Structural Generalizability: The Case of Similarity Search

Yodsawalai Chodpathumwan*
yodsawalai.c@tggs.kmutnb.ac.th
King Mongkut's University of Technology North Bangkok

Arash Termehchy
termehca@oregonstate.edu
Oregon State University

Stephen A. Ramsey
stephen.Ramsey@oregonstate.edu
Oregon State University

Aayam Shresta
shrestaa@oregonstate.edu
Oregon State University

Amy Glen
glena@oregonstate.edu
Oregon State University

Zheng Liu
liuzhen@oregonstate.edu
Oregon State University

ABSTRACT

Supervised and Unsupervised ML algorithms are widely used over graphs. They use the structural properties of the data to deliver effective results. It is known that the same information can be represented under various graph structures. Thus, these algorithms may be effective on some structural variations of the data and ineffective on others. One would like to have an algorithm that is effective and generalizes to all structural variations of a data graph. We define the concept of structural generalizability for algorithms over graphs. We focus on the problem of similarity search, which is a popular task and the building block of many ML algorithms on graphs, and propose a structurally generalizable similarity search algorithm. As this algorithm may require users to specify features in a rather complex language, we modify this algorithm so that it requires only simple guidance from the user. Our extensive empirical study show that our algorithms are structurally generalizable while being efficient and more effective than current algorithms.

CCS CONCEPTS

• **Information systems → Graph-based database models**; **Data mining**.

KEYWORDS

Structural Variations, Structural Generalizability, Similarity Search

ACM Reference Format:
Yodsawalai Chodpathumwan, Arash Termehchy, Stephen A. Ramsey, Aayam Shresta, Amy Glen, and Zheng Liu. 2021. Structural Generalizability: The Case of Similarity Search. In *Proceedings of the 2021 International Conference on Management of Data (SIGMOD '21), June 20–25, 2021, Virtual Event, China.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3448016.3457316

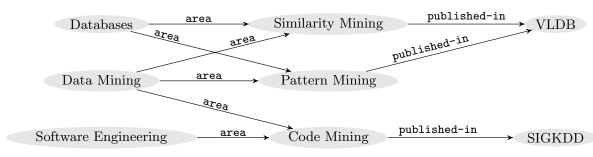
1 INTRODUCTION

ML on Graphs. Unsupervised and supervised machine learning (ML) methods are widely used over graph data [29, 43–45, 47, 48, 53, 57]. To deliver effective predictions, these methods usually leverage structural or topological features of the data that quantify relationships between entities. For example, consider the DBLP dataset (*dblp.uni-trier.de*) that stores information about papers, conferences, and research areas in computer sciences whose fragments are shown in Figure 1(a). Assume that a user wants to find the most similar research area to *Data Mining* in this dataset according to their conferences and publications. Similarity search algorithms usually use the structure of the graph to measure the degree of similarity between entities in the data. For example, SimRank [29] is a well-known unsupervised similarity search algorithm over graphs that quantifies the similarity between two entities according to how likely two random surfers will meet each other if they start from the two entities [29, 43, 47, 48, 53, 57]. Thus, it correctly finds *Data Mining* to be more similar to *Databases* than to *Software Engineering* in Figure 1(a). Oversimplifying a bit, since *Data Mining* and *Databases* share relatively more neighbors, i.e., papers, over this dataset, they have a relatively high SimRank score.

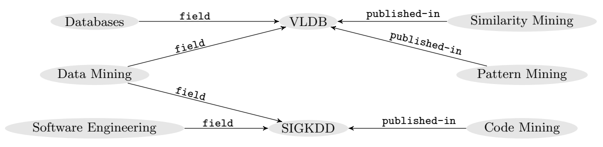
Structural Variation & Heterogeneity. It is established that different datasets usually represent essentially the same information under different forms and structures [1, 4, 8, 10, 11, 16, 17, 21, 28, 35, 46, 55]. A well-known example of such structural variations is (de-)normalization in relational and XML data where the original and normalized databases represent the same information under different structures [1, 3, 4, 46, 55]. As another example, consider the SIGMOD Record bibliographic dataset (*sigmod.org/publications*) whose fragments are shown in Figure 1(b). Intuitively, this dataset represent essentially the same information about the same set of *paper*, *conference*, and *research area* entities as the fragment of DBLP in Figure 1(a). Figure 2 depicts the schematic fragments of the DBLP and SIGMOD Record datasets shown in Figure 1. As Figure 2 illustrates, each dataset has its own way of representing these entities and their relationships. For example, DBLP connects directly each paper to its research areas and conferences. Given that all papers in a conference share the same set of research areas, one can choose the structure of SIGMOD Record to represent this information and connect research areas of a paper to its conference. This dataset represents the relationship between the research area of a paper using a path through the conference of the paper instead of a direct link as in Figure 1(a). As another example, Figure 3 shows fragments of WSU course database (*cs.washington.edu/research/xmldatasets*) and Alchemy UW-CSE database (*alchemy.cs.washington.edu/data/uw-cse*). These two databases represent the same information about courses, instructors and course offerings in a university under two

*Works done while at the University of Illinois at Urbana-Champaign

SIGMOD '21, June 20–25, 2021, Virtual Event, China
© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-8343-1/21/06…$15.00
https://doi.org/10.1145/3448016.3457316

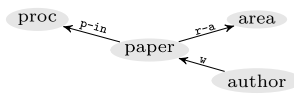


(a) Fragments of DBLP

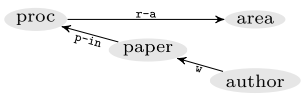


(b) Fragments of SIGMOD Record

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

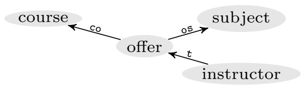


(a) DBLP

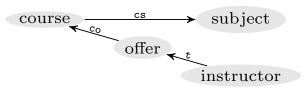


(b) SIGMOD Record

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



(a) WSU



(b) Alchemy UW-CSE

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

different structures. We work with bioinformatics experts to analyze large graph datasets from various data sources and have observed that they often restructure their graph data to meet certain efficiency-oriented or data quality goals. For instance, if two nodes are far apart in the data and are often queried together in some lookup queries, the experts add new edges that connect these entities directly to run lookup queries faster. This modification does *not* add any new information to the database as the new edge can be inferred from the (long) path between the corresponding entities.

Generalizability Over Structural Variations. As structural or topological features may deliver different values across the different structural variations of the same information, ML methods that rely on such features may return inaccurate results on some structures. For example, SimRank finds *Data Mining* more similar to *Software Engineering* than *Databases* on the data fragment in Figure 1(b), which is clearly not accurate. Thus, the accuracy of SimRank depends on the form under which the information is presented rather than its content. Due to the constant data evolution [20, 23, 33, 41, 52] and growing need for analyzing different datasets with a great deal of structural variations [6, 7, 10, 19, 21, 22, 35, 42], similarity features and models are often used across datasets with different forms and structures. Thus, the accuracy of current similarity features and models is unstable and unpredictable, e.g., they may perform well on the datasets used to develop or train these features but poorly on others. It has been a central problem in ML and data analysis to measure or improve the robustness and generalizability of features and models against variations in the content of datasets [31, 36, 38, 39]. For the same reason, it is also important to ensure that similarity models and algorithms generalize to and are effective on *unseen structures*, i.e., the structure of the information other than the ones used to develop or train features and models.

