Structurally Robust Similarity Search

Yodsawalai Chodpathumwan*
TGGS, KMUTNB
Bangkok, Thailand
yodsawalai.c@tggs.kmutnb.ac.th

Arash Termehchy
Oregon State University
Corvallis, OR, USA
termehca@oregonstate.edu

Steven Ramsey
Oregon State University
Corvallis, OR, USA
ramsey@oregonstate.edu

Aayam Shresta
Oregon State University
Corvallis, OR, USA
shrestaa@oregonstate.edu

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.

PVLDB Reference Format:
Y. Chodpathumwan, and A. Termehchy. Structurally Robust Similarity Search. *PVLDB*, 12(xxx): xxxx-yyyy, 2019.
DOI: https://doi.org/10.14778/xxxxxxx.xxxxxxx

1. INTRODUCTION

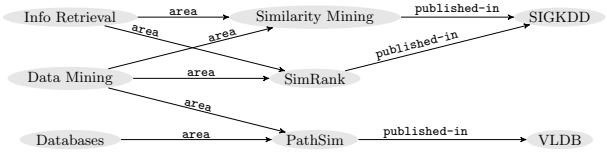
Data variety is ubiquitous in data management as different data sources may represent the same information in different forms [15, 31, 16, 25, 1]. A key goal of data management is to devise query interfaces that can hide schematic variations from users so that they do *not* have to reformulate their queries over schematic variations to get the same results [11, 1]. Recently, researchers have noticed that the results of unsupervised and supervised machine learning algorithms over structured data also depend on the schema and structure of the underlying database [36, 41, 9].

As an example, consider two bibliographic datasets from DBLP (*dblp.uni-trier.de*) and SIGMOD Record (*sigmod.org/publications*) whose fragments are shown in Figure 1. Intuitively, these databases represent essentially the same set of relationships between the same set of *paper*, *conference*, and *research area* entities. But, each dataset has its own way of organizing these entities and their relationships. For example, DBLP connects 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) for this information and connects research areas directly to conferences instead of papers. Assume that a user wants to find the most similar research area to *Data Mining* according to their conferences and publications. SimRank [26] is a well-known similarity search algorithm over graphs that finds two node, i.e. two entities, to be (structurally) similar if they both are similar to another node in the graph. It implements this idea using random walks over the graph. It finds *Data Mining* to be more similar to *Info Retrieval* than to *Databases* in Figure 1(a). In Figure 1(b), however, it declares *Data Mining* more similar to *Databases* than to *Info Retrieval*.

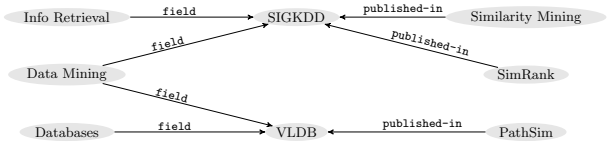
Hence, to use a data analytics algorithm, the user has to restructure the database to find the structure over which the algorithm delivers accurate results. Since there is *not* any clear guideline on how to find such a desirable structure for the algorithm, one has to do this through trial and error, which takes a great deal of time and effort. Bioinformatics experts frequently use graph databases. In our consultation with them, we have observed interesting cases in which these experts modify the structure of their graph databases to meet certain efficiency-oriented or data quality goals, but these modifications changes the results of answering similarity queries over these databases. For example, 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. This will help them to rewrite their lookup queries and run them 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.However, it undesirably changes

*Worked done at University of Illinois at Urbana-Champaign, Urbana, IL, USA

Proceedings of the VLDB Endowment, Vol. 12, No. xxx
ISSN 2150-8097.
DOI: https://doi.org/10.14778/xxxxxxx.xxxxxxx



(a) Fragments of DBLP



(b) Fragments of SIGMOD Record

Figure 1: Example of two bibliography databases. The nodes on the left, middle, and right are research areas, papers, and conferences, respectively.

the results of some similarity queries over the transformed database and make them less effective.

Ideally, one should design an algorithm that returns essentially the same accurate results over all possible schematic variations of a dataset. There are robust algorithms over certain types of schematic variations [33, 36, 41, 9]. 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 [41], or make significant and/or complex modifications to the current ones [33]. 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.

In this paper, we propose a novel approach to the problem of similarity search over graph databases that realizes the aforementioned goals. Similarity search is both a popular type of queries over graphs and an important building block of other graph analytics tasks, such as pattern query matching and community detection [38, 39, 26, 49]. To find and explore structural variations of a graph database, we leverage and extend a results in database literature [43, 12, 25], which state that constraints on a relational database give rise to its schematic variations. We extend this result for graph databases. For example, the variation in Figure 1 will *not* preserve the information in the database in Figure 1(a) unless all papers in a conference share the same set of research areas. Otherwise, it is *not* clear what research areas must connect to a conference in Figure 1(b). We show that given some mild assumptions, the constraints that induce structural variations over graph databases are in the form of tuple-generating dependencies (tgd), which are a well-known family of constrains over databases [1, 7]. In particular, we make the following contributions.

- We formalize invertible structural variations in graph databases, and define structural robustness of a similarity search algorithm (Section 3).
- We show that existing similarity search algorithms are *not* robust because of the limited types of relationships they use to measure the degree of similarity. Thus, we propose a robust algorithm called *RelSim*, which extends some well-known similarity search algorithm (*PathSim* [35, 34]) to use a sufficiently expressive set of patterns.
- RelSim requires users to specify the relationship between nodes that is used to compute similarity score between the nodes, which may be relatively complex and hard for users to provide them. We propose an algorithm that leverages some 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 6). Our empirical studies also show that our algorithm is efficient over large databases.

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 [5, 46, 17]. 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 [7, 6, 1]. Two widely used type of constraints are *tuple-generating* and *equality-generating* dependencies [7, 6]. 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* $(x_1, \texttt{area}, x_3) \wedge (x_3, \texttt{published-in}, x_4) \wedge (x_2, \texttt{published-in}, x_4) \rightarrow (x_1, \texttt{area}, x_2)$.

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 [18, 10, 46, 5]. 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 $\exists\bar{y}((z_1, p_1, z'_1) \wedge ... \wedge (z_k, p_k, z'_k))$ where p_i is an RPQ and $z_i, z'_i \in \{x_1, \ldots, x_n, y_1, \ldots, y_m\}$, $1 \leq i \leq k$ [46, 5]. 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 [35, 48, 40, 26, 50, 27, 21, 2, 34]. 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 answers for the same query across databases that contain the same information content. Researchers use the concept of invertible transformation to formalize the equivalence of information stored in different databases [25, 16]. 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 [25, 16]. We denote a transformation from S to T as Σ_{ST}. For example, consider a transformation $\Sigma_{1a,1b}$ from schema of the database in Figure 1(a) to the schema of the one in Figure 1(b), which changes the structure of the database in Figure 1(a) such that the research areas associated to 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 [15, 5]. 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 $(I, J) \models \Sigma_{ST}$ [15, 5]. 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., $\Sigma_{TS}(\Sigma_{ST}(I)) = I$. 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 $\Sigma_{ST}(I)$ contains sufficient information to rebuild the I and has at least the same amount of information as I. As Σ_{ST} maps each database of schema S to a database with at least the same amount of information in schema T, schema T has at least the same information capacity as S. In this case, 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 a transformation $\Sigma_{1a,1b}$ from schema of the database in Figure 1(a) to the schema of the 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 to a paper connect 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 still have the amount of information needed to recover the information of the database in Figure 1(a) from the transformed databases. We will formally explain the relationship between the constrain on the database in Figure 1(a) and invertibility of $\Sigma_{1a,1b}$ and $\Sigma_{1a,1c}$ in Section 3.2.2.*

Our definition of inverse extends the notion of Fagin-inverse used in the context of relational data exchange [12]. It, however, is slightly different from the notion of Fagin-inverse due to the fact that we have a different objective than the ones in data exchange. 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. Our goal is to compare the results of an algorithm over a single database and all its structural variations. The answers to a similarity query in different databases of S may be different. Thus, we are interested in the transformations that map a single database under a schema to multiple ones under the transformed schema. Consequently, the inverse of such a transformation maps multiple databases under the transformed schema only to 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 its invertible transformation. Two (ranked) list 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 $\Sigma_{ST}(I) \in \texttt{Inst}(T)$.*

