Towards Representation Independent Similarity Search Over Graph Databases

Yodsawalai Chodpathumwan, Amirhossein Aleyasen
University of Illinois
{ychodpa2,aleyase2}@illinois.edu

Arash Termehchy
Oregon State University
termehca@oregonstate.edu

Yizhou Sun
Northeastern University
yzsun@ccs.neu.edu

ABSTRACT

Finding similar entities is a fundamental problem in graph data analysis. Similarity search algorithms usually leverage the structural properties of the database to quantify the degree of similarity between entities. However, the same information can be represented in different structures and the structural properties observed over particular representations may not hold for the alternatives. These algorithms are effective on some representations and ineffective on others. We define the property of representation independence for similarity search algorithms as their robustness against transformations that modify the structure of databases but preserve the information content. We introduce a widespread group of such transformations called *relationship reorganizing*. We propose an algorithm called **R-PathSim**, which is provably robust under relationship reorganizing. Our empirical results show that current algorithms except R-PathSim are highly sensitive to the data representation and R-PathSim is as efficient and effective as other algorithms.

Keywords

Structural similarity search; Database transformation; Representation independence

1. INTRODUCTION

Finding similar or strongly related entities is a fundamental problem in graph data management [5, 6, 9]. It is a building block of algorithms for various important problems, such as similarity query processing and community detection [5, 8]. Since the properties of *similar* or *related* entities cannot be precisely defined, current similarity search algorithms use intuitively appealing heuristics that leverage information about the links between entities. For instance, *Random Walk with Restart* (RWR) quantifies the degree of similarity between two entities as the likelihood that a random surfer visits one of the entities in the database given it starts and keeps re-starting from the other entity [9]. *SimRank* evaluates the similarity between two entities according to how likely two random surfers will meet each other if they start from the two entities [5]. Figure 1a shows fragments of IMDb (*imdb.com*), which contains information about movies, actors, and characters. IMDb represents relationship between a character, its movie, and the actor who played the character by connecting these entities through some edges. Assume that a user asks for the most similar movie to *Star Wars III* in Figure 1a. RWR and SimRank find *Star Wars III* more similar to *Star Wars V* (RWR-score = 0.061, SimRank-score = 0.213) than to *Jumper* (RWR-score = 0.060, SimRank-score = 0.185) which is arguably an effective answer.

Generally, there is no canonical representation for a particular set of content and people often represent the same information in different, i.e., non-isomorphic, structures [1, 3]. Database designers may represent the same information in one form or another for reasons such as improving the running time of queries and reducing redundancy [1]. The choice of representation may unintentionally influence the output of current similarity search algorithms. Consider the excerpts of Freebase (*freebase.com*) in Figure 1b which contains the same information as Figure 1a. It differs with Figure 1a only in how it represents the relationships between a character, its movie, and its actor: it connects them to a common node labeled *starring*. As opposed to their results over Figure 1a, RWR and SimRank find *Star Wars III* more similar to *Jumper* (RWR-score = 0.014, SimRank-score = 0.076) than to *Star Wars V* (RWR-score = 0.011, SimRank-Score = 0.074) in Figure 1b.

The power of similarity search algorithms, however, remains out of the reach of most users as the algorithms are usable only by trained data analysts who can predict which algorithms are likely to be effective for particular representation of the dataset. Since these algorithms do not normally offer any clear description of their desired representations, users have to rely on their own expertise and use trial and error to find the representations in which the algorithm returns effective results. However, we want our algorithms to be used by ordinary users, not just experts who know the internals of these algorithms and can restructure the data. Furthermore, the structures of databases constantly evolve and users may have to repeat the process of finding the right representation and restructuring their data accordingly.

One approach to solve the problem is to run a similarity search algorithm over all possible representations of a dataset and select the representation(s) with the most accurate answers. However, it is undecidable to compute all possible representations of a graph database [3]. Thus, it is infeasible to generate and run algorithms over them. Researchers have proposed the idea of *universal relation* to

CIKM'16 , October 24-28, 2016, Indianapolis, IN, USA

DOI: http://dx.doi.org/10.1145/2983323.2983673

achieve some level of schema independence for SQL queries over relational databases [1]. One may extend this idea and define a universal representation in which all graph databases can be represented and develop similarity search algorithms that are effective over this representation. Nevertheless, the experience gained from the idea of universal relation, indicates that such representation may not always exist [1]. Furthermore, it may not be practical to force developers to organize their data in and create their algorithms for a particular style of representation.

