COST-EFFECTIVE DATA STRUCTURAL PREPARATION

BY

YODSAWALAI CHODPATHUMWAN

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2018

Urbana, Illinois

Doctoral Committee:

Professor Marianne Winslett, Chair
Professor Jiawei Han
Professor Aditya Parameswaran
Professor Renée J. Miller, University of Toronto

ABSTRACT

People structure and represent their data in many different ways. One factor to consider in choosing between different representations is how the structure will affect the effectiveness of algorithms that run over the data. In fact, before sophisticated analytics can be performed, one must usually go through a data preparation phase, where the structural representation of the data is changed to be more suitable for the particular analytics procedure that will be performed. This is necessary because individual analytics algorithms are effective only for certain kinds of structural representations of their input data. Unfortunately, analytics algorithms do not come with a clear description of their desired representation. Hence, time and expertise is required to identify and materialize a suitable representation for each analytics task. In this dissertation, we address this issue in data preparation.

Our first contribution focuses on the concept of design independence, in which the intent is to create an analytics algorithm that is effective regardless of the choices of data representations. The benefit of becoming more design independent is that it will reduce or, in the most favorable outcome, remove the cost of manually finding and preparing the most effective structure or schema for the data. In this part of our work, we consider common variations of data source structure that preserve its content. For the analytics task of similarity search, we propose an algorithm that satisfies the design independence property against the studied variations. We then generalize our findings for other structural variations, and prove that it is design independent with respect to these structural variants. We show that humans find its answers at least as desirable as those provided by existing similarity search algorithms.

In the case where design independence is not achievable, we address the data preparation issue by proposing an algorithm that finds a cost-effective structure to be imposed on an unstructured dataset. Under this approach, structural information is added to the data source to improve the effectiveness of an algorithm running over the data. We leverage the information from an existing domain of concepts or an ontology to add structure to the data collection in the form of annotations. Because each concept may require different amounts of resources and time in annotating and/or maintaining the data source, we would like to find a set of affordable concepts that improves the effectiveness of an algorithm the most. This is called the cost-effective conceptual design problem. Previous works on this topic assumed that a domain of concepts is simply an unorganized set of concepts. However, real-world domains are often organized, in the form of taxonomies for example. Hence, in

this dissertation, we explore a new version of the cost-effective conceptual design problem, using taxonomies of concepts and considering multi-concept queries.

To my family and friends.

ACKNOWLEDGMENTS

This thesis would not have been possible without the support of many people. First of all, many thanks to my two advisers, Professor Marianne Winslett and Professor Arash Termehchy, for their vision and guidance throughout my years of studies and researches in the subject. Without them, I might have been lost in finding the way to complete this thesis. I would like to thank Marianne for supporting me in various life aspects, from my research, academic lessons to even how to drive a car. I would like to thank Arash for his kindness and patience in supervising me and never giving up on me for all this time. I will be forever extremely grateful to them.

I thank my committee members Professor Jiawei Han, Professor Aditya Parameswaran and Professor Renée J. Miller for their insightful comments on my research topic. I also thank Professor Elsa Gunter who allowed me to assist her teaching of CS421. I would like to also thank Lisa Pierce, my ESL instructor who helped me with my studies and practice in English conversation both inside and outside the classroom.

I would like to thank my collaborators Ali Vakilian and Amirhossein Aleyasen for their assistance in many important parts of my thesis. I thank my colleagues Vahid Ghadakchi, Ben McCamish, Jose Picado, Parisa Ataie, Huong Luu, Anku Adhikari, Hong Wei Ng and many others for providing feedback on my works, lively group meetings and research discussions. I am also thankful to the undergraduate students Godfrey Yeung, Xiayao Qian and Jiwoong Youn for their hard work in data hunting and preparation for my research.

I also appreciate all the support of my friends Thasphon Chuenchujit, Chayapa Darayon, Sakulbuth Ekvittayaniphon, Theerasit Issaranon, Woraprach Kusolthossakul, Panadda Nonthanum Pongsakorn Suppakittpaisarn, Tana Wattanawaroon and many others for my wonderful time in Urbana-Champaign.

Lastly, I am grateful to my family, especially my aunt Sansanee, for her support and encouragement throughout my entire life.

In closing, I would like to thank the National Science Foundation (NSF grant No. 1421247) who provided financial support to complete this work.

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

Before data can be queried and analyzed, developers must refine and manipulate it so that it can be effectively used in various tasks including data analytics. Generally, the tasks of data preparation include both data cleaning and data validation. In the case of data collected from multiple sources, the task also includes manipulation and transformation of data and its metadata. In fact, it has been known and agreed in the research community that one of the most important parts and most time-consuming and difficult processes in any data science project is data preparation [1]. However, available resources and time for preparing data are limited. Over large-scale databases, database managers have to choose which actions to perform during data preparation to ensure the effectiveness of future analytics tasks over the data. In many cases, changes to the structure of the data are involved.

The information needs of users over structured data range from seeking exact answers to precise queries to searching for entities or patterns similar to a given query [2–5], discovering interesting patterns [6–13], or predicting relationships and concepts [5, 14, 15]. Meanwhile, available datasets have also become more heterogeneous and more complex. Research communities have proposed numerous supervised and unsupervised algorithms to solve these exploration and analytical tasks over structured data such as similarity query processing, inexact pattern matching or relationship prediction [2, 5, 14, 16]. Since the properties of interesting and desirable answers are not precisely defined in such queries, these algorithms use intuitively appealing heuristics to choose answers that are interesting and most likely to satisfy users. In many cases, choices of database structure affect the heuristic results. Hence, researchers and developers have to spend their time and resources in converting those data into the desired structure for their algorithms to be effective.

To see why choices of data structure affect the results of database analytics, consider the excerpts of Freebase[1] in Figure 1.1(b) and the excerpts of IMDb[2] in Figure 1.1(a), which is a database harvested mainly from information in Wikipedia about entities and their relationships, from the same set of movies, their characters and the actors who played them, respectively. Figure 1.1(b) shows the same set of entities and relationships as Figure 1.1(a). It differs with Figure 1.1(a) only in terms of how it represents relationships between characters, movies and actors, in which it connects them to a common node labeled *starring*. Database researchers have recognized that different, i.e., non-isomorphic, structures can contain the same information [17, 18]. Various link-based similarity search algorithms over graph databases, such as Random Walk with Restart (RWR) [5] and SimRank [2], use prox-

[1] *www.freebase.com*
[2] *www.imdb.com*

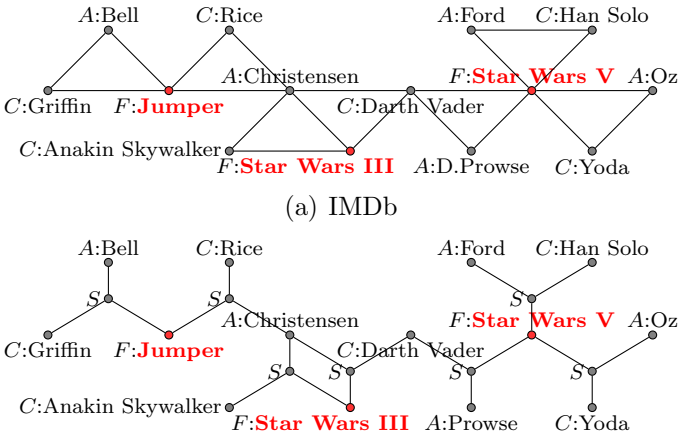


Figure 1.1: Fragments of IMDb and Freebase, where A, C, F, and S refer to *actor*, *character*, *film* and *starring*, respectively.

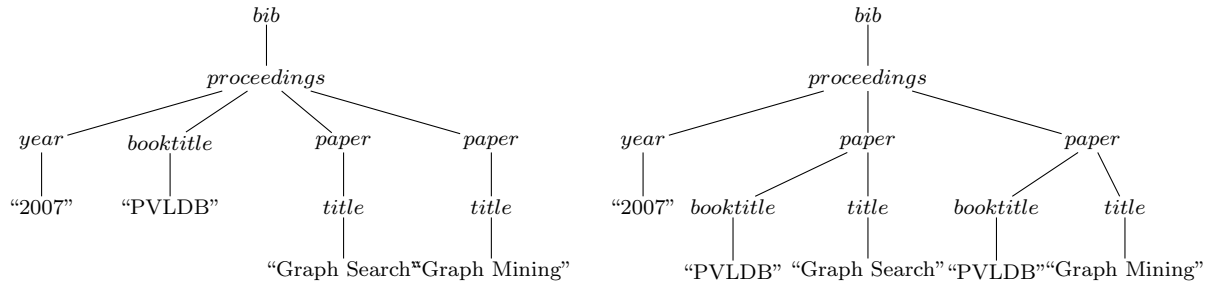


Figure 1.2: Fragments of bibliographic databases.

imity-based heuristics. RWR evaluates how likely an entity will be visited if a random surfer starts and keeps re-starting from the query entity. SimRank evaluates the similarity between two entities according to how likely it is that the two random surfers will meet each other if they start from the two entities. RWR and SimRank find *Star Wars III* more similar to *Star Wars V* than to *Jumper* in Figure 1.1(a), but find *Star Wars III* to be more similar to *Jumper* in Figure 1.1(b).

Another conventional example of structural variations is a database *normalization*. Normalization is a process that restructures data in order to reduce redundancy and improve integrity of the data. For instance, consider that a path *paper*/*booktitle* is repeated under every subtree of *preceedings* in Figure 1.2(b). The database manager may normalize the data into the fragment shown in Figure 1.2(a). Previous works over XML databases have shown that many keyword query interfaces are only effective over one of these schemas [19].

Generally, there is no canonical representation for a particular set of content, and dif-

ferent people often represent the same information using different structural representations [17, 18]. Thus, database managers have to restructure their databases to some *proper representations* to effectively run the algorithms. However, algorithms do not usually provide clear descriptions of their desired representations. It is often the case that database managers have to rely on their own expertise to find the desired structural representation of data. Another intuitive approach is to run an algorithm over all structural variations of a database and select the one with the most accurate answers. Nevertheless, computing all possible structural representations of a database is undecidable [18]. Restricting the search over particular types of structures may still result in a large number of representations to explore. This method also requires a large amount of time and resources to prepare and transform a large database into different structures.

Another approach to reduce such preparation cost is to define a *universal representation* to which all possible representations of a database can be transformed and develop algorithms that are effective over this representation. However, the experience gained with the concept of a universal representation indicates that it may not always exist [17].

This problem persists beyond the data preparation phase. Structures of large-scale databases tend to also evolve over time as data sources evolve and applications change. Hence, we should move beyond the need for constant expert attention in data management in order to keep analytics algorithms effective.

To reduce cost of data preparation due to the problem of organizational heterogeneity and evolution of large-scale data, we propose a property of data analytics that considers *design independence*, i.e., the ability to deliver the same answers regardless of the choice of structural representations of data. There have been efforts that address the problem of design independence in various database tasks [19–21]. Unfortunately, these methods are constructed and proved for specific structural variations. For instance, work done in [21] only considers database normalization. In this dissertation, we aim to address the problem over more varieties of structural variations, and to generalize the problem for any possible structural variations.

Ideally, we would like to have an algorithm that satisfies the property of design independence. Unfortunately, achieving design independence over drastic changes or differences in data structures may be costly or impractical. In other cases, database managers may also want to add structural information to improve the effectiveness of an algorithm over the database. Hence, in parallel to the approach of achieving design independence, we would like to address the data preparation problem for effective and efficient query answering over large-scale database using what is known about data representation. That is, we want to select and add a structural design, or rather a *conceptual design*, to a database in order to

maximize the improvement in effectiveness of an algorithm. For example, consider the taxonomy of concepts shown in Figure 5.4 and database of medical excerpts shown in Figure 5.1. Database managers may add structural information to the database by annotating their data to the fragments shown in Figure 5.2 or Figure 5.3. In these examples, the annotation choice of Figure 5.2 is likely to improve query answering more than the design choice of Figure 5.3 because the concepts of the former design are more specific. However, due to the higher specificity of the concepts, the cost of producing the annotation shown in Figure 5.2 is likely to be more expensive than the annotation shown in Figure 5.3. Hence, there is a trade-off between the cost of selecting a design and the effectiveness of the algorithm running over the database.

The available financial or computational resources of an enterprise are limited. It is often the case that organizing or adding a complete structure or structural concepts to a database is infeasible. In fact, researchers have recognized the overheads and costs of curating and organizing large data sets [22–24]. For example, some researchers have recently considered the problem of selecting data sources for fusion such that the marginal cost of acquiring a new data source does not exceed its marginal gain, where cost and gain are measured using the same metric, e.g., US dollars [22]. Generally, conceptual designs have been created and manually maintained by experts in a domain of interest. Such efforts are costly in terms of both manual labor and time especially over large-scale databases even with the help of machine learning. Hence, the conceptual design becomes a *cost-effectiveness* problem. More formally, a cost-effective conceptual design is a problem that seeks to find a design such that an algorithm returns the most effective answers possible while the cost of that design is within a given budget [23, 25]. Researchers have previously examined the problem of selecting a cost-effective subset of concepts from a set of concepts for annotation [25]. Nevertheless, real-world domains are usually maintained in richer structures such as a taxonomies rather than an unorganized set. Hence, in this dissertation, we address the problem for when concepts are organized in taxonomies.

1.1 DISSERTATION OUTLINE

The rest of this dissertation is organized as follows. Chapter 2 discusses previous related works. Chapter 3 discusses steps towards design independence of analytics algorithms through a subset of structural variations that preserve information content. We then present an effective similarity search algorithm that is provable to be design independence under the discussed structural variations. Chapter 4 generalizes our approach for structural design independence through the observation that structural variations beyond simple renaming

require a database constraint. We leverage the information of those constraints to predict possible structural variations that preserve information content. Then we present a relationship expression language that we prove is necessary to ensure robustness and effectiveness of analytics algorithms such as similarity search. Chapter 5 recognizes the case when design independence is not achievable, and discusses how to cost-effectively select and add structural design to a database. We then conclude our dissertation and point out possible future research directions in Chapter 6

CHAPTER 2: RELATED WORKS

2.1 DATA EXCHANGE, DATA INTEGRATION AND SCHEMA MAPPINGS

Data exchange is the act of passing data from one application or platform to another, where the source and destination may use different representations for the same information. The data exchange problem is the question of how to convert the data to the new representation as accurately as possible. This problem has been investigated in the research community over the past several decades. One of the earliest works that recognize the problem of data exchange is EXPRESS [26], dating back to 1977, which presented a low-level tool for conversion of hierarchical databases. Since then, the data exchange problem has been studied in many data models [27–32].

The data exchange problem became even more important where many applications require to run over various data sources. The data exchange problem is closely related to the data integration problem as both are involving management over heterogeneity of data formats. The goal of data integration is to synthesize data from different sources into a unified view. In data exchange, the goal is to materialize target database instances corresponding to the source data as accurately as possible. Both problems involve the restructuring of data representations; the materialization, however, is not emphasized in data integration.

In both data exchange and data integration, *schema mapping*s are a fundamental notion that specifies how a data under one representation can be transformed into a data under a different representation. Typically, schema mappings are written in high-level declarative languages, describing relationships between different schemas at the logical level without physical-level implementation details. More formally, a schema mapping consists of source schema, target schema and a set of logical formulas over source and target schemas.

Consider a schema mapping between two schemas. It is, in fact, possible that a source instance may not have any solution. On the other hand, even if the solution exists, it is also not uncommon that there are multiple (or infinitely many) non-isomorphic solutions for a single source instance. This arises the question of which solution should be materialized as an answer to the data exchange problem of an input source instance. To address this question, the concept of *universal* solutions is introduced by Fagin et al. in [33]. Under their definition, there is a homomorphism from the universal solution to every other solution of an input instance. Hence, one can obtain every solution of the input instance from a given universal solution using a homomorphism. The authors observe that universal solutions are the preferred solutions in the problem of data exchange because they are the most general

solutions.

The *chase* is a computation method for testing and enforcing the implications of a given set of dependencies in a database systems [34, 35]. The procedure is adapted for use in data exchange. It is shown in [33] that a variant of chase procedure can be used to check the existence of a universal solution and compute one if it exists. Generally, it is possible that there may not exist a finite chase for arbitrary set of dependencies. However, under some fairly general restrictions, it has been shown that there exists a polynomial-time algorithm to compute a universal solution based on the chase procedure [33].

It must be noted that, even though all universal solutions for a single source instance are homomorphically equivalent, they may not be isomorphic to each other. One of the subsequent related questions is whether some universal solutions are better or more preferred than others. Using *minimality* as the key criterion, the authors in [36] discussed that all universal solutions share a unique (up to isomorphism) common *core* when viewed as relational structures. The core is, therefore, the smallest and the best universal solution for a source instance in terms of space complexity when materializing the solution. Generally, computation of the core for arbitrary relational structures is an **NP**-hard problem unless $\mathbf{P} = \mathbf{NP}$. It is again shown in [36] that, within certain broad classes of schema mappings, there exists a polynomial time computation to find the core of universal solutions.

One of the fundamental tasks in data exchange is query answering over the target schema. Typically, the task focuses on computing *certain* answers of queries [29–32, 37] The task, indeed, was considered earlier in the study of incomplete databases. However, the challenge of the problem in the context of data exchange, as well as data integration, is that queries are not posed on a single database, but rather on the set of all databases satisfying certain specifications. The *certain answers* problem is the decision problem of whether the fact in the source database can be found on the (possibly infinite) set of target databases.

Clearly, by the definition of certain answers, the computation for the problem is infeasible due to the computation over sets of infinite solutions of a schema mapping. Instead, in data integration, the approach to address this problem is to rewrite queries over the target instance to queries over the source instance. In data exchange, the materialized solution is used to obtain the certain answers of queries. However, since there may be infinitely many solutions, researchers try to identify situations where it is feasible to evaluate certain answers. It is shown in [29] that the certain answers of a *union* of conjunctive queries can be obtained by evaluating the query on a universal solution. The paper also discusses other variations of a query such as conjunctive queries with inequalities; however, the complexity of the problem can be intractable [29].

Nevertheless, data exchange has been extensively studied and explored during the past few

decades. Early works mainly focus on relational database and later in XMLs. A system such as Clio is built to support data exchange between any combination of XML and relational databases [27, 28, 38]. The system leads to study of many fundamental issues in data exchange and data management including the concept of universal solutions [36]. The main studies of data exchange over XML databases are published in [30]. The studies of the subject particularly over graph databases, however, received small attention and are explored much later than other traditional databases. One of the earliest works on data exchange for graph databases is done by Calvanese et al., which addresses view-based query rewriting for graph database [39]. More recent in-depth theoretical studies of data exchange and schema mappings over graph databases appeared in [40], which focused on simplification of schema mappings, and in [40], which focuses on the data exchange problem in the context of navigational queries.

Like the work on data exchange, we are interested in the problem of obtaining equivalent answers to queries posed over different representations of information. Our work differs in that we focus on more complex analytic tasks than traditional database queries; this is important because unlike a traditional query, these analytics tasks tend to rely heavily on the structure when computing their answers, so that changes in the structure lead to changes in the answer. As a simple example, consider a program that computes the distance between two entities in a graph by determining the length of the shortest path between them. Adding the transitive closure of the edges will change the answer. Traditional database queries do not make this kind of use of structure.

We also differ from the work on data exchange in that our goal is never to materialize a particular representation of the data. Instead, our goal is to find algorithms for analytics tasks that give equally effective answers over all representations that exactly preserve the content of the data.

2.2 DESIGN INDEPENDENCE

Work on design independence aims to ensure that an algorithm will return the same effective results over a dataset, no matter how its data is represented. We must note that traditional problems for data exchange are focusing on algorithms that identify and compute certain answers. Given that the setting of the design independence problem assumes equivalence of information between databases, in some sense, the certain answers are guaranteed. Hence, design independence is focusing more on algorithms which compute and return a ranked list of answers for a query, and the user is prepared to deal with some degree of uncertainty.

In addressing the problem of design independence, the work was built on the area of data management which seeks to characterize equivalent content between two databases. In data exchange and data integration, researchers have previously studied and explored the properties of information preservation and query preservation for relational and XML schema mappings [18, 41]. The original purpose of the studies is to identify schema mappings where a query over a data source can be translated to a new query over the target schema. Research on design independence, however, seeks to forgo query translation entirely and works to identify the extent to which this is possible, i.e., the set of schema mappings such that an analytic algorithm will return the same answer over any pair of databases related by these mappings.

One generic and early approach to solve the problem is to define a *universal* schema in which all representations of database can be transformed and/or used to develop the desired algorithms. In [42], researchers have proposed a strong form of logical data independence based on representing a database under a *universal-relation* model. Despite the effort, experience gained from the idea of the universal relation indicates that such a universal schema does not always exist. Further, studies in data exchange have shown that the complexity of materializing a database from schema mappings is considerably impractical for a very large database.

Researchers have previously proposed a keyword query interface over XML datasets that provably returns the same answers across databases with different structures but equivalent information content [19, 20]. There is also an effort on the same problem for learning algorithms over relational databases [21]. Unfortunately, these methods address the problem only over a subset of all possible schematic variations. For instance, the work done in [21] is based only on schema normalization. It is unclear how to adapt these efforts to create a design-independent algorithm for other representation variations or other schema mappings in general.

2.3 CONCEPTUAL DESIGN

Conceptual design is a traditional topic in data management [43]. If a dataset has no structure, e.g., it is natural language text, then one approach to improve query-answering effectiveness is to add structure to it, by annotating portions of the document with the concepts that they refer to. As discussed earlier, it is too expensive to annotate a large dataset with all potentially relevant concepts. Thus given a budget for annotation and a set of concepts, the goal of cost-effective conceptual design is to decide what to annotate in order to improve query-answering effectiveness the most, without exceeding the budget. Researchers

have examined the problem of selecting cost-effective designs from an unorganized set of concepts for annotation [25]. It has been shown that, over a set of unorganized concepts, finding the most cost-effective conceptual design is **NP**-hard [25]. In the same work, the authors proposed a fully-polynomial-approximation algorithm and a greedy algorithm to address the problem over the case of unorganized concepts.

Nevertheless, concepts are usually organized into richer structures such as a taxonomy rather than an unorganized set. In this thesis, we investigate the cost-effective design problem in this new setting.

CHAPTER 3: TOWARDS STRUCTURAL DESIGN INDEPENDENCE

One way to reduce the cost of data preparation is to eliminate the need to transform the data into a different structural representation. Our first step in order to reduce the cost of data preparation is by an effort to remove the need in transforming the structured data into another representation. In this chapter, we consider fairly general structural variations of data and discuss their fundamental properties in order to design an algorithm that is design independent against these structural variations. Since analytics tasks include many applications that involve examination of the database in order to draw certain conclusions, we make a case study of one particular application, namely, similarity search over graph databases.