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

Definition 1 does not specify the language of invertible transformations. To characterize the structural variations of a schema, one needs to express invertible transformations in some language. Researchers usually use *declarative (schema) mappings* to express schematic variations in graph and relational databases [15, 12, 5]. Roughly speaking, a transformation between schemas S and T is expressed as a set of first order 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 [5]. 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. *Consider a transformation $\Sigma_{1a,1b}$ from schema of the database in Figure 1(a) to the one in Figure 1(b). It adds an edge* `field` *for every path* `area`$\cdot$`published-in` *and keeps existing edges of label* `published-in`*. It is expressed as mapping with two rules of $(x_1, \texttt{published-in}, x_2) \rightarrow (x_1, \texttt{published-in}, x_2)$ and $(x_1, \texttt{area} \cdot \texttt{published-in}, x_2) \rightarrow (x_1, \texttt{field}, x_2)$. The inverse of $\Sigma_{1a,1b}$ can also be expressed as the set of following rules: $(x_1, \texttt{field} \cdot \texttt{published-in}^-, x_2) \rightarrow (x_1, \texttt{area}, x_2)$ and $(x_1, \texttt{published-in}, x_2) \rightarrow (x_1, \texttt{published-in}, x_2)$. 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 $(x_1, \texttt{published-in}, x_2) \rightarrow (y_1, \texttt{keyword-of}, x_1)$, where y_1 is an existential variable that represents the nodes that contain keywords of the paper x_1. The inverse of $\Sigma_{1a,1c}$ is the same as the inverse of $\Sigma_{1a,1b}$ as it does not require the information about keywords to reconstructs the original database.*

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 $\bar{u} \in [[\phi_S(\bar{y})]]_I$ [5]. We use the closed world semantic for schema mappings [23]. 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 [15, 12, 5]. 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 the information of I [12]. 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 Characterizing Structural Variations

Next, we present the full characterization of schemas that have structural variations. It helps us to identify the set of all invertible transformations 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 $\Sigma_{ST} \circ \Sigma_{TR}$, is a transformation from S to R such that if $J = \Sigma_{ST}(I)$ and $K = \Sigma_{TR}(J)$, then $K = (\Sigma_{TR} \circ \Sigma_{ST})(I)$.

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 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 [43] and the ones on the inverse of a relational schema mapping in [12]. According to Proposition 1, if there is *not* any constraint on a graph schema, it will *not* have any invertible structural variations. In other words, the constraints on schema S determines its structural variations. Also, based on this proposition, the constraints on S that gives rise to invertible structural variations are in form of tgds.

EXAMPLE 4. *The composition $\Sigma_{1a,1b} \circ \Sigma_{1a,1b}^{-1}$ 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 [13, 5]. Roughly speaking, each rule in $\Sigma_{ST} \circ \Sigma_{TR}$ 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) [13, 5, 7]. Assume that the conclusion of a rule in Σ_{ST} contains an atom (x, exp, y) with an existentially quantified variable. Also, assume that there is an atom in the premise of Σ_{ST}^{-1} that matches (x, exp, y). In this case, one needs second-order logic to express this composition [13]. 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 [7, 1]. Thus, in this paper, we focus our attention to the first order constraints as explained in Section 2. Existentially quantified variables are also introduced in an atom (x, exp, y) by using concatenation, Kleene star, or disjunction, i.e., $+$, in exp.

If the atoms in the premise of Σ_{ST}^{-1} 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 [13, 5]. For instance, 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 tgd constrains . In this case the shared atoms between the premises of rules in Σ_{ST}^{-1} 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. In this case, each rule in $\Sigma_{ST} \circ \Sigma_{ST}^{-1}$ is created by replacing an atom (x, l, y) in the premise of a rule in Σ_{ST}^{-1} 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 form (x, l^{-1}, y) in the premise of rules in Σ_{ST}^{-1}.

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 structure of similar types of inverse of schema mappings over relational databases [12]. Thus, the composition of Σ_{ST} and Σ_{ST}^{-1} 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, $\Sigma_{ST} \circ \Sigma_{ST}^{-1}$ 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 $\Sigma_{ST}^{-1} \circ \Sigma_{ST}$. 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. Table 1 summarizes the notations used in our framework.

3.2.3 Identifying Inverses

Over relational databases, there has been numerous works on identifying and computing (somewhat) inverses of a transformation [12, 14, 4]. Relational transformation, however, lacks the notion of recursion. Hence, we cannot use the proposed methods to compute the inverse of any transformation written using navigation language such as RPQ. To adopt existing methods, we reduce the RPQ as described in Section 2 to the following grammar.

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

where n is some constant natural number. We called this query language as *reduced-RPQ*. That is, the Kleene-star notion is replaced by a path with finite length of n, and the disjunction is not allowed. In this case, given a transformation whose set of logical formulas are written in *reduced-RPQ*, we can rewrite the transformation to be similar to that of relational. For instance, the reduced-RPQ $(x, l_1 \cdot l_2, y)$ can be rewritten as $l_1(x, z) \wedge l_2(z, y)$, by adding additional node variable z to the formula. Following the method proposed in [12] that define a *canonical $\mathcal{S}$-inverse*, which is an inverse of a transformation T (if there is) over a schema $\mathcal{S}$ whose set of constraint is Γ. We must note that the set of tgds in Γ must also be weakly acyclic for Γ to be finite chasable.

Let I_R be a one-tuple instance containing only the fact $R(\bar{x})$ where R is a label in $\mathcal{S}$, and I_R^Γ be a finite result of chasing I_R with Γ. Let J_R^Γ be a finite result of chasing I_R^γ with T. The inverse is a set of full tgds whose L.H.S. contains fact from J_R^Γ and R.H.S. contains fact from I_R^Γ. Consider that Γ consists of relations of two variables. The chase results in both J_R^Γ and I_R^Γ also contain relations of two variables. Since each formulas in the inverse is full tgd and each atom is a relation of two variables, e.g., $R(x, y)$, then we can convert these tgds to the one written in conjunctive reduced-RPQ. Nevertheless, computing an inverse is not a subject of this paper, and is open for future work.

Table 1: Summary of the notations.

Notation	Definition
$\texttt{Inst}(S)$	Databases of schema S
Σ_{ST}	Transformation from schema S to T
$\Sigma_{ST}(I)$	Transformations of I based on Σ_{ST}
Σ_{ST}^{-1}	The inverse of Σ_{ST}
$\Sigma_{ST} \circ \Sigma_{TR}$	The composition of transformations

4. ROBUST SIMILARITY SEARCH

4.1 Robustness of Current Methods

To the best of our knowledge, frequently used structural-based similarity search algorithms are based on random walk, e.g., RWR [39], pairwise random walk, e.g., SimRank [26] and P-Rank [50], or path-constrained framework, e.g., PathSim, [35, 34]. There are 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 [29, 49].

Similarity score computed by algorithms that use random walks and pairwise random walks are largely influenced by the topology of the graph. Because some information preserving transformations may modify the topological structure of a database, a structural-based similarity search algorithms such as RWR and SimRank are not robust under these variations as shown in our empirical studies in Section 6. Similar to these algorithms, there are algorithm 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 [49]. 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 the invertible transformations that modify the topology of the database. Invertible transformation may change the length and number of paths and walks between two nodes. For example, the length of the path between nodes *SIGKDD* and *Data Mining* is one in the database in Figure 1(a) and is two in its variation in Figure 1(b). One may significantly reduce or increase the length of paths between two entities in a database and its invertible transformations using the same type of variations. Thus, these methods may deliver a different similarity scores for the same pairs of nodes over a a database and its invertible transformations.