Our Contributions. In this paper, we define structural generalizability and robustness over a wide range of structural variations in graph data and investigate the generalizability of similarity features and algorithms in the face of such structural variations. We focus on the problem of similarity search as it is both a popular type of queries over graphs and an important building block of other ML models and data analysis tasks, such as pattern query matching, community detection, and clustering [2, 47, 48, 57]. We propose novel methods to similarity search that are robust against a wide range of structural variations in graphs. Instead of developing new methods from scratch, we build our method on top of the current similarity search approaches. This approach will make it easier to achieve robustness in current data analytics systems. In particular, we make the following contributions.

- We define structural generalizability and robustness of a similarity search algorithm (Section 3). We build a theoretical framework that captures a sufficiently general notion of structural generalizability and a wide variety of structural variations on graphs.
- We show that existing similarity search methods are not robust and do not generalize against structural variations because of the limited types of relationships they use to measure the degree of similarity. We propose a robust algorithm called *RelSim*, which extends a well-known similarity search algorithm (*PathSim* [40, 43]) by using a sufficiently expressive set of patterns. We establish a relationship between the structural robustness of similarity search methods and the expressivity of the languages they use to specify heuristics and quantify the similarity of nodes, e.g., the language that expresses the types of paths used for random walks. We show that, roughly speaking, the more expressive the language is, the more generalizable the similarity algorithm is. Interestingly, this is in contrast with the generalizability of ML models against variations in the data content where an expressive model is harder to generalize than a less expressive one [36].
- It is very time-consuming to compute features represented in an overly expressive language over graphs [24, 54]. we propose a language that is sufficiently expressive to generalize over popular variations and its expressed features are computed efficiently.
- RelSim requires users to specify the relationship between nodes to compute their similarity score using the aforementioned proposed language. As it may be hard for users to work with such a complex language, we propose an algorithm that leverages simple and high-level 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 algorithms are efficient over large data.

The proofs of our theoretical results are in [12].

2 GRAPH STRUCTURE AND CONSTRAINTS

Basic Notations. 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 [8, 18, 54]. We denote an edge from node u to node v whose label is a as (u, a, v). We say that $(u, a, v) \in D$ $(u, v \in V)$ whenever $(u, a, v) \in E$.

Schema and Constraints. A schema S is a pair $(\mathcal{L}, \Gamma_S)$ in which $\mathcal{L}$ and Γ_S are (finite) sets of labels and constraints, respectively. Constraints restrict the instances of a schema. They are usually expressed as logical formulas over schemas [1, 9, 10]. Two widely used type of constraints are **tuple-generating** and **equality-generating** dependencies [9, 10]. A tuple-generating dependency (*tgd* for short) over schema $\mathcal{L}$ is in the form of $\forall\bar{x}(\phi(\bar{x}) \rightarrow \exists\bar{y}\psi(\bar{x}, \bar{y}))$ where $\bar{x}$ and $\bar{y}$ are sets of variables, and ϕ and ψ are logical formulas in a query language over $\mathcal{L}$. An equality-generating dependency (*egd* for short) over schema $\mathcal{L}$ is in the form of $\forall\bar{x}(\phi(\bar{x}) \rightarrow x_1 = x_2)$ where $\bar{x}$ is a set of variables, $x_1, x_2 \in \bar{x}$, and ϕ is logical formulas in a query language over $\mathcal{L}$.

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

Tgds and egds are arguably the most popular and frequently used types of database constraints and generalize popular constraints, such as function and multi-valued dependencies [1]. 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 *instance* (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 $\texttt{Inst}(S)$.

Graph Query Languages. A commonly studied query language over graph databases is **conjunctive regular path queries** (conjunctive RPQ), which is used to express constraints over graph databases [8, 13, 21, 54]. The **RPQ** p over schema $\mathcal{L}$ is defined as

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

in which ϵ is an empty label, $^-$ is a reverse traversal of an edge, $\cdot$ is a concatenation, $+$ is a disjunction, and $*$ is a Kleene star. To avoid parentheses and ambiguity, it is assumed that the reverse traversal has the highest priority, then Kleene star, then concatenation and then disjunction. Example of an RPQ over a schema of a database shown in Figure 1(b) is $\texttt{field} \cdot \texttt{published-in}^-$. The RPQ p defines a binary relation over database nodes. More precisely, the result of evaluating p on database D is a set of pairs of nodes in D such that there is a path defined by p between the two nodes. We denote the result of evaluating p over D as $[[p]]_D$. For example, given label a in the schema of D, the result of $[[a]]_D$ is a set of pairs of nodes $\{(u, v)\}$ where there is an edge with label a from u to v. Let $\bar{x} = (x_1, \ldots, x_n)$ and $\bar{y} = (y_1, \ldots, y_m)$ be tuples of distinct variables. A conjunctive RPQ is a formula $\phi(\bar{x})$ of the form $\exists\bar{y}((z_1, p_1, z'_1) \wedge \ldots \wedge (z_k, p_k, z'_k))$ where p_i is an RPQ and $z_i, z'_i \in \{x_1, \ldots, x_n, y_1, \ldots, y_m\}$, $1 \leq i \leq k$ [8, 54]. (z_i, p_i, z'_i) is an **atom** of $\phi(\bar{x})$.

Similarity Queries. A query q over database $I(V_I, E_I) \in \texttt{Inst}(S)$ is a node id in V_I [2, 25, 29, 30, 40, 43, 49, 56, 58]. The answers to q over I is a ranked list of node ids in I.

3 GENERALIZABILITY AND VARIATIONS

3.1 Structural Generalizability

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

This definition of transformations may *not* be sufficiently powerful to capture structural variations. Assume 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 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 [8, 16]. It maps each database $I \in \texttt{Inst}(S)$ to at least one database $J \in \texttt{Inst}(T)$. One denotes the fact that J is a transformation of I under Σ_{ST} as $(I, J) \models \Sigma_{ST}$ [8, 16]. For brevity and by the abuse of notation, we show the transformed databases of I under Σ_{ST} as $\Sigma_{ST}(I)$.

3.1.2 Invertible Transformations. 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., $\Sigma_{TS}(\Sigma_{ST}(I))$ is exactly I. In other words, the composition of Σ_{ST} and Σ_{TS}, shown as $\Sigma_{TS} \circ \Sigma_{ST}$ is equivalent to the identity transformation *id* that maps each database only to itself. 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 $\Sigma_{ST}(I)$. That is, each database in $\Sigma_{ST}(I)$ has at least the same amount of information as I. We say that S has at least as much information as T and denote their relationship as $S \preceq_{\Sigma_{ST}} T$ or simply $S \preceq T$ if Σ_{ST} is clear from the context.