In this paper, we propose the property of *representation independence* for similarity search algorithms, i.e., the ability to deliver the same answers regardless of the choices of structure for organizing the data. To the best of our knowledge, the property of representation independence has not previously been explored for similarity search algorithms and/or graph databases. We believe that the key to the success of building representation independent analytics in general and similarity search algorithms in particular is to modify current algorithms to become representation independent instead of developing completely new algorithms. Current algorithms are being used in industry and it is easier for organizations to modify these algorithms rather than using new algorithms. Our contributions are as follows.

- We formally define the representation independence of a similarity search algorithm as its robustness under transformations that modify the structure of database but preserve its information content.
- We introduce a group of content-preserving transformations called *relationship reorganizing*. We show that current similarity search algorithms are not representation independent. We extend a current similarity search algorithm called PathSim and develop an algorithm, namely Robust-PathSim (*R-PathSim*), that is representation independent under relationship reorganizing.
- We empirically study the representation independence of well-known similarity search algorithms under relationship-reorganizing using real-world databases. Our results indicate that relationship-reorganizing transformation considerably affects the output of all algorithms but R-PathSim. Our results show that R-PathSim is as effective and as efficient as current algorithms.

The full proofs for our theorems can be found at [2].

2. BACKGROUND

2.1 Related Work

The architects of relational model envisioned the desirable property of *logical data independence*. Oversimplifying a bit, this meant that an exact query should return the same answers no matter which schema is chosen for the data [1]. Developers usually achieve logical data independence by creating a set of views over the database, which keeps the application unaffected from modifications in the database schema [1]. However, characteristics of the ideal representations for similarity search are not clearly defined. Hence, it takes far more time and much deeper expertise to find and maintain the proper representation for the data. Moreover, our goal is to achieve representation independent without making users create and tune data representations.

Researchers have proposed query interfaces over tree-shaped XML data that return the same answers to a keyword query

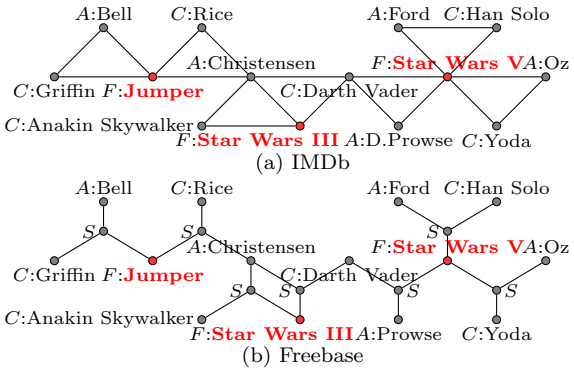


Figure 1: Fragments of IMDb and Freebase, where A, C, F, and S refer to *actor*, *character*, *film* and *starring*, respectively.

over databases with equivalent content but different choices of structure [7]. We, however, introduce and study the concept of representation independence for a different problem and data model. The task of similarity search has a different semantic than keyword search and requires different types of algorithms. Further, graph databases are more complex than tree-shaped XML databases and offer novel challenges in defining the concept of representation independence and developing representation independent algorithms.

2.2 Data Model

Let *dom* be a fixed and countably infinite set of values. To simplify our definitions, we assume the members of *dom* are strings. Let L be a finite set of labels. Each member of L denotes a *semantic type* in a domain of interest, e.g. *actor* in movie domain. A database D defined over L is a graph $D = (V, E, \mathcal{L}, \mathcal{A})$, where V is the set of nodes, $E \subseteq V \times V$ is the set of edges, $\mathcal{L}$ is a total function from V to L that assigns a label to each node, and $\mathcal{A}$ is a function from V to *dom* that assigns values to nodes in V. We denote the set of all databases whose labels belong to L as $\mathbf{L}$. Real-world databases often contain nodes without any value to represent relationships between or categorize entities [4]. Figure 1b is an example of using nodes without values to represent relationships between entities. About 30% of 1.23 billion RDF triples collected from the Web contain nodes without values [4]. We call the nodes with values *entities* [5, 6]. We assume that each label in L denotes a semantic type for entities or for nodes without values, but not for both.

We denote a similarity query q, query for short, over database D as (v), where v is an entity in D. Query $q = (v)$ seeks for entity nodes other than v in D that are similar to v [6, 5, 9, 8]. For example, query ($film$:Star Wars III) over Figure 1b asks for other entities similar to the node $film$:Star Wars III in this database. A similarity search algorithm returns a ranked list of entities as the result of query q. We denote the result of query q over database D using algorithm S by $q_S(D)$. If S is clear from the context, we denote $q_S(D)$ as $q(D)$.