Two entities may be related via multiple paths in a database where each path may represent a different type of relationship. The degree of similarity between two entities may largely depend on the type of relationship between them. For instance, consider again Figure 1. A user may be interested in finding similar papers based on the conferences in which they are published rather than their common research areas. As another example, 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 and other users may like to find similar ones based on their shared conferences. 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, follow this approach [35, 34]. They allow users to supply a path template, called meta-path, that specifies the type of relationship between entities in their queries. A meta-path in Figure 1 is m_1 : $\texttt{published-in} \cdot \texttt{published-in}^-$, which reflects the relationship between two papers through the conference in which they are both published. Each instance of a meta-path in database D is a path in D whose sequence of edge labels match the sequence of labels in the meta-path. For example, *Similarity Mining*·$\texttt{published-in}$·*SIGKDD*·$\texttt{published-in}^-$·*SimRank* is an instance of m_1 in Figure 1(b). The PathSim score of entities u and v in a database D given a meta-path 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 depends 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 meta-path* 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* Info Retrieval *than to* Databases. *However, in Figure 1(b), the same user may use the meta-path* p_2 : $\texttt{field} \cdot \texttt{field}^-$ *to compute similarity scores between research areas. This meta-path finds that both* Info Retrieval *and* Databases *are equally similar to* Data Mining.

4.2 Achieving Robustness

Example 5 illustrates that there may *not* be any meta-path over some representations of a dataset to express a desired relationship. If a user wants to find the similarity of research areas based on their shared conferences, she can use p_2 over the representation in Figure 1(b), but she cannot find any meta-path in Figure 1(a) that expresses such a relationship. A candidate could be p_1 but it has additional information of the set of papers published in the conferences. On the other hand, the user may like to measure the similarities of research areas based on their shared conferences and the papers published in those conferences; therefore, she uses meta-path p_1 in Figure 1(a). Nevertheless, there is not any meta-path in Figure 1(b) that exactly expresses that relationship. The user should use the expression that also includes information about publications, and the represented relationship should be based on shared conferences.

One may solve this problem by using a language that is more expressive than the set of meta-paths to express relationships 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* p_2 *to be able to 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 the them in Figure 1(b). The resulting pattern in Figure 1(b) expresses an equivalent relationship to what* p_1 *represents over Figure 1(a).*

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 new relationship language that is expressive enough to represent equivalent relationships across various representations of the same dataset. 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 [5]. 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 same results as p_1 over Figure 1(a), the user should use the pattern p_4 : $\texttt{field}\cdot[\texttt{published-in}^-]\cdot[\texttt{published-in}^-]\cdot\texttt{field}^-$ to compute similarity score between research areas. That is, similar research areas are based on shared conferences, and the strength of this relation is based on the number of publications published in that conferences.

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

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

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 relationship 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 graph 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 $\bar{s}=\langle \hat{s_m},...,\hat{s_1}\rangle$ where, for each $i=1...m$, $\hat{s_i}=s_i$ if s_i represents a node and $\hat{s_i}=s_i^-$ 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 p^n is a concatenation of n p's, and $\widetilde{p}$ is a string of a copy of 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

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

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

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.

THEOREM 1. *Given schemas S and T, for every transformation Σ_{ST} and every pattern p over S, there exists a pattern p' over T such that, for every database $D\in\texttt{Inst}(S)$ and $J\in\Sigma_{ST}(D)$ $\forall u,v\in D$, $|\mathcal{I}_D^{u,v}(p)|=|\mathcal{I}_J^{u,v}(p')|$.*

First, we prove the following proposition.

PROPOSITION 3. *Given a schema S, $a\in S$, p, p_1 and p_2 are arbitrary RREs over S, and a database $D\in\texttt{Inst}(S)$ 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) $|\mathcal{I}_D^{u,u}([p])|=|\mathcal{I}_D^{u,u}(p\cdot\lceil\lceil p^-\rfloor\rfloor)|$

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. 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 '+'.

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 $k>0$ transformation rules in its inverse s.t. $\phi_1(x_1,x_2,\bar{x})\rightarrow(x_1,a,x_2)$, ..., $\phi_k(x_1,x_2,\bar{x})\rightarrow(x_1,a,x_2)$. Because each rule is acyclic, one can construct a pattern $p'_{i,j}$ that traverses $\phi_j(\bar{x})$ from x_1 to x_2 for each $j=1...k$. We have that, $\forall u,v\in D$, $(u,a,v)\in D$ iff $\bigvee_{j=1...k}\phi_j(u,v,\bar{x})$ iff $(u,p'_{i1}+...+p'_{ik},v)\in\Sigma_{ST}(D)$. Let $p_i'=\lceil\lceil p'_{i1}+...+p'_{ik}\rfloor\rfloor$. 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.

Next we show that $p'_j = p_i$, for each $i \neq j$. Consider if $p'_j = p'_i$, where $i \neq j$, and there is no other such p'_j. There must exist a transformation rule in the inverse of Σ_{ST} that maps to multiple labels in S, and there is no rule that maps to each of those labels. Hence, Σ_{ST} is not information preserving.

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 transformation with an acyclic premise 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 along the premise, and requires an indicator in the relationship language where two variables along the traversal are the same, e.g., starting and ending nodes in a cycle are the same. For instance, consider the relationship pattern representing the premise of a cyclic tgd $(x_1, \mathtt{a}, x_2) \wedge (x_2, \mathtt{b}, x_3) \wedge (x_3, \mathtt{c}, x_4) \wedge (x_1, \mathtt{d}, x_3) \wedge (x_2, \mathtt{e}, x_4) \rightarrow (x_1, \mathtt{f}, x_4)$. It is *not* possible to rewrite this pattern to an equivalent one without a conjunction ($\wedge$). That is, the premise must be rewritten in the form (x_1, exp, x_4) for some RRE *exp*. For instance, to remove the conjunction between in $(x_1, \mathtt{a}, x_2) \wedge (x_2, \mathtt{b}, x_3)$, one may write $(x_1, \mathtt{a} \cdot \mathtt{b}, x_3)$. However, because x_2 is specified in $(x_2, \mathtt{e}, x_4)$, 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 expression. By adding conjunction to the relationship language, as the patterns become more complex, it will take longer to compute the similarity scores between nodes. The result of Theorem 1 extends for general tgd constraints if conjunction is added to our proposed relationship language.

The following is an immediate result of Theorem 1.

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 $\forall D \in \mathtt{Inst}(S)$, $\forall u, v \in D$, $\mathit{sim}_p(u, v, D) = \mathit{sim}_{M(p)}(u, v, \Sigma_{ST}(D))$.*

Corollary 1 guarantees that, for each pairs 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 define RWR or SimRank scores between entities based on a particular relationship pattern between entities [35]. 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 the 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. However, when given a relationship pattern, a hop is defined only if there is a walk that follows and completes 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 1, 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 consider 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 $\forall D \in \mathtt{Inst}(S)$, $\forall u, v \in D$, $\mathit{RWR}_p(u, v, D) = \mathit{RWR}_{M(p)}(u, v, \Sigma_{ST}(D))$ and $\mathit{SimRank}_p(u, v, D) = \mathit{SimRank}_{M(p)}(u, v, \Sigma_{ST}(D))$.*

PathSim is shown to be more effective than RWR and SimRank [35]. Thus, we focus on our extension of PathSim.

4.3 Computing Similarity Scores

For an expression with only concatenations, the number of RRE instances can be computed using *commuting matrix* [35]. Given labels $l_1, ..., l_m$ in a schema S, a commuting matrix of an expression $p = l_1 \cdot ... \cdot l_m$ 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 $\frac{2\mathbf{M}_p(u,v)}{\mathbf{M}_p(u,u)+\mathbf{M}_p(v,v)}$ [35].

We extend the computation of commuting matrix for RRE expressions 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 arbitrary expressions 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 $p_1 = p_2$, $\mathbf{M}_{\lceil p \rceil} = \mathbf{M}_p > \mathbf{0}$, 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 expressions p over database D still follow 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 number of other operations, and $|V|$ denotes the number of nodes in D. Therefore, RelSim still has the same complexity as that of PathSim.

Nevertheless, one may reduce the complexity by exploiting the use of parenthesis when constructing an expression p. For instance, consider an expression $p = l_1 \cdot l_2 + l_1 \cdot l_3$ for some labels l_1, l_2 and l_3. The commuting matrix for p can be computed as $\mathbf{A}_{l_1}\mathbf{A}_{l_2} + \mathbf{A}_{l_1}\mathbf{A}_{l_3}$. One may rewrite p as $l_1 \cdot (l_2 + l_3)$ which can be computed as $\mathbf{A}_{l_1}(\mathbf{A}_{l_2} + \mathbf{A}_{l_3})$.

