(a), the algorithm ignores producing an RRE such as `published-in`·`published-in`$^-$. However, an RRE such as `area`·`published-in` is valid because `area` appears in the conclusion of the constraint. Hence, this filtering helps reduce the space and running time of aggregate RelSim over a set of relationship patterns returned by Algorithm 1.

7 EMPIRICAL EVALUATION

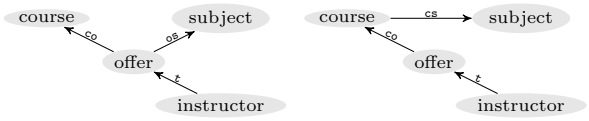
Datasets: We use 4 datasets in our experiments: DBLP, Microsoft Academic Search (MAS), WSU course dataset, and a Biomedical dataset (BioMed). *DBLP* consists of 1,227,602 nodes and 2,692,679 edges, which contains bibliographic information of publications in computer science. We add information about the research areas for each conference in DBLP from information extracted from Microsoft Academic Search (MAS). Figure 2(a) depicts the schema of DBLP. We also use a subset of Microsoft Academic Search data with 44,068 nodes and 44,220 edges. MAS contain information about papers, conferences, areas, e.g., *Databases*, and keywords of each paper and/or area, e.g., *indexing*. WSU course database[1] contains information about courses, instructors and course offerings in the university. The dataset consists of 1,124 nodes and 1,959 edges. Figure 3(a) depicts the schema of WSU dataset. The Biomedical dataset (BioMed), is made available to us as a part of an NIH funded project and contains information about genetic conditions, diseases, drugs,

[1] *cs.washington.edu/research/xmldatasets*



(a) DBLP (b) SIGMOD Record

Figure 2: Schema fragments of bibliographic databases. `p-in`, `r-a` and `w` denote edge labels `published-in`, `research-area` and `writes`, respectively.



(a) WSU (b) Alchemy UW-CSE

Figure 3: Variations for course databases. `cs`, `os`, `t` and `co` denote edge labels `course-subject`, `offering-subject`, `teach`, and `offering-course`, respectively.

and their relationships. Figure 4 depicts a fragment of BioMed. It consists of 43,307 nodes and 1,742,970 edges.

Settings: We compare robustness, effectiveness and efficiency of RelSim with RWR [47] using a restart probability of 0.8, SimRank [29] using a damping factor of 0.8, and PathSim [43]. Since previous studies show that the PathSim similarity computation method is more effective than those of SimRank and RWR [43], we use RelSim with the similarity score computation of PathSim, i.e., Equation 1. We implement all algorithms using MATLAB 8.5 on a Linux server with 64GB memory and two quad-core processors.

Result Overview: Our empirical study indicates that current algorithms are *not* robust over invertible transformations. Moreover, our proposed algorithm are also generally more robust than other algorithms over transformations in which the original database may have less information than the transformed one. This indicates that our algorithm may also tolerate losing some information during restructuring of the data better than other algorithms. Our results also indicate that our algorithms returns at least as effective or more effective results than other algorithms over real-world datasets and query workloads. Finally, we show that although RelSim uses a more expressive relationship language than PathSim, it is as efficient as PathSim and runs efficiently over large databases. We have also evaluated the efficiency of RelSim that uses simple input patterns as explained in Section 5. Our results over a large real-world database and query workload with varying number of constraints on the database and the length of input patterns suggest that RelSim is efficient over a large database with relatively large number of constraints and long input patterns.

7.1 Structural Robustness

We use DBLP, WSU and BioMed databases to evaluate the structural robustness of RWR, SimRank, PathSim and RelSim. As SimRank

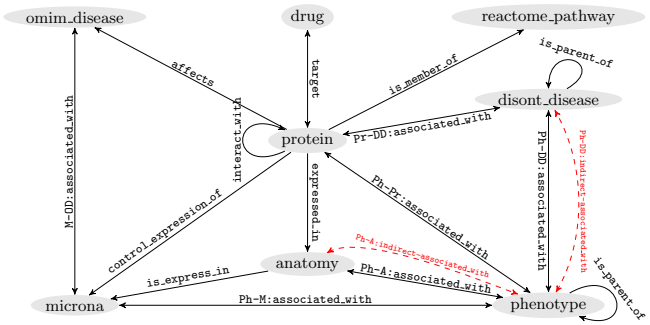


Figure 4: Schema fragment of BioMed dataset

takes too long to finish on full DBLP dataset, we do this evaluation using a subset of DBLP with 24,396 nodes and 98,731 edges.