3. REPRESENTATION INDEPENDENCE

Intuitively, a representation independent similarity search algorithm should return the same list of entities for the same query across databases that represent the same information. Researchers have defined the conditions under which relational or XML schemas represent the same information [3, 1, 7]. Graph databases, however, do not generally follow

strict schemas. Hence, we extend the ideas on comparing information contents of databases for graph data model.

Transformation T is a function from a set of databases $\mathbf{L}$ to a set of databases $\mathbf{K}$, denoted as $T : \mathbf{L} \rightarrow \mathbf{K}$. For instance, consider the set of labels $L_1 = \{actor, film, char\}$ and $L_2 = \{actor, film, char, starring\}$. The databases in Figures 1a and 1b belong to $\mathbf{L_1}$ and $\mathbf{L_2}$, respectively. One may define transformation $T_{IMDb2Freebase} : \mathbf{L_1} \rightarrow \mathbf{L_2}$, which replaces every triangle between nodes of labels *film*, *character*, and *actor* with a subgraph whose nodes have the same labels and values of the nodes in the triangle and are connected to a single new node with label *starring*. $T_{IMDb2Freebase}$ maps the database in Figure 1a to the one in Figure 1b.

Intuitively, transformation T is invertible if one can reconstruct database D from the information in $T(D)$. For example, transformation $T_{IMDb2Freebase}$ is invertible as one can reconstruct the original database, e.g., Figure 1a, using the information in its transformed one, e.g., Figure 1b. More formally, $T : \mathbf{L} \rightarrow \mathbf{K}$ is *invertible* iff there is a transformation $T^{-1} : \mathbf{K} \rightarrow \mathbf{L}$ such that for all $D \in \mathbf{L}$ we have $T^{-1}(T(D)) = D$. Because the transformed database of an invertible transformation contain sufficient information to build the original database, the original and transformed databases contain essentially the same information [1, 3].

To precisely define representation independence over a transformation T, we should make sure that users can pose the same set of queries over databases D and $T(D)$. Similarity search queries over a database D are entities of D, thus, D and $T(D)$ should contain the same set of entities. Moreover, similarity search algorithms generally view the labels of nodes as their semantic types [6]. For example, they assume that the nodes with label *film* in Figure 1a represent entities from the same semantic type, while the entities of labels *film* and *actor* belong to different semantic types. Thus, for these algorithms to return the same results over a transformation T, T should map entities of the same label in the original database D to entities with the same label in the transformed database $T(D)$. We consider two data values equal iff they are lexicographically equal. Our approach also supports other definitions of equality between data values.

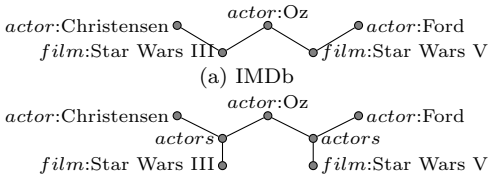
DEFINITION 1. *Transformation $T : \mathbf{L} \rightarrow \mathbf{K}$ that transforms database $D = (V, E, \mathcal{L}, \mathcal{A})$ to $T(D) = (V_T, E_T, \mathcal{K}, \mathcal{A}_T)$ is entity preserving iff there is a bijective mapping M between entities in V and V_T such that*

- $\forall v \in V$, $\mathcal{A}(v) = \mathcal{A}_T(M(v))$
- $\forall v_1, v_2 \in V$, $\mathcal{L}(v_1) = \mathcal{L}(v_2)$ *iff* $\mathcal{K}(M(v_1)) = \mathcal{K}(M(v_2))$.

For example, transformation $T_{IMDb2Freebase}$ is entity preserving. By the abuse of notation, we denote the entity in database $T(D)$ that is mapped to the entity v in database D, as $T(v)$. To simplify our definitions, we assume that transformations do not rename the labels in databases. Our results extend for the transformations that rename labels.

If a transformation is both invertible and entity preserving, it is *similarity preserving*. Each similarity preserving transformation T maps a databases D to a database $T(D)$ that has the same information and the same set of possible queries as D. Hence, it is possible to design a similarity search algorithm that returns essentially the same answers for every query over D and $T(D)$.