Hence, the latter helps reduce the required matrix operations by one multiplication. One may also use sparse matrices operations or any existing fast matrices multiplication to reduce the time complexity [45]. Further, consider that certain patterns (or sub-patterns) may be frequently used. Their commuting matrices, hence, are repeatedly constructed. To reduce such repetitive computation and overall running time, we may pre-materialize those matrices then look them up in later computation. For instance, one can pre-materialize and store all commuting matrices $\mathbf{M}_{l_1 \cdot l_2}$ of pattern $l_1 \cdot l_2$, for every pair of labels l_1 and l_2. To compute instances of pattern $a \cdot b \cdot c$, one only needs to look up $\mathbf{M}_{a \cdot b}$ and $\mathbf{M}_{b \cdot c}$ and performs a single multiplication $\mathbf{M}_{a \cdot b}\mathbf{M}_{b \cdot c}$. Nevertheless, we do *not* explain and discuss any details of such optimization as it is out of the scope of this paper. To compute the similarity scores using patterns with kleene-star p^*, one has to consider all possible repetition of p as we have $\mathcal{I}(p^*) = \mathcal{I}(\epsilon + p + p \cdot p + \ldots)$. To compute such patterns efficiently, we limit the maximum number of repetitions in p^* to a given number.

5. SIMPLIFYING RelSim

The relationship language presented in Section 4, is relatively complicated and hard to construct or interpret for average users. For instance, an RRE p_4 : `field`$\cdot$ [`pubslihed-in`$^-$] $\cdot$ [`pubslihed-in`$^-$]$\cdot$ `field`$^-$ is less intuitive than an RRE p_2 : `field`$\cdot$`field`$^-$ over Figure 1(b). Generally, users would like to spend less effort to express their queries. One way to achieve this goal is to enable users to submit their patterns using a relatively smaller and intuitive subset of operations in our proposed pattern language such as concatenation. In addition, users may also like to measure the degree of similarity of two entities using a set of related relationships to get a more holistic view of their similarities. For example, users may want to use both p_2 and p_4 to compute similarity between pairs of research areas over Figure 1(b) to find the overall similarity of research areas. In this section, we propose a robust similarity search algorithm whose input is an RRE pattern that uses only concatenations and/or reverse traversals. We call such a pattern *simple*. Our algorithm leverages the constrains in the database to generate a set of patterns *related* to the input one and use the full expressivity of RRE. Our algorithm, then, compute and aggregate the similarity scores of these patterns. This way, the user can both use a relatively simple language to specify relationships between entities, which is as simple as the one used by PathSim, and take advantage of complex relationships between entities to get effective and robust answers.

Algorithm 1 finds a subset of RREs by minimally modifying a simple pattern given by the user such that the results of Corollary 1 holds for the aggregated scores of all patterns. Each RRE returned by the algorithm represents a relationship pattern that may include or exclude some information to or from the input pattern according to the database constraints.

For instance, given an input p_2 : `field`$\cdot$ `field`$^-$ over Figure 1(b), the algorithm returns a set of RREs including both p_2 and p_4 : `field`$\cdot$ [`published-in`$^-$] $\cdot$ [`published-in`$^-$] $\cdot$ `field`$^-$. These RREs are then used to compute aggregate scores over each RRE in the returned set using Equation 1.

Specifically, Algorithm 1 takes a simple pattern $p = l_1 \cdot \ldots \cdot l_n$ from a user in addition to a database schema $S = (\mathcal{L}, \Gamma)$. Let (r, i) denote an RRE r which is processed up to label l_i in p. Given (r, i), let s denote the the remaining unprocessed sub-path of p, e.g., $s = l_{i+1} \cdot \ldots \cdot l_n$. Algorithm 1 examines each sub-path $exp : l_{i+1} \cdot l_j$ of s for some $i + 1 \leq j \leq n$. Then, it uses Algorithm 2 to find a set $\mathcal{R}$ of possible replacement pattern for exp according to each constraint in S (Line 11). If a replacement exp' exists for exp, the algorithm replaces exp in s with exp' and marks that all labels up to l_j has been processed, e.g., $(r \cdot exp', j)$ (Line 14). In addition, the algorithm also keeps the case where exp is not replaced, e.g., $(r \cdot exp, j)$ or $(r \cdot l_{i+1}, i+1)$ when consider only $j = i+1$ (Line 8). Overall, Algorithm 1 fines all possible combinations of replacement over each sub-path of p.

We provide a toy example of running Algorithm 1 as follows. Consider a simple pattern $p = \mathtt{a} \cdot \mathtt{b} \cdot \mathtt{c} \cdot \mathtt{d}$ and a single constraint γ. Starting from $(\epsilon, 0)$, the algorithm adds $(\mathtt{a}, 1)$ to *processing*. Here, we have the remaining unprocessed sup-path $s = \mathtt{a} \cdot \mathtt{b} \cdot \mathtt{c} \cdot \mathtt{d}$ of p. Then the algorithm examines each of following sub-paths of s: $\mathtt{a}$, $\mathtt{a} \cdot \mathtt{b}$, $\mathtt{a} \cdot \mathtt{b} \cdot \mathtt{c}$ and $\mathtt{a} \cdot \mathtt{b} \cdot \mathtt{c} \cdot \mathtt{d}$. Assume Algorithm 2 returns a set $\mathcal{R}$ consisting of $(\mathtt{a}, \mathtt{w}_1)$ and $(\mathtt{a} \cdot \mathtt{b} \cdot \mathtt{c}, \mathtt{w}_2)$ for a simple pattern s and constraint γ. Hence, the algorithm adds $(\mathtt{w}_1, 1)$ and $(\mathtt{w}_2, 3)$ to *processing*. Then the algorithm continues with the next member in *processing*. Consider $(\mathtt{w}_1, 1)$ where the remain-

Algorithm 1: `Path Modifier`

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

1 $done \leftarrow \{\}$
2 $processing \leftarrow \{(\epsilon, 0)\}$ `// For each pair` $(r, i) \in processing$, r `denotes an RRE 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*
7 **continue**
8 Add $(r \cdot l_{i+1}, i + 1)$ to *processing*
9 $s \leftarrow l_{i+1} \cdot l_{i+2} \cdot \ldots l_n$
10 **foreach** $\gamma \in \Gamma$ **do**
11 $\mathcal{R} \leftarrow \mathcal{R} \cup$ `SubPathModifierPerConstraint`(γ, s)
12 **foreach** $j \geq i + 1$ **do**
13 **foreach** $(l_{i+1} \cdot l_{i+2} \cdot \ldots \cdot l_j, exp') \in \mathcal{R}$ **do**
14 Add $(r \cdot exp', j)$ to *processing*
15 **return** $\mathcal{E}_p \leftarrow done$

Algorithm 2: `Sub-path Modifier Per Constraint`

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

1 $G_\gamma \leftarrow$ a graph representing ϕ_γ
2 **foreach** $i > 0, j \geq i$ **do**
3 $exp \leftarrow l'_i \cdot l'_{i+1} \cdot \ldots \cdot l'_j$
4 **if** a path exp 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 $exp' : v_g \hookrightarrow_H v_h$ that traverse H from v_g to v_h and visit each edge of H once
7 Add (exp, exp') to $\mathcal{R}$
8 Add (exp^-, exp'^-) to $\mathcal{R}$
9 **return** $\mathcal{R}$

ing unprocessed sub-path s is now $\mathtt{b}\cdot\mathtt{c}\cdot\mathtt{d}$. It first adds $(\mathtt{a}\cdot\mathtt{b}, 2)$ to *processing*. Assume Algorithm 2 returns a set $\mathcal{R}$ consisting of $(\mathtt{b}\cdot\mathtt{c}\cdot\mathtt{d}, \mathtt{w}_3)$ for the this new value of s. It then adds $(\mathtt{a}\cdot\mathtt{w}_3, 4)$ to *processing*. Since 4 is the length of p, $\mathtt{a}\cdot\mathtt{w}_3$ is marked as *done* and will be added to the final results in $\mathcal{E}_p$. The algorithm repeats these steps until all members in *processing* are processed.