DBLP dataset satisfies constraint $(paper_1, \texttt{r-a}, area) \wedge (paper_1, \texttt{p-in}, proc) \wedge (paper_2, \texttt{p-in}, proc) \rightarrow (paper_2, \texttt{r-a}, area)$. We transform this database to a database with the structure shown in Figure 2(b), which follows the style of SIGMOD Record database. We call this transformation DBLP2SIGM. We randomly sample 100 proceedings based on their node degrees as our query workload over these datasets.

WSU course dataset satisfies the constraint $(offer_1, \texttt{os}, subject) \wedge (offer_1, \texttt{co}, course) \wedge (offer_2, \texttt{co}, course) \rightarrow (offer_2, \texttt{os}, subject)$. We transform WSU course database to the graph structure of Alchemy UW-CSE database[2] whose structure is shown in Figure 3(b). We call this transformation WSUC2ALCH. We also randomly sample 100 courses from WSU based on their degrees as our query workload for these datasets.

BioMed dataset satisfies $(phenotype_1, \texttt{is-parent-of}, phenotype_2) \wedge (phenotype_1, \texttt{associated-with}, anatomy) \rightarrow (phenotype_2, \texttt{indirect-associated-with}, anatomy)$ and $(phenotype_1, \texttt{is-parent-of}, phenotype_2) \wedge (disease, \texttt{associ- ated-with}, phenotype_1) \rightarrow (disease, \texttt{indirect-associated-with}, phenotype_2)$. We transform the BioMed dataset such that all edges of label `indirect-associated-with` are removed. We denote the transformation over BioMed dataset as BioMedT. The structure of the transformed BioMed dataset is also shown in Figure 4 excluding all dashed edges. The main goal of using this dataset in the NIH project is to discover the drugs that are closely related to queried diseases. Since we use this dataset to also evaluate the effectiveness of our algorithms, we have obtained a set of 30 diseases and their relevant drugs from experts in the domain of the data. Since paths between diseases and drugs are asymmetric, we cannot compute similarity scores using PathSim formula over this dataset. Instead, we evaluate the queries using HeteSim [40], which extends PathSim to support asymmetric paths, e.g., finding similarity between different entity types.

We measure the structural robustness of each method by comparing its ranked list of results for the same query over different datasets with the same information but different structural representations. We adopt normalized Kendall's tau measurement to compare two ranked lists. The value of normalized Kendall's tau varies between 0 and 1 where 0 means two lists are identical and 1 means one list is the reverse of another. As users are normally interested in highly ranked answers, we compare top 5 and 10 answers.

[2] *alchemy.cs.washington.edu/data/uw-cse*

Table 1: Average ranking differences

	DBLP2SIGM		WSUC2ALCH		BioMedT	
	top 5	top 10	top 5	top 10	top 5	top 10
RWR	.447	.412	.259	.253	.130	.112
SimRank	.455	.410	.387	.341	.405	.385
PathSim	.608	.590	.310	.247	.438	.461

Table 1 shows the average ranking differences for top 5 and 10 answers returned by RWR, SimRank, PathSim and/or HeteSim. We do *not* report the results of RelSim because it returns the same answers over all transformations. According to Table 1, the outputs of all current algorithms are significantly different across databases under these information-preserving transformations.

We also evaluate the robustness of our algorithm when a transformation is invertible and adds extra information. We define a transformation, called DBLP2SIGMX, similar to DBLP2SIGM, which also adds additional nodes that connect authors to proceedings in the SIGMOD Record schema. These nodes indicate the relationship of what authors has a paper in what proceedings. Each of these nodes is connected to both the corresponding author and the proceedings in wich she has published at leas one paper. DBLP2SIGMX is invertible and has the same inverse as DBLP2SIGM. It takes more than a day to run SimRank and RWR over DBLP and BioMed. Thus, to compare the impact of adding information on all algorithms, we have performed these experiments over the small DBLP dataset, and a subset of BioMed datasets which has 4,125 nodes and 60,176 edges. We randomly selected 100 queries for DBLP dataset and use the same query workload used in Section 6.1 for BioMed dataset. We also use the relationship patterns for PathSim, HeteSim and RelSim that are used in the previous robustness experiments. Table 2 shows the robustness of all algorithms, including RelSim, under this transformation, which indicate that RelSim is the only robust algorithm over this transformation.