DEFINITION 2. *Similarity search algorithm S is representation independent under similarity preserving transformation $T : \mathbf{L} \rightarrow \mathbf{K}$ iff for each database $D \in \mathbf{L}$ and $T(D) \in \mathbf{K}$*



(a) IMDb

(b) Movielicious (www.netwalkapps.com/movies-xml-format)

Figure 2: Fragments of movie databases.

and every query q over D, there is a bijective mapping N between $q(D)$ and $T(q)(T(D))$ such that

- $\forall v \in q(D)$ *and* $N(v) \in T(q)(T(D))$, $N(v) = T(v)$
- $\forall u, v \in q(D)$, *v appears before u iff $N(v)$ appears before $N(u)$ in $T(q)(T(D))$.*

The first condition in Definition 2 guarantees that the answers to q over D and $T(q)$ over $T(D)$ contain the same set of entities. Its second condition ensures that these entities appear at the same order in results of q over D and $T(q)$ over $T(D)$. According to Definition 2, if answers v and u are placed at the same position, in $q(D)$, $T(v)$ and $T(u)$ must also appear at the same position in $T(q)(T(D))$.

The result of a query is a list of entities, where each entity is shown by its semantic type and value. A database may have several entities with equal values from of the same semantic type. Hence, it may not be possible to check the first condition of Definition 2 using only the semantic types and values of the entities in the results of a query. One may assign a unique (printable) id to each entity in the database to address this problem. [3]. To simplify our framework and definitions, we assume that databases do not contain entities that belong to the same semantic type and have equal values. Our results extend for other cases.

4. RELATIONSHIP REORGANIZATION

4.1 Relationship-Reorganizing Transformations

Generally speaking, a relationship-reorganizing transformation T maps database D to database $T(D)$ such that D and $T(D)$ contain the same set of entities and relationships, but they may represent these relationships in different forms. More specifically, D and $T(D)$ may express the same relationship between the same set of entities using some edges or some nodes without values. For example, Figure 2b uses a set of edges to represent the relationship between a movie and its actors. However, Figure 2a expresses the same relationship between the same set of entities by a node without value, i.e., *actors*. In this section, we formally define this type of representational variation. First, we find patterns that represent relationships between entities in a database. Then, we define the conditions under which two patterns represent the same information. Finally, we define a relationship-reorganizing transformation as a bijective mapping between patterns that represent the same information in the original and transformed databases.

A **walk** in database is a sequence of nodes and edges where each edge's endpoints are the preceding and following nodes in the sequence. We show a walk in database D as a sequence of nodes $[v_0, \ldots, v_n]$, such that v_i are nodes and (v_{i-1}, v_i) ,$0 \leq i \leq n$, are edges in D. Intuitively, a walk represents some relationship between its entities. For example, walk [*actor*:Ford, *actors*, *film*:Star Wars V] in Figure 2a shows that actor *Ford* has played in movie *Star Wars V*. To simplify our framework, we assume that each database is a sim-

ple graph: it has at most one edge between each two nodes and does not have any loop at each node. Our framework extends for other cases. We are interested in walks that express relationships between entities. Hence, we consider only walks that start and end with entities.

Some walks contain consecutive forward and backward traverses from an entity to a node without value. For example, walk [*actor*:Ford, *actors*, *film*:Star Wars V, *actors*, *film*:Star Wars V] in Figure 2b expresses the relationship between actor *Ford* and movie *Star Wars V*. It contains consecutive forward and backward traverses from entity *film*:Star Wars V to the node without value *actors*. The information expressed by this walk can be represented using a shorter walk [*actor*:Ford, *actors*, *film*:Star Wars V], which does not contain any consecutive forward and backward traverses from *film*:Star Wars V to *actors*. Hence, unless otherwise noted, we consider only walks that does *not* have any consecutive forward and backward traverses from an entity to a node without value because they do not contain any information regarding the relationship between entities or their information can be expressed by shorter walks.

The **meta-walk** of a walk $[v_1, \cdots, v_n]$ in database $D = (V, E, \mathcal{L}, \mathcal{A})$ is a sequence of labels $[\mathcal{L}(v_1), \cdots, \mathcal{L}(v_n)]$. For example, the meta-walk of walk [*actor*:Ford, *actors*, *film*:Star Wars V] in Figure 2b is [*actor*, *actors*, *film*]. Each meta-walk represents a pattern of relationship between entities of certain semantic types. Some meta-walks represent basically the same relationships between the same sets of semantic types. For instance, meta-walk [*actor*, *film*] in Figure 2a and meta-walk [*actor*, *actors*, *film*] in Figure 2b represent the relationship of starring in movies between the same set of actors and movies. Next, we define the conditions under which two meta-walks represent the same relationship between the same set of entities. Given database $D = (V, E, \mathcal{L}, \mathcal{A})$, the value of an entity node $e \in V$ is the pair $\mathcal{L}(v) : \mathcal{A}(v)$. The value of a walk $w = [v_0, \ldots, v_n]$ is the tuple $[a_0, \ldots, a_m]$, $m \leq n$ such that a_0 and a_m are the values of v_0 and v_n, respectively, and for all $0 \leq i < j \leq n$ and $0 \leq i', j' \leq m$ if $a_{i'}$ and $a_{j'}$ are the values of entity nodes v_i and v_j, respectively, then $i' < j'$. For instance, the value of walk [*actor*:Ford, *actors*, *film*:Star Wars V] is [*actor*:Ford, *film*:Star Wars V]. Values of two walks are **equal** iff they have equal arities and their corresponding positions contain the same label and equal values. Two walks are **content equivalent** iff their values are equal. We show content-equivalent walks w and x as $w \equiv x$. We denote the set of walks in database D whose meta-walks are p as $p(D)$. Meta-walks p_1 in database D_1 and p_2 in database D_2 are **content equivalent** iff there is a bijection $M : p_1(D_1) \to p_2(D_2)$ where for all $w \in p_1(D_1)$, $w \equiv M(w)$. For instance, meta-walks [*actor*, *film*] in Figure 2b and [*actor*, *actors*, *film*] in Figure 2a are content equivalent. We denote content-equivalent meta-walks p_1 and p_2 as $p_1 \equiv p_2$.