Before we explain Algorithm 2, we would like to describe a graph representing the premise of a constraint. Given a constraint $\gamma : \phi_\gamma(\bar{x}) \rightarrow \psi_\gamma(\bar{x})$, let us first assume that, for each atom (x_i, exp, x_j) in ϕ_γ, exp *cannot* be written as $exp_1 \cdot exp_2$ for some non-empty RPQs exp_1 and exp_2. If such atom exists, the atom is rewritten as $(x_i, exp_1, x') \wedge (x', exp_2, x_j)$ for some fresh variable x'. A representing graph of the premise 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 pattern between each pair of those variables in ϕ_γ. More specifically, a node $v_i \in V$ if and only if x_i is a variable in ϕ_γ, and an edge $(v_i, exp, v_j) \in E$ if and only if (x_i, exp, x_j) is in ϕ_γ. For instance, given a constraint $\gamma : (x_1, \mathtt{a}, x_2) \wedge (x_2, \mathtt{b}, x_3) \wedge (x_3, \mathtt{c}+\mathtt{d}, x_4) \rightarrow (x_1, \mathtt{e}, x_4)$, the graph $G_{pre}(\gamma)$ is shown in Figure 2.

Next, we would like to explain the underlying idea of the replacement patterns found by Algorithm 2. Consider that each constraint implies an information-preserving transformation that may add or remove an edge from the current schema. For instance, consider a constraint γ_{1a} as described in Example 1 over a schema S_{1a} of a database shown in Figure 1(a). Following Example 3, a transformation from S_{1a} may add an edge label `field` between two nodes u_1 and u_2 for every path `area`·`published-in` from u_1 to u_2. The transformation then removes edges of label `area` resulting in a target schema S_{1b} of a database as shown in Figure 1(b). That is, the corresponding RRE over S_{1a} to `field` over S_{1b} is $p_{field} : \lceil\lceil \mathtt{area}\cdot\mathtt{published\text{-}in}\rfloor\rfloor$. Consider $G_{pre}(\gamma)$ which is the representing graph of the premise of constraint γ_{1a} or $G_{pre}(\gamma_{1a})$. Because edges $(v_1, \mathtt{area}, v_3)$ and $(v_3, \mathtt{published\text{-}in}, v_4)$ exist in $G_{pre}(\gamma)$, the path `area`·`published-in` from u_1 to u_2 matches the same pattern from v_1 to v_4 over $G_{pre}(\gamma)$. Hence, we can find an RRE pattern p_{field} which is one of the traversals from v_1 to v_4 over $G_{pre}(\gamma)$.

Based on the aforementioned observation, given a constraint γ and a simple pattern r, we describe Algorithm 2 that finds associated RREs over a graph $G_{pre}(\gamma)$ to r. Generally, for each simple pattern exp, the algorithm constructs one or more RREs exp' from possible traversals over $G_{pre}(\gamma)$ along the path exp. Then each sub-path of the user's pattern matching exp is replaced with each corresponding exp' in Algorithm 1.

For Algorithm 2, we briefly describe a recursive procedure to compute an RRE $v_s \hookrightarrow_G v_t$ that traverses a graph G from node v_s to v_t as follows. Consider that one may adopt a depth-first or 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 a constraint to be acyclic, then 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 $[p^{i,k}]$. We then 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 the graph of 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.

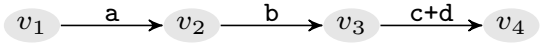


Figure 2: A representing graph $G_{pre}(\gamma)$ of a constraint $\gamma : (x_1, \mathtt{a}, x_2) \wedge (x_2, \mathtt{b}, x_3) \wedge (x_3, \mathtt{c}+\mathtt{d}, x_4) \rightarrow (x_1, \mathtt{e}, x_4)$

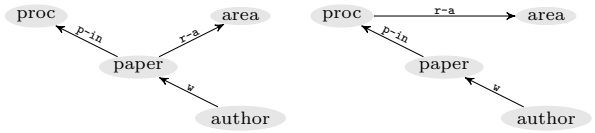
As an example, given a graph $G_{pre}(\gamma)$ as shown in Figure 2 and a simple pattern $\mathtt{a}\cdot\mathtt{b}$, Possible RRE patterns that traverse this graph from v_1 to v_3, i.e. $v_1 \hookrightarrow_{H \subseteq G_{pre}(\gamma)} v_3$, are $\mathtt{a}\cdot\mathtt{b}$, $\lceil\lceil\mathtt{a}\cdot\mathtt{b}\rfloor\rfloor$, $\mathtt{a}\cdot\mathtt{b}\cdot[\mathtt{c}+\mathtt{d}]$ and $\lceil\lceil\mathtt{a}\cdot\mathtt{b}\rfloor\rfloor\cdot[\mathtt{c}+\mathtt{d}]$. In this case, Algorithm 2 adds all $(\mathtt{a}\cdot\mathtt{b}, exp')$ to the returned set $\mathcal{R}$ where exp' is one of the above patterns, except the first one.

The complexity of Algorithm 1 largely depends on the number of replacement patterns found from Algorithm 2. In Algorithm 2, since each simple pattern p found in G when constructing the traversal can be either p or $\lceil\lceil p\rfloor\rfloor$, this algorithm is indeed exponential in the number of simple patterns. However, each simple pattern is a subgraph of G that is a path graph, e.g., a connected tree whose nodes have degree 2 except two terminal nodes. Hence, the number of simple patterns is $D = 1 + \sum_{v \in V_{\deg>2}} (\deg(v) - 1)$ where $V_{\deg>2}$ is a set of nodes in G whose degrees are greater than 2. Since $\sum_{v \in V(G)} \deg(v) = 2|E(G)|$. Hence, the procedure is $O(2^{|E(G)|})$. Further, a linear-time algorithm that finds all connected subgraphs G of G_γ is described in [24] based on either depth-first-search or breath-first-search procedures. Also, since the algorithm iterates over all sub-path of the input pattern, the total complexity of Algorithm 2 is $O(n^2(|V(G)| + 2^{|E(G)|}))$. Nevertheless, constraints of a schema are given and usually have small number of terms compared to the size of databases. We may view them as some small constants C. Hence, Algorithm 2 is $O(n^{2C})$. Since Algorithm 1 replaces each label or sub-expression of the user input with possible RREs returned by Algorithm 2, the complexity of Algorithm 1 is $O((n^{2C})^n) = O(n^{2Cn})$.

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

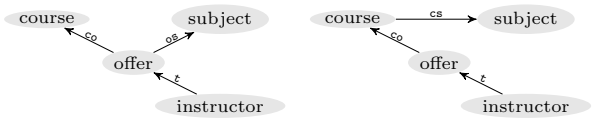
Thus, a similarity search algorithm that compute aggregate similarity scores over a set of RREs returned by Algorithm 1 is structurally robust.

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 our algorithms. We show that a structural variation actually needs I to have non-trivial constrains. Thus, as far as our algorithm uses 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



(a) DBLP (b) SIGMOD Record

Figure 3: 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 4: Variations for course databases. `cs`, `os`, `t` and `co` denote edge labels `course-subject`, `offering-subject`, `teach`, and `offering-course`, respectively.

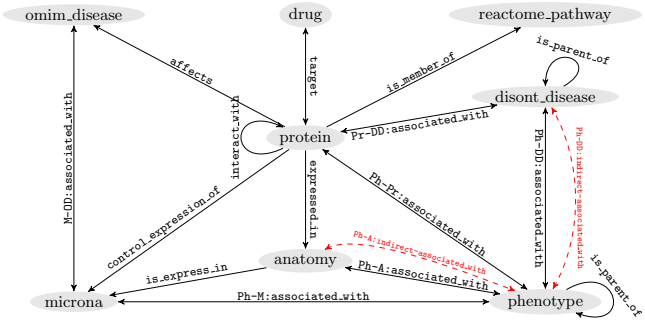


Figure 5: Schema fragment of BioMed dataset

and the running time of its similarity search. The details of our optimizations are in the Appendix 7.

Our approach simplifies the language of patterns but users still need to know the schema of the database to formulate their queries. This is more difficult on very large and heterogeneous databases, such as Yago (*www.mpi-inf.mpg.de/yago-naga*), which contain thousands of edges types. One may use path suggestion and auto-completion tools based on the popularity of queries paths to make users' work easier. Users may also use schema exploration tools, e.g., [44], to learn the schema easier.

6. EMPIRICAL EVALUATION

Datasets & Settings: 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 3(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 4(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, and their relationships. Figure 5 depicts a fragment of BioMed. It consists of 43,307 nodes and 1,742,970 edges.