Furthermore, some real-world data transformations may *not* be invertible and lose some information. Hence, we also measure the robustness of algorithms when a small fraction of edges between nodes are removed during the transformation. In this experiment, we create two new transformations for DBLP and BioMed, namely DBLP2SIGM(.95) and BioMedT(.95), respectively. DBLP2SIGM(.95) and BioMedT (.95) restructure DBLP and BioMed similar to that of DBLP- 2SIGM and BioMedT, respectively. These transformation first restructure the database and then randomly remove 5% of the total number of edges from the transformed databases. Table 2 shows the average ranking differences for top 5 and top 10 answers returned by RWR, SimRank, PathSim (HeteSim) and RelSim using the same datasets, the query workloads and relationship patterns described in the previous experiment. The result indicates that RelSim is generally more robust than other methods when the transformed databases has a slightly less information than the original ones.

7.2 Effectiveness

We evaluate the effectiveness of RRE over BioMed and MAS databases. We use the same query workload used in Section 6.1 for BioMed Since each disease query relates only to a single drug, we

Table 2: Average ranking differences over transformations that modify information

	DBLP2SIGM		BioMedT(.95)		DBLP2SIGM(.95)	
	top 5	top 10	top 5	top 10	top 5	top 10
RelSim	0	0	.750	.144	0 .170	.298
RWR	.696	.752	.530	.500	.835	.640
SimRank	.790	.750	.143	.344	.790	.750
PathSim	.364	.239	.927	.927	.423	.452

Table 3: Average MRR of algorithms over BioMed.

BioMed dataset	RWR	SimRank	HeteSim	RelSim
original	.010	.062	.077	.077
under BioMedT	.010	.062	.072	.077

use *Mean Reciprocal Rank* (MRR) to evaluate the effectiveness of the algorithms. *Reciprocal Rank* (RR) of a list of answers to a query is $1/p$ where p is the position of the first relevant answer in the returned list of answers. MRR is the average of RR over a set of queries.

Table 3 shows average MRR of RWR, SimRank, HeteSim and RelSim over original BioMed dataset and BioMed under BioMedT. According to our discussion with the experts, these queries are very hard to answer effectively by using only the structural patterns in the data set and without consulting external sources of knowledge and even a slight improvement in the accuracy of the returned answers may save a great deal of time and effort in their research. The overall results show that RelSim are more effective than other algorithms. This implies that the use of RRE language helps to improve the effectiveness of the algorithm. We have also used the BioMed and query workload to evaluate the effectiveness of similarity search method proposed in Section 5. We have used the simple pattern given to HeteSim for this experiment. The MRR of the final results is close to the one delivered by RelSim algorithm that takes RRE patterns as input, which is as effective or more effective than other algorithms. These results also indicate that RelSim is as effective or more effective than similar algorithms.

7.3 Efficiency

We evaluate the query processing time of RelSim and PathSim over DBLP and BioMed datasets using the query workloads reported in Section 7.1. As explained in Section 6.1, it takes almost a day to run SimRank and RWR over these datasets. First, we evaluate the query processing time of RelSim and PathSim for the case where the user provides an exact relationship pattern (Section 4). All reported running times in this section assume that the commuting matrices of all meta-paths, i.e., simple RRE patterns that use only concatenation and reversal operations, up to size 3 are materialized and pre-loaded in main memory for both RelSim and PathSim. Theoretically, both RelSim and PathSim have the same time complexity. However, the expressiveness of RRE used in RelSim allows the specified relationship pattern to be more complex than the expression used by PathSim. To compare the efficiency between these two algorithms, we first pick a pattern over each database as a reference. Then, for each referenced pattern, we find the corresponding RRE pattern p_R for RelSim and the closest correspondent simple pattern, i.e., meta-path, p_P for PathSim. For instance, a referenced pattern over DBLP is `p-in·r-a`. Over DBLP under DBLP2SIGM transformation, the correspondent patterns for RelSim and PathSim