3.1 BACKGROUND

Finding similar or strongly related entities is a fundamental and important problem in graph data management [2–5, 44]. It is a building block of algorithms for various important problems, such as similarity query processing, pattern query matching, and community detection [2, 8, 9, 45, 46]. Since the properties of *similar* or *related* entities cannot be precisely defined, current similarity and proximity 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 [5]. *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 [2]. Figure 1.1(a) shows fragments of IMDb[1], which contains information about movies, actors and characters. To represent the relationship between a character, its movie and the actor who played the character, IMDb connects these entities through some edges. Assume that a user asks for the most similar movie to *Star Wars III* in Figure 1.1(a). Since the RWR and SimRank score of *Star Wars V* (RWR-score = 0.061, SimRank-score = 0.213) are larger than those of *Jumper* (RWR-score = 0.060, SimRank-score = 0.185), RWR and SimRank find *Star Wars III* more similar to *Star Wars V* than to *Jumper*, 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 [17, 18].

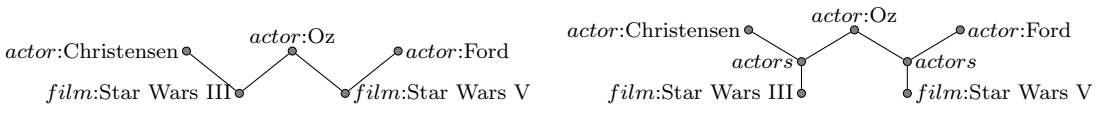
[1] *www.imdb.com*

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 [17]. The choice of representation, however, may unintentionally influence the output of current similarity search algorithms. As we have shown in Chapter 1 by considering the excerpts of Freebase[2] in Figure 1.1(b), Figure 1.1(b) contains information about exactly the same set of entities and relationships as Figure 1.1(a). It differs with Figure 1.1(a) 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 1.1(a), 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 1.1(b).

The power of similarity search algorithms, however, remains out of the reach of most users as today's similarity search algorithms are usable only by trained data analysts who can predict which algorithms are likely to be effective for particular representations of a dataset. Because a similarity search algorithm may not be effective on the representation of input database, users have to restructure the input database to some proper representation over which the algorithm returns effective results, i.e., delivers the answers that a domain expert would judge as relevant. 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 such representations. 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 large 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 structural representations of a dataset and select the representation(s) with the most accurate answers. Nevertheless, because most similarity algorithms are unsupervised, there is no validating data available to measure the effectiveness of these algorithms over certain representations. Moreover, it is undecidable to compute all possible structural representations of a graph database [18]. If we restrict the set of possible representations, a database may still have enormous representational variations. For example, the number of vertical decompositions of a relational table may be exponential in the number its attributes [17]. As graph databases have less restrictive schemas than relational databases, they may have more representational variations and need more time to generate and run algorithms over them. Researchers have proposed the idea of *universal relation* to achieve some level of schema

[2] *www.freebase.com*



(a) IMDb (b) Movielicious

Figure 3.1: Fragments of movie databases.

independence for SQL queries over relational databases [17]. 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 [17]. Furthermore, it may not be practical to force developers to organize their data in and create their algorithms for a particular style of representation.

We propose the property of *design independence* for similarity search algorithms, i.e., the ability to deliver the same answers regardless of the choices of structure for organizing the data. In our context, a *design* of a database denotes structural representation, e.g., schema, of the database. To the best of our knowledge, the property of design independence has not previously been explored for similarity search algorithms and/or graph databases. We believe that the key to the success of building design independent analytics in general and similarity search algorithms in particular is to modify current algorithms to become design 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. They have also been shown empirically to be effective over certain representations of databases, which provide evidences that their reasonable modifications may be effective over more databases.

3.2 RELATED WORKS

The architects of relational models envisioned the desirable property of *logical data independence*. Oversimplifying a bit, this means that an exact query should return the same answers regardless of the schema chosen for the data [17, 47]. One may consider the idea of design independence as an extension of the principle of logical data independence for similarity and proximity search algorithms. Nevertheless, these ideas differ in an important aspect. One may achieve logical data independence for database applications by creating a set of views over the database, which keep the application unaffected from modifications in the database schema [17]. However, characteristics of the ideal representations for similarity and proximity search algorithms are not clearly defined. Also, graph databases follow far

less rigid schemas and are more amenable to change than relational databases. Hence, it takes far more time and more in-depth expertise to find the proper representation as well as create and maintain the mapping between the database and this representation.

Researchers have proposed keyword query interfaces over tree-shaped XML data that return the same answers to a keyword query over databases with equivalent content but different choices of structure [19]. We, however, introduce and study the concept of design independence for a different problem and data model. The task of similarity and proximity 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 design independence and developing design independent algorithms.

Researchers have also analyzed the stability of random walk algorithms in graphs against relatively small perturbations in the data [48–50]. We also seek to instill *structural robustness* in graph mining algorithms, but we are targeting robustness in a new dimension: robustness in the face of variations in the representation of data. Researchers have provided systems that help users with transforming and wrangling their data [51–54]. We also address the problem of data preparation but using a difference approach: *eliminating the need to wrangle the data*.

Researchers have proposed several normal forms for relational and tree-shaped XML schemas [17, 55, 56]. Nevertheless, we focus on finding design independent similarity search algorithms rather than transforming the database to a particular representation with some desirable properties. Moreover, because similarity search algorithms usually operate over graph databases without rigid schemas, our transformations are defined over a much less restrictive schemas than relational schemas or XML DTDs. Our entity-rearranging transformations somewhat resemble normalization/de-normalization in relational and tree-shaped XML databases. Our transformations, however, modify the connections between entities in the database instead of creating or removing duplicates. They are also defined over graph databases rather than relational or tree-shaped databases.

Blank nodes represent the existence of resources without any global identifier, i.e., existential variables, in RDF databases [57, 58]. As blank nodes often convey redundant information, researchers have proposed methods to remove them from RDF databases [57, 58]. However, our goal is not to remove certain nodes from a database. Further, because our databases do not contain any existential variable, we use a different approach to ensure that our transformations do not modify the information content of a database. For instance, as opposed to our transformations, the mappings that eliminate blank nodes may not be invertible. Some serializations of RDF data, such as RDF/XML, may assign labels and iden-

tifiers within the scope of a document to blank nodes in the document [58]. Our framework covers these applications of blank nodes. Nevertheless, it also addresses the representational shifts over databases that do not contain any blank node. Researchers have proposed algorithms to convert RDF data sets that contain certain relationships in *RDF Schema*, such as *rdfs:subClassOf*, to some normal forms [57]. Our transformations, however, are not limited to particular set of relationships.

Schema mapping has been an active research area for the last three decades [17]. In particular, researchers have defined schema mappings over graph databases as constraints in some graph query language in the context of data exchange [32]. As opposed to the transformations in our work, the original and transformed databases in those settings may not represent the same information. We also focus on evaluating the design independence of similarity search algorithms rather than traditional questions in schema mapping and data exchange, such as computing the transformed database instances.

3.3 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 [58, 59]. Figure 1.1(b) is an example of using nodes without values to represent relationships between entities. It may be easier to express complex relationships using nodes without values [59]. About 30% of 1.23 billion RDF triples collected from the Web contain nodes without values [58]. We call the nodes with values *entities* [2, 4]. We assume that each label in a set of labels L is used to denote 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 [2, 4, 5, 9]. For example, query ($film$:Star Wars III) over Figure 1.1(b) 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 $q(D)$.

3.4 DESIGN INDEPENDENCE

Intuitively, a design 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 [17–19]. 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 1.1(a) and 1.1(b) 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 1.1(a) to the one in Figure 1.1(b).

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 1.1(a), using the information in its transformed one, e.g., Figure 1.1(b). However, a transformation that removes the edges between each *film* node and its neighboring *actor* and *character* nodes from Figure 1.1(a) is not invertible because there is not sufficient information in the transformed database to recover the relationship between *film*, *actor*, and *character* nodes. More formally, $T : \mathbf{L} \rightarrow \mathbf{K}$ is *invertible* if and only if 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 [17, 18].

To precisely define design 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, hence, 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 [4]. For example, they assume that the nodes with label *film* in Figure 1.1(a) 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 if and only if they are lexicographically equal: they have the same length and contain the same characters in the same positions. Our approach can also support other definitions of equality between data values.

Definition 3.1. *Transformation* $T : \mathbf{L} \rightarrow \mathbf{K}$ *that transforms database* $D = (V, E, \mathcal{L}, \mathcal{A})$ *to* $T(D) = (V_T, E_T, \mathcal{K}, \mathcal{A}_\mathcal{T})$ *is entity preserving if and only if there is a bijective mapping* M *between entities in* V *and* V_T *such that*

- *For all entities* $v \in V$*, we have* $A(v) = A_T(M(v))$*.*
- *For all entities* $v_1, v_2 \in V$ *that* $\mathcal{L}(v_1) = \mathcal{L}(v_2)$*, we have* $\mathcal{K}(M(v_1)) = \mathcal{K}(M(v_2))$*.*

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 3.2. *Similarity search algorithm* S *is design independent under similarity preserving transformation* $T : \mathbf{L} \rightarrow \mathbf{K}$ *if and only if, for each database* $D \in \mathbf{L}$ *and* $T(D) \in \mathbf{K}$ *and every query* q *over* D*, there is a bijective mapping* N *between* $q(D)$ *and* $T(q)(T(D))$ *such that*

- *for all entities* $v \in q(D)$ *and* $N(v) \in T(q)(T(D))$*, we have* $N(v) = T(v)$
- *entity* v *appears before entity* u *in* $q(D)$ *if and only if* $N(v)$ *ranks before* $N(u)$ *in* $T(q)(T(D))$*.*

The first condition in Definition 3.2 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 3.2, 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 $T(q)(T(D))$. In short, **if an algorithm is design independent against a transformation, then it is (structurally) robust against that particular structural variation of representations defined by the transformation. We call the the degree of such property a *structural robustness* of an algorithm.** We must note that the two terms, *design independence* and *structural robustness*, may be used interchangeably throughout this dissertation.

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 3.2 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 [18]. 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.

3.5 RELATIONSHIP REORGANIZATION

3.5.1 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 3.1(b) uses a set of edges to represent the relationship between a movie and its actors. However, Figure 3.1(a) 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. For example, $w_1 =$ [*actor*:Ford, *actors*, *film*:Star Wars V] is a walk in Figure 3.1(a). Intuitively, a walk represents some relationship between its entities. For example, walk w_1 in Figure 3.1(a) shows that actor *Ford* has played in movie *Star Wars V*. One may use paths to capture relationships between entities in a database [4]. But, we show in Section 3.6 that walks represent more varieties of relationships than paths, which enables us to achieve design independence over more transformations. To simplify our framework, we assume that each database is a simple graph: it has at most one edge between a pair of 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 traversals 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 3.1(b) expresses the relationship between actor *Ford* and movie *Star Wars V*. It contains consecutive forward and backward traversals 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 traversals from *film*:Star Wars V to *actors*. Another example of such walks in Figure 3.1(b) is [*film*:Star Wars V, *actors*, *film*:Star Wars V]. This walk does not provide any information regarding the relationships between entities in the database. Hence, unless otherwise noted, we consider only walks that does *not* have any consecutive forward and backward traversals 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 3.1(b) 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 3.1(a) and meta-walk [*actor*, *actors*, *film*] in Figure 3.1(b) 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* if and only if they have equal arities and their corresponding positions contain the same label and equal values. Two walks are *content equivalent* if and only if their values are equal. For instance, walk [*actor*:Ford, *film*:Star Wars V] in Figure 3.1(a) and walk [*actor*:Ford, *actors*, *film*:Star Wars V] in Figure 3.1(b) are content equivalent. We show content-equivalent walks w and x as $w \equiv x$. Let $p(D)$ denote the set of walks in database D whose meta-walk is p.

Definition 3.3. *Meta-walks p_1 in database D_1 and p_2 in database D_2 are* content equivalent *if and only if there is a bijection $M : p_1(D_1) \to p_2(D_2)$ where for all $w \in p_1(D_1)$, $w \equiv M(w)$.*

Meta-walks [*actor*, *film*] in Figure 3.1(b) and [*actor*, *actors*, *film*] in Figure 3.1(a) 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 this section. However, this straightforward definition ignores some interesting transformations. For example, intuitively the databases in Figure 3.1(a) and Figure 3.1(b) contain the same information. But, there is not any meta-walk in Figure 3.1(a) that is content equivalent to meta-walk p_3=[*actor*, *actors*, *actor*] in Figure 3.1(b). By looking closely at the Figure 3.1(b) and original Movielicious[3] 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 3.1(b). Hence, if a transformation maps p_4 to a content-equivalent meta-walk in Figure 3.1(a), 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 $w \sqsubseteq x$, if and only if 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 because the edge (v_1, v_3) is not in x_1. Meta-walk p is a subwalk of meta-walk r, denoted as $p \sqsubseteq r$, if and only if 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 3.1(b).

Definition 3.4. *Given meta-walks p and p' in database D, p'* includes *p if and only if 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.*
- *p' has at least one entity label that is not in p.*

For example, meta-walk [*actor*, *actors*, *film*, *actors*, *actor*] includes [*actor*, *actors*, *actor*] in Figure 3.1(b). A meta-walk p in database D is *maximal* if and only if it is not included in any other meta-walk. For instance, [*actor*, *actors*, *film*, *actors*, *actor*] is maximal in Figure 3.1(a). 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

[3] *www.netwalkapps.com/movies-xml-format*

meta-walks in the set of databases $\mathbf{L}$. Similarly, we denote the set of all maximal meta-walks in $\mathbf{L}$ as $\mathcal{P}_{\max}(\mathbf{L})$.

Definition 3.5. *Transformation $T : \boldsymbol{L} \to \boldsymbol{K}$ is* relationship reorganizing *if and only if 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 3.1(b) to Figure 3.1(a) and Figure 1.1(a) to Figure 1.1(b) are relationship-reorganizing. Using Definitions 3.4 and 3.5, we have the following theorem.

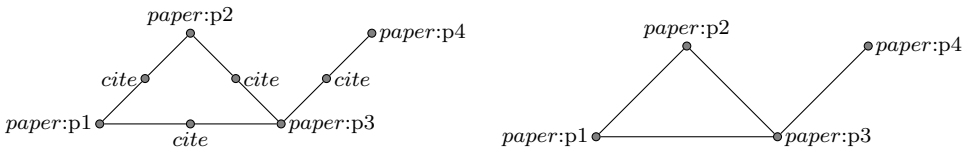
Theorem 3.1. *Every relationship-reorganizing transformation is similarity preserving.*

Proof. Let T be a relationship-reorganizing transformation. Definition 3.5 implies that T is entity preserving. To show that T is invertible, we prove that given databases D and E, if $T(D) = T(E)$ then $D = E$. That is, a walk belongs to D if and only if it is in E. Consider walk $w \in p(D)$. If p is maximal, there is a bijection M that bijectively maps w to walk $w' \in M(p)(T(D))$. Because $T(D) = T(E)$, w' is bijectively mapped to a walk $w'' \in p(E)$ where $w \equiv w''$. Hence, $w = w''$. If p is not maximal, Definition 3.4 implies that w is bijectively mapped to a walk w''' where $w \subseteq w'''$. Using similar argument, we have that w''' exists in D if and only if it exists in E, and so is w.

3.5.2 Towards 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 [5], pairwise random walk, e.g., SimRank [2] and P-Rank [3], or relationship-constrained framework, e.g., PathSim [4, 44]. 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 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 3.1, we have shown that RWR and SimRank return different results over a database and its relationship reorganization in Figure 1.1.



(a) DBLP-citation (b) SNAP

Figure 3.2: Fragments of two citation databases

PathSim measures the similarity between entities over a given relationship [4]. For example, it may compute the similarity of two movies in a movie database based on their common actors. PathSim uses meta-walks to represent relationships between entities. For instance, the relationship between two movies in Figure 1.1(b) 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 3.2 shows fragments of DBLP[4], called DBLP-citation, and SNAP[5] that contain information about citations. Consider the meta-walk s = [*paper*, *cite*, *paper*, *cite*, *paper*] in Figure 3.2(a), and its corresponding meta-walk s' = [*paper*, *paper*, *paper*] in Figure 3.2(b). s has a walk between entities $p3$ and $p4$, x = [*paper*:p3, *cite*, *paper*:p4, *cite*, *paper*:p4]. But, there is no corresponding walk of meta-walk s' between $p3$ and $p4$ in Figure 3.2(b). Hence, PathSim reports $p1$ to be more similar to $p2$ than $p3$ in Figure 3.2(a), but considers $p1$ to be more similar to $p3$ than $p2$ in Figure 3.2(b). 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 3.5.1, non-informative walks either do not provide any information about the relationship between entities or their information can be represented by a shorter walk. Figure 3.2(a) and Figure 3.2(b) 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 robust under relationship-reorganizing transformations. Using Definition 3.4 and 3.5, we

[4] *dblp.uni-trier.de*
[5] *snap.stanford.edu*

have the following theorem.

Theorem 3.2. *Let $T : \boldsymbol{L} \to \boldsymbol{K}$ 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 $|p(e, f, D)| = |r(T(e), T(f), T(D))|$.*

Proof. Suppose p is maximal. According to Definition 3.5, there is a maximal meta-walk $T(p)$ in $T(D)$ s.t. $p \equiv T(p)$. Because there is a bijection that maps each informative walk of p to an informative walk of $T(p)$ with equal value, we have $|p(e, f, D)| = |T(p)(T(e), T(f), T(D))|$. If p is not maximal, according to Definition 3.4, we can find a maximal meta-walk p' in D that includes p s.t. $|p(e, f, D)| = |p'(e, f, D)|$. Using similar arguments to when p is maximal, we have that there exists a maximal meta-walk $T(p')$ in $T(D)$ s.t. $|T(p')(T(e), T(f), T(D))| = |p'(e, f, D)|$. Hence, $|p(e, f, D)| = |T(p')(T(e), T(f), T(D))|$.

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 [4] 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 $M^i_p = \prod_{s \in S}(M_s - M^d_s) \prod_{r \in R} M_r$.

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 [4]. We follow the same method to compute R-PathSim scores efficiently.

Users may not know the structure of the database and cannot supply any input meta-walk. One may solve this problem by computing the (weighted) average of similarity scores over maximal meta-walks between entities [4]. Definition 3.5 provides that there is a bijection between all maximal meta-walks in a database and its relationship-reorganizing transformation. Also, Theorem 3.2 guarantees that R-PathSim returns equal scores for each maximal meta-walk over a database and its transformation. Hence, the combined similarity scores are equal in the original and transformed databases.

In order to find a set of maximal meta-walks, we first find a set of meta-walks in the database. Then we check if the meta-walks found are maximal or not. Algorithm 3.1 provides a framework on checking whether a given meta-walk is maximal. The underlying idea is that, if a meta-walk p is not maximal, there exists a meta-walk p' that includes p. That is, p' must contain an additional entity label to p. Using Definition 3.4, we check whether each walk of p is a subwalk of exactly one walk of p'. If there is p' that includes p, then p is not maximal. Otherwise, p is maximal. The running time of Algorithm 3.1 is $O(nd^3m)$ where n is the size of a given meta-walk p, d is the average degree of nodes, and m is the number of walks of p in the database.

For further optimization, if there are many meta-walks between the query node and the candidate answers in the database, one may save processing time by limiting the set of meta-walks over which the aggregated score is computed. One may do so by selecting the maximal meta-walks $p = rr^{-1}$, where r^{-1} is a meta-walk that is the reverse of r, such that r contains only distinct entity labels and only a given number of entity labels. Definition 3.5 guarantees that, for each maximal meta-walk r, there is exactly one maximal meta-walk r' in the transformed database such that $r \equiv r'$. Further, the number of entity labels of r and r' must be the same. That is, if p is used over the database, then $p' = r'r'^{-1}$ is also used over its transformation . Similar to Theorem 3.2, we have that the R-PathSim score computed using p over the database and using p' over its transformation are equal. Therefore, the aggregated R-PathSim score computed over these sets are equal across the original database and its transformations.

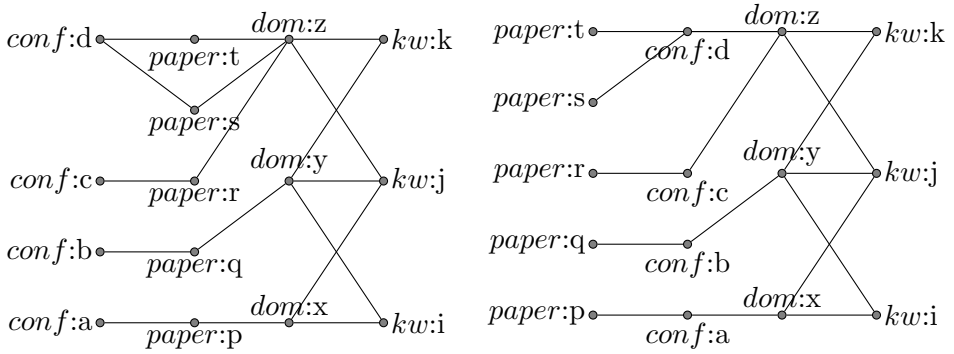
Next, we discuss why it is possible to modify PathSim and create a design independent algorithm and whether it is feasible to apply similar changes to RWR and SimRank to make them design independent over relationship-reorganizing transformations. The concept of meta-walk (meta-path) captures how relationships between entities are represented in a

Algorithm 3.1: Check a meta-walk if it is maximal

Input: Database $D = (V, E, L, A)$, Meta-walk $p = [l_1, ..., l_n]$
Output: ACCEPT if p is maximal, or REJECT if p is not maximal

```
1  foreach i = 2...n − 1 do
2      S_i ← set of all meta-walks [l_i, l] or [l_i, l', l] in D where l is an entity label and l' is
         not an entity label
3      foreach r ∈ S_i do
4          foreach w = [v_1, ..., v_n] ∈ p(D) do
5              if there exists no walk or more than two walks in r(D) from v_i then
6                  Go to process next r ∈ S_i   // Assume p' = [l_1, ..., l_i]rr^{-1}[l_i, ..., l_n] where
                     p ⊏ p'.  p' does not include p.
7              end
8          end
9          // Each walk of p is a subwalk of exactly one walk of p'.  Hence, p'
               includes p.
10         return REJECT
11     end
12 end
13 return ACCEPT
```

database. Relationship reorganizing transformations do not add any new type of relationship to or remove any existing type of relationship from a database. Because R-PathSim quantifies the amount of similarity separately for each type of relationship between two entities, it has a chance of returning equal scores over a database and its relationship reorganizing transformations. R-PathSim also leverages the concept of meta-walk to detect and ignore the spurious walks in each meta-walk that may not be present in some representations of the database. RWR and SimRank, however, do not support the concepts similar to meta-walk, so it is not clear how they can be modified to be robust over relationship reorganizing transformations. Nevertheless, one can define RWR or SimRank score between two entities in a database for a given meta-walk [4]. RWR and SimRank may be modified within this context to ignore the non-informative walks between entities to provide the equal scores over relationship reorganizing transformations. Analyses of such extensions are interesting subjects of future work.