We compare robustness, effectiveness and efficiency of RelSim with RWR [39] using a restart probability of 0.8, SimRank [26] using a damping factor of 0.8, and PathSim [35]. Since previous study show that the PathSim similarity computation method is more effective than those of SimRank and RWR [35], we use RelSim with the similarity computation score 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.

6.1 Structural Robustness

We use DBLP, WSU and BioMed databases to evaluate the structural robustness of RWR, SimRank, PathSim and RelSim. Because SimRank takes too long to finish on full DBLP dataset, we perform 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 3(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 4(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 database satisfies ($phenotype_1$, `is-parent-of`, $phenotype_2$) $\wedge$ ($phenotype_1$, `associated-with`, $anatomy$) $\rightarrow$ ($phenotype_2$, `indirect-associated-with`, $anatomy$) and ($phenotype_1$, `is-parent-of`, $phenotype_2$) $\wedge$ ($disease$, `associated-with`, $phenotype_1$) $\rightarrow$ ($disease$, `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 5 excluding all dashed edges. A 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 [34], which extends PathSim to support asymmetric paths, e.g., finding similarity between different entity types.

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

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

Table 2: Average ranking differences

	DBLP2SIGM		WSU2Alchemy		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

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. Because users are normally interested only in highly ranked answers, we compare only top 5 and 10 answers.

Table 2 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. For PathSim, we use the expressions $\texttt{p-in}^- \cdot \texttt{r-a} \cdot \texttt{r-a}^- \cdot \texttt{p-in}$ and $\texttt{r-a} \cdot \texttt{r-a}^-$ over DBLP and SIGMOD Record structures, respectively. For RelSim, we use the same expression as that for PathSim over DBLP and use an RRE $[\texttt{p-in}^-] \cdot \texttt{r-a} \cdot \texttt{r-a}^- \cdot [\texttt{p-in}^-]$ over SIGMOD. Over WSU and Alchemy UW-CSE, we use the simple patterns $\texttt{co}^- \cdot \texttt{os} \cdot \texttt{os}^- \cdot \texttt{co}$ and $\texttt{cs} \cdot \texttt{cs}^-$, respectively, for PathSim. For RelSim, we use the same expression over WSU, but we use $[\texttt{co}^-] \cdot \texttt{cs} \cdot \texttt{cs}^- \cdot [\texttt{co}^-]$ over Alchemy UW-CSE. For BioMed dataset, we consult an expert and obtain an RRE p_{ex} : $\texttt{target}\cdot\texttt{express-in}\cdot (\texttt{Ph-A:associated-with}+\texttt{Ph-A:indirect-associated-with}) \cdot (\texttt{Ph-DD:associated-with}+\texttt{Ph-DD:indirect-associated-with})$. The corresponding RRE over the transformed BioMed dataset is p_{ex}^T : $\texttt{target}\cdot\texttt{express-in}\cdot (\texttt{Ph-A:associated-with} + \lceil\lceil\texttt{is-parent-of}^-\cdot\texttt{Ph-A:associated-with}\rfloor\rfloor) \cdot (\texttt{Ph-DD:associated-with}+ \lceil\lceil\texttt{is-parent-of}^-\cdot\texttt{Ph-DD:associated-with}\rfloor\rfloor)$. Since HeteSim does *not* support RRE, we compute the similarity scores between a pair of drug *dr* and disease *dd* by averaging similarity scores computed over the following patterns: $\texttt{target}\cdot\texttt{express-in}\cdot L_1\cdot L_2$, where L_1 is either $\texttt{Ph-A:associated-with}$ or $\texttt{Ph-A:indirect-associated-with}$ and L_2 is either $\texttt{Ph-DD:associated-with}$ or $\texttt{Ph-DD:indirect-associated-with}$. Since edges of label $\texttt{indirect-associated-with}$ do *not* exist in the transformed database, our best attempt to construct an equivalent expression to p_{ex}^T for HeteSim results in the following patterns: $\texttt{target}\cdot\texttt{express-in}\cdot L_1' \cdot L_2'$, where L_1' is either $\texttt{Ph-A:associated-with}$ or $\texttt{is-parent-of}^-\cdot\texttt{Ph-A:associated-with}$ and L_2' is either $\texttt{Ph-DD:associated-with}$ or $\texttt{is-parent-of}^-\cdot\texttt{Ph-DD:associated-with}$. According to Table 2, the outputs of all current algorithms are significantly different across databases under these invertible transformations.

Furthermore, real world data transformation may induce some small information lost during the process. That is, a transformation may not be invertible. Hence, we also measure the robustness of algorithms when some small fraction of relations between entities are removed during the transformation. In this experiment, we create 2 extra 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 DBLP2SIGM and BioMedT, respectively. To reflect information lost, they also randomly remove 5% of the total edges of the transformed databases. It takes more than a few days to run SimRank and RWR over both DBLP and BioMed datasets. Thus, to compare the impact of reducing information on all algorithms, we have performed these experiments over subsets of DBLP and BioMed datasets, which have 18,995 and 4,125 nodes, and 24,962 and 60,176 edges, respectively. We followed the same approach in choosing queries for all methods and the same relationship patterns for PathSim, HeteSim and RelSim as used in the previous experiments to evaluate robustness. Table 3 shows the average ranking differences for top 5 and top 10 answers returned by RWR, SimRank, PathSim (HeteSim) and RelSim. These results indicate that RelSim is generally more robust than other methods for the cases that original and transformed databases do *not* contain exactly the same information. The only exception is the top-5 results for SimRank and RWR over BioMed dataset.

Table 3: Average ranking differences over transformations that slightly reduce information

	DBLP2SIGM(.95)		BioMedT(.95)	
	top5	top10	top5	top10
RelSim	.170	.298	.750	.144
RWR	.696	.640	.530	.500
SimRank	.790	.750	.143	.344
PathSim/HeteSim	.423	.452	.927	.927

6.2 Effectiveness

We evaluate the effectiveness of RelSim over MAS and BioMed. For query workload over MAS, we randomly sample 100 conferences based on their degrees in the dataset. To provide the ground truth, for a given conference q, we manually label other conferences in three categories: *similar*, *quite-similar* and *least-similar*. A conference is considered similar to q when they share the same research area. A conference is considered quite-similar to q when they are connected to strongly related research area. Otherwise, the conference is considered least-similar to q. For example, *Data Mining* and *Databases* are strongly related, but *Databases* and *Computer Vision* are not. We use Normalized DCG (nDCG) to evaluate the effectiveness because it supports multiple levels of relevance for returned answers [30]. The value of nDCG varies between 0 and 1 where the high values indicate more effective ranking. We compare the effectiveness of RelSim with PathSim and report the values of nDCG for top 5 (nDCG@5) and top 10 (nDCG@10) answers. For BioMed, we obtain the query workload from an expert. We use 30 queries of diseases with their relevant drugs from the domain experts. Since each disease query relates only to a single drug, we 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.

Over MAS, we compute similarities between conferences based on the keywords in their domains. We use the pattern $\texttt{pc}\cdot\texttt{pd}\cdot\texttt{da}\cdot\texttt{da}^-\cdot\texttt{pd}^-\cdot\texttt{pc}^-$ and $\lceil\lceil\texttt{pc}\cdot\texttt{pd}\rfloor\rfloor\cdot\texttt{da}\cdot\texttt{da}^-\cdot\lceil\lceil\texttt{pd}^-\cdot\texttt{pc}^-\rfloor\rfloor$ for PathSim and RelSim, respectively. The average

Table 4: The MRR of RWR, SimRank, HeteSim and RelSim over BioMed databases.

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

nDCG@5 (nDCG@10) for RelSim and PathSim are 1.0 (1.0) and 0.969 (0.901), respectively. RelSim significantly outperforms PathSim. This is because, nodes about paper should not influence the similarity score between conferences based on the keywords of their domains. Since the language of relationship pattern used by PathSim is less expressive, the pattern between conferences and keywords always include papers. Hence, it deems conferences with more papers to be more similar although they may not have many common keywords. The RRE language used by RelSim is more expressive, and so it could avoid this problem. For example, the top 5 answers returned by RelSim for query *SIGKDD* are *ICDM*, *IDEAL*, *PAKDD*, *PJW* and *PKDD*. However, the the top 5 answers returned by PathSim for the same query are *ICOMP*, *IC-AI*, *ICAIL*, *ICALP* and *ICANN*.