Table 4: Average query processing time in seconds.

	single pattern		using Algorithm 1	
	DBLP	BioMed	DBLP	BioMed
RelSim	.035	.473	.034	.511
PathSim	.024	.267	.027	.477

are p_R : `[p-in⁻]·r-a` and p_P : `r-a`, respectively. Then we compare the running time of PathSim using p_P with the running time of RelSim using p_R and report the results. Table 4 show the average query processing time for a single pattern per query of RelSim and PathSim (under "single pattern"). Overall, RelSim is slower than PathSim because RelSim uses more complex and longer patterns than those used by PathSim. Nevertheless, the running time of RelSim is still relatively short over large datasets.

Next, we measure the efficiency of RelSim that incorporates Algorithm 1 introduced in Section 5 with the the optimization techniques to ignore non-relevant constraints. In this version, RelSim takes a simple pattern as an input. Hence, we supply the same pattern to both RelSim and PathSim, and compare their query processing times. We use the same relationship patterns over DBLP and BioMed as described in Section 7.1. The average query processing times per query of RelSim and PathSim are reported in Table 4 (under "using Algorithm 1"). Overall, the running time of RelSim is slightly slower than PathSim due to the procedure of Algorithm 1. This result also shows that making RelSim more usable does *not* increase its running time considerably.

We also evaluate the scalability of the version of RelSim that takes simple patterns as input as described in Section 5 over BioMed dataset. Since the running time of this version depends on the number of tgd constraints on the original database and the length of the input simple pattern, we measure its running time over different values of these parameters. Because our datasets contain only two constraints, we test the scalability of our algorithm over a randomly generated set of constraints from only BioMed. Af a matter of fact, we pick BioMed as it allows us to generate and test many constraints due to its rich structure. We generate each constraint by using a coin flipping random generator over each edge label in BioMed to decide whether an atom of the constraint contains such edge. We limit the number of atoms in the premise of each constraint to be between 2 and 5, and a single atom in its conclusion. For the input pattern, each simple pattern is randomly generated such that it traverses the data graph from a drug to a disease. We measure the running time by setting the number of constraints to 1, 5, 10, 20 and 40, and vary the length, i.e., number of edge labels, of the input simple pattern between 4 and 10. We use the same query workload as the one described in Section 7.1. We report the average similarity query running times of RelSim per query over 5 runs of each setting in Figure 5. The Running time for pattern size of 9 for 40 constraints are omitted due to long running time. Overall, RelSim is reasonably fast for a small number of constraints and up to the length of 8 for the input pattern. The algorithm slows down when the number of constraint or the length of the input pattern goes up, especially when there are 20 or more constraints. We also tested the version of RelSim in Section 5 without our proposed optimization techniques for ignoring non-relevant constraints. The algorithm takes days to finish for 5 constraints or longer for more constraints for patterns of size more than 7.

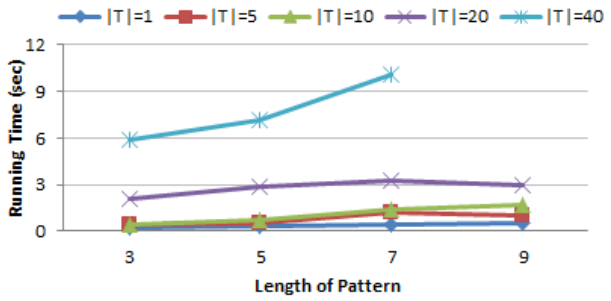


Figure 5: Running times of RelSim in various settings

8 RELATED WORK

There are robust algorithms over certain types of schematic variations [11, 37, 45, 49]. They, however, have two major shortcomings. First, they are robust only over a subset of frequently occurring schematic variations. Because they leverage the properties special to the variation over which they are robust, it is *not* clear how to generalize these algorithms to be robust against other schematic variations. Second, current schematically robust systems generally either propose new algorithms [49], or make significant and/or complex modifications to the current ones [37]. However, current algorithms have been widely adapted and it is costly to replace them with new ones. Hence, one should aim at making current algorithms robust to schematic variations using simple modifications. Moreover, current algorithms are shown to be effective over some data representations. Thus, a robust version of them will be effective over more representations.