(a) Original representation (b) Alternative representation

Figure 3.3: Fragments of some representations for MAS data with FDs *paper* → *conf* and *conf*→ *dom*.

3.6 ENTITY REARRANGEMENT

3.6.1 Entity-Rearranging Transformation

Different databases may represent the same relationship between a set of entities by connecting them using different sets of edges. Consider Figure 3.3 that shows the original and an alternative representation for Microsoft Academic Search[6] (MAS for short) data. Both databases contain entities of semantic types paper, conference, domain and keyword, which are labeled as *paper*, *conf*, *dom* and *kw*, respectively. The domains of papers and conferences show their areas, e.g., *database* and *data mining*. The keyword entities contain the keywords of domains, e.g., *indexing* for *database* domain. Each paper is published in only one conference and each conference belongs to only one domain. The database in Figure 3.3(a) expresses the relationship between a paper and its conference and domain by connecting each paper to both its conference and its domain. On the other hand, the database in Figure 3.3(b) represents the same relationship by connecting each paper to its conference and connecting each conference to its domain. We call this representational shift that rearranges entities in a database an *entity-rearranging* transformation.

Because each paper in Figure 3.3(b) has only one conference and each conference has only one domain, we can switch the relative position of a paper and a conference, and get Figure 3.3(a) without losing or adding any relationship to the one represented in Figure 3.3(b). On the other hand, assume that a paper can be published in multiple conferences from different domains in a database that follows the representation of Figure 3.3(b). If we rearrange the positions of papers, conferences, and domains in this database according to the represen-

[6] *www.academic.research.microsoft.com*

tation in Figure 3.3(a), each conference of a paper will be connected to all domains of every conference in which the paper is published. Hence, we unintentionally add new relationships between conferences and domain that are not available in the original database. Also, we will not be able to recover the original set of relationships between a conference of its domain in the original database. Hence, an entity-rearranging transformation preserves the information of a database if certain entities in the database satisfy some dependencies. The following definition formalizes these dependencies. Let $l(D)$ denote all nodes in database D with label l.

Definition 3.6. *Given meta-walk $p = [l_1, \ldots, l_n]$ in the set of databases $\boldsymbol{L}$, $\boldsymbol{L}$ satisfies* functional dependency *(FD) $l_1 \xrightarrow{p} l_n$ if and only if for every $D \in \boldsymbol{L}$ if walks $[e, \ldots, f]$ and $[e, \ldots, g]$ of meta-walk p are in D, then $f = g$.*

For example, the FDs in Figure 3.3(a) are *paper* $\xrightarrow{p_1}$ *conf*, *paper* $\xrightarrow{p_2}$ *dom*, and *conf* $\xrightarrow{p_3}$ *dom*, where p_1 = [*paper*,*conf*], p_2 = [*paper*,*conf*,*dom*] and p_3 = [*conf*,*dom*]. Given meta-walk $p = [l_1, l_2]$, we write the FD $l_1 \xrightarrow{p} l_2$ as $l_1 \rightarrow l_2$ for brevity.

Intuitively, an entity-rearranging transformation should preserve the label and values of entities, the relationships between entities, and the FDs of a database to preserve its information. For example, there is a bijection between entities in Figure 3.3(a) and Figure 3.3(b) that preserves their labels and values. Moreover, if there is not any FD between some entities, an entity-rearranging transformation must not rearrange them. In other words, if we have edge (e, f) in database D and there is no FD between e and f, an entity-rearranging transformation T must map e and f to entities $T(e)$ and $T(f)$ in $T(D)$ with edge $(T(e), T(f))$. Similarly, if there is no edge between the aforementioned entities in D, there must not be any edge between them in $T(D)$. Furthermore, the transformed database must satisfy essentially the same FDs as the original database. That is, if there is an FD between entities of semantic types l_1 and l_2 in the original database, there must be an FD between the entities of l_1 and l_2 in the transformed database. Otherwise, as explained in the preceding paragraph, the transformation may introduce spurious relationships between entities. However, the corresponding FDs in the original and transformed databases may be represented using different meta-walks. For instance, FD *conf* $\xrightarrow{[conf,paper,dom]}$ *dom* in Figure 3.3(a) is mapped to *conf* $\rightarrow$ *dom* in Figure 3.3(b). The following definition formalizes the aforementioned intuitions. Let $F_{\mathbf{L}}$ denote the set of FDs satisfied by the set of databases $\mathbf{L}$.

Definition 3.7. *A transformation $T : \boldsymbol{L} \rightarrow \boldsymbol{K}$ that maps database $D = (V_D, E_D, \mathcal{L}, \mathcal{A}_D)$ to database $T(D) = (V_{T(D)}, E_{T(D)}, \mathcal{K}, \mathcal{A}_{T(D)})$ is* entity rearranging *if and only if there is a bijection $M : V_D \rightarrow V_{T(D)}$ such that*

- *for all* $v \in V_D$, $\mathcal{L}(v) = \mathcal{K}(M(v))$ *and if* v *is an entity,* $\mathcal{A}_D(v) = \mathcal{A}_{T(D)}(M(v))$.
- *for all* $(u, v) \in V_D$ *where neither* $\mathcal{L}(u) \to \mathcal{L}(v)$ *nor* $\mathcal{L}(v) \to \mathcal{L}(u)$ *are in* $F_{\boldsymbol{L}}$*, we have* $(u, v) \in E_D$ *if and only if* $(M(u), M(v)) \in E_{T(D)}$.
- *for all* $(u, v) \in V_D$ *where neither* $\mathcal{L}(u) \to \mathcal{L}(v)$ *nor* $\mathcal{L}(v) \to \mathcal{L}(u)$ *are in* $F_{\boldsymbol{L}}$*, we have* $(u, v) \in E_D$ *if and only if* $(M(u), M(v)) \in E_{T(D)}$.
- *there is a bijection* $N : F_{\boldsymbol{L}} \to F_{\boldsymbol{K}}$ *such that if* $N(l_1 \xrightarrow{p} l_2) = l_1 \xrightarrow{p'} l_2$*, for all entities* $e, f \in V_D$*,* $p(e, f, D)$ *is empty if and only if* $p'(M(e), M(f), T(D))$ *is empty.*

Using Definitions 3.6 and Definition 3.7, we have the following.

Theorem 3.3. *Every entity-rearranging transformation is similarity preserving.*

Proof. Let T be an entity-rearranging transformation. We need to prove that given databases D and E such that $T(D) \cong T(E)$, then $D \cong E$. That is, a walk exists in D if and only if it exists in E. Let $w = [u, v]$ be in $p(D)$. If w follows the second condition in Definition 3.7, then $[M(u), M(v)]$ exists in $T(D)$ (and $T(E)$). So $[u, v]$ must also exist in E. Otherwise, if w does not follow the second condition, w must be a walk of some meta-walk p for some FD. Using the third condition of Definition 3.7, then p is bijectively mapped to a meta-walk p' in $T(D)$. Also, there must exist a walk $w' \in p'(M(u), M(v), T(D))$ (and in $T(E)$). Using similar arguments, p must exist in E where w' is in $T(E)$ if and only if w is in E. Assume $w = [u_1, ..., u_{k>2}]$ is in D. We have that each $[u_i, u_{i+1}]$, $i = 1...k-1$, exists in D if and only if it exists in E. Therefore, w also exists in D if and only if it exists in E. Because the first condition in Definition 3.7 implies that each entity-rearranging transformation is entity preserving, every entity-rearranging transformation is similarity preserving.

Entity-rearranging transformations resemble (de-)normalization in relational and tree-shaped XML databases [17, 55]. They, however, modify the connections between entities in the database instead of removing duplicates and are defined over graph databases that follow less restrictive schemas than relational schemas or DTDs.

3.6.2 Extension of R-PathSim

Because entity-rearranging transformations modify the topology of a database, RWR and SimRank are not robust under these transformations. For example, consider the entity-rearranging transformation between Figure 3.3(a) and Figure 3.3(b). RWR and SimRank find *paper*:p to be more similar to *paper*:r than *paper*:t in Figure 3.3(b). However, they find *paper*:p to be more similar to *paper*:t than *paper*:r in Figure 3.3(a). R-PathSim and PathSim are also not robust under entity-rearranging transformations. A user may like to

find conferences similar to *conf*:b based on their common keywords using meta-walk p_1 = [*conf*, *dom*, *kw*, *dom*, *conf*] in Figure 3.3(b). R-PathSim finds *conf*:a and *conf*:c equally similar to *conf*:b. The meta-walk that represents the closets relationship to p_1 in Figure 3.3(a) is p_2 = [*conf*, *paper*, *dom*, *kw*, *dom*, *paper*, *conf*]. However, using meta-walk p_2, R-PathSim finds *conf*:a more similar to *conf*:b than *conf*:c in Figure 3.3(a).

We observe that meta-walk p_2 does not exactly represent the same information as meta-walk p_1 because p_2 contains additional entity labels, i.e., *paper*. Hence, a walk in p_1 may correspond to several walks in p_2. This causes R-PathSim to produce different rankings for the same query over Figures 3.3(a) and 3.3(b). To return the same answers over Figure 3.3(a) and 3.3(b), one may look for a structure in Figure 3.3(a) that represents exactly the same information that p_1 expresses in Figure 3.3(b). Every walk of p_1 represents the fact that a conference belongs to a certain domain and does not contain any information about the number of papers published in the conference. Hence, we extend the definition of meta-walk to ignore the number of occurrences of certain entities in a walk. For example, we define meta-walk p_3 in Figure 3.3(a) whose walks express the fact that entities of labels *conf* and *dom* are connected through *paper* entities without any regard to the number of papers between them. This meta-walk treats all walks between each pair of entities of label *conf* and *dom* through *paper* entities as a single walk. We show p_3 as [*conf*, $\overline{\textit{paper}}$, *dom*, *kw*, *dom*, $\overline{\textit{paper}}$, *conf*]. We call $\overline{\textit{paper}}$ a *skip-label* in p_3. Using a skip-label in the meta-walk indicates that the user is interested in whether a connection between entities in the meta-walk exists. Meta-walk p_3 has the same number of walks in Figure 3.3(a) as p_1 has in Figure 3.3(b). Hence, R-PathSim will deliver the same ranking for query *conf*:b over Figure 3.3(a) using meta-walk p_3 and Figure 3.3(b) using meta-walk p_1.

Furthermore, we may have to use a more complex meta-walk in a database to express the same information as a simpler meta-walk in the entity-rearranging transformation of the database. For instance, a user may like to find similar conferences based on the meta-walk $p2$ in Figure 3.3(a). However, she must use a more complex meta-walk [*conf*, *paper*, *conf*, *dom*, *kw*, *dom*, *conf*, *paper*, *conf*] to obtain the same results in Figure 3.3(b). Instead of stopping at the candidate answer label, this meta-walk goes beyond and traverses back to the candidate answer label. We call this type of meta-walks *meta-walks with repeated entities*.

Hence, a relationship between the same set of entities may be represented by normal meta-walks, as defined in Section 3.5 in a database, but using meta-walks with a skip-label or repeated entities on its entity-rearranging transformations. To be robust over entity-rearranging transformations, R-PathSim should consider meta-walks with skip-label and repeated entities in addition to the meta-walks defined in Section 3.5. Thus, we extend R-PathSim to consider these types of meta-walks. Nevertheless, if we allow a skip-label for

every label in each meta-walk, R-PathSim has to pre-compute the commuting matrices for a large number of meta-walks. Hence, we would like to identify a minimal set of meta-walks with skip-labels that capture all relationships in a database. Then, R-PathSim can use them to deliver the same results over the database and its entity rearrangements. First, according to Definition 3.7, if neither meta-walk p nor any of its subwalks is a meta-walk for any FD over a database D, there is a meta-walk r over an entity-rearranging transformation of D, $T(D)$, such that r and p have equal number of walks in D and $T(D)$, respectively. Thus, R-PathSim score of entities over these meta-walks are equal over D and $T(D)$. Hence, we do not assign any skip-label to meta-walks that are not part of any FD in a database. Second, consider meta-walk $s = [l_1, \ldots, l_k]$ in database D where $l_1 \rightarrow l_2$, $l_2 \rightarrow l_3$, ..., $l_{k-1} \rightarrow l_k$ hold in D. If there is a walk from entity e of label l_1 and an entity f of label l_k in D, there will be exactly one walk between e and f in D because of the FDs over s. Hence, every meta-walk created by setting some of the labels in s to skip-label have the same number of walks in D as s has. Intuitively, setting some labels in s to skip-labels will not express any new useful relationship between entities in s. Moreover, because R-PathSim returns the same similarity score for two entities using s and its modifications, we will consider only s.

Next, we prove that the aforementioned extension of R-PathSim is design independent over entity-rearranging transformations. Let $\mathbf{L}$ be a set of databases whose set of labels is L. We define a binary relation $\prec$ between labels l_1 and l_2 in L where $l_1 \prec l_2$ if and only if there is a meta-walk p such that $l_1 \xrightarrow{p} l_2 \in F_{\mathbf{L}}$. The set of labels $S \subseteq L$ is a **chain** if and only if $\prec$ is a total order over S. S is a *maximal chain* if and only if there is no $R \subseteq L$ such that $S \subsetneq R$ and R is a chain. For instance, because we have *paper* $\rightarrow$ *conf*, *paper* $\rightarrow$ *dom* and *conf* $\xrightarrow{[conf,paper,dom]}$ *dom* in Figure 3.3(a), {*paper*,*conf*,*dom*} is a maximal chain. By the abuse of notation, we let F_S denote the set of FDs in $\mathbf{L}$ whose labels are in S. In this work, we focus on a set of databases whose sets of maximal chains are mutually exclusive. MAS databases whose fragments are shown in Figure 3.3(a) and 3.3(b) are examples of such databases.

Theorem 3.4. *Given an entity-rearranging transformation $T : \boldsymbol{L} \rightarrow \boldsymbol{K}$, for all $D \in \boldsymbol{L}$, T bijectively maps each meta-walk p in D to a meta-walk r in $T(D)$ such that for all entities e and f in D, $|p(e, f, D)| = |r(T(e), T(f), T(D))|$.*

Proof. Let us define *internal* labels of a meta-walk $p = [l_1, ..., l_n]$ as labels $l_2, ...l_{n-1}$. Given a meta-walk p in D, one can write p as a concatenation of meta-walks $s_1...s_m$ where m is the smallest value s.t. each s_i, $i = 1...m$, satisfies exactly one of the following conditions: (1) s_i is not a meta-walk of any FD in $\mathbf{L}$, (2) all internal labels of s_i are skip-labels, or (3) $s_i = [l'_1, ..., l'_k]$ where $l'_1 \rightarrow l'_2, ..., l'_{k-1} \rightarrow l'_k \in F_L$ (or $l'_k \rightarrow l'_{k-1}, ..., l'_2 \rightarrow l'_1 \in F_L$). Clearly,

no s_i satisfies both conditions (1) and (3), or both (2) and (3). Because we set the labels of meta-walks that appear in an FD to skip-labels, no s_i satisfies both conditions (1) and (2). Suppose there are more than one concatenations for p that satisfies the aforementioned conditions. Without losing generality, assume $p = s_1 s_2 = s'_1 s'_2$ where $s_1 \neq s'_1$, $s_1 = [l_1, ..., l_k]$, $s_2 = [l_k, ..., l_n]$, and $l_{k-1}, ..., l_n$ are in s'_2. If s_1 and s_2, or s'_1 and s'_2, satisfy condition (1), then p satisfies condition (1). Thus, $m = 2$ is not the smallest number that satisfies the aforementioned concatenation for p. Hence, s_1 and s_2 together do not satisfy condition (1). If s_1 and s_2 satisfy condition (2), then the internal labels of s'_1 or s'_2 will contain both skip-labels and l_k, and s'_1 or s'_2 will not satisfy any of the aforementioned conditions. Hence, s_1 and s_2 together do not satisfy condition (2). Similarly, if s'_1 and s'_2 satisfy condition (2), s_1 or s_2 will not satisfy any of the conditions. Thus, s'_1 and s'_2 together do not satisfy condition (2). Let s_1, s_2, s'_1 and s'_2 satisfy condition (3). Thus, either $l_1 \rightarrow l_2, ..., l_{k-1} \rightarrow l_k$ or $l_k \rightarrow l_{k-1}, ..., l_2 \rightarrow l_1$ hold in F_L. Let $l_1 \rightarrow l_2, ..., l_{k-1} \rightarrow l_k$ hold in F_L. Because we focus on only the databases whose sets of maximal chains are mutually exclusive, $l_k \rightarrow l_{k-1}$ does not hold in F_L Hence, for s'_2 to satisfy condition (3), $l_{k-1} \rightarrow l_k, ..., l_{n-1} \rightarrow l_n$ must hold in F_L. That is, p itself satisfies condition (3). Similarly, we prove that if $l_k \rightarrow l_{k-1}, ..., l_2 \rightarrow l_1$ hold in F_L, p satisfies condition (3). Thus, $m = 2$ is not the smallest number that satisfies the aforementioned concatenation for p. Hence, there is only one concatenation of p that satisfies the aforementioned conditions.

Let $s_i = [l'_1, ..., l'_k]$. Suppose s_i satisfies condition (1). Using Definition 3.7, there is a bijection between walks of s_i and walks of $T(s_i)$ s.t. $|s_i(e, f, D)| = |T(s_i)(T(e), T(f), T(D))|$. Suppose s_i satisfies condition (2). Because we set the labels of only meta-walks used in an FD to skip-labels, we have $l'_1 \xrightarrow{s_i} l'_k$ (or $l'_1 \xleftarrow{s_i} l'_k$). By Definition 3.7, T bijectively maps $l'_1 \xrightarrow{s_i} l'_k$ to some $l'_1 \xrightarrow{r_i} l'_k$ in $F_{\mathbf{K}}$ s.t. $s_i(e, f, D) = \emptyset$ if and only if $r_i(T(e), T(f), T(D)) = \emptyset$. If $s_i(e, f, D) \neq \emptyset$, then $|s_i(e, f, D)| = 1$ and $r_i(T(e), T(f), T(D)) \neq \emptyset$. Let r_i^* be obtained by changing all internal labels of r_i to skip-labels. If $r_i = [l'_1, l'_2]$, then $r_i^* = r_i$. If $r_i(T(e), T(f), T(D)) \neq \emptyset$, then $|r_i^*(T(e), T(f), T(D))| = 1$. Hence, $|s_i(e, f, D)| = |r_i^*(T(e), T(f), T(D))|$. Suppose s_i satisfies condition (3). If $s_i(e, f, D) \neq \emptyset$, then $|s_i(e, f, D)| = 1$. Similar to the case where s_i satisfies condition (2), we prove that T bijectively maps s_i to r_i^* s.t. $|s_i(e, f, D)| = |r_i^*(T(e), T(f), T(D))|$.

The end node of each walk in s_i is the start node of a walk s_{i+1}. Hence, by Definition 3.7, the end node of each walk of $T(s_i)$ is the start node of a walk of $T(s_{i+1})$. Let r be the meta-walk created by concatenating $T(s_i)$'s. Each walk of r is a concatenation of walks of $T(s_i)$ in $T(D)$.

Similar to Section 3.5.2, one may compute a single similarity between a pair of entities

by computing the average of R-PathSim scores over all meta-walks between the pair of entities. Theorem 3.4 guarantees this aggregated similarity score for each pair of entities and their mapping over entity-rearranging transformations are equal. We use the same methods discussed in Section 3.5.2 to pre-compute and compute the score of meta-walks with skip-labels and repeated entities. Our results here introduce a new method to make a similarity search algorithm design independent. Because the same relationship may be expressed in several forms over different representations of the same data, the algorithm should consider more varieties of relationships.

3.7 EMPIRICAL EVALUATION

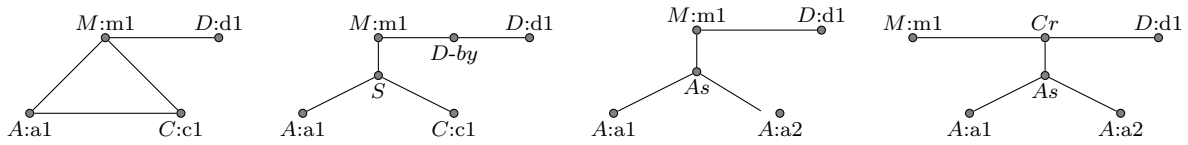
3.7.1 Experiment Settings

We use 5 datasets in our experiments. We use a subset of DBLP[4] data with 1,227,602 nodes and 2,692,679 edges, which contains information about publications in computer science. We add information about the area for each conference in DBLP from MAS[6]. Figure 3.5(a) shows fragments of DBLP. We also use a subset of Microsoft Academic Search data with 44,068 nodes and 44,220 edges whose fragments are shown Figure 3.3(a). We use Arxiv High Energy Physics paper citation graph from SNAP[5] with 34,536 nodes and 42,158 edges whose fragments are shown in Figure 3.2(b). We use a subset of IMDb data with 2,409,252 nodes and 7,525,281 edges whose fragments are shown in Figure 3.4(a). We also use WSU course database[7] with 1,124 nodes and 1,959 edges, which contains information about courses, instructors, and course offerings. Figure 3.6(a) shows fragments of this dataset. We implement our and other algorithms using MATLAB 8.5 on a Linux server with 64GB memory and two quad core processors.

3.7.2 Structural Robustness

In this experiment, we would like to investigate the property of design independence of the studied algorithms. Hence, we would like to measure the degree of structural robustness of these algorithms. 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 3, 5 and 10 answers.

[7] *www.cs.washington.edu/research/xmldatasets*



(a) IMDb (IMDB) (b) Freebase (FB) (c) Movielicious (MVL) (d) ASM

Figure 3.4: Fragments of movies databases where A, M, C, D, S, As, Cr and *D-by* denotes *actor*, *movie*, *character*, *director*, *starring*, *actors*, *credit* and *directed-by*.

Table 3.1: Average ranking differences for all transformations.

		IM2MV	IM2AS	IM2FB	DB2SI	WS2AL
Top 3	RWR	0.473	0.505	0.170	0.482	0.300
	SimRank	0.411	0.458	0.333	0.481	0.440
	PathSim	-	-	-	0.641	0.320
Top 5	RWR	0.444	0.459	0.158	0.447	0.259
	SimRank	0.365	0.392	0.337	0.455	0.387
	PathSim	-	-	-	0.608	0.310
Top 10	RWR	0.404	0.415	0.155	0.412	0.253
	SimRank	0.343	0.348	0.322	0.410	0.341
	PathSim	-	-	-	0.590	0.247

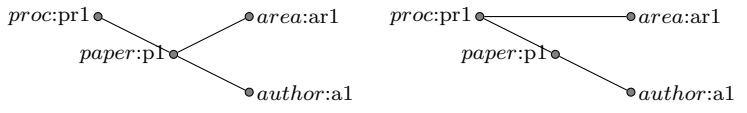
Relationship Reorganization

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 and the damping factor of SimRank to 0.8. We reorganize IMDb database to the structures of Freebase (FB), Movielicious (MVL) and a structure used in a class assignment from Evergreen Valley College[8] (ASM) whose fragments are shown in Figure 3.4(b), Figure 3.4(c) and Figure 3.4(d), respectively. 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. The sizes of data graphs used in IM2MV and IM2AS are 47,835 nodes and 130,916 edges for IMDb, 69,121 nodes and 152,202 edges for MVL, and 90,407 nodes and 173,488 edges for ASM. The sizes of data graphs used in IM2FB is 50,052 nodes and 139,998 edges for IMDb and 96,718 nodes and 139,998 edges for FB.