Table 4 shows the MRR values 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. According to the experts, 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.

In addition, consider that RelSim uses the same similarity metric to that of PathSim over MAS and HeteSim over BioMed, but RelSim is shown to be more effective. This implies that the use of RRE language helps to improve the effectiveness of the algorithm.

6.3 Efficiency

We evaluate the query processing time of RelSim and PathSim over DBLP and BioMed datasets using the query workloads reported in Section 6.1. 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 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. The average query processing time for a single pattern per query of RelSim (PathSim) over DBLP and BioMed dataset are 0.035 (0.024) and 0.473 (0.267) seconds, respectively. The reason that RelSim is slower than PathSim as RelSim uses more complex and longer patterns than those used by PathSim. But, the running time is still relatively short over large datasets.

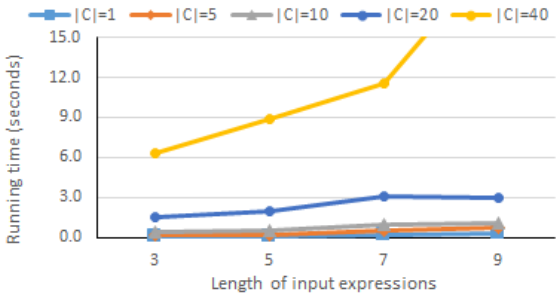


Figure 6: Running times of RelSim given various parameter settings

Next, we measure the efficiency of RelSim that incorporates Algorithm 1 introduced in Section 5 and applied the filter discussed in Section **??**. 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 6.1. The average query processing time per query of RelSim (PathSim) over DBLP and BioMed dataset are 0.034 (0.024) and 0.511 (0.477) seconds, respectively. 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.

Since the complexity of RelSim is exponentially large, we evaluate the scalability of RelSim after applying our proposed optimization as follows. Since the complexity of RelSim depends on the number of given constraints and the length of input simple pattern, we measure the running time of the algorithm by fixing one of the components while varying the remaining component to analyze how each component affects the running time. We test our algorithm over randomly generated constraints whose numbers of atoms are between 2 and 5. We measure the running time by setting the number of constraints to 1, 5, 10, 20 and 40, and vary the length of an input simple pattern, p, between 4 and 10. The length of a simple pattern is defined as the number of labels in the pattern. We report the average running times of RelSim with proposed optimization per query over 5 runs of each settings in Figure 6.

7. RELATED WORK

The notion of inverse of a schema mapping has been extensively studied in relational data exchange. Since some schema mappings lose significant amount of information of the original database, it is *not* possible to define an inverse for them such that it fully restores the information of the original database. Thus, researchers have proposed less restrictive notions of inverse, which may *not* fully recover the mapped information. A *maximum-recovery* inverse of

a mapping recovers as much source information as possible from the mapped data [4]. Arenas et al. further define the notion of maximum recovery based on classes of queries on the source database: a L-maximum (L-full) recovery inverse of a mapping Σ recovers the maximum possible amount (all) of information that can be retrieved by query language L over the source database [3]. Generally, the less expressive L is, the less information a L-maximum recovery inverse recovers. Our goal is to find structural variations over which a similarity algorithm returns the same results. Thus, the information related to similarity queries must be available to the similarity search algorithm on all structural variations. Thus, our notion of inverse is most similar to the Fagin-inverse for relational data. Further discussion on the related work is in Section A of the appendix.

8. ADDITIONAL AUTHORS

9. REFERENCES

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases: The Logical Level*. Addison-Wesley, 1995.

[2] I. Antonellis, H. Garcia-Molina, and C. Chang. Simrank++: Query Rewriting through Link Analysis of the Click Graph. In *VLDB*, 2008.

[3] M. Arenas, J. Pérez, J. L. Reutter, and C. Riveros. Inverting schema mappings: Bridging the gap between theory and practice. *PVLDB*, 2(1):1018–1029, 2009.

[4] M. Arenas, J. Pérez, and C. Riveros. The recovery of a schema mapping: Bringing exchanged data back. *ACM Trans. Database Syst.*, 34(4):22:1–22:48, Dec. 2009.

[5] P. Barcelo, J. Perez, and J. Reutter. Schema Mappings and Data Exchange for Graph Databases. In *ICDT*, 2013.

[6] C. Beeri and M. Y. Vardi. A proof procedure for data dependencies. *J. ACM*, 31(4), Sept. 1984.

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

[8] A. Borodin, G. Roberts, J. S. Rosenthal, and P. Tsaparas. Link Analysis Ranking: Algorithms, Theory, and Experiments. *ACM Trans. Inter. Tech.*, 5(1), 2005.

[9] Y. Chodpathumwan, A. Aleyasen, A. Termehchy, and Y. Sun. Towards representation independent similarity search over graph databases. In *CIKM*, 2016.

[10] I. Cruz, A. Mendelzon, and P. Wood. A graphical query language supporting recursion. In *SIGMOD*, page 323330, 1987.

[11] E. Codd. Does Your DBMS Run By the Rules? *ComputerWorld*, 1985.

[12] R. Fagin. Inverting schema mappings. *ACM Trans. Database Syst.*, 32(2), 2007.

[13] R. Fagin, P. G. Kolaitis, L. Popa, and W.-C. Tan. Composing schema mappings: Second-order dependencies to the rescue. *ACM Trans. Database Syst.*, 30(4):994–1055, Dec. 2005.

[14] R. Fagin, P. G. Kolaitis, L. Popa, and W. C. Tan. Quasi-inverses of schema mappings. *ACM Trans. Database Syst.*, 33(2), 2008.

[15] R. Fagina, P. Kolaitis, R. Miller, and L. Popa. Data exchange: semantics and query answering. *TCS*, 336(1), 2005.

[16] W. Fan and P. Bohannon. Information Preserving XML Schema Embedding. *TODS*, 33(1), 2008.

[17] W. Fan, Y. Wu, and J. Xu. Functional dependencies for graphs. In *SIGMOD*, SIGMOD '16, pages 1843–1857, New York, NY, USA, 2016. ACM.

[18] N. Francis and L. Libkin. Schema mappings for data graphs. In *PODS*, PODS '17, pages 389–401, New York, NY, USA, 2017. ACM.

[19] B. Ghadiri Bashardoost, C. Christodoulakis, S. Hassas Yeganeh, R. J. Miller, K. Lyons, and O. Hassanzadeh. Vizcurator: A visual tool for curating open data. In *WWW*, 2015.

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

[21] G. He, H. Feng, C. Li, and H. Chen. Parallel SimRank Computation on Large Graphs with Iterative Aggregation. In *KDD*, 2010.

[22] J. Heer, J. Hellerstein, and S. Kandel. Predictive interaction for data transformation. In *CIDR*, 2015.

[23] A. Hernich, L. Libkin, and N. Schweikardt. Closed world data exchange. *ACM Trans. Database Syst.*, 36(2):14:1–14:40, June 2011.

[24] J. Hopcroft and R. Tarjan. Algorithm 447: Efficient algorithms for graph manipulation. *Commun. ACM*, 16(6):372–378, June 1973.

[25] R. Hull. Relative Information Capacity of Simple Relational Database Schemata. 1984.

[26] G. Jeh and J. Widom. Simrank: A measure of structural-context similarity. In *KDD*, 2002.

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

[28] S. Kandel, A. Paepcke, J. Hellerstein, and J. Heer. Wrangler: Interactive visual specification of data transformation scripts. In *CHI*, 2011.

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

[30] C. Manning, P. Raghavan, and H. Schutze. *An Introduction to Information Retrieval*. Cambridge University Press, 2008.

[31] S. Melnik, A. Adya, and P. A. Bernstein. Compiling mappings to bridge applications and databases. In *SIGMOD*, 2007.

[32] A. Ng, A. Zheng, and M. Jordan. Stable Algorithms for Link Analysis. In *SIGIR*, 2001.

[33] J. Picado, A. Termehchy, A. Fern, and P. Ataei. Schema independent relational learning. In *Proceedings of the 2017 ACM International Conference on Management of Data*, SIGMOD '17, pages 929–944, New York, NY, USA, 2017. ACM.