Naturally, content-equivalent meta-walks represent the same sets of relationships between the same sets of entities. Thus, if a transformation bijectively maps each meta-walk in database D_1 to its content-equivalent meta-walk in database D_2, D_1 and D_2 represent the same information. We formally prove this intuition later in the section. However, this straightforward definition ignores some interesting transformations. For example, intuitively the databases in Figure 2a and Figure 2b contain the same information. But, there is not any meta-walk in Figure 2a that is content equivalent to meta-walk p_3=[*actor*, *actors*, *actor*] in Figure 2b. By looking closely at the Figure 2b and original Movielicious data, we observe that the node *actors* always groups actors that play in the same movie. Thus, each walk of p_3 is a part of a walk of meta-walk p_4= [*actor*, *actors*, *film*, *actors*, *actor*] in Figure 2b. Hence, if a transformation maps p_4 to a content-equivalent meta-walk in Figure 2a, it also preserves the information of p_3. Generally, some meta-walks contain other meta-walks. If a transformation preserves the information of a meta-walk, it will preserve the information of its contained meta-walks. Let us formalize this relationship between meta-walks. A walk w is a **subwalk** of walk x, shown as $w \sqsubseteq x$, iff w is a subsequence of x. For example, walk $[v_1, v_2, v_3]$ is a subwalk of walk $x_1 = [v_1, v_2, v_4, v_2, v_3]$, but walk $[v_1, v_3]$ is not a subwalk of x_1. Meta-walk p is a subwalk of meta-walk r, denoted as $p \sqsubseteq r$, iff a walk of p is a subwalk of a walk of r. For example, [*actor*, *actors*, *actor*] is a subwalk of [*actor*, *actors*, *film*, *actors*, *actor*] in Figure 2b.

DEFINITION 3. *Given meta-walks p and p' in database D, p'* **includes** *p iff*

- *there is a bijection M between $p(D)$ and $p'(D)$ such that for every walk $w \in p(D)$, we have $w \sqsubseteq M(w)$ and w and $M(w)$ start at the same node and end at the same node.*
- *there exists an entity label l whose occurrence in p' is more than in p, and the closest entity labels to the left and to the right of l in p' are not l.*

For example, meta-walk [*actor*, *actors*, *film*, *actors*, *actor*] includes [*actor*, *actors*, *actor*] in Figure 2b. A meta-walk p in database D is **maximal** iff it has a walk in D and it is not included in any other meta-walk. For instance, [*actor*, *actors*, *film*, *actors*, *actor*] is maximal in Figure 2a. Maximal meta-walks subsume the information of non-maximal meta-walks. Thus, if a transformation preserves only the information of maximal meta-walks in a database, it will preserve the information content of the database. Let $\mathcal{P}(\mathbf{L})$ denote the set of all meta-walks in the set of databases $\mathbf{L}$. We denote the set of all maximal meta-walks in $\mathbf{L}$ as $\mathcal{P}_{\max}(\mathbf{L})$.

DEFINITION 4. *Transformation $T : \boldsymbol{L} \to \boldsymbol{K}$ is* **relationship reorganizing** *iff there is a bijective mapping $M : \mathcal{P}_{\max}(\boldsymbol{L}) \to \mathcal{P}_{\max}(\boldsymbol{K})$ such that $p \equiv M(p)$.*

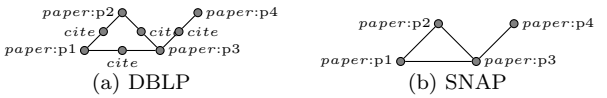
The transformations that map Figure 2b to Figure 2a and Figure 1a to Figure 1b are relationship-reorganizing. Using Definitions 3 and 4, we have the following theorem.