REFERENCES

[1] Serge Abiteboul, Richard Hull, and Victor Vianu. 1995. *Foundations of Databases: The Logical Level.* Addison-Wesley.

[2] Ioannis Antonellis, Hector Garcia-Molina, and Chi-Chao Chang. 2008. Simrank++: query rewriting through link analysis of the click graph. *Proc. VLDB Endow.* 1, 1 (2008), 408–421. https://doi.org/10.14778/1453856.1453903

[3] Marcelo Arenas. 2006. Normalization Theory for XML. *SIGMOD Rec.* 35, 4 (Dec. 2006), 57–64. https://doi.org/10.1145/1228268.1228284

[4] M. Arenas and L. Libkin. 2004. A Normal Form for XML Documents. *TODS* 29, 1 (2004).

[5] Marcelo Arenas, Jorge Pérez, and Cristian Riveros. 2009. The Recovery of a Schema Mapping: Bringing Exchanged Data Back. *ACM Trans. Database Syst.* 34, 4, Article 22 (Dec. 2009), 48 pages. https://doi.org/10.1145/1620585.1620589

[6] Magda Balazinska, Surajit Chaudhuri, Anastasia Ailamaki, Juliana Freire, Sailesh Krishnamurthy, and Michael Stonebraker. 2020. The Next 5 Years: What Opportunities Should the Database Community Seize to Maximize its Impact?. In *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*, David Maier, Rachel Pottinger, AnHai Doan, Wang-Chiew Tan, Abdussalam Alawini, and Hung Q. Ngo (Eds.). ACM, 411–414. https://doi.org/10.1145/3318464.3394837

[7] Pablo Barcelo, Jorge Perez, and Juan Reutter. 2013. Schema Mappings and Data Exchange for Graph Databases. In *ICDT*.

[8] Catriel Beeri and Moshe Y. Vardi. 1984. A Proof Procedure for Data Dependencies. *J. ACM* 31, 4 (Sept. 1984), 24. https://doi.org/10.1145/1634.1636

[9] Shai Ben-David, Ulrike von Luxburg, and David Pal. 2006. A Sober Look at Clustering Stability. In *COLT*.

[10] I. Boneva, A. Bonifati, and R. Ciucanu. 2015. Graph data exchange with target constraints. In *EDBT/ICDT Workshop GraphQ (PODS '17)*. 171–176.

[11] Yodsawalai Chodpathumwan, Amirhossein Aleyasen, Arash Termehchy, and Yizhou Sun. 2016. Towards Representation Independent Similarity Search Over Graph Databases. In *CIKM*. https://doi.org/10.1145/2983323.2983673

[12] I. Cruz, A. Mendelzon, and P. Wood. 1987. A graphical query language supporting recursion. In *SIGMOD*. 323–330.

[13] Ronald Fagin. 2007. Inverting Schema Mappings. *ACM Trans. Database Syst.* 32, 2 (2007).

[14] Ronald Fagin, Phokion G. Kolaitis, Lucian Popa, and Wang-Chiew Tan. 2005. Composing Schema Mappings: Second-order Dependencies to the Rescue. *ACM Trans. Database Syst.* 30, 4 (Dec. 2005), 994–1055. https://doi.org/10.1145/1114244.1114249

[15] Ronald Fagin, Phokion G. Kolaitis, Lucian Popa, and Wang Chiew Tan. 2008. Quasi-inverses of schema mappings. *ACM Trans. Database Syst.* 33, 2 (2008). https://doi.org/10.1145/1366102.1366108

[16] Ronald Fagina, Phokion Kolaitis, Renee Miller, and Lucian Popa. 2005. Data exchange: semantics and query answering. *TCS* 336, 1 (2005).

[17] Wenfei Fan and Philip Bohannon. 2008. Information Preserving XML Schema Embedding. *TODS* 33, 1 (2008).

[18] Wenfei Fan, Yinghui Wu, and Jingbo Xu. 2016. Functional Dependencies for Graphs. In *SIGMOD (SIGMOD '16)*. ACM, New York, NY, USA, 1843–1857.