[34] C. Shi, X. Kong, Y. Huang, S. Y. Philip, and B. Wu. Hetesim: A general framework for relevance measure in heterogeneous networks. *TKDE*, (10), 2014.

[35] Y. Sun, J. Han, X. Yan, S. P. Yu, and T. Wu. PathSim: MetaPath-Based Top-K Similarity Search in Heterogeneous Information Networks. In *VLDB*, 2011.

[36] J. Tang, C. Li, and Q. Mei. Learning representations of large-scale networks. In *KDD*, 2017.

[37] A. Termehchy, M. Winslett, Y. Chodpathumwan, and A. Gibbons. Design Independent Query Interfaces. *TKDE*, 2012.

[38] H. Tong and C. Faloutsos. Center-piece subgraphs: Problem definition and fast solutions. In *KDD*, 2006.

[39] H. Tong, C. Faloutsos, and J. Pan. Fast random walk with restart and its applications. In *ICDM*, 2006.

[40] H. Tong, H. Qu, and H. Jamjoom. Measuring Proximity on Graphs with Side Information. In *ICDM*, 2008.

[41] B. Q. Truong, S. S. Bhowmick, and C. Dyreson. *SINBAD: Towards Structure-Independent Querying of Common Neighbors in XML Databases*, pages 156–171. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[42] M. Vardi. The universal-relation data model for logical independence. *IEEE Software*, 5, 1988.

[43] M. Y. Vardi. On decomposition of relational databases. In *FOCS*, pages 176–185, Nov 1982.

[44] G. VegaGorgojo, M. Giese, and L. Slaughter. Exploring semantic datasets with rdf surveyor. In *ISWC*, 2017.

[45] V. V. Williams. Breaking the coppersmith-winograd barrier. *E-mail address: jml@ math. tamu. edu*, 2011.

[46] P. T. Wood. Query languages for graph databases. *SIGMOD Rec.*, 41(1), Apr. 2012.

[47] S. H. Yeganeh, O. Hassanzadeh, and R. J. Miller. Linking Semistructured Data on the Web. In *WebDB*, 2011.

[48] W. Yu and X. Lin. IRWR: Incremental Random Walk with Restart. In *SIGIR*, 2012.

[49] J. Zhang, J. Tang, C. Ma, H. Tong, Y. Jing, and J. Li. 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, pages 1445–1454, New York, NY, USA, 2015. ACM.

[50] P. Zhao, J. Han, and Y. Sun. P-Rank: A Comprehensive Structural Similarity Measure over Information Networks. In *CIKM*, 2009.

APPENDIX

.1 The case of 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., $\phi(\bar{x}) \rightarrow \phi(\bar{x})$. As we have simplified our analysis in Section 3.2 where the conclusion of a constraint is an atom of a single label, e.g., (x_i, a, x_j) where $x_i, x_j \in \bar{x}$ and a is schema label. That is a trivial constraint is of the form $(x_i, a, x_j) \rightarrow (x_i, a, x_j)$. 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 are meaningless in term of putting restriction in a database, we ignore any occurrence of trivial constraints and treat them as if they do not exist.

In relational database, it has been proved by Hull that there is *not* any information-preserving variation of a relational schema without any constraint beyond simple renaming of the scheme elements [25]. We show that this result also holds for graph databases.

THEOREM 2. *Given two schemas $S = (\mathcal{L}_S, \Gamma_S)$ and $T = (\mathcal{L}_T, \Gamma_T)$ where $\Gamma_S = \Gamma_T = \emptyset$, if there exists Σ_{ST} such that $S \overset{\Sigma_{ST}}{\equiv} T$ then $S = T$ or there is a bijection between $\mathcal{L}_S$ and $\mathcal{L}_T$.*

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

Similar to the idea of trivial constraint, there always exists a transformation from a schema S to schema T, $T \equiv S$, whose transformation constraints are also trivial. We call such transformation an *identity* transformation. 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 $\{(x_1, a, x_2) \wedge (x_2, b, x_3) \rightarrow (x_1, c, x_3), (x_1, l, x_2) \rightarrow (x_1, l, x_2), l \in S\}$ is information preserving. 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 cannot produce an RRE $\lceil\lceil a \cdot b \rfloor\rfloor$ over S which is mapped to c over T because the expression involves two separate constraints. In fact, with only a trivial constraint, our algorithms do not modify an input expression.

Further, with the assumption that a constraint γ^* exists for each constraint γ in a given schema, we have the following theorem regarding constraints of T.

THEOREM 3. *Given two schemas $S = (\mathcal{L}_S, \Gamma_S)$ and $T = (\mathcal{L}_T, \Gamma_T)$ where $\Gamma_S = \emptyset$, if there exists Σ_{ST} such that $S \overset{\Sigma_{ST}}{\equiv} T$ then there exists a bijection between $\mathcal{L}_S$ and some $L' \subseteq \mathcal{L}_T$ where there is no constraint whose conclusion contains label in L', and for each $l \in \mathcal{L}_T \setminus L'$, there exists a constraint $\lambda(\bar{x}) \rightarrow (x_1, l, x_2)$ in Γ_T where l does not appear in λ, for some CRPQ f.*

Following Theorem 3, we have that if a schema S contains no constraint, every information preserving transformation from S preserves all edges (or up-to renaming of those edges). We call this type of transformation an *easy* transformation, and they are information preserving.

COROLLARY 2. *Every transformation from a schema without a constraint is easy.*

THEOREM 4. *Every easy transformation is information preserving.*

Based on Theorem 3 and Corollary 2, to avoid the issue of trivial constraints. Our proposed algorithms should ignore producing any expression 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. S is information equivalent 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 1, we have that an expression l over S is mapped to an expression $r : \lceil\lceil x_1 \hookrightarrow_{G_\phi} x_2 \rfloor\rfloor$. However, r is not a simple

expression 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 disallowed or every l should be replaced with $x_1 \hookrightarrow_{G_\phi} x_2$ that does not contains a skip-operation.

.2 Filtering Constraints Based on Conclusions

We have simplified the usage of our framework in Section 5 so that users can take advantage of it by submitting only simple patterns. Using the 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 provides a method to reduce the set of RREs in order to improve the efficiency of RelSim while ensuring its effectiveness and robustness.

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 information-preserving 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 information-preserving 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 information-preserving 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}.*

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. Further details of discussion about *easy* transformation and the filtering method for them can be found in Appendix 7.

Some constraint may induce an information-preserving 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 information-preserving 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 information-preserving. 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 even though the constraint is not trivial or of the form discussed in the end of Section 5. 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 should ignore producing an RRE such as $\texttt{published-in} \cdot \texttt{published-in}^-$. However, an RRE such as $\texttt{area} \cdot \texttt{published-in}$ 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.

A. ADDITIONAL RELATED WORK

One generic approach to achieve schematic robustness is to define a *universal schema* to which all possible representations of a database can be transformed and use and/or develop algorithms that are effective over this representation. Nevertheless, the experience gained from the idea of universal relation, indicates that such representation does not often exist [1, 42]. One also has to transform their database to the universal representation, which may be quite complex considering the intricacies associated with defining such a representation and not practical for a large database.

Researchers have also analyzed the stability of random walk algorithms in graphs against relatively small perturbations in the data [32, 20, 8]. We also seek to instill robustness in graph mining algorithms, but we are targeting robustness in a new dimension: robustness in the face of variations in the representation of data. Researchers have provided systems that help users with transforming and wrangling their data [28, 22, 19, 47]. We also address the problem of data preparation but using a difference approach: *eliminating the need to wrangle the data.*

Researchers have defined schema mappings over graph databases as constraints in some graph query language in the context of data exchange [5]. We focus on evaluating the robustness of similarity search algorithms rather than traditional questions in schema mapping and data exchange, such as computing the transformed database instances.

Researchers have proposed a keyword query interfaces over XML dataset that returns the same answers across normalized and denormalized XML databases with equivalent information content [37]. Authors in [9] have presented a similarity search algorithm over graph databases that is robust over a data variation of representing relationships between entities as a node or edge in a database. Our proposed method goes beyond robustness over a fixed variation and provides a way to develop similarity search methods that is robust across a wide variety of structural variations. We also propose a usable interface for using such a system.