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

3.1.3 Structurally Generalizable Algorithms.

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

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

An algorithm is generalizable over a set of transformations if it is generalizable over all its members.

3.2 Structural Variations

3.2.1 Expressing Transformations.

To characterize the structural variations of a schema, one needs to express invertible transformations. Researchers usually use **declarative (schema) mappings** to express schematic variations in graph and relational databases [8, 14, 16]. Roughly speaking, a transformation between schemas S and T is expressed as a set of logical formulas $\phi_S(\bar{x}) \rightarrow \psi_T(\bar{y})$ where $\phi_S(\bar{x})$ and $\psi_T(\bar{y})$ are queries over schemas S and T, respectively. More precisely, transformation Σ_{ST} between graph schemas S and T is a finite set of rules $\phi_S(\bar{x}) \rightarrow \psi_T(\bar{y})$ such that $\bar{x} \subseteq \bar{y}$ and $\phi_S(\bar{x})$, i.e., **premise**, and $\psi_T(\bar{y})$, i.e., **conclusion**, are conjunctive RPQs over S and T, respectively [8]. In each rule $\phi_S(\bar{x}) \rightarrow \psi_T(\bar{y})$, every variable in $\bar{x}$ is universally quantified and every one in the set of $\bar{y}$ either belongs to $\bar{x}$ or is existentially quantified.

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

Transformation Σ_{ST} maps each database $I \in \texttt{Inst}(S)$ to $J \in \texttt{Inst}(T)$ if for each rule $\phi_S(\bar{x}) \rightarrow \psi_T(\bar{y})$ in Σ_{ST}, we have $\bar{u} \in [[\psi_T(\bar{x})]]_J$ if $\bar{u} \in [[\phi_S(\bar{y})]]_I$ [8]. We use the closed world semantic for schema mappings [26]. That is, transformation Σ_{ST} maps $I \in \texttt{Inst}(S)$ to only databases whose nodes and edges are constructed using the mapping rules. The alternative semantic is the open world semantic in which the transformations of $I \in \texttt{Inst}(S)$ under Σ_{ST} may contain nodes and edges that are *not* created as the result of the schema mapping rules [8, 14, 16]. A consequence of this semantic is that the inverse of transformation Σ_{ST} will map each database $\Sigma_{ST}(I)$ to other databases in addition to I, e.g., all databases in S that include at least the same amount of information as I [14]. Our goal, however, is to compare the results of similarity queries over the original database I and its variations. Thus, we use the closed world semantic for schema mappings.

3.2.2 Information Preservation.

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

Given transformations Σ_{ST} from schema S to T, its inverse Σ_{ST}^{-1} is a transformation from T back to S. Thus, the composition $\Sigma_{ST}^{-1} \circ \Sigma_{ST}$ will map the database $I \in \texttt{Inst}(S)$ to (only) itself. In other words, the composition of $\Sigma_{ST}^{-1} \circ \Sigma_{ST}$ 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 $S \preceq_{\Sigma_{ST}} T$, 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 [51] and the ones on the inverse of a relational schema mapping in [14]. According to Proposition 1, the constraints on schema S determine its structural variations. Following this proposition, the constraints on S that give rise to invertible structural variations are in tgds.

Example 4. *The composition $\Sigma_{1a,1b}^{-1} \circ \Sigma_{1a,1b}$ in Example 3 results in a constraint $(x_1, \texttt{area}, x_4) \land (x_4, \texttt{published-in}, x_3) \land (x_2, \texttt{published-in}, x_3) \rightarrow (x_1, \texttt{area}, x_2)$ 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 [8, 15]. Roughly speaking, each rule in $\Sigma_{TR} \circ \Sigma_{ST}$ is created by replacing an atom (x, exp, y) in the premise of a rule in Σ_{TR} by the premise of a rule in Σ_{ST} whose conclusion matches (x, exp, y) [8, 10, 15]. Assume that the conclusion of a rule in Σ_{ST} contains an atom (x, exp, y) with an existentially quantified variable, which may be stated explicitly in the rule or introduced by using concatenation, Kleene star, or disjunction, i.e., +, in exp. Also, assume that there is an atom in the premise of Σ_{ST}^{-1} that matches (x, exp, y). In this case, one needs second-order logic to express this composition [15]. If this happens to a transformation and its inverse, then the set of constraints on the original schema will be tgds in second order logic. To the best of our knowledge, there has *not* been any work on database constraints over graph (or relational) databases that are in languages more expressive than the first order logic, e.g., second order logic, and the database constraints in the first order logic are by far more widely used than the ones expressed in higher order logics [1, 10]. Thus, we focus our attention to the first order constraints as explained in Section 2. If the atoms in the premise of Σ_{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 [8, 15]. Therefore, we consider only transformations whose composition with their inverses can be expressed in first order logic.

As shown in Example 4, the composition of transformation $\Sigma_{1a,1b}$ and its inverse, and transformation $\Sigma_{1a,1c}$ and its inverse can be expressed as such tgds. In this case the shared atoms between the premises of rules in Σ_{ST}^{-1} and the conclusions of rules in Σ_{ST} will be in form of (x, l, y) or (x, l^-, y) where l is a label in schema T. Thus, each rule in $\Sigma_{ST}^{-1} \circ \Sigma_{ST}$ 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^-, y) by exchanging the positions of x and y). This method naturally extends to the atoms of the form (x, l^-, 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 the structure of similar types of inverse of schema mappings over relational databases [14]. 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. A tgd without any existential variable in its conclusion is a *full tgd*. Hence, $\Sigma_{ST}^{-1} \circ \Sigma_{ST}$ is a set of full tgds over S.

The set of tgds introduced by Proposition 1 is necessary to have invertible transformations for schema S, but it is *not* sufficient. We show that S must satisfy an additional group of tgds to have invertible variations. Let σ denote the set of tgd constraints in $\Sigma_{ST}^{-1} \circ \Sigma_{ST}$. 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 $(x, l', y) \rightarrow \texttt{FALSE}$.

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 invertible transformations simply as transformations. Table 1 summarizes the notations used in our framework.

Table 1: Summary of the notations.