For query workload, we randomly sample 50 movies in IMDb database based on their degrees.

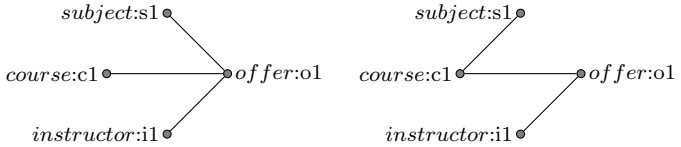
Table 3.1 shows the average ranking differences for top 3, 5, and 10 answers returned by RWR and SimRank over IM2MV, IM2AS, and IM2FB transformations. Because R-PathSim delivers the same rankings over these transformations, we have omitted the results for R-

[8] *www.evc-cit.info/cit041x/assignment/_css.html*



(a) DBLP (b) SIGMOD Rec. structure

Figure 3.5: Fragments of bibliographic databases.



(a) WSU (b) Alchemy UW-CSE

Figure 3.6: Fragments of course databases.

PathSim. Because each entity label and its consecutive entity labels in every meta-walk over FB, MVL, and ASM data are different, all walks used in the computation of PathSim are informative, thus, PathSim is robust over these transformations according to Theorem 3.2. Hence, we omit the results of PathSim from the table. According to Table 3.1, the rankings produced by RWR and SimRank varies considerably over relationship-reorganizing transformations.

As we have shown in Section 3.5, PathSim is not robust under certain relationship reorganizing transformations. We use the SNAP dataset and re-organize it to the structure of DBLP-citation as depicted in Figure 3.2(a). For query workload, we randomly sample 50 papers from SNAP based on their degrees. We use [*paper*, *paper*, *paper*] meta-walk on SNAP and [*paper*, *citation*, *paper*, *citation*, *paper*] on DBLP-citation for PathSim and R-PathSim. The average ranking differences for top 3, 5 and 10 answers of PathSim are 0.564, 0.522 and 0.495, respectively. Hence, the output of PathSim varies significantly over some relationship-reorganizing transformations.

Entity Rearrangement

We use DBLP and WSU course databases to evaluate the robustness of RWR, SimRank, PathSim, and R-PathSim over entity-rearranging transformations. Because SimRank and RWR take too long to finish on full DBLP dataset, we perform the following experiments using a subset of DBLP. with 24,396 nodes and 98,731 edges. The FDs in DBLP database are *paper* $\rightarrow$ *proc*, *paper*$\rightarrow$ *area*, and *proc* $\xrightarrow{[proc,paper,area]}$ *area*. We transform this database to a database that follows the structure of SIGMOD Record from *sigmod.org/publications*, where the information about each collection of papers is directly connected to the node

that represents the collection. Figure 3.5(b) shows fragments of this database. The FDs in this database are *paper* → *proc*, *proc*→*area*, and $paper \xrightarrow{[paper,proc,area]} area$. We call this transformation DB2SI. We randomly sample 100 proceedings based on their degrees in DBLP dataset as our query workload. The FDs in WSU database are *offer*→*course*, *offer*→*subject*, and $course \xrightarrow{[course,offer,subject]} subject$. Figure 3.6 depicts the transformation of our WSU Course dataset to the structure of the Alchemy UW-CSE[9] database. We call this transformation W2AL. The FDs in Alchemy UW-CSE database are *offer*→*course*, *offer* $\xrightarrow{[offer,course,subject]}$ *subject*, and *course*→ *subject*. We randomly sample 100 courses from WSU based on their degrees as our query workload. Table 3.1 shows the average ranking differences for top 3, 5 and 10 answers from RWR, SimRank and PathSim under DB2SI and WS2AL. We use meta-walks [*proc*, *area*, *proc*] and [*proc*, *paper*, *area*, *paper*, *proc*] over DBLP and SIGMOD Record, respectively, for PathSim and R-PathSim. We use meta-walks [*course*, *offer*, *subject*, *offer*, *course*] and [*course*, *subject*, *course*] over WSU and Alchemy UW-CSE, respectively. Because R-PathSim returns the same answers over both transformations, we do not report its results. According to Table 3.1, the outputs of all algorithms, except R-PathSim, are significantly different over entity-rearranging transformations.

3.7.3 Efficiency

We evaluate the efficiency of R-PathSim and PathSim over full IMDb and DBLP data. 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. As we have explained in Section 3.7.2, DBLP dataset satisfy some FDs. Thus, we use it to measure the influence of using meta-walks with skip-labels in the running time of R-PathSim. To explore the impact of both detecting informative walks and using meta-walks with skip-labels on the efficiency of R-PathSim, we add a node without value, called *authors*, that groups authors of the same paper in DBLP dataset. This modification introduces non-informative walks to the database. We call the resulting database DBLP+, which contains 1,905,092 nodes and 3,370,169 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 [4]. MVL, DBLP, and DBLP+ have 16, 16, and 22 meta-walks with sizes less or equal to 3, respectively. It takes 49.5, 154.6, and 156.1 seconds for R-PathSim to precompute and store the commuting matrices of these meta-walk for MVL, DBLP and DBLP+, respectively, which are reasonable for a pre-processing step. We have executed PathSim over the same datasets and get almost equal

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

Table 3.2: Average query time in seconds.

	Size of meta-walks	MVL	DBLP	DBLP+
R-PathSim	3	0.058	0.025	0.020
	5	0.036	0.035	0.046
	7	0.068	0.343	0.233
PathSim	3	0.058	0.024	0.018
	5	0.036	0.030	0.046
	7	0.068	0.347	0.227

running times as the ones of R-PathSim.

We randomly select 100 movies from MVL and 100 proceedings based on their degrees from DBLP and DBLP+ and use them as our query workloads. Table 3.2 shows the average query processing time of R-PathSim and PathSim per query per meta-walk for meta-walks up to size 7 given that commuting matrices up to size 3 are materialized. The result indicates that the additional steps in R-PathSim do not significantly increase its running time compared to that of PathSim.

3.7.4 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. For example, *Data Mining* and *Databases* domains are strongly related, but *Databases* and *Computer Vision* are not. We use Normalized DCG (nDCG) to compare the effectiveness of R-PathSim and PathSim because it supports multiple levels of relevance for returned answers [4, 60]. 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.

In the first experiment, we use meta-walk [*conf*, *paper*, *citation*, *paper*, *citation*, *paper*, *conf*] to find similar conferences based on their papers' citations. Since R-PathSim considers only informative walks of this meta-walk, it will return different results than PathSim. The average nDCG@5 (nDCG@10) for R-PathSim and PathSim are 0.264 (0.315) and 0.261 (0.313), respectively. Although the value of nDCG for R-PathSim is higher than PathSim, the difference is not statistically significant according to the paired t-test at significant level of 0.05. In the second experiment, we evaluate the effectiveness of using meta-walks with skip-labels. We compute the similarities of conferences based on the keywords in their

domains. PathSim uses meta-walk [*conf*, *paper*, *domain*, *keyword*, *domain*, *paper*, *conf*] and R-PathSim uses meta-walk [*conf*, $\overline{paper}$, *domain*, *keyword*, *domain*, $\overline{paper}$, *conf*]. The average nDCG@5 (nDCG@10) for R-PathSim and PathSim are 1.0 (1.0) and 0.640 (0.616), respectively. R-PathSim significantly outperforms PathSim. Entities of type *paper* should not play a role in computing the similarity of conferences based on the keywords of their domains. Nevertheless, PathSim considers papers in determining these similarities. Hence, it deems conferences with more papers more similar, while they may not have that many common keywords. R-PathSim avoids this problem by treating *paper* as a skip-label. For example, the top 5 answers of R-PathSim for query *SIGKDD* are *ICDM*, *IDEAL*, *PAKDD*, *PJW* and *PKDD*. However, the top 5 answers of PathSim for the same query are *ICOMP*, *IC-AI*, *ICAIL*, *ICALP* and *ICANN*.

CHAPTER 4: STRUCTURAL DESIGN INDEPENDENCE THROUGH DATABASE CONSTRAINTS

Our work discussed in Chapter 3 is indeed limited to particular structural variations of databases. Hence, the proposed algorithm, SR-PathSim, may not be robust against some other possible variations. Nevertheless, the underlying idea of our approach in Chapter 3 is actually based on an increase in the expressiveness of a language that describes relationships between entities over a data graph. Following such approach, in this chapter, we would like to generalize our solution to address the problem of design independence. In particular, we leverage the properties of a database due to its imposed constraints to predict possible structural variations. Then we discuss and prove expressivity of the relationship expressing language over a data graph that is necessary for the property of design independence.

4.1 BACKGROUND

Data variety is ubiquitous in data management as different data sources may represent the same information in different forms [29, 61–63]. In particular, databases from the same data model may represent essentially the same information in different structures or schemas [17, 18, 29, 41, 42, 58, 64]. A classic example of these structural variations is schema normalization and de-normalization in relational databases [17, 65]. From the early days of database research, an important goal has been to devise query interfaces that can hide schematic variations from users so that they do not have to reformulate their queries over schematic variations to get the same results [42, 47, 66].

Recently, researchers have noticed that the results of unsupervised and supervised machine learning algorithms over structured data also depend on the schema and structure of the underlying database [20, 21, 67, 68]. In particular, machine learning algorithms leverage topological and structural characteristics of the graph datasets, e.g., number of edges between two nodes in a graph, whose values may change across different schemas and structures for the same dataset. Thus, these algorithms may deliver accurate results over certain schemas and structures and be substantially less accurate over other structures used to represent the same information.

As an example, consider two bibliographic datasets from DBLP[1] and SIGMOD Record[2] whose fragments are shown in Figure 4.1. Intuitively, these databases represent essentially the same set of relationships between the same set of *paper*, *conference*, and *research area*

[1] *dblp.uni-trier.de*

[2] *sigmod.org/publications*

entities. But, each dataset has its own way of organizing these entities and their relationships. For example, DBLP connects each paper to its research areas and conferences. Given that all papers in a conference share the same set of research areas, one can also choose the structure in Figure 4.1(b) for this information and connects research areas directly to conferences instead of papers. Assume that a user wants to find the most similar research area to *Data Mining* according to their conferences and publications. SimRank [2] is a well-known similarity search algorithm over graphs that finds two node, i.e. two entities, to be (structurally) similar if they both are similar to another node in the graph. It implements this idea using random walks over the graph. It finds *Data Mining* to be more similar to *Info Retrieval* than to *Databases* in Figure 4.1(a). In Figure 4.1(b), however, it declares *Data Mining* more similar to *Databases* than to *Info Retrieval*.

Hence, to use a data analytics algorithm, the user has to restructure the database to find the structure over which the algorithm delivers accurate results. Since there is *not* any clear guideline on how to find such a desirable structure for the algorithm, one has to do this through trial and error, which takes a great deal of time and effort from experts who are familiar with the details of the algorithms. This issue becomes increasingly important as a user often has to analyze datasets from multiple sources where each dataset has its own style of structuring information. Moreover, normal users who *cannot* write programs to change the structure of the data are *not* able to use these algorithms effectively.

Ideally, one would like to design an algorithm that returns essentially the same accurate results over all possible schematic variations of a dataset. Researchers have proposed algorithms that are robust over certain types of schematic variations of the underlying data [20, 21, 67, 68]. These methods, unfortunately, have two major shortcomings. First, they are robust only over a subset of all possible and popular schematic variations. Because they leverage the properties special to the variation over which they are robust, it is not clear how to generalize these algorithms to be robust against other schematic variations. For example, Picado et al. propose a learning algorithm over relational databases that is robust over schema normalization [21], which uses the foreign-key-primary-key links created as the result of normalization. As such links may not be produced in other schematic variations, it is not clear how one generalizes this algorithm to other schematic shifts.

Second, current schematically robust systems generally either propose new algorithms [20], or make significant and complex modifications to the current algorithms [21]. However, current well-known data analytics algorithms already been widely adapted to solve various problems. Hence, to make these algorithms schematically robust, it is easier for the organization to keep using the current algorithms or make straightforward and simple modifications to them. Moreover, they have been shown empirically to be effective in multiple domains.

Thus, a schematically robust version of them will be effective over more representations. Therefore, the ideal approach to achieve the vision of structurally robust data analytics should make rather simple modifications to the current algorithms and create a version that is robust in the face of all popular structural variations.

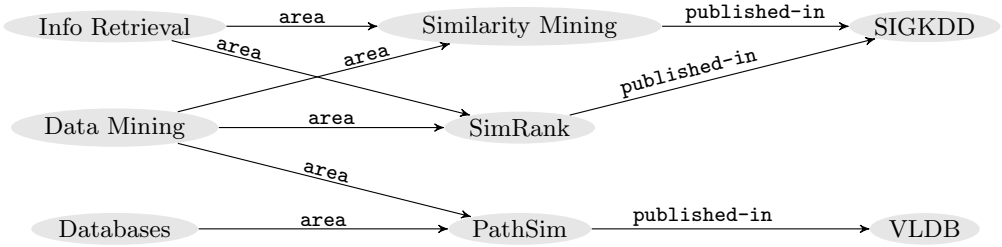
One generic method is to run an algorithm and over all possible variations of a validation subset of the database and select the representation with the most accurate answers. Nonetheless, databases have a large number of possible structural variations [17, 18]. For example, a relational table may have exponential number of normalizations. Moreover, such a validation subset is not generally available for unsupervised methods, such as similarity search. This approach also requires the underlying database be transformed to the desired representation, which may not be practical for a large and/or constantly evolving databases that are used by varieties of algorithms, where each is effective over a different representation.

We propose such an approach for the problem of similarity search over graph databases. Our first observation is to leverage a result by Hull [41], which states that there is *not* any information-preserving variation of a relational schema without any constraint beyond simple renaming of the scheme elements. In other words, constraints give rise to schematic variations. We extend this result for graph databases and predicts schematic variations for a database according to the constraint in the database. Then, we provide an algorithm that computes information-preserving variations of a given pattern in the graph given a set of constraints. We show that these variations may go beyond the simple paths, which are used to represent relationships in similarity search, and require a more expressive language to be expressed over graphs. Finally, we extend an available and well-known similarity search algorithm called *PathSim* to compute the similarities using the extended set of patterns.

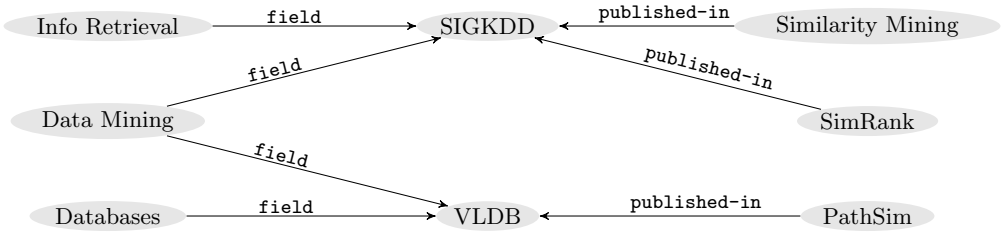
We must note that, while our work discusses structural variations, we do not assume that such variations are given to users. That is, we do not provide a procedure to transform a user query according to a given structural variation. Instead, we provide an ability for users to express their intentions across various structure of databases, and guarantee the robustness (and effectiveness) of the algorithm.

4.2 RELATED WORKS

One generic approach to achieve schematic robustness is to define a *universal schema* to which all possible representations of a database can be transformed and use and/or develop algorithms that are effective over this representation. Nevertheless, the experience gained from the idea of universal relation, indicates that such representation does not often exist [17, 42]. One also has to transform their database to the universal representation, which may



(a) Fragments of DBLP



(b) Fragments of SIGMOD Record

Figure 4.1: Example of two bibliography databases.

be quite complex considering the intricacies associated with defining such a representation and not practical for a large database.

Researchers have also analyzed the stability of random walk algorithms in graphs against relatively small perturbations in the data [48–50]. We also seek to instill robustness in graph mining algorithms, but we are targeting robustness in a new dimension: robustness in the face of variations in the representation of data. Researchers have provided systems that help users with transforming and wrangling their data [51–54]. We also address the problem of data preparation but using a difference approach: *eliminating the need to wrangle the data*.

Schema mapping has been an active research area for the last three decades [17]. In particular, researchers have defined schema mappings over graph databases as constraints in some graph query language in the context of data exchange [32]. As opposed to the transformations in our work, the original and transformed databases in those settings may not represent the same information. We also focus on evaluating the robustness of similarity search algorithms rather than traditional questions in schema mapping and data exchange, such as computing the transformed database instances.

Researchers have previously proposed a keyword query interfaces over XML dataset that is provably returns the same answers across databases with equivalent information content but using different data structures [19]. Over graph database, our work in [68] have proposed a steps towards the same goal over graph databases. They presented an extension to a similarity search algorithm so that the algorithm is schematically robust across databases that may present edge label using a relationship node, e.g., non-entity nodes or nodes without

values. In the work of this chapter, we investigating more on this concept and also focus on a wider variety of representation shifts that preserves information content.

4.3 GRAPH DATABASES AND CONSTRAINTS

We fix a countably infinite set of node ids denoted by $\mathcal{V}$. Let $\mathcal{L}$ be a finite set of labels. A *database* D over $\mathcal{L}$ is a directed graph (V, E) in which V is a finite subset of $\mathcal{V}$ and $E \subseteq V \times L \times V$. We denote an edge from node u to node v whose label is a as (u, a, v). We say that $(u, a, v) \in D$ whenever $(u, a, v) \in E$. Similarly, we say that $v \in D$ whenever $v \in V$.

Database constraints restrict the instances of a schema. Constraints are usually expressed as logical formulas over the labels in databases [69–71]. Two well-known and widely used types of constraints are *tuple-generating* and *equality-generating* dependencies [69, 70]. 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 4.1. *Database shown in Figure 4.1(a) contains a constraint* $(x_1, \texttt{area}, x_3) \wedge (x_3, \texttt{published-in}, x_4) \wedge (x_2, \texttt{published-in}, x_4) \rightarrow (x_1, \texttt{area}, x_2)$.

A commonly studied query language over graph databases is *conjunctive regular path queries* (conjunctive RPQ), which is used to express tgd and egd constraints over graph databases [32, 71–73]. The *RPQ* p over schema $\mathcal{L}$ is defined by the following grammar.

$$p := \epsilon \mid a\ (a \in \mathcal{L}) \mid a^{-}\ (a \in \mathcal{L}) \mid p \cdot p \mid p + p \mid p^{*} \tag{4.1}$$

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 4.1(b) is $\texttt{field} \cdot \texttt{published-in}^{-}$. The RPQ p defines a binary relation over database nodes. More precisely, the result of evaluating p on database D is a set of pairs of nodes in D such that there is a path defined by p between the two nodes. We denote the result of evaluating p over D as $[[p]]_D$. For example, given label a in the schema of D, the result of $[[a]]_D$ is a set of pairs of nodes $\{(u, v)\}$ where there is an edge with label a from u to v. Let $\bar{x} = (x_1, \ldots, x_n)$ and $\bar{y} = (y_1, \ldots, y_m)$ be tuples of distinct variables. A conjunctive RPQ is a

formula $\phi(\bar{x})$ of the form $\exists\bar{y}((z_1, p_1, z'_1) \wedge ... \wedge (z_k, p_k, z'_k))$ where p_i is an RPQ and $z_i, z'_i \in \{x_1, \ldots, x_n, y_1, \ldots, y_m\}$, $1 \leq i \leq k$ [32, 73]. We call (z_i, p_i, z'_i) an *atom* of $\phi(\bar{x})$.

A schema S is a pair $(\mathcal{L}, \Gamma_S)$ in which $\mathcal{L}$ is a (finite) set of labels and Γ_S is a finite set of constraints. By the abuse of notation, we say that a label $l \in S$ if $l \in \mathcal{L}$ and a constraint $\gamma \in S$ if $\gamma \in \Gamma_S$. Each *database* of schema $S = (\mathcal{L}, \Sigma_S)$ is a database over $\mathcal{L}$ such that all constraints in Σ_S holds. Database constraints are generally expressed as sentences in some logical language, usually a subset of the first order logic [17]. Tgds and egds are arguably the most popular and frequently used types of database constraints [17]. They generalize popular constraints, such as functional and multi-valued dependencies, and provide a reasonable trade-off between the expressivity and efficiency of constraint checking and implication. A *full tgd* does *not* have any existential variable in its conclusion. We denote the set of all databases of schema S as $\texttt{Inst}(S)$.

4.4 STRUCTURAL ROBUSTNESS AND VARIATIONS

4.4.1 Structural Robustness

Following the discussion of design independence in Chapter 3, intuitively, a structurally-robust query answering algorithm should return essentially the same (list of) answers for the same query across databases that contain the same information content. Researchers have leveraged the concept of invertible transformation to formalize the equivalence of information stored in different databases [18, 41]. A *transformation* from schema S to schema T is a (computable) function from $\texttt{Inst}(S)$ to $\texttt{Inst}(T)$, which maps each database of S to a database of T. We denote a transformation from S to T as Σ_{ST}. 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 $\Sigma_{ST}(I)$ to I, i.e., $\Sigma_{TS}(\Sigma_{ST}(I)) = I$. In other words, the composition of $\Sigma_{ST}(I)$ and $\Sigma_{TS}(I)$, shown as $\Sigma_{ST}(I) \circ \Sigma_{TS}$ is equivalent to the identity transformation id that maps each database to itself. In this case, we call Σ_{TS} the *inverse* of Σ_{TS} and denote it as Σ_{ST}^{-1}. If there is an invertible transformation from schema S to T, one can reconstruct the information in each database I from the information available in $\Sigma_{ST}(I)$. In other words, the transformed database contains sufficient information to rebuild the original one. Thus, $\Sigma_{ST}(I)$ has at least the same amount of information I. As Σ_{ST} maps each database of schema S to a database with at least the same amount of information in schema T, schema T has at least the same information capacity as S.

Transformation Σ_{ST} establishes a bijection between $\texttt{Inst}(S)$ and $\texttt{Inst}(T)$, if the domain of Σ_{ST}^{-1} is exactly $\texttt{Inst}(T)$, i.e., all databases of schema T. In this case, we call transformation

Σ_{ST} *information-preserving*, and schemas S and T are *information-equivalent*. We denote the fact that S and T are *information-equivalent* using transformation Σ_{ST} as $S \overset{\Sigma_{ST}}{\equiv} T$ or simply as $S \equiv T$ if the information about Σ_{ST} is *not* needed.