THEOREM 1. *Every relationship-reorganizing transformation is similarity preserving.*

4.2 Toward Robust Similarity Algorithms

To the best of our knowledge, the most frequently used methods for similarity search on graph database are based on random walk, e.g., RWR [9], pairwise random walk, e.g., SimRank [5], or relationship-constrained framework, e.g., PathSim [6]. There are other similarity measures, such as common neighbors, $Katz_\beta$ measure, hitting time, and commute time, which can be considered as special cases of aforementioned heuristics. Hence, we discuss similarity search methods based on these three frameworks.

Methods that use random walk and pairwise random walks leverage the topology of a graph database to measure the degree of similarities between entities. A relationship-reorganizing transformation may remove many edges from and add many



(a) DBLP (b) SNAP

Figure 3: Fragments of two citation databases

new nodes and edges to a database. Thus, it may radically modify the database topology. For example, a relationship-reorganizing transformations may drastically change the degree of a node and modify the probability that random surfers visit the node. Hence, these methods cannot always return the same answers over the original and the transformed database for the same query. In Section 1, we have shown that RWR and SimRank return different results over a database and its relationship reorganization in Figure 1.

PathSim measures the similarity between entities over a given relationship [6]. PathSim uses meta-walks to represent relationships between entities. For instance, the relationship between two movies in Figure 1b based on their common actors is expressed by [*film*,*actors*,*actor*,*actors*,*film*]. Let $p(e, f, d)$ be a set of walks of meta-walk p from entity e to entity f in database D. PathSim measures the similarity between e and f according to the input meta-walk p as $s(e, f) = \frac{2\times|p(e,f,D)|}{|p(e,e,D)|+|p(f,f,D)|}$. PathSim considers walks with and without consecutive forward and backward traverses from an entity to a node without value when it computes $s(e, f)$.

PathSim may return different answers for the same queries over the same relationship on a database and its relationship reorganizations. Figure 3 shows fragments of DBLP from *dblp.uni-trier.de*, and SNAP from *snap.stanford.edu* that contain information about citations. Consider the meta-walk s = [*paper*, *cite*, *paper*, *cite*, *paper*] in Figure 3a, and its corresponding meta-walk s' = [*paper*, *paper*, *paper*] in Figure 3b. s has a walk between entities $p3$ and $p4$, x = [*paper*:p3, *cite*, *paper*:p4, *cite*, *paper*:p4]. But, there is no walk of meta-walk s' between $p3$ and $p4$ in Figure 3b. Hence, PathSim reports $p1$ to be more similar to $p2$ than $p3$ in Figure 3a, but considers $p1$ to be more similar to $p3$ than $p2$ in Figure 3b. PathSim returns different answers because it considers walks with consecutive forward and backward traverses from an entity to a node without value, such as x.

From here onward, we call a walk without consecutive forward and backward traverses from an entity to a node without value **informative**, and **non-informative** otherwise. As discussed in Section 4.1, non-informative walk either do not provide any information about the relationship between entities or their information can be represented by a shorter walk. Figure 3a and Figure 3b show that non-informative walks may be present in a database but be absent from its relationship-reorganizing transformations. Hence, if we modify PathSim so that it computes similarity scores using only informative walks, it will be representation independent under relationship-reorganizing transformations. Using Definition 3 and 4, we have the following theorem.

THEOREM 2. *Let $T : \boldsymbol{L} \rightarrow \boldsymbol{K}$ be a relationship-reorganizing transformation and p be a meta-walk in $D \in \boldsymbol{L}$. There is a meta-walk r in $T(D)$ such that for each pair of entities e and f in D, we have $|p(e, f, D)| = |r(T(e), T(f), T(D))|$.*

Given entities e and f and a meta-walk p in database D and their corresponding entities and meta-walk in $T(D)$, $T(e)$, $T(f)$, and r, the numerator and denominator of $s(e, f)$ will be respectively equal to the numerator and denominator of $s(T(e), T(f))$. Hence, the modification of PathSim will return equal similarity scores for queries over a database and its relationship-reorganizing transformation. We call this extension of PathSim, *Robust-PathSim* (*R-PathSim*).