Notation	Definition
$S = (\mathcal{L}, \Gamma)$	Schema S with label set $\mathcal{L}$ and constraints Γ
$\texttt{Inst}(S)$	Databases of schema S
$[[p]]_D$	Results of evaluating pattern p on database D
Σ_{ST}	Transformation from schema S to T
$\Sigma_{ST}(I)$	Transformations of I based on Σ_{ST}
$\phi_S(\bar{x}) \rightarrow \psi_T(\bar{y})$	A transformation rule.
Σ_{ST}^{-1}	The inverse of Σ_{ST}
$\Sigma_{TR} \circ \Sigma_{ST}$	The composition of transformations

4 GENERALIZABLE SIMILARITY SEARCH

4.1 Generalizability of Current Methods

4.1.1 Similarity Search Methods. To the best of our knowledge, frequently used structural-based similarity search features and algorithms are based on random walks, e.g., RWR [48], pairwise random walk, e.g., SimRank [29] and P-Rank [58], or path-constrained framework, e.g., PathSim [43] and HeteSim [40]. There are also other similarity search algorithms that extend the aforementioned algorithms such as common neighbors, $Katz_\beta$ measure, commute time, and sampled set of random paths/walks between nodes [32, 57]. The algorithms that are not path-constrained, e.g., SimRank and RWR, are mainly developed to measure similarity over graphs where all edges have the same label [29, 48]. However, these algorithm are sometimes used over data graphs with multiple edge labels [43]. They have also been used as a basis for methods developed for data graphs with multiple labels [40, 43]. We consider both the original and the extended versions of SimRank and RWR for completeness.

4.1.2 Topology-Based Methods. Similarity scores computed by algorithms that use random walks and pairwise random walks are largely influenced by the topology of the graph. Because some transformations may modify the topological structure of data, 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. There are similar algorithms that use the idea of random walks by randomly picking some walks between two nodes and randomly traversing certain number steps on each walk [57]. The more similar two nodes are, the more likely it is for one to reach to one of them by starting from another one using the aforementioned method. As they use similar core ideas to RWR and SimRank, their results are also influenced by transformations that modify the data topology. An invertible transformation may change the length and the number of paths and walks between two nodes. For example, the length of the path between nodes *VLDB* and *Pattern Mining* is one in Figure 1(a) and two in its variation (Figure 1(b)). Thus, they may deliver different similarity scores for the same pairs of nodes in a database and its invertible transformations.

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

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

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 methods largely depend on the representation of the underlying relationships.

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

4.2 Achieving Generalizability

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

On the other hand, the user may like to measure the similarity of research areas based on their shared conferences and also the papers published in those conferences. Over Figure 1(a), she can use pattern p_1 to help measure the similarity. However, there is no pattern that expresses that relationship in Figure 1(b). One may solve this problem by using a language that is more expressive than the sequence of relationship labels to express relationship types between entities in a database.

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

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

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

4.2.2 Toward a Sufficiently Expressive Pattern Language. The aforementioned insight suggests that to achieve generalizability against structural variations, the similarity search algorithm should use more complex patterns to express relationships between and measure the similarity of entities than the current ones. Nevertheless, one should be careful not to increase the expressivity of the relationship language too much as it takes a long time to find all instances of a complex pattern and compute its similarity score in a large database. We present a relationship expressing language that is expressive enough to represent equivalent relationships across various representations of the same datasets. We also show that using this 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 language is structurally generalizable.

To implement the solution presented in Example 7, one may use the idea of nested operation in the nested regular expression (NRE) language [8]. Let $[p]$ denote a nested path of p where a path $(u, [p], u)$ exists if and only if there exists a node v such that a path (u, p, v) exists. To achieve the same results as p_1 over Figure 1(a), the user should use the pattern p_4 : `field`· [`pub-in`$^-$] · [`pub-in`$^-$] · `field`$^-$ to compute a similarity score between research areas. That is, similar research areas are based on shared conferences, and the strength of this relationship is based on the number of publications published in those conferences. Now, we define our extension to NRE called **rich-relationship expression** (RRE), on schema S as

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

where [ ] and $\lceil\lceil\ \rfloor\rfloor$ are **nested** and **skip** operations, respectively.

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

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

$$\mathcal{I}_D(a) = \{(u, v, \langle u, a, v \rangle) \mid (u, a, v) \in D\}$$
$$\mathcal{I}_D(p^-) = \{(v, u, \bar{s}) \mid (u, v, s) \in \mathcal{I}_D(p)\}$$
$$\mathcal{I}_D(p_1 \cdot p_2) = \{(u, v, s_1 \bullet s_2) \mid \forall w, (u, w, s_1) \in \mathcal{I}_D(p_1) \text{ and } (w, v, s_2) \in \mathcal{I}_D(p_2)\}$$

The rules for patterns with other constructs are in Appendix A.

We define the **set of instances of an RRE** for a particular pair of nodes u and v in database D as

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

If database D is clear from the context, we may write $\mathcal{I}_D^{u,v}(p)$ and $\mathcal{I}_D(p)$ as $\mathcal{I}^{u,v}(p)$ and $\mathcal{I}(p)$, respectively. For the remaining of this paper, we assume all relationship patterns are RREs. We have the following result regarding the number of instances of RRE across structural variations of graphs. We prove the following theorem by induction on the number of disjunctions and concatenated labels in a given RRE pattern.

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

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

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

4.2.3 Relationship Between Transformations & Pattern Languages. 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 edges (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.

A cyclic premise allows multiple traversals from one variable to another in the premise, and requires an indicator in the relationship language whether two variables along the traversal are the same, e.g., starting and ending nodes in a cycle are the same. In this case, it is *not* possible to rewrite this pattern over the premise to an equivalent one without a conjunction ($\wedge$). For instance, consider the pattern representing the premise of a cyclic tgd $(x_1, \mathsf{a}, x_2) \wedge (x_2, \mathsf{b}, x_3) \wedge (x_3, \mathsf{c}, x_4) \wedge (x_1, \mathsf{d}, x_3) \wedge (x_2, \mathsf{e}, x_4) \rightarrow (x_1, \mathsf{f}, x_4)$. Assume we want to rewrite the pattern over this premise similar to (x_1, exp, x_4) for some RRE exp. To remove the conjunction in $(x_1, \mathsf{a}, x_2) \wedge (x_2, \mathsf{b}, x_3)$, one may write $(x_1, \mathsf{a} \cdot \mathsf{b}, x_3)$. However, because x_2 is specified in (x_2, e, x_4), x_2 cannot be removed, and so this conjunction is necessary. Hence, the language to properly express this relationship pattern should be a conjunctive RRE.

We restrict our attention to transformations with acyclic premises in order to reduce the expressivity of the relationship language and keep the computation of similarity scores efficient. Since conjunctive RRE is more complex, it will take longer to compute the similarity scores between nodes. Regardless, the result of Theorem 1 extends for general tgd constraints if a conjunction is added to our proposed relationship expression language.

4.2.4 Extending SimRank and RWR Using RRE. One may extend RWR or SimRank so that the similarity measurement is based on a particular relationship pattern between entities [43]. RWR computes a similarity score between nodes u and v in a dataset using the steady-state probability that a random walk from u will stay at v. SimRank, on the other hand, computes the score based on the probability that two random walks from u and v are met at a vertex in the data graph. Technically, the probability of a random walk from u to v computes the chance that a walk from u *hop*s from a node to its neighbor repeatedly until reaching v. Each hop, hence, is intuitively defined as a single edge between two nodes. In this extended RWR or SimRank, given a relationship pattern, a hop is defined only if a walk follows the given pattern from one node to another node. Following this idea, we may use the same measurement as SimRank and RWR to compute similarity scores over a relationship pattern as similarly specified in RelSim. We prove the following proposition similar to Theorem 1. Let $\mathtt{RWR}_p(u, v, D)$ and $\mathtt{SimRank}_p(u, v, D)$ denote a similarity score between nodes u and v computed using RWR and SimRank scoring function that only considers walks that follows RRE p.