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

Definition 4.1. *Given schemas S and T such that $S \overset{\Sigma_{ST}}{\equiv} T$, an algorithm is robust under Σ_{ST} if it returns equivalent answers for every input query q over databases $I \in \texttt{Inst}(S)$ and $\Sigma_{ST}(I) \in \texttt{Inst}(T)$.*

An algorithm is robust under a set of transformations if it is robust under all members of the set. Using the definition of robustness, we have the following proposition.

Proposition 4.1. *Given schemas S and T such that $S \overset{\Sigma_{ST}}{\equiv} T$, if there is not any bijection between node ids in S and T, no algorithm will be robust under Σ_{ST}.*

Proof. Given database $I \in \texttt{Inst}((S))$, let $\Sigma_{ST}(I)$ have some node id v that is *not* present in I. In this case, every (effective) algorithm should return an empty answer for the input query v over I but a non-empty list of answer(s) over $\Sigma_{ST}(I)$. According to Definition 4.1, there will *not* be any robust algorithm under Σ_{ST}. The same argument holds for the case where I has a node id that is *not* present in $\Sigma_{ST}(I)$.

Thus, we restrict our attention in this chapter to transformations between schemas S and T that map database $I \in instS$ to $J \in \texttt{Inst}(T)$ such that there is bijection between nodes of I and J. For simplicity, if a node v is in I, we also refer to its bijection in J as v.

4.4.2 Structural Variations

Definition 4.1 does not specify the language of (information-preserving) transformations. To characterize the structural variations of a schema, one needs to express information-preserving transformations in some language. Researchers usually use *declarative (schema) mappings* to express schematic variations in graph and relational databases [32, 74]. Roughly speaking, a transformation between schemas S and T is expressed as a set of first order logical formulas $\phi_S(\bar{x}) \rightarrow \psi_T(\bar{y})$ where $\phi_S(\bar{x})$ and $\psi_T(\bar{y})$ are queries over schemas S and T, respectively. More precisely, transformation Σ_{ST} between graph schemas S and T is a finite

set of rules $\phi_S(\bar{x}) \rightarrow \psi_T(\bar{x})$ such that $\phi_S(\bar{x})$, i.e., *premise*, and $\psi_T(\bar{y})$, i.e., *conclusion*, are conjunctive RPQs over S and T, respectively [32].

Example 4.2. *Consider a transformation $\Sigma_{1a,1b}$ from schema shown in Figures 4.1(a) to the schema shown in Figures 4.1(b) that adds an edge* `field` *for every path* `area`·`published-in`*, e.g.,* $(x_1, \texttt{area} \cdot \texttt{published-in}, x_2) \rightarrow (x_1, \texttt{field}, x_2)$*, and keeps all existing edges of label* `published-in`*, e.g.,* $(x_1, \texttt{published-in}, x_2) \rightarrow (x_1, \texttt{published-in}, x_2)$*. Because of a constraint as described in Example 4.1, this transformation is information preserving. Intuitively, this particular constraint over Figure 4.1(a) implies that every paper published in the same conferences are related to the same set of research areas. Hence, one may change the structure in Figure 4.1(a) such that a research area is first connected to a conference, then a paper is connected to the conference it is published in as described in $\Sigma_{1a,1b}$.*

Transformation Σ_{ST} maps each database $I \in \texttt{Inst}(S)$ to $J = \Sigma_{ST}(I) \in instT$ if for each rule $\phi_S(\bar{x}) \rightarrow \psi_T(\bar{x})$ in Σ_{ST}, we have $\bar{u} \in [[\psi_T(\bar{x})]]_J$ if $\bar{u} \in [[\phi_S(\bar{x})]]_I$.

In the aforementioned definition, the mapped database may contain more information than the original one. For example, the mapped database may have more nodes than the original one. We further limit the language of transformations to enforce the restrictions of Proposition 4.1. First, if the conclusion of a rule in transformation Σ_{ST} contains existentially quantified variables, for at least some database I, $\Sigma_{ST}(I)$ will have some node ids that are *not* present in I. Hence, we restrict our attention to the transformations in which the conclusion of each rule does *not* have any existentially quantified variable. In this case, we can write the transformation as a set of rules in the form of $\phi_S(x, y) \rightarrow (x, exp, y)$ where (x, exp, y) is an RPQ atom over T [32]. Second, based on Proposition 4.1, exp must be either in the form of (x, l, y) or (x, l^-, y) where l is a label in schema T. Otherwise, if exp contains concatenation, i.e., $\cdot$, or Kleene star, i.e., $*$, at least one database $I \in \texttt{Inst}(S)$ will be mapped to a database $J \in \texttt{Inst}(T)$ such that there is no bijection between the nodes in I and J. For example, a rule $(x, l_S, y) \rightarrow (x, l_T.p_T, y)$, where l_S is a label in S, and l_T and p_T are labels in schema T, maps database $v_1 \xrightarrow{l_S} v_2$ in $\texttt{Inst}(S)$ to $v_1 \xrightarrow{l_T} v_3 \xrightarrow{p_T} v_2$ in $\texttt{Inst}(T)$ where there is not any bijection between their nodes. Moreover, if exp in the rule $\phi_S(x, y) \rightarrow (x, exp, y)$ contains disjunction, i.e., $+$, the transformation will not be a function. Hence, in the rest of this chapter, we assume that each rule in every transformation is in the form $\phi_S(x, y) \rightarrow (x, l, y)$ where l is a label in schema T. Our results extends for the case of $\phi_S(x, y) \rightarrow (x, l^-, y)$ by exchanging the position of x and y in the rule.

We also use the *closed world semantic* for our transformations [75]. In other words, given transformation Σ_{ST} from schema S to T, for every edge (v, l, u) in database $J \in \texttt{Inst}(T)$, which is a transformation of $I \in \texttt{Inst}(S)$, we have a rule $\phi_S(x, y) \rightarrow (x, l, y)$ in

Σ_{ST} where $(v, u) \in [[\phi_S(x, y)]]_I$. Using other semantics, e.g., the open world semantic [32], the schema mapping may not express a function and/or the transformed database may have more information, e.g., nodes, than the original one.

4.4.3 Information Preservation

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

We have the following proposition by applying the definitions of the inverse and composition of transformations.

Proposition 4.2. *Given schemas S and T such that $S \overset{\Sigma_{ST}}{\equiv} T$, for all databases $I \in \texttt{Inst}(S)$ and $J \in \texttt{Inst}(T)$, we have $I \models \Sigma_{ST}^{-1} \circ \Sigma_{ST}$ and $J \models \Sigma_{ST} \circ \Sigma_{ST}^{-1}$.*

Proposition 4.2 extends the results on the lossless decomposition of a relational schema [76] and the ones on the inverse of a relational schema mapping in [74].

Each rule in $\Sigma_{ST} \circ \Sigma_{TR}$ is created by replacing an atom (x, l, y) in the premise of a rule in Σ_{TR} by the premise of a rule in Σ_{ST} whose conclusion is (x, l, y). According to the definition of transformation, the resulting set of rules express exactly the composition of $\Sigma_{ST} \circ \Sigma_{TR}$. 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 schema S. Hence, $\Sigma_{ST} \circ \Sigma_{ST}^{-1}$ is a set of full tgds. The set of tgds introduced by Proposition 4.2 is necessary to have information-preserving transformations for schema S, but it is *not* sufficient. We show that S must satisfy an additional group of tgds to have information-preserving variations. Let σ denote the set of tgd constraints in $\Sigma_{ST}^{-1} \circ \Sigma_{ST}$. Given σ, we create another group of tgd constraints over S, denoted as σ^*, as follows. For each tgd constraint in σ whose conclusion is in the form of $\chi_1(x, y) \rightarrow (x, l^-, y)$, we replace it with constrain $\chi_1(y, x) \rightarrow (y, l, x)$. Then, for all tgds with the same atom in their conclusions, i.e., $\chi_1(x, y) \rightarrow (x, l, y), ..., \chi_m(x, y) \rightarrow (x, l, y)$ in σ, we construct the constraint $(x, l, y) \rightarrow \chi_1(x, y) \vee ... \vee \chi_m(x, y)$. For each label l' in $\mathcal{L}_S$ that does *not* appear in a conclusion of any constraint in σ, we create the constraint $(x, l', y) \rightarrow \texttt{FALSE}$, which means that there is *not* any database in $\texttt{Inst}(S)$ with any edge whose label is l'.

Proposition 4.3. *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 information-preserving with inverse Σ_{TS} if and only if, for every database $I \in \texttt{Inst}(S)$, we have $I \models \sigma \wedge \sigma^*$.*

Proof. The necessity is proved using Proposition 4.2. Let Σ_{ST} maps database $I \in \texttt{Inst}(S)$ to J, and Σ_{TS} maps J to I'. Since I and I' satisfy σ, they both contain every node and edge that is generated by applying Σ_{TS} to J. Since I and I' satisfy σ^*, every node and edge in both I and I' is created by applying at least one rule in Σ_{TS} to J. Because I and I' do *not* contain any node or edge in addition to the ones generated by applying Σ_{TS} to J, we have $I = I'$. Thus, Σ_{ST} establishes a bijection between $\texttt{Inst}(S)$ and $\texttt{Inst}(T)$.

Example 4.3. *Similar to Example 4.2, one can define a transformation $\Sigma_{1b,1a}$ from the schema of Figures 4.1(b) to the schema of Figures 4.1(a). $\Sigma_{1b,1a}$ consists of two rules: $(x_1, \textit{field}\cdot\textit{published-in}^-, x_2) \rightarrow (x_1, \textit{area}, x_2)$ and $(x_1, \textit{published-in}, x_2) \rightarrow (x_1, \textit{published-in}, x_2)$. The composition $\Sigma_{1b,1a} \circ \Sigma_{1a,1b}$ results in a constraint $(x_1, \textit{area}, x_4) \wedge (x_4, \textit{published-in}, x_3) \wedge (x_2, \textit{published-in}, x_3) \rightarrow (x_1, \textit{area}, x_2)$ which is equivalent to the constraint of the database shown in Figure 4.1(a) as described in Example 4.1.*

According to the definition of information-preserving transformations, if Σ_{ST} is information-preserving so as Σ_{ST}^{-1}. Thus, the results of Proposition 4.3 also applies to the databases of schema T. In other words, if γ denotes $\Sigma_{TS} \circ \Sigma_{ST}$, for each $J \in \texttt{Inst}(T)$, we have $J \models \gamma \wedge \gamma^*$. We should note that the composition of two transformations whose rules are written in conjunctive RPQ language *cannot* always be expressed as a tgd in conjunctive RPQ [32]. Hence, we focus our attention in this chapter to transformation Σ_{ST} such that $\Sigma_{ST} \circ \Sigma_{ST}^{-1}$ is a tgd in conjunctive RPQ. We refer an information-preserving transformation simply as a transformation.

4.5 ROBUST SIMILARITY SEARCH

4.5.1 Robustness of Current Methods

To the best of our knowledge, frequently used structural-based similarity search algorithms are based on random walk, e.g., RWR [5], pairwise random walk, e.g., SimRank [2] and P-Rank [3], or path-constrained framework, e.g., PathSim, [4, 44]. There are other similarity search algorithms that extend the aforementioned algorithms such as common neighbors, $Katz_\beta$ measure, and commute time [77].

Similarity score computed by algorithms that use random walks and pairwise random walks are largely influenced by the topology of the graph. Because some information preserving transformations may modify the topological structure of a database, a structural-based similarity search algorithms such as RWR and SimRank are not robust under these variations as shown in our empirical studies in Section 4.9.

Two entities may be related via multiple relationship patterns, e.g., paths, in a database where each pattern may represent a different type of relationship. The degree of similarity between two entities may largely depend on the type of relationship between them. For instance, consider again Figure 4.1. A user may be interested in finding similar papers based on the conferences in which they are published rather than their common research areas. As another example, consider a database with researchers, conferences in which they publish, and their affiliations. Some users may want to find similar researchers from the point of view of their affiliations, but other users may like to find similar ones based on their shared conferences. Thus, one often has to consider the type of relationship between two entities to define an effective similarity metric with a clear semantic. Path-constrained similarity search algorithms, such as PathSim, follow this approach [4, 44]. They allow users to supply a path template, called meta-path, that specifies the type of relationship between entities in their queries. A meta-path in Figure 4.1 is $m_1 : \texttt{published-in} \cdot \texttt{published-in}^{-}$, which reflects the relationship between two papers through the conference in which they are both published. Each instance of a meta-path in database D is a path in D whose sequence of edges labels match the sequence of labels in the meta-path. For example, using the notion of a *walk* as defined in Chapter 3, [*Similarity Mining*, `published-in`, *SIGKDD*, `published-in`$^{-}$, *SimRank*] is an instance of m_1 in Figure 4.1(b). The PathSim similarity score between two entities u and v in a database D given a meta-path p is

$$\texttt{sim}_p(u, v, D) = \frac{2 \times |u \rightsquigarrow_p v|}{|u \rightsquigarrow_p u| + |v \rightsquigarrow_p v|} \tag{4.2}$$

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

Example 4.4. *Consider two representations of bibliographic data in Figure 4.1. Suppose a user wants to find similar research areas to* Data Mining *based on their shared conferences. In Figure 4.1(a), the user uses meta-path* $p_1 : \texttt{area} \cdot \texttt{published-in} \cdot \texttt{published-in}^{-} \cdot \texttt{area}^{-}$ *to represent the relationship and compute similarity scores between research areas. PathSim then finds* Data Mining *more similar to* Info Retrieval *than to* Databases. *However, in Figure 4.1(b), the same user may use meta-path* $p_2 : \texttt{field} \cdot \texttt{field}^{-}$ *to compute similarity*

scores between research areas. This meta-path finds that both Info Retrieval *and* Databases *are equally similar to* Data Mining.

4.5.2 Achieving Robustness

Example 4.4 illustrates that there may *not* be any meta-path over some representations of a dataset to express a desired relationship. If a user wants to find the similarity of research areas based on their shared conferences, she can use p_2 over the representation in Figure 4.1(b), but she cannot find any meta-path in Figure 4.1(a) that expresses such a relationship. A candidate could be p_1, but it has additional information of the set of papers published in the conferences. On the other hand, the user may like to measure the similarities of research areas based on their shared conferences and the papers published in those conferences; therefore, she uses meta-path p_1 in Figure 4.1(a). Nevertheless, there is not any meta-path in Figure 4.1(b) that exactly expresses that relationship. The user should use the expression that also includes information about publications, and the represented relationship should be based on shared conferences.

One may solve this problem by using a language that is more expressive than the set of meta-paths to express relationships between entities in a database.

Example 4.5. *Following Example 4.4, one can create an equivalent relationship to the one expressed by p_2 in Figure 4.1(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 4.5 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 4.1(a) to the one conveyed by p_2 over Figure 4.1(b). Hence, one has to define and add an operation that implements the aforementioned skipping behavior to the language that describes relationships between entities.

On the other hand, the user may find a relationship expression p_1 and the results returned over Figure 4.1(a) to be more desirable. That is, similarity of research areas are based on shared conferences and the publications in those conferences.

Example 4.6. *Following Example 4.4, one can modify p_2 to be able to visit the publications of conference while visiting a conference. This way, we will get a relationship between two research areas that takes into consideration both the conferences and publications shared*

between the them in Figure 4.1(b). The resulting pattern in Figure 4.1(b) expresses an equivalent relationship to what p_1 represents over Figure 4.1(a).

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

We present a new relationship language that is expressive enough to represent equivalent relationships across various representations of the same dataset. We also show that using this relationship language, there is a similarity algorithm that returns equal similarity scores between every pair of corresponding entities over different representations of the same information. More precisely, an algorithm that computes a similarity score using Equation 4.2 where p is written in our proposed language is structurally robust.

To implement the solution presented in Example 4.6, one may use the idea of nested operation in the nested regular expression (NRE) language [32]. Let $[p]$ denote a nested path of p where a path $(u, [p], u)$ exists if and only if there exists a node v such that a path (u, p, v) exists. To achieve same results as p_1 over Figure 4.1(a), the user should use the pattern $p_4 : \texttt{field} \cdot [\texttt{pubslihed-in}^-] \cdot [\texttt{pubslihed-in}^-] \cdot \texttt{field}^-$ to compute similarity score between research areas. That is, similar research areas are based on shared conferences, and the strength of this relation is based on the number of publications published in that conferences.

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

$$p := \epsilon \mid a \ (a \in S) \mid p^- \mid p^* \mid p \cdot p \mid p + p \mid [p] \mid \lceil\lceil p \rfloor\rfloor \tag{4.3}$$

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

Since Equation 4.2 used the number of instances of a specified relationship pattern when calculating the similarity score, we define an *instance* of an RRE as follows. An instance of some RRE in a graph database D is a ternary relation (u, v, s) representing a graph traversal over D from node u to node v whose actual traversal are recorded in a sequence s. Each entry in the recorded sequence s is either a node id, an edge label or a string of pattern. Equivalence between two RRE instances is defined naturally by entry-wise comparison.

Given a sequence $s = \langle s_1, ..., s_m \rangle$ and $t = \langle t_1, ..., t_n \rangle$ of m and n entries, respectively, let $s \bullet t = \langle s_1, ..., s_m, t_2, ..., t_n \rangle$ which is defined only if $s_m = t_1$; and let $\bar{s} = \langle \grave{s_m}, ..., \grave{s_1} \rangle$ 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

$$\mathcal{I}_D(\epsilon) = \{(u, u, \langle u\rangle) \mid u \text{ is a node in } D\} \tag{4.4}$$

$$\mathcal{I}_D(a) = \{(u, v, \langle u, a, v\rangle) \mid (u, a, v) \in D\} \tag{4.5}$$

$$\mathcal{I}_D(p^-) = \{(v, u, \bar{s}) \mid (u, v, s) \in \mathcal{I}_D(p)\} \tag{4.6}$$

$$\mathcal{I}_D(p_1 \cdot p_2) = \{(u, v, s_1 \bullet s_2) \mid \forall w, (u, w, s_1) \in \mathcal{I}_D(p_1) \text{ and } (w, v, s_2) \in \mathcal{I}_D(p_2)\} \tag{4.7}$$

$$\mathcal{I}_D(p_1 + p_2) = \{(u, v, s) \mid (u, v, s) \in \mathcal{I}_D(p_1) \cup \mathcal{I}_D(p_2)\} \tag{4.8}$$

$$\mathcal{I}_D(p^*) = \{(u, v, s) \mid (u, v, s) \in \mathcal{I}_D(\epsilon) \cup \mathcal{I}_D(p) \cup \mathcal{I}_D(p^2) \cup ...\} \tag{4.9}$$

$$\mathcal{I}_D(\lceil\lceil p\rfloor\rfloor) = \{(u, v, \langle u, \widetilde{p}, v\rangle) \mid \exists s, (u, v, s) \in \mathcal{I}_D(p)\} \tag{4.10}$$

$$\mathcal{I}_D([p]) = \{(u, u, s \bullet \langle v, u\rangle) \mid \forall v, (u, v, s) \in \mathcal{I}_D(p)\} \tag{4.11}$$

where $\widetilde{p^n}$ is a concatenation of n p's, and $\widetilde{p}$ is a string of a copy of p with all $\lceil\lceil\ \rfloor\rfloor$ removed, e.g., $\widetilde{\lceil\lceil a \cdot b\rfloor\rfloor} = a \cdot b$. We define the definition of instances of an RRE for a particular pair of nodes u and v in database D such that $\mathcal{I}_D^{u,v}(p) = \{(u, v, s) \mid \forall s, (u, v, s) \in \mathcal{I}_D(p)\}$. Further, if a database D is clear from context, we may write $\mathcal{I}_D^{u,v}(p)$ and $\mathcal{I}_D(p)$ simply as $\mathcal{I}^{u,v}(p)$ and $\mathcal{I}^{u,v}(p)$, respectively. For the remaining of this chapter, we assume all relationships are RREs.

Proposition 4.4. *Given a schema S, $a \in S$, p, p_1 and p_2 are arbitrary RRE expressions over S, and a database $D \in \texttt{Inst}(S)$ where nodes u and v are in D, the following properties hold.*

(1) If $\mathcal{I}_D^{u,v}(p) \neq \emptyset$, then $|\mathcal{I}_D^{u,v}(\lceil\lceil p\rfloor\rfloor)| = 1$. Otherwise, $|\mathcal{I}_D^{u,v}(\lceil\lceil p\rfloor\rfloor)| = 0$.

(2) $\mathcal{I}_D^{u,v}(\lceil\lceil a\rfloor\rfloor) = \mathcal{I}_D^{u,v}(a)$

(3) $|\mathcal{I}_D^{u,v}(p_1 \cdot p_2)| = \sum_{w \in D} |\mathcal{I}_D^{u,w}(p_1)||\mathcal{I}_D^{w,v}(p_2)|$

(4) If $(u, p_1, v) \in D$ if and only if $(u, p_2, v) \in D$, then $|\mathcal{I}_D^{u,v}(\lceil\lceil p_1\rfloor\rfloor)| = |\mathcal{I}_D^{u,v}(\lceil\lceil p_2\rfloor\rfloor)|$.

(5) $|\mathcal{I}_D^{u,u}([p])| = |\mathcal{I}_D^{u,u}(p \cdot \lceil\lceil p^-\rfloor\rfloor)|$

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

Given a transformation $\gamma : \phi(\bar{x}) \to (x_1, a, x_2)$, one can construct an undirected graph

$G_\gamma = (V, E)$ such that $V = \{\bar{x}\}$ and E is a set of an edge (x_i, p, x_j) where (x_i, p, x_j) is an atom in $\phi(\bar{x})$. We say that γ is acyclic if G_γ contains *no* cycle. In the following theorem, we assume that the premise of every transformations are acyclic.

Theorem 4.1. *Given two schemas S and T, for every transformation Σ_{ST} where $S \overset{\Sigma_{ST}}{\equiv} T$, for every pattern p over S, there exists a pattern p' over T such that, $\forall D \in \texttt{Inst}(S)$, $\forall u, v \in D$, $|\mathcal{I}_D^{u,v}(p)| = |\mathcal{I}_{\Sigma_{ST}(D)}^{u,v}(p')|$.*

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

Suppose $p = \lceil\lceil r \rfloor\rfloor$ for some pattern r over S. By Proposition 4.4(4), if there exists a pattern r' over T s.t. $|\mathcal{I}_D^{u,v}(r)| > 0$ if and only if $|\mathcal{I}_{\Sigma_{ST}(D)}^{u,v}(r')| > 0$, we have $|\mathcal{I}_D^{u,v}(p)| = |\mathcal{I}_{\Sigma_{ST}(D)}^{u,v}(\lceil\lceil r' \rfloor\rfloor)|$. Also, by Proposition 4.4(5), one may write $p \cdot \lceil\lceil p^- \rfloor\rfloor$ instead of $[p]$. Hence, we may consider a pattern p without any use of $\lceil\lceil\ \rfloor\rfloor$ or $[\ ]$. Further, since $\mathcal{I}(p*) = \mathcal{I}(\epsilon) \cup \mathcal{I}(p) \cup \mathcal{I}(p^2) \cup ...$, if there exists p' such that $|\mathcal{I}_D^{u,v}(p)| = |\mathcal{I}_{\Sigma_{ST}(D)}^{u,v}(p')|$, then $|\mathcal{I}_D^{u,v}(p^*)| = |\mathcal{I}_{\Sigma_{ST}(D)}^{u,v}(p'^*)|$. Hence, we may also consider a pattern p without the use of $*$.