The computation of R-PathSim is similar to that of PathSim [6] with extra steps of detecting and ignoring non-informative walks. The *commuting matrix* of meta-walk $p = [l_1, \cdots, l_k]$ in database D is $M_p = A_{l_1 l_2} A_{l_2 l_3} \cdots A_{l_{k-1} l_k}$, where $A_{l_i l_j}$ is the adjacency matrix between nodes of labels l_i and l_j in D. Each entry $M_p(i, j)$ represents the the number of walks between entities $i \in l_i(D)$ and $j \in l_j(D)$. Given commuting matrix M_p, we can compute the PathSim score between i and j as $\frac{2M_p(i,j)}{M_p(i,i)+M_p(j,j)}$. However, R-PathSim uses only the informative walks. A meta-walk whose walks may not be informative is in the form of $p = [l_1, \cdots, l_i, x_{n_i}, \ldots, x_{m_i}, l_i, \ldots, l_k]$, $1 \leq i \leq k$ where l_i's are entity labels and $x_{n_i}, \cdots, x_{m_i}$ are labels of nodes without values. Meta-walk p may have non-informative walks because it contains meta-walks $s_i = [l_i, x_{n_i}, \ldots, x_{m_i}, l_i]$. Let M_{s_i} be the commuting matrix of s_i. The diagonal entries in M_{s_i} contain the number of non-informative walks of s_i. Let $M^d_{s_i}$ denote a diagonal matrix of M_{s_i}. Matrix $M_{s_i} - M^d_{s_i}$ contains the number of informative walks of s_i. To compute the number of informative walks of meta-walk p, we first find subwalks of p that start and end with same entity label and their remaining labels are non-entity labels. We call this set of meta-walks S and denote the rest of the subwalks of p R. The number of informative walks of p between each pair of entities in D is $M_p = \prod_{s\in S}(M_s - M^d_s) \prod_{r\in R} M_r$.

It may take a long time to compute the commuting matrix of a relatively long meta-walk in query time. Also, it is not feasible to precompute and store the commuting matrices for every possible meta-walk. PathSim precomputes commuting matrices for relatively short meta-walks. Then, PathSim concatenates them in the query time to get the commuting matrix of a longer meta-walk. This approach efficiently computes PathSim scores [6]. We follow the same method to compute R-PathSim scores efficiently.

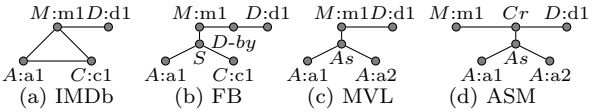
5. EMPIRICAL EVALUATION

Datasets & Setting: We use 3 datasets in our experiments. We use Arxiv High Energy Physics paper citation graph from SNAP with 34,536 nodes and 42,158 edges whose fragments are shown in Figure 3b. We use a subset of IMDb data with 2,409,252 nodes and 7,525,281 edges whose fragments are shown in Figure 4a. We also use a subset of Microsoft Academic Search data with 44,068 nodes and 44,220 edges whose fragments are shown Figure 5. We implement our and other algorithms using MATLAB 8.5 on a Linux server with 64GB memory and two quad core processors.

Robustness: We use normalized Kendall's tau to compare ranked lists. The value of normalized Kendall's tau varies between 0 and 1 where 0 means the two lists are identical and 1 means one list is the reverse of the other. As users are interested in the highly ranked answers, we compare top 5 and 10 answers. Because it takes too long to run SimRank and RWR over full IMDb dataset, we use the largest subset of IMDb with 47,835 nodes and 130,916 edges over which we can run SimRank and RWR reasonably fast to evaluate their robustness. We set the restart probability of RWR to 0.2 and the damping factor of SimRank to 0.8. We reorganize IMDb database to the structures of Freebase (FB), Movielicious (MVL) and a structure from

		IM2MV	IM2AS	IM2FB
Top 5	RWR	0.444	0.459	0.158
	SimRank	0.365	0.392	0.337
Top 10	RWR	0.404	0.415	0.155
	SimRank	0.343	0.348	0.322

Table 1: Average ranking differences for all transformations.



(a) IMDb (b) FB (c) MVL (d) ASM

Figure 4: Fragments of movies databases where A, M, C, D, S, As, Cr and $D\text{-}by$ denote *actor*, *movie*, *character*, *director*, *starring*, *actors*, *credit* and *directed-by*.