Proposition 3. *Given a database instance D of a schema S, for every transformation Σ_{ST} for some schema T, there is a mapping M between a set of patterns over S and a set of patterns over T such that, for a given pattern p over S, we have $\forall D \in \mathtt{Inst}(S)$, $\forall u, v \in D$, $\mathtt{RWR}_p(u, v, D) = \mathtt{RWR}_{M(p)}(u, v, \Sigma_{ST}(D))$ and $\mathtt{SimRank}_p(u, v, D) = \mathtt{SimRank}_{M(p)}(u, v, \Sigma_{ST}(D))$.*

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

4.3 Computing Similarity Scores

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

We extend the computation of commuting matrix for RREs as follows. Given matrices $\mathbf{X}$ and $\mathbf{Y}$, let $>$ be a Boolean operation such that each entry (i, j) of $\mathbf{X} > \mathbf{Y}$ is 1 if $\mathbf{X}(i, j) > \mathbf{Y}(i, j)$ or 0 otherwise, and $\mathbf{diag}\{\mathbf{X}\}$ denote a diagonal matrix of $\mathbf{X}$. Given a label a and arbitrary patterns p, p_1 and p_2 over database D in schema S, we have $\mathbf{M}_a = \mathbf{A}_a$, $\mathbf{M}_{p^-} = \mathbf{M}_p^T$, $\mathbf{M}_{p_1 \cdot p_2} = \mathbf{A}_{p_1}\mathbf{A}_{p_2}$, $\mathbf{M}_{p_1+p_2} = \mathbf{A}_{p_1} + \mathbf{A}_{p_2}$ if $p_1 \neq p_2$, $\mathbf{M}_{p_1+p_2} = \mathbf{A}_{p_1} = \mathbf{A}_{p_2}$ if $p_1 = p_2$, $\mathbf{M}_{\lceil\lceil p \rfloor\rfloor} = \mathbf{M}_p > \mathbf{0}$, and $\mathbf{M}_{[p]} = \mathbf{diag}\{\mathbf{M}_p(\mathbf{M}_p^T > \mathbf{0})\}$ where $\mathbf{0}$ denotes a matrix whose entries are zero.

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

5 SIMPLIFYING RELSIM

5.1 Usable Pattern Language

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

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

5.2 Generating RREs From a Simple Pattern

Algorithm 1 finds a set $\mathcal{E}_p$ of RREs by minimally modifying the input simple pattern p such that the results of Corollary 1 and Proposition 3 hold for the aggregated scores of all patterns in $\mathcal{E}_p$. For example, given an input p_2 : `field`$\cdot$ `field`$^-$ over Figure 1(b), the algorithm returns a set of RREs whose members are including both p_2 and p_4 : `field`$\cdot$ [`pub-in`$^-$] $\cdot$ [`pub-in`$^-$] $\cdot$ `field`$^-$. Our method computes and aggregates the similarity scores of pattern in the set of RREs returned by Algorithm 1 using Equation 1. If p_5: `area` $\cdot$ `pub-in` $\cdot$ `pub-in`$^-$ $\cdot$ `area`$^-$ is the input of Algorithm 1 over Figure 1(a), the algorithm returns a set of RREs whose members are p_5 and p_6 : $\lceil\lceil$`area`$\cdot$`pub_in`$\rfloor\rfloor \cdot \lceil\lceil$`pub_in`$^-\cdot$`area`$^-\rfloor\rfloor$. According to the mapping defined in Corollary 1 between two data fragments in Figures 1(b) and Figure 1(a), p_2 and p_4 map to p_6 and p_5, respectively. Thus, there is a one-to-one mapping between the sets of RRE returned by the algorithm over data fragments. By computing the scores through counting, one finds that the similarity results for p_2 over Figure 1(b) and p_5 over Figure 1(a) are the equal.

More precisely, let Algorithm 1 take a simple pattern $p = l_1 \cdot \ldots \cdot l_n$ over a schema $S = (\mathcal{L}, \Gamma)$. Let (r, i) denote a generated RRE r by Algorithm 1 from p, which is processed up to label l_i in p. Given (r, i), let s be the the remaining unprocessed sub-pattern of p, e.g.,

Algorithm 1: `PatternGenerator`

```
Input: schema $S = (\mathcal{L}, \Gamma)$, simple pattern $p = l_1 \ldots l_n$ over $S$
Output: subset $\mathcal{E}_p$ of RREs over $S$
1  $done \leftarrow \{\}$; $processing \leftarrow \{(\epsilon, 0)\}$; // For $(r,i) \in processing$, $r$
       is an RRE $r$ processed up to the position of label $l_i$ in $p$
2  foreach $(r, i) \in processing$ do
3      Remove $(r, i)$ from $processing$;
4      if $i \geq n$ then
5          Add $r$ to $done$; continue;
6      Add $(r \cdot l_{i+1}, i + 1)$ to $processing$; // Use original pattern
7      // Modify each sub-pattern of $s = l_{i+1} \cdot l_{i+2} \cdot \ldots l_n$
8      foreach $\gamma \in \Gamma$ do
9          $\mathcal{R} \leftarrow \mathcal{R} \cup$ ModPatternRefsPerConstraint$(\gamma, s)$;
10     foreach $j \geq i + 1$ do
11         foreach $(l_{i+1} \cdot l_{i+2} \cdot \ldots \cdot l_j, e') \in \mathcal{R}$ do
12             Add $(r \cdot e', j)$ to $processing$;
13 return $\mathcal{E}_p \leftarrow done$;
```

Algorithm 2: `ModPatternRefsPerConstraint`

```
Input: constraint $\gamma$, simple pattern $s = l'_1 \cdot \ldots \cdot l'_m$
Output: set $\mathcal{R} = \{(e, e')\}$ where $e'$ is a corresponding RRE to $e$
        which is a sub-pattern of $s$
1  $G_\gamma \leftarrow$ the premise graph representing $\gamma$;
2  foreach $i > 0, j \geq i, j \leq m$ do
3      $e \leftarrow l'_i \cdot l'_{i+1} \cdot \ldots \cdot l'_j$;
4      if a path $e$ from some $v_g$ to $v_h$ exists in $G_{pre}(\gamma)$ then
5          foreach connected subgraph $H$ of $G_{pre}(\gamma)$ do
6              Find all RREs $e' : v_g \hookrightarrow_H v_h$ that traverse $H$ from $v_g$
                   to $v_h$ and visit each edge of $H$ once;
7              Add $(e, e')$ and $(e^-, e'^-)$ to $\mathcal{R}$;
8  return $\mathcal{R}$
```

$s = l_{i+1} \cdot \ldots \cdot l_n$. Algorithm 1 examines each sub-pattern $e : l_{i+1} \cdot l_j$ of s for some $i + 1 \leq j \leq n$. Then, it uses Algorithm 2 to find a set $\mathcal{R}$ of RREs for e according to each constraint in S (Line 9). If Algorithm 2 generates RRE e' for e, Algorithm 1 replaces e in s with e' and marks that all labels up to l_j as processed, e.g., $(r \cdot e', j)$ (Line 12). It also includes the input pattern p in the set (Line 6).