Assume $p = p_1 + ... + p_m$ where $p_1, ..., p_m$ are distinct and contain no '+'.

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

Clearly, if $p_i = a$ or $p_i = a^-$ where $a \in S$ and $a \in T$, then the statement holds. Otherwise, since Σ_{ST} is information preserving, there exists $k > 0$ transformation rules in its inverse s.t. $\phi_1(x_1, x_2, \bar{x}) \rightarrow (x_1, a, x_2)$, ..., $\phi_k(x_1, x_2, \bar{x}) \rightarrow (x_1, a, x_2)$. Because each rule is acyclic, one can construct a pattern $p'_{i,j}$ that traverses $\phi_j(\bar{x})$ from x_1 to x_2 for each $j = 1...k$. We have that, $\forall u, v \in D$, $(u, a, v) \in D$ if and only if $\bigvee_{j=1...k} \phi_j(u, v, \bar{x})$ if and only if $(u, p'_{i1} + ... + p'_{ik}, v) \in \Sigma_{ST}(D)$. Let $p_i' = \lceil\lceil p'_{i1} + ... + p'_{ik} \rfloor\rfloor$. By Proposition 4.4(4), $|\mathcal{I}_D^{u,v}(p_i)| = |\mathcal{I}_{\Sigma_{ST}(D)}^{u,v}(p_i')|$. The proof extends for the case where $p = a^-$.

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

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

Using an induction over the number of disjunction over p, we have that there exists a pattern $p' = p'_1 + ... + p'_k$ s.t. the theorem holds.

We restrict our attention to a transformation with an acyclic premise in order to reduce the expressivity of the relationship language and keep the computation of similarity scores efficient. A cyclic premise allows multiple traversals from one variable to another along the premise, and requires an indicator in the relationship language where two variables along the traversal are the same, e.g., starting and ending nodes in a cycle are the same. For instance, consider the relationship pattern representing the premise of a cyclic tgd $(x_1, \mathtt{a}, x_2) \wedge (x_2, \mathtt{b}, x_3) \wedge (x_3, \mathtt{c}, x_4) \wedge (x_1, \mathtt{d}, x_3) \wedge (x_2, \mathtt{e}, x_4) \rightarrow (x_1, \mathtt{f}, x_4)$. It is *not* possible to rewrite this pattern to an equivalent one without a conjunction, e.g., $\wedge$. That is, the premise must be rewritten in the form (x_1, exp, x_4) for some RRE exp. To remove the conjunction between in $(x_1, \mathtt{a}, x_2) \wedge (x_2, \mathtt{b}, x_3)$, one may write $(x_1, \mathtt{a} \cdot \mathtt{b}, x_3)$. However, because x_2 is specified in $(x_2, \mathtt{e}, x_4)$, x_2 cannot be removed, and so this conjunction is necessary. Hence, the language to properly express this relationship pattern should be a conjunctive RRE expression. By adding conjunction to the relationship language, as the patterns become more complex, it will take longer to compute the similarity scores between nodes. The result of Theorem 4.1 extends for general tgd constraints if conjunction is added to our proposed relationship language.

The following corollary is an immediate result of Theorem 4.1.

Corollary 4.1. *Given a database D of a schema S, for every transformation Σ_{ST} for some schema T, there is a mapping M between the set of patterns over S and the set of patterns over T such that, for a given pattern p over S, we have that $\forall D \in \mathtt{Inst}(S)$, $\forall u, v \in D$, $\mathit{sim}_p(u, v, D) = \mathit{sim}_{M(p)}(u, v, \Sigma_{ST}(D))$.*

Corollary 4.1 guarantees that, for each pair of entities u and v and pattern p between them over a dataset, one can always find an 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 4.2 to compute similarity on RRE patterns *Relationship-Similarity* (SR-PathSim).

One may define RWR or SimRank scores between entities based on a particular relationship pattern between entities [4]. 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 a 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 it reaches v. Each hop, hence, is intuitively defined as a single edge between two nodes. However, when given a relationship pattern, a hop is defined only if there is a walk that follows and completes the given pattern from one node to another node. Following this idea, we can use the same measurement as SimRank and RWR to compute similarity scores over a relationship pattern as similarly specified in SR-PathSim. Using a similar proof to Theorem 4.1, we prove the following proposition. Let $\texttt{RWR}_p(u, v, D)$ and $\texttt{SimRank}_p(u, v, D)$ denote a similarity score between nodes u and v computed using RWR and SimRank scoring function that only consider walks that follows RRE p.

Proposition 4.5. *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 $\forall D \in \texttt{Inst}(S)$, $\forall u, v \in D$, $\textit{RWR}_p(u, v, D) = \textit{RWR}_{M(p)}(u, v, \Sigma_{ST}(D))$ and $\textit{SimRank}_p(u, v, D) = \textit{SimRank}_{M(p)}(u, v, \Sigma_{ST}(D))$.*

Proof. The proof is similar to the one of Theorem 4.1. Specifically, RWR and SimRank assume that the weight of connectivity between two nodes in a data graph depends on the number of instances of the relationship pattern in the database. The weight matrices are then used to compute similarity score between two nodes.

4.5.3 Computing Similarity Scores

For an expression with only concatenations, the number of RRE instances can be computed using *commuting matrix* [4]. Given labels $l_1, ..., l_m$ in a schema S, a commuting matrix of an expression $p = l_1 \cdot ... \cdot l_m$ over database D is $\mathbf{M}_p = \mathbf{A}_{l_1}\mathbf{A}_{l_2}...\mathbf{A}_{l_m}$ where $\mathbf{A}_{l_i}$ is an adjacency matrix that represents a number of edges of label l_i between pairs of nodes in D. Each entry $\mathbf{M}_p(u, v)$ represents the number of instances of p from node u to node v in D. Given a commuting matrix, we can compute a similarity score $\texttt{sim}_p(u, v, D)$ as $\frac{2\mathbf{M}_p(u,v)}{\mathbf{M}_p(u,u)+\mathbf{M}_p(v,v)}$ [4].

We extend the computation of commuting matrix for RRE expressions as follows. Given matrices $\mathbf{X}$ and $\mathbf{Y}$, let $>$ be a boolean operation such that each entry (i, j) of $\mathbf{X} > \mathbf{Y}$ is 1 if $\mathbf{X}(i, j) > \mathbf{Y}(i, j)$ or 0 otherwise, and $\mathbf{diag}\{\mathbf{X}\}$ denote a diagonal matrix of $\mathbf{X}$. Given a label a and arbitrary expressions p, p_1 and p_2 over database D in schema S, we have $\mathbf{M}_a = \mathbf{A}_a$, $\mathbf{M}_{p^-} = \mathbf{M}_p^T$, $\mathbf{M}_{p_1 \cdot p_2} = \mathbf{A}_{p_1}\mathbf{A}_{p_2}$, $\mathbf{M}_{p_1+p_2} = \mathbf{A}_{p_1} + \mathbf{A}_{p_2}$ if $p_1 \neq p_2$, $\mathbf{M}_{p_1+p_2} = \mathbf{A}_{p_1} = \mathbf{A}_{p_2}$ if $p_1 = p_2$, $\mathbf{M}_{\lceil p \rfloor \rfloor} = \mathbf{M}_p > \mathbf{0}$, 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 expressions p over database D still follow standard matrix operations, the complexity is bounded by $O(m|V|^3 + n|V|^2)$ where m de-

notes the number of matrix multiplications, e.g., the number of concatenations and nested operations in p, n denotes number of other operations, and $|V|$ denotes the number of nodes in D. Therefore, SR-PathSim still has the same complexity as that of PathSim.

Nevertheless, one may reduce the complexity by exploiting the use of parenthesis when constructing an expression p. For instance, consider an expression $p = l_1 \cdot l_2 + l_1 \cdot l_3$ for some labels l_1, l_2 and l_3. The commuting matrix for p can be computed as $\mathbf{A}_{l_1}\mathbf{A}_{l_2} + \mathbf{A}_{l_1}\mathbf{A}_{l_3}$. One may rewrite p as $l_1 \cdot (l_2 + l_3)$ which can be computed as $\mathbf{A}_{l_1}(\mathbf{A}_{l_2} + \mathbf{A}_{l_3})$. Hence, the latter helps reduce the required matrix operations by one multiplication. One may also use sparse matrices operations or any existing fast matrices multiplication to reduce the time complexity [78]. Further, consider that certain patterns (or sub-patterns) may be frequently used. Their commuting matrices, hence, are repeatedly constructed. To reduce such repetitive computation and overall running time, we may pre-materialize those matrices then look them up in later computation. For instance, one can pre-materialize and store all commuting matrices $\mathbf{M}_{l_1 \cdot l_2}$ of pattern $l_1 \cdot l_2$, for every pair of labels l_1 and l_2. To compute instances of pattern $a \cdot b \cdot c$, one only needs to look up $\mathbf{M}_{a \cdot b}$ and $\mathbf{M}_{b \cdot c}$ and performs a single multiplication $\mathbf{M}_{a \cdot b}\mathbf{M}_{b \cdot c}$. Nevertheless, we do *not* explain and discuss any details of such optimization as it is out of the scope of this dissertation.

To compute the similarity scores using patterns with kleene-star p^*, one has to consider all possible repetitions of p as we have $\mathcal{I}(p^*) = \mathcal{I}(\epsilon + p + p \cdot p + \ldots)$. To compute such patterns efficiently, we limit the maximum number of repetitions in p^* to a given number.

4.5.4 Connection Between Languages of Constraints and Relationships

Our defined language to express relationship is indeed related to our assumption in the language of database constraints. For instance, consider a constraint $(x_1, a \cdot b, x_2) \rightarrow (x_1, c, x_2)$ over a schema of label $\{a, b, c\}$. One may define a transformation that removes edge of label c in a target database. That is, a path (u, c, v) in a source database will be mapped to a path $(u, a \cdot b, v)$ in the target database, but they may have different strength depends on the number of nodes along the path $a \cdot b$. Hence, we introduced the use of *skipped* operation when expressing relationship. In fact, we show that there is no simpler language than RRE that guarantees robustness given a tgd constraint.

Theorem 4.2. *RRE is necessary to guarantee structural robustness between two database under information-preserving transformations whose schema mapping language is conjunctive RPQ.*

Proof. Consider a full tgd $f(\bar{x}) \rightarrow g(\bar{x})$ where f and g represents relationship patterns and

$\bar{x}$ denotes entities in the graph database. The smallest relation presented between entities in a graph is shown with a single relation label. Suppose g denotes a single relation label l between entities x_1 and x_2. Without losing generality, assume $g(\bar{x}) = l(x_1, x_2)$. That is, to guaranteed robustness, a transformation maps, at most, $f(\bar{x})$ to $l(\bar{x})$. Since there exists one or more instance of $f(\bar{x})$ per $l(x_1, x_2)$, the skipped operation over $f(\bar{x})$ (or equivalent) between x_1 and x_2, is necessary to satisfy the bijective mapping between their instances, e.g., there is at most one $l(x_1, x_2)$ in the database. Inversely, let $h(\bar{x})$ denote $f(\bar{x}) - l(x_1, x_2)$. Let $h_1(\bar{x})$ and $h_2(\bar{x})$ denote relationship patterns of $h(\bar{x})$ from x_1 and from x_2, respectively. To bijectively map $f(\bar{x})$ to $g'(\bar{x})$ where $l(x_1, x_2)$ is in $g'(\bar{x})$, one must increase at least the same amount of information, e.g., $h_1(\bar{x})$ and $h_2(\bar{x})$, to ensure equivalent instances between $f(\bar{x})$ and $g'(\bar{x})$. Following the definition of a nested operation, the operation is necessary to satisfy bijective mapping between instances of $f(\bar{x})$ and $g'(\bar{x})$.

Nevertheless, there arising a question of whether RRE suffices to guarantee the robustness when a language of database constraint is more complex than what we have discussed. For instance, a nested regular expression (NRE) can be used as a language of database constraints or schema mapping [32]. It is interesting to see whether a more expressive relationship language is necessary or whether it is possible to guarantee a robustness. Such discussion is indeed far beyond our analysis as we only strict our constraint in a form of a full tgd. This is because a nested operation may require a use of an embedded tgd due to its semantic, e.g., $[[\,[p]\,]]_D = \{(u, u) \mid \exists v, (u, p, v) \in D\}$. Despite the shortcomings of our analysis and solutions, it is still interesting and challenging for future research to address these questions.

4.6 SIMPLIFYING SR-PATHSIM

The relationship language presented in Section 4.5, is relatively complicated and hard to construct or interpret for average users. For instance, an RRE $p_4 : \texttt{field} \cdot [\texttt{pubslihed-in}^-] \cdot [\texttt{pubslihed-in}^-] \cdot \texttt{field}^-$ is less intuitive than an RRE $p_2 : \texttt{field} \cdot \texttt{field}^-$ over Figure 4.1(b). Generally, users would like to spend less effort to express their queries. One way to achieve this goal is to enable users to submit their patterns using a relatively smaller and intuitive subset of operations in our proposed pattern language such as concatenations. In addition, users may also like to measure the degree of similarity of two entities using a set of related relationships to get a more holistic view of their similarities. For example, users may want to use both p_2 and p_4 to compute similarity between pairs of research areas over Figure 4.1(b) to find the overall similarity of research areas. In this section, we propose a robust similarity search algorithm whose input is an RRE pattern that uses only concatenations and/or reverse

traversals. We call such a pattern *simple*. Our algorithm leverages the constrains in the database to generate a set of patterns *related* to the input one and use the full expressivity of RRE. Our algorithm, then, computes and aggregates the similarity scores of these patterns. This way, the user can both use a relatively simple language to specify relationships between entities, which is as simple as the one used by PathSim, and take advantage of complex relationships between entities to get effective and robust answers.

Previous studies in database theory have shown that query rewriting has been widely used in query optimization [17, 79]. In many cases, the system uses schema dependencies and constraints to help plan query rewriting. For instance, one may optimize and rewrite their queries by utilizing database constraints such as functional dependencies and inclusion dependencies to remove unnecessary joins [17]. This query optimization helps to improve the effectiveness and efficiency of query answering in the database. Following the idea of query optimization with database constraints, our system utilizes existing constraints to add or remove some information related to the user's relationship pattern. It then constructs a set of related patterns to the user's input. Intuitively, our generated set of related patterns is effective. This is because the user input may not include possible information or may include more information than what the user intends in her input pattern. For instance, in the schema of Figure 4.1(a), the user's input is $\texttt{area}\cdot\texttt{published-in}$. However, her underlying intention is for a relationship pattern between a research area and a conference regardless of the number of publications of the research area in that conference. In this case, we use a constraint information of the schema and find that there is an alternative representation as shown in Figure 4.1(b) in which a direct edge `field` between a conference and a research area exists. This `field` edge satisfies the user intention; however, it does not exist in Figure 4.1(a). Hence, we modify the user's expression over Figure 4.1(a) to $\lceil\lceil\texttt{area}\cdot\texttt{published-in}\rfloor\rfloor$ which is equivalent to `field` in Figure 4.1(b).

Algorithm 4.1 finds a subset of RREs by minimally modifying a simple pattern given by the user such that the results of Corollary 4.1 holds for the aggregated scores of all patterns. Each RRE returned by the algorithm represents a relationship pattern that may include or exclude some information to or from the input pattern according to the database constraints. These patterns also represent relationships that may be found in a database under different structures but contains equivalent information content. For instance, given an input $p_2 : \texttt{field}\cdot\texttt{field}^-$ over Figure 4.1(b), the algorithm returns a set of RREs including both p_2 and $p_4 : \texttt{field}\cdot[\texttt{pubslihed-in}^-]\cdot[\texttt{pubslihed-in}^-]\cdot\texttt{field}^-$. These RREs can then be used in computing aggregate similarity scores over each RRE in the returned set using Equation 4.2.

Specifically, Algorithm 4.1 takes a simple pattern $p = l_1 \cdot ... \cdot l_n$ from a user in addition to a database schema $S = (\mathcal{L}, \Gamma)$. Let (r, i) denote an RRE r which is processed up to label l_i

Algorithm 4.1: `Path Modifier`

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

```
1  done ← {}
2  processing ← {(ϵ, 0)} // For each pair (r, i) ∈ processing, r denotes an RRE processed up to the
       position of label l_i in p
3  foreach (r, i) ∈ processing do
4      Remove (r, i) from processing
5      if i ≥ n then
6          Add r to done
7          continue
8      Add (r · l_{i+1}, i + 1) to processing
9      s ← l_{i+1} · l_{i+2} · ...l_n
10     foreach γ ∈ Γ do
11         R ← R ∪ SubPathModifierPerConstraint(γ, s)
12     foreach j ≥ i + 1 do
13         foreach (l_{i+1} · l_{i+2} · ... · l_j, exp′) ∈ R do
14             Add (r · exp′, j) to processing
15 return E_p ← done
```

in p. Given (r, i), let s denote the the remaining unprocessed sub-path of p, e.g., $s = l_{i+1} \cdot ... \cdot l_n$. Algorithm 4.1 examines each sub-path $exp : l_{i+1} \cdot l_j$ of s for some $i+1 \leq j \leq n$. Then, it uses Algorithm 4.2 to find a set $\mathcal{R}$ of possible replacement pattern for exp according to each constraint in S (Line 11). If a replacement exp' exists for exp, the algorithm replaces exp in s with exp' and marks that all labels up to l_j has been processed, e.g., $(r \cdot exp', j)$ (Line 14). In addition, the algorithm also keeps the case where exp is not replaced, e.g., $(r \cdot exp, j)$ or $(r \cdot l_{i+1}, i+1)$ when consider only $j = i+1$ (Line 8). Overall, Algorithm 4.1 fines all possible combinations of replacement over each sub-path of p.

We provide a toy example of running Algorithm 4.1 as follows.

Example 4.7. *Consider a simple pattern* $p = \mathtt{a} \cdot \mathtt{b} \cdot \mathtt{c} \cdot \mathtt{d}$ *and a single constraint* γ*. Starting from* $(\epsilon, 0)$*, the algorithm adds* $(\mathtt{a}, 1)$ *to processing. Here, we have the remaining unprocessed sup-path* $s = \mathtt{a} \cdot \mathtt{b} \cdot \mathtt{c} \cdot \mathtt{d}$ *of* p*. Then the algorithm examines each of the following sub-paths of* s*:* $\mathtt{a}$*,* $\mathtt{a} \cdot \mathtt{b}$*,* $\mathtt{a} \cdot \mathtt{b} \cdot \mathtt{c}$ *and* $\mathtt{a} \cdot \mathtt{b} \cdot \mathtt{c} \cdot \mathtt{d}$*. Assume Algorithm 4.2 returns a set* $\mathcal{R}$ *consisting of* $(\mathtt{a}, \mathtt{w}_1)$ *and* $(\mathtt{a} \cdot \mathtt{b} \cdot \mathtt{c}, \mathtt{w}_2)$ *for the simple pattern* s *and the constraint* γ*. Hence, the algorithm adds* $(\mathtt{w}_1, 1)$ *and* $(\mathtt{w}_2, 3)$ *to processing. Then the algorithm continues with the next member in processing. Consider* $(\mathtt{a}, 1)$ *in processing, where the remaining unprocessed sub-path* s *is* $\mathtt{b} \cdot \mathtt{c} \cdot \mathtt{d}$*. It first adds* $(\mathtt{a} \cdot \mathtt{b}, 2)$ *to processing. Assume Algorithm 4.2 returns a set* $\mathcal{R}$ *consisting of* $(\mathtt{b} \cdot \mathtt{c} \cdot \mathtt{d}, \mathtt{w}_3)$ *for the new value of* s*. It then adds* $(\mathtt{a} \cdot \mathtt{w}_3, 4)$ *to processing. Since*

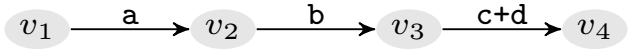


Figure 4.2: A representing graph $\mathcal{G}(\phi)$ of a conjunctive RPQ $\phi(\bar{x}) : (x_1, \mathtt{a}, x_2) \wedge (x_2, \mathtt{b}, x_3) \wedge (x_3, \mathtt{c} + \mathtt{d}, x_4)$.

4 is the length of p, $\mathtt{a} \cdot \mathtt{w_3}$ *is marked as done and will be added to the final results in* $\mathcal{E}_p$*. The algorithm repeats these steps until all members in processing are processed.*

Before we explain Algorithm 4.2, we would like to describe a graph representing a conjunctive RPQ, e.g., the premise of a constraint. Given a non-empty conjunctive RPQ $\phi(\bar{x})$, let us first assume that, for each atom (x_i, exp, x_j) in ϕ, exp *cannot* be written as $exp_1 \cdot exp_2$ for some non-empty expressions exp_1 and exp_2. If such atom exists, the atom is rewritten as $(x_i, exp_1, x') \wedge (x', exp_2, x_j)$ for some fresh variable x'. A representing graph of a conjunctive RPQ $\phi(\bar{x})$, denoted by $\mathcal{G}(\phi)$, is a directed graph (V, E) whose nodes are variables in ϕ and edges are labeled by the RPQ pattern between each pair of those variables in ϕ. More precisely, a node $v_i \in V$ if and only if x_i is a variable in ϕ, and an edge $(v_i, exp, v_j) \in E$ if and only if (x_i, exp, x_j) is in ϕ. For instance, given a conjunctive RPQ $\phi(\bar{x}) : (x_1, \mathtt{a}, x_2) \wedge (x_2, \mathtt{b}, x_3) \wedge (x_3, \mathtt{c} + \mathtt{d}, x_4)$, the graph $\mathcal{G}(\phi)$ is shown in Figure 4.2.

Next, we would like to explain the underlying idea of the replacement patterns found by Algorithm 4.2. Consider that each constraint implies an information-preserving transformation that may add or remove an edge from the current schema. For instance, consider a constraint γ_{1a} as described in Example 4.1 over a schema S_{1a} of a database shown in Figure 4.1(a). Following Example 4.2, a transformation $\Sigma_{1a,1b}$ from S_{1a} may add an edge label `field` between two nodes u_1 and u_2 for every path `area`·`published-in` from u_1 to u_2. The transformation then removes edges of label `area` resulting in a target schema S_{1b} of a database as shown in Figure 4.1(b). That is, the corresponding RRE over S_{1a} to `field` over S_{1b} is $p_{field} : \lceil\lceil \mathtt{area} \cdot \mathtt{published\text{-}in} \rfloor\rfloor$. Consider $\mathcal{G}(\phi_{\Sigma_{1a,1b}})$ which is the representing graph of the premise of transformation $\Sigma_{1a,1b}$ or $\phi_{\Sigma_{1a,1b}}$. Because edges $(v_1, \mathtt{area}, v_3)$ and $(v_3, \mathtt{published\text{-}in}, v_4)$ exist in $\mathcal{G}(\phi_{\Sigma_{1a,1b}})$, the path `area`·`published-in` from u_1 to u_2 matches the same pattern from v_1 to v_4 over $\mathcal{G}(\phi_{\Sigma_{1a,1b}})$. Hence, we can find an RRE pattern p_{field} which is one of the traversals from v_1 to v_4 over $\mathcal{G}(\phi_{\Sigma_{1a,1b}})$.