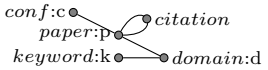


Figure 5: Fragments of MAS databases.

evc-cit.info/cit041x/assignment_css.html (ASM) whose fragments are shown in Figure 4. We denote the transformations from IMDb to Freebase as IM2FB, from IMDb to Movielicious as IM2MV, and from IMDb to ASM as IM2AS. Since MVL and ASM structures do not have any *character*, we remove character nodes in IMDb when applying IM2MV and IM2AS transformations. For query workload, we randomly sample 50 movies in IMDb database based on their degrees. Table 1 shows the average ranking differences for top 5 and 10 answers returned by RWR and SimRank over IM2MV, IM2AS, and IM2FB transformations. Because R-PathSim delivers the same rankings over these transformations, we have omitted the results for R-PathSim. Since each entity label and its consecutive entity labels in every meta-walk over IMDb are different, all walks used in the computation of PathSim are informative. Corresponding meta-walks in FB, MVL and ASM to those meta-walks in IMDb also follow the same properties. Similar to R-PathSim, PathSim is robust over these transformations. Thus, we omit the results of PathSim from the table. According to Table 1, the rankings produced by RWR and SimRank varies considerably over relationship-reorganizing transformations. We have shown in Section 4 that PathSim is not robust under certain relationship-reorganizing transformations. We use SNAP dataset and reorganize it to the structure of DBLP as depicted in Figure 3a. We randomly sample 50 papers from SNAP based on their degrees for query workload. We use [*paper*, *paper*, *paper*] on SNAP and [*paper*, *citation*, *paper*, *citation*, *paper*] on DBLP for PathSim. The average ranking differences for top 5 and 10 answers of PathSim are 0.522 and 0.495, respectively. Hence, the output of PathSim varies significantly over relationship-reorganizing transformations.

Efficiency: We evaluate the efficiency of R-PathSim and PathSim over full IMDb dataset. We transform IMDb to Movielicious (MVL) structure that contains both informative walks and non-informative walks to evaluate the impact of detecting informative walks in R-PathSim. This results in a database of 1,272,253 nodes and 2,886,494 edges. We precompute and store commuting matrices for meta-walks of size, i.e., number of labels, up to 3 to be used in query processing as done in PathSim [6]. MVL has 16 meta-walks with sizes less or equal to 3, respectively. It takes 49.5 seconds for R-PathSim to precompute and store the commuting matrices of these meta-walk which are reasonable for a preprocessing step. We have executed PathSim over the same datasets and get almost equal running times as the ones of R-PathSim. We randomly select 100 movies from MVL based on their degrees and use them as our query workloads. Given that commuting matrices up to size 3 are materialized, the average query processing time in second per query per meta-walk for meta-walks of size 5 and 7 are .033 and .023 for R-PathSim, and .029 and .021 for PathSim. Efficiency evaluation over other datasets can be found at [2].

Effectiveness: We evaluate the effectiveness of R-PathSim over the Microsoft Academic Search (MAS) dataset. For query workload, we randomly sample 50 conferences based on their degrees from the dataset. To provide the ground truth, given a conference q, we manually group all other conferences in three categories: *similar*, which contains all conferences that have the same domain as q; *quite-similar*, which includes the conferences in the domains that are closely related to the domain of q; and *least-similar* that contain conferences in the domains that are not strongly related to the domain of q. We use Normalized DCG (nDCG) to compare the effectiveness of R-PathSim and PathSim. The value of nDCG ranges between 0 and 1 where higher values show more effective ranking. We report the values of nDCG for top 5 (nDCG@5) and top 10 (nDCG@10) answers. We use meta-walk [*conf*, *paper*, *citation*, *paper*, *citation*, *paper*, *conf*] to find similar conferences based on their papers' citations. The average nDCG@5 (nDCG@10) for R-PathSim and PathSim are .264 (.315) and .261 (.313), respectively.

6. CONCLUSION

We postulated that a similarity search algorithm should return essentially the same answers for the same query over different representations of a database. We introduced a family of widespread representational shift over graph databases called relationship-reorganizing transformation. We showed that current similarity search algorithms are not representation independent and proposed a new algorithm that is representation independent under this transformation.

7. ACKNOWLEDGMENTS

This work is supported by NSF grant number 1421247.

8. REFERENCES

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases: The Logical Level*. Addison-Wesley, 1994.

[2] Y. Chodpathumwan, A. Aleyasin, A. Termehchy, and Y. Sun. Representation Independent Proximity and Similarity Search. 2015, arXiv:1508.03763 [cs.DB].

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

[4] A. Hogana, M. Arenas, A. Mallea, and A. Polleres. Everything you always wanted to know about blank nodes. *Web Semantics*, 2014.

[5] G. Jeh and J. Widom. SimRank: A Measure of Structural-context Similarity. In *KDD*, 2002.

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

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

[8] H. Tong and C. Faloutsos. Center-Piece Subgraphs: Problem Definition and Fast Solutions. In *KDD*, 2006.

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