Before we explain Algorithm 2, we define the **premise graph** of a constraint. Consider a constraint $\gamma : \phi_\gamma(\bar{x}) \rightarrow \psi_\gamma(\bar{x})$. We assume that for each atom (x_i, e, x_j) in ϕ_γ, e is *not* written as $e_1 \cdot e_2$ for some non-empty RPQs e_1 and e_2. If such atom exists, we rewrite it as $(x_i, e_1, x') \wedge (x', e_2, x_j)$ for some fresh variable x'. The **premise graph** of γ, denoted by $G_{pre}(\gamma)$, is a directed graph (V, E) whose nodes are variables in ϕ_γ and edges are labeled by the RPQ patterns between each pair of those variables in ϕ_γ. More precisely, a node $v_{x_i} \in V$ if and only if x_i is a variable in ϕ_γ, and an edge $(v_{x_i}, e, v_{x_j}) \in E$ if and only if (x_i, e, x_j) is in ϕ_γ. The premise graph of constraint γ_1 : $(x_1, \texttt{area}, x_3) \wedge (x_3, \texttt{pub-in}, x_4) \wedge (x_2, \texttt{pub-in}, x_4) \rightarrow (x_1, \texttt{area}, x_2)$ over Figure 1(a) is $v_1 \xrightarrow{\text{area}} v_3 \xrightarrow{\texttt{pub-in}} v_4 \xleftarrow{\texttt{pub-in}} v_2$.

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

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

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

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

Proposition 4. *Given a database D of schema S and a equivalent schema T under transformation Σ, for every simple pattern p_S over S, there exists a simple pattern p_T over T such that, $\forall u, v \in D$, $\sum_{p \in \mathcal{E}_{p_S}} \texttt{sim}_p(u, v, D) = \sum_{p' \in \mathcal{E}_{p_T}} \texttt{sim}_{p'}(u, v, \Sigma(D))$*

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

5.3 Complexity of Generating RREs

The complexity of Algorithm 1 largely depends on the number of patterns generated in Algorithm 2. Since each simple pattern p found in G_γ can be either p or $\lceil\lceil p \rfloor\rfloor$, this algorithm is exponential in the number of p's. However, since G_γ is acyclic, each p is a path. Hence, the number of simple pattern p's is $1 + \sum_{v \in V_{\deg>2}} (\deg(v) - 1)$ where $V_{\deg>2}$ is a set of nodes in G whose degrees are greater than 2. Since $\sum_{v \in V(G)} \deg(v) = 2|E(G)|$, the procedure is $O(\exp(|E(G)|))$. There is also a linear-time algorithm $O(V(G))$ that finds all connected subgraphs G of G_γ [27]. Since the number of iterations over all sub-patterns (Line 2) is polynomial in the length of p, the total

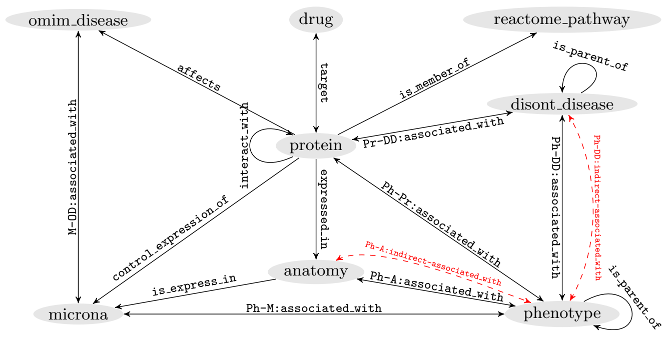


Figure 4: Schema fragment of DD dataset

complexity of Algorithm 2 is $O((n^2)(\exp(|E(G)|) + |V(G)|))$ where n is the number of nodes in the input simple pattern. Constraints of a schema usually have a very small number of terms compared to the size of databases. We may view the exponent factor as a (small) constant. Hence, Algorithm 2 is $O(n^2)$ where n is usually small [43]. Let $O(M)$ denote the number of patterns generated from Algorithm 2. Consider that there are possible $O(\exp(k))$ possible sub-patterns to the user input pattern with k nodes. Also, algorithm 1 may replace each of those sub-patterns by the generated patterns from Algorithm 2. Hence, the complexity of Algorithm 1 is $O(M^{\exp(k)})$. The value of k, i.e., the number of nodes in the submitted simple pattern is also significantly smaller than the size of the database. We have used some optimizations in our implementation of Algorithm 2, which improve the running time of the algorithm. These methods are explained in the full version of the paper in [12].

6 EMPIRICAL STUDY

Datasets: We use the following datasets in our experiments. *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. Figure 2(a) depicts the schema of DBLP. We also use a subset of Microsoft Academic Search (*MAS*) 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*. The *WSU* course database consists of 1,124 nodes and 1,959 edges. Figure 3(a) depicts the schema of WSU dataset. The *Linked-MDB* dataset collected from *www.cs.toronto.edu/~oktie/linkedmdb* contains information about movies, actors, and characters those actors have played in movies. It has 21,039 nodes and 48,084 edges. The Drug and Disease dataset (DD), is made available to us as a part of an NIH funded project and contains information about diseases, drugs, pathways, and their relationships. Figure 4 depicts a fragment of DD. It consists of 43,307 nodes and 1,742,970 edges. The *Gene* dataset has also been used in the same NIH program and has detailed information about interactions of genes and drugs. It has 77,520 nodes and 428,940 edges.

Settings: We compare robustness, effectiveness and efficiency of RelSim with RWR [48] using a restart probability of 0.8, SimRank [29] using a damping factor of 0.8, and PathSim [43]. Since previous studies show that the PathSim similarity computation method is more effective than those of SimRank and RWR [43], we use RelSim

Table 2: Average ranking differences

	DBLP2SIGM		WSUC2ALCH		DDT	
	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

with the similarity score computation of PathSim, i.e., Equation 1. We implement all algorithms using MATLAB 8.5 on a Linux server with 64GB memory and two quad-core processors.

6.1 Structural Robustness

We use DBLP, WSU and DD databases to evaluate the structural generalizability of RWR, SimRank, PathSim and RelSim. As SimRank takes too long to finish on full DBLP dataset, we do this evaluation using a subset of DBLP with 24,396 nodes and 98,731 edges.