Based on the aforementioned observation, given a simple pattern r and a constraint γ whose set of induced transformations is $\mathbb{T}^{\gamma}$, we describe Algorithm 4.2 that finds associated RREs over graphs representing those induced transformations to r. Generally, for each simple pattern exp, the algorithm constructs one or more RREs exp' from possible traversals over $\mathcal{G}(\phi_{\Sigma_{1a,1b}})$ along the path exp, for some $\Sigma_{1a,1b} \in \mathbb{T}^{\gamma}$. Then each sub-path of the user's pattern matching exp is replaced with each corresponding exp' in Algorithm 4.1. We defer

Algorithm 4.2: `Sub-path Modifier Per Constraint`

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

$\mathbb{T}^\gamma \leftarrow$ a set of predicted transformations induced by γ over S
$\mathbb{R} \leftarrow \cup\Sigma_{ST} \in \mathbb{T}^\gamma$ `// a set of all transformation rules`
foreach $i > 0, j \geq i$ **do**
    $exp \leftarrow l'_i \cdot l'_{i+1} \cdot ... \cdot l'_j$
    if a path exp from some v_g to v_h exists in $\mathcal{G}(\phi_\Sigma)$ for some $\Sigma \in \mathbb{R}$ **then**
        Find all RREs $exp' : v_g \hookrightarrow_H v_h$ that traverse $\mathcal{G}(\phi_\Sigma)$ from v_g to v_h and visit each edge in $\mathcal{G}(\phi_\Sigma)$ once
        Add (exp, exp') to $\mathcal{R}$
        Add (exp^-, exp'^-) to $\mathcal{R}$
return $\mathcal{R}$

our discussion in finding the set $\mathbb{T}^\gamma$ later in Section 4.7.

For Algorithm 4.2, we briefly describe a recursive procedure to compute an RRE $v_s \hookrightarrow_G v_t$ that traverses a graph G from node v_s to v_t as follows. Consider that one may adopt a depth-first or breath-first search algorithm to find all paths, i.e. simple patterns, from node v_i to node v_j in G. An RRE pattern of all n paths that traverse from node v_i to node v_j is $p_1^{i,j} + ... + p_n^{i,j}$. Since we assume a constraint to be acyclic, then n is exactly 1. Let us denote this pattern as $p^{i,j}$. At each node v_i which connects to some leaf node v_k in G, we construct a pattern $[p^{i,k}]$. We then concatenate $[p^{i,k}]$ at the front of any pattern from v_i or at the end of any pattern to v_i. We mark each edge as visited when each pattern $p^{i,j}$ or $[p^{i,k}]$ is constructed. The base case is to first construct a pattern $p^{s,t}$. The procedure ends when all edges in the graph of G are visited. We must note that each constructed $p^{i,j}$ can also be written as $\lceil\lceil p^{i,j} \rfloor\rfloor$, which results in multiple patterns of this traversal.

Example 4.8. *Given a graph $mG(G)$ as shown in Figure 4.2 and a simple pattern* $\mathtt{a} \cdot \mathtt{b}$*, Some possible RRE patterns that traverse this graph from v_1 to v_3, i.e., $v_1 \hookrightarrow_{\mathcal{G}(G)} v_3$, are* $\mathtt{a} \cdot \mathtt{b} \cdot [\mathtt{c} + \mathtt{d}]$ *and* $\lceil\lceil \mathtt{a} \cdot \mathtt{b} \rfloor\rfloor \cdot [\mathtt{c} + \mathtt{d}]$*. In this case, Algorithm 4.2 adds all* $(\mathtt{a} \cdot \mathtt{b}, exp')$ *to the returned set $\mathcal{R}$ where exp' is one of the above patterns.*

The complexity of Algorithm 4.1 largely depends on the number of replacement patterns found from Algorithm 4.2. In Algorithm 4.2, for each induced transformation, since each simple pattern p found in a graph G when constructing the traversal can be either p or $\lceil\lceil p \rfloor\rfloor$, this algorithm is indeed exponential in the number of simple patterns. However, each simple pattern is a subgraph of G that is a path graph, e.g., a connected tree whose nodes have degree 2 except two terminal nodes. Hence, the number of simple patterns

in G 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 $O(\exp\{|E(G)|\})$ for each transformation. The value of $|E(G)|$ is the number of atoms in a transformation. We later show that this number is generally the same as the size of a constraint. Also, since the algorithm iterates over all sub-paths of the input pattern, the total complexity of Algorithm 4.2 is $O(n^2 \cdot |\mathbb{T}| \cdot \exp\{|E(G)|\}$ where n is the number of labels in the input pattern. Nevertheless, constraints of a schema are given and usually have small number of terms compared to the size of databases. Further, unless the constraints are changed regularly, the set $\mathbb{T}$ is constant regardless of the user's input, and can be pre-materialized. We may view the value $|\mathbb{T}| \cdot \exp\{|E(G)|$ as some constant C. Hence, Algorithm 4.2 is $O(n^2)$. Since Algorithm 4.1 replaces each label or sub-expression of the user input with possible RREs returned by Algorithm 4.2, the complexity of Algorithm 4.1 is $O((n^2)^P) = O(n^{2P})$ where P denote the number of replaced sub-paths. In the worst case analysis, P is as large as $O(n^2)$. We, however, later empirically show in Section 4.9 that not all sub-paths of the user input is replaced, and so P is much smaller than n^2.

Proposition 4.6. *Given a database D of schema S and a equivalent schema T under information preserving transformation Σ, for every simple expression p_S over S, there exists a simple expression p_T over T such that, $\forall u, v \in D$, $\sum_{p \in \mathcal{E}_{p_S}} \mathtt{sim}_p(u, v, D) = \sum_{p' \in \mathcal{E}_{p_T}} \mathtt{sim}_{p'}(u, v, \Sigma(D))$*

Proof. (sketch) Let $f_{crpq}(v_g \hookrightarrow_G v_h)$ denote a CRPQ representing an RRE $exp' : v_g \hookrightarrow_G v_h$ in Algorithm 4.2. Consider a transformation $\Sigma : f_{crpq}(v_g \hookrightarrow_G v_h) \rightarrow (x_g, l, x_h)$ for some variables x_g and x_h corresponding to v_g and v_h, respectively, and some label $l \notin S$. Clearly, Σ is information preserving, and $\lceil\lceil v_g \hookrightarrow_G v_h \rfloor\rfloor$ is mapped to l according to Theorem 4.1. Similarly, there exists an RRE exp'' over T that maps to l for some Σ'. By transitivity, exp' is mapped to exp'' according to Theorem 4.1. We must note that Σ always exists, and Algorithm 4.2 yields all possible such Σ''s. Hence, using similar arguments to Theorem 4.1 and Corollary 4.1, we have that our proposition holds.

Thus, a similarity search algorithm that compute aggregate similarity scores over a set of RREs returned by Algorithm 4.1 is structurally robust.

4.7 INFORMATION-PRESERVING VARIABILITY PREDICTIONS

In Section 4.6, we have presented an algorithm that finds a set of *related* relationship patterns to a given input simple pattern. However, the algorithm assumes that one may find

a set of induced transformations of a known schema's constraint. In this section, we would like to discuss and present an algorithm that helps to predict such structural variability given a constraint.

4.7.1 Beyond Renaming

As a constraint is a full tgd, one may write a *trivial* constraint over a schema S as $\gamma_a : (x_1, a, x_2) \rightarrow (x_1, a, x_2)$ where a is a label in S. However, every database that has label a satisfies γ_a regardless of whether the database is an instance of S. In fact, if a is a label in both schemas $S_1 = (\mathcal{L}, \Gamma)$ and $S_2 = (\mathcal{L}, \Gamma \cup \{\gamma_a\})$, then $\texttt{Inst}(S_1) = \texttt{Inst}(S_2)$. Intuitively, this constraint is meaningless in the term of putting a restriction over databases of the schema. Hence, for the remaining of this chapter, we ignore any occurrence of a trivial constraint whose premise and conclusion are equivalent, and treat such constraints as if they *do not exist*.

For relational databases, it has been proved by Hull that there is *not* any information-preserving variation of a relational schema without any constraint beyond simple renaming of the scheme elements [41]. We show that this result also holds for a graph database.

Theorem 4.3. *Given two schemas $S = (\mathcal{L}_S, \Gamma_S)$ and $T = (\mathcal{L}_T, \Gamma_T)$ where $\Gamma_S = \Gamma_T = \emptyset$, if there exists Σ_{ST} such that $S \overset{\Sigma_{ST}}{\equiv} T$, then $S = T$ or there is a bijection between $\mathcal{L}_S$ and $\mathcal{L}_T$.*

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

Based on these findings, we have that, for a representation variations beyond renaming, either the source schema or the target schema must contain a non-empty set of constraints,

e.g., there exists one or more non-trivial constraint. Since an inverse of an information preserving transformation is also information preserving, without losing generality, we assume that the set of constraints for a source schema is non-empty.

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

Theorem 4.4. *Given two schemas $S = (\mathcal{L}_S, \Gamma_S)$ and $T = (\mathcal{L}_T, \Gamma_T)$ where $\Gamma_S = \emptyset$, if there exists Σ_{ST} such that $S \overset{\Sigma_{ST}}{\equiv} T$, then there exists a bijection between $\mathcal{L}_S$ and some $L' \subseteq \mathcal{L}_T$ where there is no constraint whose conclusion contains label in L', and, for each $l \in \mathcal{L}_T \setminus L'$, there exists a constraint $f(\bar{x}) \to (x_1, l, x_2)$ in Γ_T where l does not appear in f, for some CRPQ f.*

Proof. It is implied by Theorem 4.3 that $|\mathcal{L}_S| \leq |\mathcal{L}_T|$. Further, using similar arguments in the proof of Theorem 4.3, we have that if there exists no $L' \subseteq \mathcal{L}_T$ such that a bijection between $\mathcal{L}_S$ and L' exists, then there is no bijection between $\texttt{Inst}(S)$ and $\texttt{Inst}(T)$.

Consider a set of instances $V \subseteq \mathcal{V}$. We have that, since there is no constraints whose conclusion contain a label in L', for each $I \in V \times L' \times V$, there exists $I' \in \texttt{Inst}_V(T)$ and $I \subseteq I'$. Suppose there is no constraint $f(\bar{x}) \to (x_1, l, x_2)$, for some $l \in \mathcal{L}_T \setminus L'$. For some subset $V \subseteq \mathcal{V}$, there must also exists a database J containing edge of label $\mathcal{L}_T \setminus L'$ and there is no $K \in V \times L' \times V$ s.t. $K \subseteq J$. Therefore, $|\texttt{Inst}_V(T)| > |V \times L' \times V| = |\texttt{Inst}_V(S)|$. That is, a surjective mapping between $\texttt{Inst}_V(S)$ and $\texttt{Inst}_V(T)$, there exists an instance in $\texttt{Inst}_V(S)$ that maps to multiple instances in $\texttt{Inst}_V(T)$. In addition, since the bijection between $\mathcal{L}_S$ and L' exists, for any $V' \neq V \subseteq \mathcal{V}$, $|\texttt{Inst}_{V'}(S)| \leq |\texttt{Inst}_{V'}(T)|$. Using similar arguments to Theorem 4.3, we show that there is no bijective mapping between $\texttt{Inst}(S)$ and $\texttt{Inst}(T)$. Therefore, S and T are not information equivalent.

Following Theorem 4.4, we have that if a schema S contains no constraint, every information preserving transformation from S preserves all edges (or up-to renaming of those edges). We later discuss in the following section and call this type of transformation an *easy* transformation.

Corollary 4.2. *Every transformation from a schema without a constraint is easy.*

4.7.2 Variability induced by a single constraint

Consider that if a transformation Σ_{ST} over schema S is information preserving, then there exists an inverse transformation Σ_{ST}^{-1} such that every database instance D of S is mapped to itself by the composition $\Sigma_{ST}^{-1} \circ \Sigma_{ST}$. As shown in Proposition 4.3, each database instance

D of S must satisfy $\Sigma_{ST}^{-1} \circ \Sigma_{ST}$. Suppose there exists a constraint γ in S. Each database D must also satisfy γ. Hence, we define that **a transformation induced by a constraint γ over schema S** is **an information preserving transformation Σ_{ST} whose inverse Σ_{ST}^{-1} satisfies $\Sigma_{ST}^{-1} \circ \Sigma_{ST} \equiv \gamma$.** We denote a transformation from S to a target schema T induced by γ as Σ_{ST}^{γ}.

Following the proposed definition and Proposition 4.3, we design a high-level framework in predicting information preserving transformations induced by a single constraint as follows.

> *Given schemas S and T, and a constraint γ over S, for each candidate transformation Σ_{ST}, construct possible transformations Σ_{TS}, if there exists Σ_{TS} such that $\Sigma_{TS} \circ \Sigma_{ST}$ is equivalent to γ, then Σ_{ST} is a transformation induced by γ over S.*

Clearly, this proposed framework is impractical due to the construction of an infinite candidate set of Σ_{ST} and Σ_{TS}. Hence, in the remaining of this section, we will discuss and leverage the property of an induced transformation in order to filter and limit the set of candidate transformations Σ_{ST} and their inverse transformations Σ_{TS} (if one exists).

Consider a (full) tgd constraint $\gamma : \phi(\bar{x}) \rightarrow \bigwedge_{i=1...k}(x_{i_1}, a_i, x_{i_2})$. We have that if a database satisfies γ, then it also satisfies $\gamma_i : \phi(\bar{x}) \rightarrow (x_{i_1}, a_i, x_{i_2})$ for each $i = 1...k$. In order to simplify our analysis in the remaining of this section, we assume that the conclusion of each tgd constraint contains only a single atom. That is, for a tgd constraint γ whose conclusion contains multiple atoms, we replace γ with $\gamma_1, ..., \gamma_k$. In addition, if a database satisfies a constraint $\phi_1(\bar{x}_1) \vee \phi_2(\bar{x}_2) \rightarrow (x_1, a, x_2)$, we have that the database satisfies both $\phi_1(\bar{x}_1) \rightarrow (x_1, a, x_2)$, and $\phi_2(\bar{x}_2) \rightarrow (x_1, a, x_2)$. Similarly, if a database satisfies $\omega : \psi(\bar{x}) \wedge (x_m, p_1 + ... + p_k, x_n) \rightarrow (x_1, a, x_2)$ where $p_1, ..., p_k$ are some RPQs without '+', then it also satisfies $\omega_i : \psi(\bar{x}) \wedge (x_m, p_i, x_n) \rightarrow (x_1, a, x_2)$ for each $i = 1...k$. Hence, one may replace a constraint with multiple constraints without '+' in any of their atoms. To simplify our analysis, we also assume no use of '+' in our tgd constraints. Further, as implied by Proposition 4.3, we assume that a constraint γ^* exists whenever a full tgd constraint γ exists for a schema S.

Intuitively, in order to modify a database structure, a transformation may add or remove edges of certain relation labels. For instance, a transformation removes edges of label `area` from Figure 4.1(a) and adds new edges label `field` as shown in Figure 4.1(b). Nevertheless, we show in Proposition 4.7 that, given a constraint γ, a transformation induced by γ cannot remove any edge of a label that does not appear in the conclusion of γ.

Proposition 4.7. *Given a schema S with a constraint $\gamma : \phi_\gamma(\bar{x}) \rightarrow (x_1, a, x_2)$, for every transformation Σ_{ST}^{γ} from S to a target schema T, there exists a mapping $M : S \rightarrow T$ such that $(x, l, y) \rightarrow (x, M(l), y)$, for all $l \neq a \in S$, in Σ_{ST}^{γ}.*

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

Consider that each transformation rule $(x, l, y) \rightarrow (x, M(l), y)$ simply renames each edge label l to a new edge label $M(l)$ in the target schema. To reduce the complexity of our model and analysis, we refer to $M(l)$ in the target schema as l and always assume that this transformation rule exists. `published-in` is an example of such label in the transformation between the two databases presented in Figure 4.1.

We then show in Proposition 4.8 that a transformation induced by γ cannot add any new edge whose label is in the premise of γ. That is, a transformation from the schema of the database shown in Figure 4.1(a) to the one shown in Figure 4.1(b) cannot add any new `published-in` edge in Figure 4.1(b) which does not exist in the database in Figure 4.1(a).

Proposition 4.8. *Given a schema with a constraint γ, for every label l in S that is not in a conclusion of γ, if $\psi(\bar{x}) \rightarrow (x_1, l, x_2)$ is in a transformation Σ^{γ}_{ST}, then $\psi(\bar{x})$ is (x_1, l, x_2).*

Proof. Suppose there exists Σ^{γ}_{ST} over S with rule $\psi(\bar{x}) \rightarrow (x_1, l, x_2)$ where $l \neq a \in S$ and $\psi(\bar{x}) \neq (x_1, l, x_2)$. Consider two minimal databases $D_1, D_2 \in \texttt{Inst}(S)$ in which D_1 satisfies $\psi(\bar{x})$ and (x_1, l, x_2), but D_2 satisfies $\psi(\bar{x})$ but $(x_1, l, x_2) \notin D_2$. We have that $(x_1, l, x_2) \in \Sigma^{\gamma}_{ST}(D_2)$. That is, $\Sigma^{\gamma}_{ST}(D_1) = \Sigma^{\gamma}_{ST}(D_2)$. Since Σ^{γ}_{ST} is information preserving, then there exists an inverse Σ^{γ}_{TS} of Σ^{γ}_{ST}. Hence, $D_1 = (\Sigma^{\gamma}_{TS} \circ \Sigma^{\gamma}_{ST})(D_1) = (\Sigma^{\gamma}_{TS} \circ \Sigma^{\gamma}_{ST})(D_2) = D_2$, which is contradiction.

Based on Propositions 4.7 and 4.8, we have that each candidate transformation Σ^{γ}_{ST} must contain the rule $t_l : (x, l, y) \rightarrow (x, l, y)$ for each label l that is not in a conclusion of γ, and the conclusion of each rule in Σ^{γ}_{ST}, except those t_l's, cannot contain any label of those l's.

Next, consider that some transformations can be easily verified whether they are information preserving without the need to compute its composition with its inverse and check if the result is equivalent to a given constraint. Let us define an *identity* rule as a transformation rule in the form of $(x, l, y) \rightarrow (x, l, y)$ for some label l. Intuitively, an identity transformation rule preserves all edges of such relation label from the source database in the target database. Let $\mathcal{I}_S$ denote a set of all identity rules over S. We call a transformation whose set of transformation rules is $\mathcal{I}_S$ an *identity* transformation. Clearly, an identity transformation and a transformation Σ_{ST} that includes $\mathcal{I}_S$ are information preserving as one can construct an inverse Σ_{TS} which contains all identity rules in $\mathcal{I}_S$. Further, consider a constraint $\gamma : \phi(\bar{x})$

$\rightarrow (x_1, l, x_2)$ over S and a transformation Σ_{ST} induced by γ that simply removes edges of label l, e.g., Σ_{ST} includes all identity rules for each label in S except l. One can construct an inverse Σ_{TS} such that it includes all identity rules of labels other than l and a transformation rule $\phi(\bar{x}) \rightarrow (x_1, l, x_2)$ which recovers all edges of label l. Hence, Σ_{ST} is also information preserving. Let us define an *easy* transformation induced by γ over a schema S, or an easy transformation for brevity, as a transformation whose set of transformation rules contains all identity rules for every label that appears in the premise of γ and all other labels in S that does not appear in γ. We call an information preserving transformation that is not easy a *non-easy* transformation. Following the above discussion about identity rules, we have the following lemma.

Lemma 4.1. *Every easy transformation is information preserving.*

Proof. Let Σ denote an easy transformation over a schema S. Every label in S is also a label of the target schema of Σ. Hence, one can construct a transformation Σ^{-1} whose set of transformation rules is $\mathcal{I}_S$. We have that $\Sigma^{-1} \circ \Sigma$ is $\mathcal{I}_S$. Clearly, $\forall D \in \texttt{Inst}(S)$, $\mathcal{I}_S(D) = D$. Hence, Σ is information preserving.

To identify if a non-easy transformation Σ_{ST} is a transformation induced by a constraint γ over S, we must check whether there exists an inverse Σ_{TS} such that $\Sigma_{TS} \circ \Sigma_{ST} \equiv \gamma$. To avoid constructing all possible transformations Σ_{TS}, we show that there exists a certain transformation Σ_{TS} for each Σ_{ST} such that Σ_{ST} is a transformation induced by γ if and only if $\Sigma_{TS} \circ \Sigma_{ST} \equiv \gamma$. Assume a given constraint is $\gamma : \phi_\gamma(\bar{x}) \rightarrow (x_1, a, x_2)$ and $(x, a, y) \rightarrow (x, a, y) \notin \Sigma_{ST}$. In order to have $\Sigma_{TS} \circ \Sigma_{ST} \equiv \gamma$, we have the following characteristics of Σ_{TS}:

- (x_1, a, x_2) must be the conclusion of σ and so is Σ^{-1},
- each atom in the premise of σ is either from the premise of Σ^{-1} or the premise of Σ, and
- Σ_{TS} contains exactly one non-identity rule.