[19] Raul Castro Fernandez, Dong Deng, Essam Mansour, Abdulhakim Ali Qahtan, Wenbo Tao, Ziawasch Abedjan, Ahmed K. Elmagarmid, Ihab F. Ilyas, Samuel Madden, Mourad Ouzzani, Michael Stonebraker, and Nan Tang. 2017. A Demo of the Data Civilizer System. In *Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017*, Semih Salihoglu, Wenchao Zhou, Rada Chirkova, Jun Yang, and Dan Suciu (Eds.). ACM, 1639–1642. https://doi.org/10.1145/3035918.3058740

[20] Dennis Fetterly, Mark Manasse, Marc Najork, and Janet L. Wiener. 2003. A Large-Scale Study of the Evolution of Web. In *WWW*.

[21] Nadime Francis and Leonid Libkin. 2017. Schema Mappings for Data Graphs. In *PODS (PODS '17)*. ACM, New York, NY, USA, 389–401. https://doi.org/10.1145/3034786.3056113

[22] Vijay Gadepally, Timothy G. Mattson, Michael Stonebraker, Fusheng Wang, Gang Luo, and George Teodoro (Eds.). 2019. *Heterogeneous Data Management, Polystores, and Analytics for Healthcare - VLDB 2018 Workshops, Poly and DMAH, Rio de Janeiro, Brazil, August 31, 2018, Revised Selected Papers.* Lecture Notes in Computer Science, Vol. 11470. Springer. https://doi.org/10.1007/978-3-030-14177-6

[23] Gourab Ghoshal and Albert Barbasi. 2011. Ranking Stability and Super-stable Nodes in Complex Networks. *Nature Communications* 2, 394 (2011).

[24] C. Gutierrez, C. Hurtado, A. Mendelzon, and Jorge Perez. 2010. Foundations of Semantic Web Databases. *JCSS* 77, 3 (2010).

[25] Guoming He, Haijun Feng, Cuiping Li, and Hong Chen. 2010. Parallel SimRank Computation on Large Graphs with Iterative Aggregation. In *KDD*.

[26] André Hernich, Leonid Libkin, and Nicole Schweikardt. 2011. Closed World Data Exchange. *ACM Trans. Database Syst.* 36, 2, Article 14 (June 2011), 40 pages. https://doi.org/10.1145/1966385.1966392

[27] John Hopcroft and Robert Tarjan. 1973. Algorithm 447: Efficient Algorithms for Graph Manipulation. *Commun. ACM* 16, 6 (June 1973), 372–378. https://doi.org/10.1145/362248.362272

[28] R. Hull. 1984. Relative Information Capacity of Simple Relational Database Schemata. *PODS*.

[29] Glen Jeh and Jennifer Widom. 2002. SimRank: a measure of structural-context similarity. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, July 23-26, 2002, Edmonton, Alberta, Canada*. ACM, 538–543. https://doi.org/10.1145/775047.775126

[30] G. Jeh and J. Widom. 2003. Scaling Personalized Web Search. In *WWW*.

[31] David Jensen and Jennifer Neville. 2002. Linkage and Autocorrelation Cause Feature Selection Bias in Relational Learning. In *Proceedings of the Nineteenth International Conference on Machine Learning (ICML '02)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 259–266.

[32] Leo Katz. 1953. A new status index derived from sociometric analysis. *Psychometrika* 18, 1 (1953), 39–43.

[33] T. Lange, M. Braun, V. Roth, and J. Buhmann. 2003. Stability-based Model Selection. In *NIPS*.

[34] Barbara Staudt Lerner. 2000. A Model for Compound Type Changes Encountered in Schema Evolution. *TODS* 25, 1 (2000).

[35] Sergey Melnik, Atul Adya, and Philip A. Bernstein. 2007. Compiling mappings to bridge applications and databases. In *SIGMOD*.

[36] Tom M. Mitchell. 1997. *Machine Learning.* McGraw-Hill, New York.

[37] Jose Picado, Arash Termehchy, Alan Fern, and Parisa Ataei. 2017. Schema Independent Relational Learning. In *Proceedings of the 2017 ACM International Conference on Management of Data (SIGMOD '17)*. ACM, New York, NY, USA, 929–944. https://doi.org/10.1145/3035918.3035923

[38] Philippe Rigollet. 2007. Generalization Error Bounds in Semisupervised Classification under the Cluster Assumption. *Journal of Machine Learning Research* 8 (2007).