DBLP dataset satisfies constraint $(paper_1, \texttt{r-a}, area_1) \wedge (paper_1, \texttt{p-in}, proceedings_1) \wedge (paper_2, \texttt{p-in}, proceedings_1) \rightarrow (paper_2, \texttt{r-a}, area_1)$. We transform this database to a database with the structure shown in Figure 2(b), which follows the style of SIGMOD Record database. We call this transformation DBLP2SIGM. We randomly sample 100 proceedings based on their node degrees as our query workload over these datasets.

WSU 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 database to the graph structure of Alchemy UW-CSE database whose structure is shown in Figure 3(b). We call this transformation WSUC2ALCH. We also randomly sample 100 courses from WSU based on their degrees as our query workload.

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

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

Table 3: Average ranking differences over transformations that modify information

	DBLP2SIGMX		DDT(.95)		DBLP2SIGM(.95)	
	top 5	top 10	top 5	top 10	top 5	top 10
RelSim	0	0	.750	.144	.170	.298
RWR	.696	.752	.530	.500	.835	.640
SimRank	.790	.750	.143	.344	.790	.750
PathSim	.364	.239	.927	.927	.423	.452

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. According to Table 2, the outputs of all current algorithms are significantly different across databases under these information-preserving transformations.

We also evaluate the generalizability of our algorithm when a transformation is invertible and adds extra information. We define a transformation, called DBLP2SIGMX, similar to DBLP2SIGM, which also adds additional nodes that connect authors to proceedings in the SIGMOD Record schema. These nodes indicate the relationship of what authors has a paper in what proceedings. Each of these nodes is connected to both the corresponding author and the proceedings in which she has published at leas one paper. DBLP2SIGMX is invertible and has the same inverse as DBLP2SIGM. Due to the long running time of SimRank and RWR over DBLP we have performed these experiments over the small DBLP dataset. We randomly selected 100 queries for DBLP dataset. The results shown in Table 3 indicate that RelSim is the only generalizable and robust algorithm over this transformation.

Some real-world transformations may *not* be invertible and lose some information. We also measure the generalizability of algorithms when a small fraction of edges between nodes are removed during the transformation. In this experiment, we create two new transformations for DBLP and DD, namely DBLP2SIGM(.95) and DDT(.95), respectively. DBLP2SIGM(.95) and DDT (.95) restructure DBLP and DD similar to that of DBLP2SIGM and DDT, respectively. These transformation first restructure the database and then randomly remove 5% of the total number of edges from the transformed databases. Table 3 shows the average ranking differences for top 5 and top 10 answers returned by RWR, SimRank, PathSim (HeteSim) and RelSim using the same datasets, the query workloads and relationship patterns described in the previous experiment. The result indicates that RelSim is more generalizable than other methods when the transformed databases has a slightly less information than the original ones.

6.2 Effectiveness

We evaluate the effectiveness of RelSim over DD, Gene, MAS, and LinkedMDB. We use the same query workload used in Section 6.1 for DD. For Gene, we are given a set of 80 drugs and the genes that are strongly influenced by them by our collaborator based on previously published data in medical domain. Each drug in this ground truth is associated by only one gene. For query workload over MAS, we randomly sample 100 conferences based on their degrees in the dataset. To provide the ground truth over MAS, 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 a strongly related research area. Otherwise, the conference is considered least-similar to q. We have also collected 23 pairs of actors from LinkedMDB data that are most similar in terms of the movies they co-starred and use them as queries over LinkedMDB data. Since over all datasets, except for MAS, each query has only one answer, e.g., each disease query relates only to a single drug, we use *Mean Reciprocal Rank* (MRR) to evaluate the effectiveness of the algorithms over all but MAS dataset. *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 and returns a value between 0 and 1 where the larger values show more effective results. We use normalized DCG (nDCG) to evaluate the effectiveness because it supports multiple levels of relevance for returned answers [34]. The value of nDCG varies between 0 and 1 where the higher values indicate more effective ranking. As we explained, it has been shown that PathSim (HeteSim for cases where query and result are from different types) are more effective than RWR and SimRank. Thus, we compare our method with PathSim (HeteSim). We use Wilcoxon Signed Rank test with the confidence interval of 95% to measure the statistical significance of the differences between the results of our method and others.

We compare RelSim and HeteSim over original DD data and DD under DDT. The MRR of RelSim and HeteSim over the original DD data is 0.077 and 0.077 and over the transformed one are 0.077 and 0.072, respectively, where the difference is significant for the transformed data. According to our discussion with the experts, these queries are very hard to answer effectively by using only the structural patterns in the data set and without consulting external sources of knowledge and even a slight improvement in the accuracy of the returned answers may save a great deal of time and effort in their research. We have also used the DD and query workload to evaluate the effectiveness of similarity search method proposed in Section 5. We have used the simple pattern given to HeteSim for this experiment. The MRR of the final results is close to the one delivered by RelSim algorithm that takes RRE patterns as input, which is as effective or more effective than HeteSim.

We compare the effectiveness of RelSim with HeteSim over Gene data. We also transform Gene data using an invertible transformation that adds a new type of edge to connect two entities in the Gene data. The MRR of RelSim and HeteSim over the original Gene data is 0.235 and 0.173 and over the transformed one are 0.235 and 0.173, respectively, where the difference is significant. We also evaluated the method proposed in Section 5 over these datasets and got a MRR of 0.235. The methods proposed in Section 4 and 5 deliver same MRR as RelSim uses a very indicative pattern to compute similarity. Since additional patterns used in Section 5 are not as indicative, both methods deliver almost the same MRR.

Next, we compare the effectiveness of RelSim with PathSim over LinkedMDB data, as the the query and answer nodes are of the same type. In the original IMDb data, *movie, actor*, and *character* are connected via a connecting node called *performance*. We have transformed LinkedMDB data using an invertible transformation that adds new direct edges between movies, actors, and characters. The MRR of RelSim and PathSim over the original LinkedMDB data is equal and 0.4887. But, the MRR of RelSim and PathSim over the transformed one are 0.489 and 0.352, respectively, where the difference is significant. We also evaluated the method proposed in Section 5 over these datasets and got a MRR of 0.405 over both original and transformed data. The results of the method proposed in Section 5 are significantly less effective that PathSim over the original LinkedMDB but more effective over the transformed one.

We also compare the effectiveness of RelSim with that of PathSim, and report the values of nDCG for top 5 (nDCG@5) and top 10 (nDCG@10) answers over MAS data. The average nDCG@5 (nDCG@10) for RelSim and PathSim are 1.0 (1.0) and 0.969 (0.901), respectively, where the difference is statistically significant. Using the algorithm in Section 5, we get the same results over this dataset.

The aforementioned results indicate that RelSim is as effective or more effective than similar algorithms. It also implies that using RRE language in RelSim makes it not only generalizable but it also improves the effectiveness of its results over various datasets. The same observation generally holds for the usable version of RelSim proposed in Section 5.

Table 4: Average query processing time in seconds.

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

6.3 Efficiency

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

Table 4 shows the average query processing time for a single pattern per query of RelSim and PathSim (under “single pattern”). Overall, RelSim is slower than PathSim because RelSim uses more complex patterns than those used by PathSim. Nevertheless, the running time of RelSim is still relatively short over large datasets.

Next, we measure the efficiency of RelSim that incorporates Algorithm 1 introduced in Section 5 with the the optimization techniques to ignore non-relevant constraints. In this version, RelSim takes a simple pattern as an input. Hence, we supply the same pattern to both RelSim and PathSim, and compare their query processing times. We use the same relationship patterns over DBLP and DD as described in Section 6.1. The average query processing times per query of RelSim and PathSim are reported in Table 4 (under "using Algorithm 1"). Overall, the running time of RelSim is slightly slower than PathSim due to the procedure of Algorithm 1. This result also shows that making RelSim more usable does *not* increase its running time considerably.

We also evaluate the scalability of the version of RelSim that takes simple patterns as input as described in Section 5 over DD dataset. Since the running time of this version depends on the number of tgd constraints on the original database and the length of the input simple pattern, we measure its running time over different values of these parameters. Because our datasets contain only two constraints, we test the scalability of our algorithm over a randomly generated set of constraints from only DD. As a matter of fact, we pick DD as it allows us to generate and test many constraints due to its rich structure. We generate each constraint by using a coin flipping random generator over each edge label in DD to decide whether an atom of the constraint contains such edge. We limit the number of atoms in the premise of each constraint to be between 2 and 5, and a single atom in its conclusion. For the input pattern, each simple pattern is randomly generated such that it traverses the data graph from a drug to a disease. We measure the running time by setting the number of constraints to 1, 5, 10, 20 and 40, and vary the length, i.e., number of edge labels, of the input simple pattern between 3 and 9. We use the same query workload as the one described in Section 6.1. We report the average similarity query running times of RelSim per query over 5 runs of each setting in Figure 5. The Running time for pattern size of 9 for 40 constraints are omitted due to long running time. Overall, RelSim is reasonably fast for a small number of constraints and up to the length of 7 for the input pattern. The algorithm slows down when the number of constraint or the length of the input pattern goes up, especially when there are 20 or more constraints. We also tested the version of RelSim in Section 5 without our proposed optimization techniques for ignoring non-relevant constraints. The algorithm takes multiple hours to finish for 5 constraints or longer for more constraints for patterns of size more than 7.

7 RELATED WORK

There are robust algorithms over certain types of structural variations in other data models [11, 37, 45, 50]. They have two major shortcomings. First, they are robust only over a subset of frequently occurring 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 variations. Second, these systems either propose new algorithms [50], or make significant and/or complex modifications to the current ones [37]. However, current algorithms have been widely adapted and it is costly to replace them with new ones. Moreover, current algorithms are shown to be effective over some representations. Thus, a robust version of them will be effective over more representations.

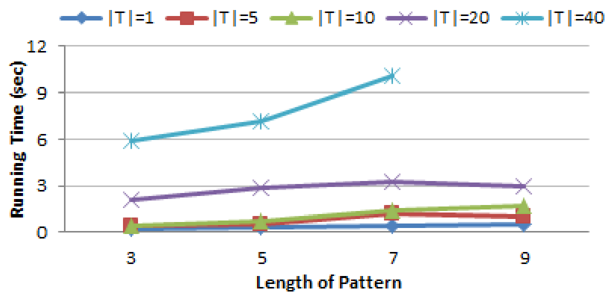


Figure 5: Running times of RelSim in various settings. $|T|$ **is the number of constraints.**

Our inverse is similar to Fagin-inverse used in relational data exchange [14]. For Fagin-inverse, a transformation and its inverse may map multiple databases under one schema to multiples ones under another schema. The answers to a similarity query in different databases of the same schema may be different. Thus, we use a notion of inverse that maps one database to multiple ones to compare the results of an algorithm over a database and all its variations with the same information. There are also other notions of inverse in which the original and target databases may *not* contain the same information [5, 6].

8 CONCLUSION & FUTURE WORK

We proposed effective and scalable similarity search algorithms that generalize over a wide structural variations of graph data. An interesting future work is to investigate the connection between the traditional notion of generalizability in ML and structural generalizability for supervised and unsupervised learning on graphs.

A SEMANTICS OF COUNTING RRE

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

where $\widetilde{p}$ is a string p with all $\lceil\lceil\ \rfloor\rfloor$ removed, e.g., $\widetilde{\lceil\lceil a \cdot b \rfloor\rfloor} = a \cdot b$.

REFERENCES

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

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

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

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

[5] Marcelo Arenas, Jorge Pérez, Juan L. Reutter, and Cristian Riveros. 2009. Inverting Schema Mappings: Bridging the Gap between Theory and Practice. *PVLDB* 2, 1 (2009), 1018–1029.

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

[7] Magda Balazinska, Surajit Chaudhuri, Anastasia Ailamaki, Juliana Freire, Sailesh Krishnamurthy, and Michael Stonebraker. 2020. The Next 5 Years: What Opportunities Should the Database Community Seize to Maximize its Impact?. In *SIGMOD* (Portland, OR, USA). ACM, 411–414. https://doi.org/10.1145/3318464.3394837

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

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

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

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

[12] Y. Chodpathumwan, A. Termehchy, S. Ramsey, A. Shresta, A. Glen, and Z. Liu. 2021. Structural Generalizability: The Case of Similarity Search. arXiv:1508.03763 [cs.DB]

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

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

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

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

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

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

[19] R. C. Fernandez, D. Deng, E. Mansour, A. A. Qahtan, W. Tao, Z. Abedjan, A. K. Elmagarmid, I. F. Ilyas, S. Madden, M. Ouzzani, M. Stonebraker, and N. Tang. 2017. A Demo of the Data Civilizer System. In *SIGMOD*. ACM, 1639–1642. https://doi.org/10.1145/3035918.3058740

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

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

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

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

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

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

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

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

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

[29] Glen Jeh and Jennifer Widom. 2002. SimRank: a measure of structural-context similarity. In *Proceedings of the 8th ACM SIGKDD Int'l Conference on Knowledge Discovery and Data Mining, July 23-26, 2002, Edmonton, Alberta, Canada.* ACM, 538–543. https://doi.org/10.1145/775047.775126

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

[31] David Jensen and Jennifer Neville. 2002. Linkage and Autocorrelation Cause Feature Selection Bias in Relational Learning. In *Proceedings of the 19th International Conference on Machine Learning (ICML '02)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 259–266.

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

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

[34] Christopher Manning, Prabhakar Raghavan, and Hinrich Schutze. 2008. *An Introduction to Information Retrieval.* Cambridge University Press.

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

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

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

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

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

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

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

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

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

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

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

[46] Arash Termehchy, Marianne Winslett, and Yodsawalai Chodpathumwan. 2011. How Schema Independent Are Schema Free Query Interfaces?. In *ICDE*.

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

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

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

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

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

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

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

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

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

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

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

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