[39] Ohad Shamir and Naftali Tishby. 1982. Cluster Stability for Finite Samples. *Annals of Probability* 10, 4 (1982).

[40] Chuan Shi, Xiangnan Kong, Yue Huang, S Yu Philip, and Bin Wu. 2014. HeteSim: A General Framework for Relevance Measure in Heterogeneous Networks. *TKDE* 10 (2014).

[41] Michael Stonebraker, Raul Castro Fernandez, Dong Deng, and Michael L. Brodie. 2017. What to do about database decay. *Commun. ACM* 60, 1 (2017), 11. https://doi.org/10.1145/3014349

[42] Michael Stonebraker and Ihab F. Ilyas. 2018. Data Integration: The Current Status and the Way Forward. *IEEE Data Eng. Bull.* 41, 2 (2018), 3–9. http://sites.computer.org/debull/A18june/p3.pdf

[43] Yizhou Sun, Jiawei Han, Xifeng Yan, Philip S. Yu, and Tianyi Wu. 2011. PathSim: Meta Path-Based Top-K Similarity Search in Heterogeneous Information Networks. *Proc. VLDB Endow.* 4, 11 (2011), 992–1003. http://www.vldb.org/pvldb/vol4/p992-sun.pdf

[44] Zequn Sun, Qingheng Zhang, Wei Hu, Chengming Wang, Muhao Chen, Farahnaz Akrami, and Chengkai Li. 2020. A Benchmarking Study of Embedding-based Entity Alignment for Knowledge Graphs. *Proc. VLDB Endow.* 13, 11 (2020), 2326–2340. http://www.vldb.org/pvldb/vol13/p2326-sun.pdf

[45] Jian Tang, Cheng Li, and Qiaozhu Mei. 2017. Learning Representations of Large-scale Networks. In *KDD*.

[46] Hanghang Tong and Christos Faloutsos. 2006. Center-Piece Subgraphs: Problem Definition and Fast Solutions. In *KDD*.

[47] H. Tong, C. Faloutsos, and J. Pan. 2006. Fast Random Walk with Restart and its Applications. In *ICDM*.

[48] Hanghang Tong, Huiming Qu, and Hani Jamjoom. 2008. Measuring Proximity on Graphs with Side Information. In *ICDM*.

[49] Ba Quan Truong, Sourav S. Bhowmick, and Curtis Dyreson. 2012. *SINBAD: Towards Structure-Independent Querying of Common Neighbors in XML Databases.* Springer Berlin Heidelberg, Berlin, Heidelberg, 156–171. https://doi.org/10.1007/978-3-642-29038-1_13

[50] M. Y. Vardi. 1982. On decomposition of relational databases. In *FOCS*. 176–185. https://doi.org/10.1109/SFCS.1982.75

[51] Yannis Velegrakis, Renee J. Miller, and Lucian Popa. 2003. Mapping Adaptation under Evolving Schemas. In *VLDB*.

[52] S. Vishwanathan, N. Schraudolph, R. Kondor, and K. Borgwardt. 2010. Graph Kernels. *JMLR* (2010).

[53] Peter T. Wood. 2012. Query languages for graph databases. *SIGMOD Rec.* 41, 1 (April 2012). https://doi.org/10.1145/2206869.2206879

[54] Rongjing Xiang, Jennifer Neville, and Monica Rogati. 2010. Modeling Relationship Strength in Online Social Networks. In *Proceedings of the 19th International Conference on World Wide Web (WWW '10)*. Association for Computing Machinery, New York, NY, USA, 981–990. https://doi.org/10.1145/1772690.1772790

[55] C. Yu and H. V. Jagadish. 2006. Efficient Discovery of XML Data Redundancy. In *VLDB*.

[56] Weiren Yu and Xuemin Lin. 2012. IRWR: Incremental Random Walk with Restart. In *SIGIR*.

[57] Jing Zhang, Jie Tang, Cong Ma, Hanghang Tong, Yu Jing, and Juanzi Li. 2015. Panther: Fast Top-k Similarity Search on Large Networks. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '15)*. ACM, New York, NY, USA, 1445–1454. https://doi.org/10.1145/2783258.2783267

[58] Peixiang Zhao, Jiawei Han, and Yizhou Sun. 2009. P-Rank: A Comprehensive Structural Similarity Measure over Information Networks. In *CIKM*.