For simplicity, we denote the non-identity rule of Σ_{TS} as $\tau' : \phi_{\tau'}(\bar{x}_{\tau'}) \rightarrow (x_1, a, x_2)$. We must note that if there exists a transformation rule $\phi_t(\bar{x}_t) \rightarrow (x_{t_1}, l'_t, x_{y_2}) \in \Sigma_{ST}$, where a is not a relation label in $\phi_t(\bar{x}_t)$, then we may have $\phi_t(\bar{x}_t)$ in $\phi_{\tau'}(\bar{x}_{\tau'})$ instead of having (x_{t_1}, l'_t, x_{y_2}) in $\phi_{\tau'}(\bar{x}_{\tau'})$ and perform the composition. Thus, we may consider only transformation rules in Σ_{ST} whose premises contain some atom with relation label a. Let $\mathcal{T}_a$ denote all transformation rules in Σ whose premises contain an atom with relation label a. Let ϕ'_γ denote a copy of ϕ_γ with all atoms with relation label a removed. Since a is not in the target schema T, we have $\phi_{\tau'}$ consists of, at most, ϕ'_γ, ϕ'_t and/or atoms from conclusions of any

Algorithm 4.3: `Transformations induced by` γ

Input: Schemas S and a set of label $\mathcal{L}_T$ for a schema T, a full tgd constraint $\gamma : \phi_\gamma(\bar{x}) \rightarrow (x_1, a, x_2)$ over S

Output: A set $\mathbb{T}^\gamma$ of transformations induced by γ over S

```
/* Assume φ_γ(x̄) = ⋀_{i=1...m}(x_{i1}, l_i, x_{i2}), l_i ∈ S.                    */
T_candid ← {t : φ_t(x̄_t) → (x_{t1}, l'_t, y_{t2}) where t is a non-identity full s-t tgd whose atoms in
  φ_t(x̄_t) are from {(x_{i1}, l_i, x_{i2}), i = 1...m}, for some l'_t ∈ T and l'_t ∉ S}
T_candid ← T_candid ∪ {(x, a, y) → (x, a, y)}
T_id ← {(x, l, y) → (x, l, y), ∀l ≠ a ∈ S}
𝕋^γ ← ∅
foreach τ ⊆ T_candid do
    Σ ← τ ∪ T_id
    if Σ is easy then
        Add Σ to 𝕋^γ
    else
        T_a ← {t ∈ τ s.t. the premise of t contains an atom with label a}
        Rename variables, for each t ∈ T_a, s.t. an atom with label a in its premises is
          (x_1, a, x_2) and other variables are fresh
        φ'_γ ← copy of φ_γ with all atoms with relation label a removed
        foreach t ∈ T_a do
            φ'_t ← copy of φ_t with all atoms whose relation labels containing a removed
        τ' ← a full s-t tgd ⋀_{t∈T_a}(φ'_t(x̄_t) ∧ (x_{t1}, l'_t, y_{t2})) ∧ φ'_γ(x̄) → (x_1, a, x_2)
        if γ ≡ τ ∘ τ' then
            Add Σ to 𝕋^γ
return 𝕋^γ
```

rule in $\mathcal{T}_a$. Thus, let τ' be a full s-t tgd $\bigwedge_{t\in\mathcal{T}_a}(\phi'_t(\bar{x}_t) \wedge (x_{t_1}, l'_t, y_{t_2})) \wedge \phi'_\gamma(\bar{x}) \rightarrow (x_1, a, x_2)$. We have that τ' is sufficient to show the existence of an inverse of Σ_{ST}. That is, we only need to perform a composition $\tau' \circ \Sigma_{ST}$ then check whether it is equivalent to γ.

Following the aforementioned discussion, Algorithm 4.3 provides an algorithmic method to generate all transformations induced by a given constraint γ. We show in Theorem 4.5 that Algorithm 4.3 is correct and complete.

Theorem 4.5. *Given a schema S with a full tgd constraint γ, Algorithm 4.3 generates all information preserving transformations induced by γ over S.*

Proof. Assume $\gamma : \phi_\gamma(\bar{x}) \rightarrow (x_1, a, x_2)$, where $\phi_\gamma(\bar{x}) = \bigwedge_{i=1...m}(x_{i_1}, l_i, x_{i_2})$ is a full tgd constraint associated with schema S.

First, we show that all candidates of transformations induced by γ over S are generated. We must note that one may write a transformation rule whose conclusion contains multiple atoms as a set of multiple rules with a single atom in the conclusion. Hence, it suffices to

generate all transformations with a single atom in their conclusions. By the construction of $\mathcal{T}_{candid}$, and Proposition 4.8, we have that every non-identity transformation rule is a member of $\mathcal{T}_{candid}$ (Line 3). Let $\mathcal{T}_{id}$ denote all identity rules of all labels in S other than a. By Proposition 4.7, a transformation Σ induced by γ over S must also include all members of $\mathcal{T}_{id}$. Hence, $\Sigma = \tau \cup \mathcal{T}_{id}$, $\exists \tau \subseteq \mathcal{T}_{candid}$ (Line 6).

Using Lemma 4.1, if Σ is easy, then Σ is information preserving and is added to $\mathbb{T}^{\gamma}$ (Line 8).

Consider the remaining case where Σ is not easy, e.g., a is a relation label in some atom in ϕ_{γ} and $(x, a, y) \rightarrow (x, a, y) \notin \Sigma$. If Σ is a transformation induced by γ, then there must exist an inverse Σ^{-1} s.t. $\gamma \equiv \Sigma^{-1} \circ \Sigma$. Because of the composition, we have the following characteristics of Σ^{-1}: (1) (x_1, a, x_2) must be the conclusion of σ and so is Σ^{-1}; and (2) each atom in the premise of σ is either from (a) the premise of Σ^{-1} or (b) the premise of Σ. Further, since $\sigma \equiv \gamma$, then Σ contains exactly one non-identity rule. For simplicity, we denote this non-identity rule of Σ as $\tau' : \phi_{\tau'}(\bar{x}_{\tau'}) \rightarrow (x_1, a, x_2)$. We must note that if there exists a transformation rule $\phi_t(\bar{x}_t) \rightarrow (x_{t_1}, l'_t, x_{y_2}) \in \Sigma$, where a is not a relation label in $\phi_t(\bar{x}_t)$, then we may have $\phi_t(\bar{x}_t)$ in $\phi_{\tau'}(\bar{x}_{\tau'})$ instead of having (x_{t_1}, l'_t, x_{y_2}) in $\phi_{\tau'}(\bar{x}_{\tau'})$ and perform composition. Thus, we may consider only transformation rules in Σ whose premises contains some atom with relation label a. Let $\mathcal{T}_a$ denote all transformation rules in Σ whose premises contain an atom with relation label a (Line 12). In addition, since a is not in the target schema T, we have $\phi_{\tau'}$ consists of, at most, ϕ'_{γ} (Line 13), ϕ'_t (Line 15) and/or atoms from conclusions of any rule in $\mathcal{T}_a$. Further, because of the characteristics of Σ^{-1}, τ' (Line 16) is necessary and sufficient to show the existence of an inverse of Σ.

Consider $\Sigma = \tau \cup \mathcal{T}_{id}$ in which τ is the only non-identity rule. It suffices to compute $\tau' \circ \tau \equiv \gamma$ in order to show that $\Sigma^{-} \circ \Sigma \equiv \gamma$. Then Σ is added to $\mathbb{T}^{\gamma}$ if the equivalence holds. This concludes the proof of the theorem.

The procedure of Algorithm 4.3 is as follows. First, the algorithm computes the candidate set $\mathcal{T}_{candid}$ of all transformation rules whose atoms in their premises are also atoms in γ, and since the generated rules must satisfy Proposition 4.8, the relation labels in their conclusions are not labels in the source schema S except the one in the conclusion of γ (Line 3).

Example 4.9. *Given a constraint*

$$\gamma : (x_3, \texttt{area}, x_2) \wedge (x_3, \texttt{published-in}, x_4) \wedge (x_1, \texttt{published-in}, x_4) \rightarrow (x_1, \texttt{area}, x_2)$$

as shown in Figure 4.1(a), $\mathcal{T}_{candid}$ consists of rules, such as

$$t_1: \quad (x_1, \texttt{published-in}, x_4) \wedge (x_3, \texttt{published-in}, x_4) \rightarrow (x1, \texttt{co-publishing}, x_3)$$
$$t_2: \quad (x_3, \texttt{area}, x_2) \wedge (x_3, \texttt{published-in}, x_4) \rightarrow (x_4, \texttt{field}, x_2),$$

and so on.

Next, let $\mathcal{T}_{id}$ denote all identity transformation rules of all labels in S except a (Line 4). Then the algorithm generates and examines each transformation whose rules are from the set of $\mathcal{T}_{candid}$ and also all members of $\mathcal{T}_{Id}$ (Line 7). Let $\mathbb{T}^{\gamma}$ denote a set of generated transformations induced by γ over S. If $\mathcal{I}_S \subseteq \Sigma$, then a transformation is easy. In addition, if a is not in the premise ϕ_γ of γ, then all generated transformations Σ are also easy. Hence, the algorithm adds all of these generated transformations to $\mathbb{T}^{\gamma}$ (Line 8-9) because they are clearly information preserving transformations induced by γ. Otherwise, the algorithm generates τ' which is a necessary inverse of Σ if Σ is a transformation induced by γ (Line 16).

Example 4.10. *Following Example 4.9, let $\tau_1 = \{t_1\}$, $\tau_2 = \{t_2\}$ and $\mathcal{T}_{id} = \{(x, \texttt{published-in}, y) \rightarrow (x, \texttt{published-in}, y)\}$. Clearly, both $\Sigma_1 = \tau_1 \cup \mathcal{T}_{id}$ and $\Sigma_2 = \tau_2 \cup \mathcal{T}_{id}$ are not easy. The rule τ_1' constructed for τ_1 is $(x2, \texttt{co-publishing}, x_4) \rightarrow (x_4, \texttt{area}, x_1)$, which is not a full s-t tgd. Hence, we indeed have that $\tau_1' = \emptyset$. As for τ_2, we have*

$$\tau_2': \quad (x_3, \texttt{published-in}, x_4) \wedge (x_1, \texttt{published-in}, x_4) \wedge (x_4, \texttt{field}, x_2) \rightarrow (x_1, \texttt{area}, x_2).$$

Then the algorithm computes $\tau_1' \circ \tau_1$ and $\tau_2' \circ \tau_2$ and determines whether they are equivalent to γ. Since $\tau_1' \circ \tau \not\equiv \gamma$ but $\tau_2' \circ \tau_2 \equiv \gamma$, the algorithm adds $\Sigma_2 = \tau_2 \cup \mathcal{I}_S$ to $\mathbb{T}^{\gamma}$.

For the details of a composition $\tau' \circ \tau$, one may follow the composition algorithm described in [80]. Intuitively, the procedure for $\tau' \circ \tau$ replaces each atom (x, l, y) in the premise of τ' with a premise of τ whose conclusion has a relation label l. For instance, following Examples 4.9 and 4.10, consider $\tau_2' \circ \tau_2$. The composition replaces $(x_4, \texttt{field}, x_2)$ in τ_2' with $(x_3, \texttt{area}, x_2)$ $\wedge$ $(x_3, \texttt{published-in}, x_4)$ from τ_2, which results in a transformation rule equivalent to γ.

Overall, the complexity of Algorithm 4.3 is $O(\exp\{\exp\{|\gamma|\}\})$ where $|\gamma|$ denotes the total number of atoms in constraint γ. This is because the algorithm first computes all candidate sets of transformation rules in Line 1, then the algorithm examines each set of those transformation rules in Line 5. Nevertheless, this amount of complexity is much smaller than the number of database instances of a given schema. Further, a constraint γ is usually pre-defined with the schema. Algorithm 4.3 can be processed once unless its schema or

its constraints are changed. Hence, the high computational cost of the algorithm is not a concern in this chapter.

Relationship between a tuple-generating dependency and a functional dependency

We must note that our works in this section mainly focus on a database constraint in the form of a full tgd. However, we have discussed and studied some property of a database due to a functional dependency constraint and its structural variations, e.g., an entity-rearranging transformation, in Chapter 3.6. A functional dependency is an example of another commonly used type of database constraints, namely equality-generating dependency, as mentioned earlier in Section 4.3. Because of the difference in the properties of these two types of database constraints, our work on this section may not generalize the case for equality-generating dependencies. Nevertheless, we would like to point out that there is actually some relationship between functional dependencies and tuple-generating dependencies in which we can exploit and show that our algorithm is also robust against an entity-rearranging transformation. We first note that an entity-rearranging transformation involves two or more functional dependencies. Consider two functional dependencies described as follows: $FD_{l_1} : (x_1, l_1, x_2) \wedge (x_1, l_1, x_3) \rightarrow x_2 = x_3$ and $FD_{l_2} : (x_1, l_2, x_2) \wedge (x_1, l_2, x_3) \rightarrow x_2 = x_3$, where l_1 and l_2 are arbitrary labels in a schema.

Proposition 4.9. *If a database D satisfies FD_{l_1} and FD_{l_2}, then D also satisfies a full tgd constraint $\gamma_{FD} : (x_1, l_1 \cdot l_2, x_2) \wedge (x_1, l_1, x_3) \rightarrow (x_3, l_2, x_2)$*

Proof. Given a database D, assume $\mathcal{I}_D(l_1 \cdot l_2) \neq \emptyset$. For every $x_1, x_2, x_3, x_4, x_5 \in D$ where (x_1, l_1, x_4), (x_4, l_2, x_2), (x_1, l_1, x_3) and (x_3, l_2, x_5) are in D, we have $x_4 = x_3$ because of FD_{l_1} and then $x_2 = x_5$ because of FD_{l_2}. Hence, (x_3, l_2, x_2) is also in D. Therefore, D also satisfies γ_{FD}.

Consider that if every database in a schema S satisfies both FD_{l_1} and FD_{l_2}, then clearly every database in S also satisfies γ_{FD}. That is, $\gamma_{FD} \in S$. Since γ_{FD} is a full tgd constraint, our variability prediction algorithm can construct a transformation T whose rules are $(x_1, l_1, x_2) \wedge (x_2, l_2, x_3) \rightarrow (x_1, l', x_3)$ and $(x, l_1, y) \rightarrow (x, l_1, y)$, and the inverse of T consists of $(x_1, l_1, x_2) \wedge (x_1, l', x_3) \rightarrow (x_2, l_2, x_3)$ and $(x, l_1, y) \rightarrow (x, l_1, y)$. This implies that a mapping between a path $l_1 \cdot l_2$ of tuples of entities (x_1, x_2, x_3) exists if and only if a path $l'^{-} \cdot l_1$ of tuples entities (x_3, x_1, x_2) exists. The specification of this transformation describes the property of an entity-rearranging transformation in Section 3.6. Hence, an entity-rearranging transformation is predictable by our work in this section, assuming that a constraint γ_{FD}

is given. We must note that such γ_{FD} is implied by the two functional dependencies. The computation of the implicated constraints is, however, out of the scope of this dissertation.

4.7.3 Variability induced by multiple constraints

Using similar arguments for a transformation induced by a single constraint, given a schema S whose set of constraints is Γ, we define that **a transformation Σ_{ST} is a transformation induced by Γ if there exists an inverse Σ_{ST}^{-1} such that every database D of S satisfies every constraint in $\Sigma_{ST}^{-1} \circ \Sigma_{ST}$.** That is, $\Sigma_{ST}^{-1} \circ \Sigma_{ST} \equiv \Gamma$. We denote a transformation induced by Γ over S as Σ_{ST}^{Γ}.

In order to generate all possible transformations induced by Γ, we show that it is not sufficient to simply construct a set of transformations induced by each constraint γ in Γ and directly union them together.

Example 4.11. *Consider a schema S whose set of labels is $\{a, b, c, d\}$ and the set of constraints Γ contains*

$$\gamma_1 : (x_1, a, x_2) \wedge (x_2, b, x_3) \rightarrow (x_1, c, x_3)$$
$$\gamma_2 : (x_1, d, x_2) \wedge (x_2, c, x_3) \wedge (x_4, c, x_3) \rightarrow (x_1, d, x_4)$$

A possible transformation Σ_{ST} induced by Γ for a target schema T is defined as follows:

$$\Sigma_{ST} : \{(x_1, d, x_2) \wedge (x_2, c, x_3) \rightarrow (x_1, e, x_3),$$
$$(x, l, y) \rightarrow (x, l, y) \,|\, l \in \{a, b\}\}.$$

Σ_{ST} is information preserving as one can construct an inverse transformation Σ_{TS} in which where $\Sigma_{TS} \circ \Sigma_{ST} \equiv \Gamma$ as follows:

$$\Sigma_{TS} : \{(x_1, a, x_2) \wedge (x_2, b, x_3) \rightarrow (x_1, c, x_3),$$
$$(x_1, e, x_3) \wedge (x_2, a, x_4) \wedge (x_4, b, x_3) \rightarrow (x_1, d, x_2),$$
$$(x, l, y) \rightarrow (x, l, y) \,|\, l \in \{a, b\}\}.$$

However, using Algorithm 4.3, we have

$$\mathbb{T}^{\gamma_1} = \{\ \Sigma_{11} : \{(x, l, y) \rightarrow (x, l, y) \,|\, l \in \{a, b, c, d\}\},$$
$$\Sigma_{12} : \{(x, l, y) \rightarrow (x, l, y) \,|\, l \in \{a, b, d\}\}\ \}$$

and

$$\begin{aligned}\mathbb{T}^{\gamma_2} = \{\ \Sigma_{21} : \{(x,l,y) \to (x,l,y) \,|\, l \in \{a,b,c,d\}\},\\ \Sigma_{22} : \{(x_1,d,x_2) \wedge (x_2,c,x_3) \to (x_1,e,x_3),\\ (x,l,y) \to (x,l,y) \,|\, l \in \{a,b,c,d\}\},\\ \Sigma_{22} : \{(x_1,d,x_2) \wedge (x_2,c,x_3) \to (x_1,e,x_3),\\ (x,l,y) \to (x,l,y) \,|\, l \in \{a,b,c\}\}\ \}.\end{aligned}$$

Any transformation that applies a transformation from $\mathbb{T}^{\gamma_1}$ and a transformation from $\mathbb{T}^{\gamma_2}$, e.g., $\Sigma_1 \cup \Sigma_2$ for some $\Sigma_1 \in \mathbb{T}^{\gamma_1}$ and $\Sigma_2 \in \mathbb{T}^{\gamma_2}$, will contain identity transformation rules of labels a, b and at least one of c or d. Hence, we cannot obtain Σ_{ST} using this procedure.

Using the settings in Example 4.11, one can similarly show that Σ_{ST} cannot be obtained by simply composing the two transformations from $\mathbb{T}^{\gamma_1}$ and $\mathbb{T}^{\gamma_2}$. This is because the intermediate schema after applying a transformation may be changed, and the other generated transformations may no longer be induced by the given constraint. Nevertheless, we observe that Σ_{ST} is generated by applying all non-identity transformation rules and shared identity transformation rules from Σ_{12} and Σ_{22}. Generalizing this idea, we can construct a transformation induced by a set of constraints Γ by computing

$$\bigcap_{\gamma \in \Gamma} \Sigma^{\gamma}_{id} \cup \bigcup_{\gamma \in \Gamma} \Sigma^{\gamma}_{other} \tag{4.12}$$

where $\Sigma^{\gamma} = \Sigma_{id} \cup \Sigma_{other}$ is a transformation induced by $\gamma \in \Gamma$, Σ^{γ}_{id} denotes all identity rules in Σ^{γ}, and Σ^{γ}_{other} are all non-identity rules in Σ^{γ}.

Example 4.12. *Following the settings of Example 4.11, a possible transformation computed by formula 4.12 using Σ_{12} induced by γ_1 and Σ_{12} induced by γ_2 is*

$$\begin{aligned}\Sigma = \{\ (x_1,d,x_2) \wedge (x_2,c,x_3) \to (x_1,e,x_3),\\ (x,l,y) \to (x,l,y) \,|\, l \in \{a,b,d\}\ \}.\end{aligned}$$

Nevertheless, we show that some transformations computed by formula 4.12 is not an information preserving transformation induced by Γ.

Example 4.13. *Following the settings in Example 4.11, let Γ also contain a constraint*

$$\gamma_3 : (x_1,b,x_2) \wedge (x_2,d,x_3) \to (x_1,a,x_3)$$

in which Algorithm 4.3 returns

$$\mathbb{T}^{\gamma_3} = \{\ \Sigma_{31} : \{(x,l,y) \to (x,l,y) \,|\, l \in \{a,b,c,d\}\},$$
$$\Sigma_{32} : \{(x,l,y) \to (x,l,y) \,|\, l \in \{b,c,d\}\}\ \}.$$

Computing formula 4.12 over Σ_{12}, Σ_{22} *and* Σ_{32} *yields a transformation*

$$\{(x_1,d,x_2) \wedge (x_2,c,x_3) \to (x_1,e,x_3), (x,b,y) \to (x,b,y)\}$$

which is not information preserving.

Fortunately, following Example 4.13, one may notice the cycle of implication between edge labels amongst the transformation Σ_{12}, Σ_{22} and Σ_{32} described as follows. First, we have only facts about relation labels b and e to recover all constraints in Γ. To recover γ_2, the inverse of Σ_{22} requires facts of relation label c which is implied by γ_1. To recover γ_1, the inverse of Σ_{12} requires facts of relations a which is implied by γ_3. Recovering γ_3 through Σ_{32} also requires facts of label d which is again implied by γ_2. Therefore, it is impossible to find an inverse such that its composition with the transformation is equivalent to Γ. Hence, we introduce Algorithm 4.4 that computes a set of transformations induced by Γ by combining all transformation induced by each constraint in Γ using formula 4.12 and guarantees that there exists no such cycle of implication amongst the combined transformations.

Algorithm 4.4 returns a set of all possible transformations induced by a set of constraints Γ over schema S. We later show in Theorem 4.6 that Algorithm 4.4 is correct and complete. The algorithm first computes a set $\mathbb{S}^{\gamma_i}$ of all possible transformations induced by each constraint $\gamma_i \in \Gamma$ using Algorithm 4.3 (Line 2). Then, the algorithm iterates over all possible lists of transformations induced by each constraint in Γ and constructs a transformation Σ using formula 4.12 (Line 4-8). For instance, following Example 4.12, we have the inverses τ_1' and τ_2' of Σ_1 and Σ_2 are $\tau_1' : (x_1,a,x_2) \wedge (x_2,b,x_3) \to (x_1,c,x_3)$ and $\tau_2' : (x_2,c,x_3) \wedge (x_2,c,x_3) \to (x_1,d,x_2)$, respectively. Clearly, we have that $\tau_1' \circ \Sigma \equiv \gamma_1$. That is, every edge of label c can be recovered, in addition to other labels appearing in identity transformation rules in Σ. For the inverse τ_2' of Σ_2, the target schema does not contain label c. However, we have already shown that label c can be recovered using τ_1'. Hence, $\tau_2' \circ \Sigma_2$ is equivalent to γ_2. If there is more constraint in Γ, the algorithm repeatedly checks whether they can be recovered (Line 15-21). If all constraints in Γ can be recovered, Σ is an information induced by Γ and is added to $\mathbb{T}^{\Gamma}$. Otherwise, there is a cycle of implication amongst transformations induced by some constraints in Γ. In this case, the algorithm ignores this transformation (Line 20).

Algorithm 4.4: `Transformations induced by` Γ

Input: Schema S, a set of m constraints $\Gamma = \{\gamma_i : \phi_i(\bar{x}_i) \rightarrow (x_{i1}, a_i, x_{i2}), i = 1...m\}$ over S
Output: A set $\mathbb{T}^\Gamma$ of transformations induced by Γ over S

```
1  /* Assume Algorithm 4.3 returns T^Γ whose members are pairs (Σ, τ') from
      Line 7 and 16, respectively.                                          */
2  Compute T^{γ_i} for each γ_i ∈ Γ
3  foreach (Σ_1, τ'_1), ..., (Σ_m, τ'_m) ∈ T^{γ_1} × ... × T^{γ_m} do
4      foreach i = 1...m do
5          Σ^i_id ← all identity rules in Σ_i
6          Σ^i_other ← Σ_i \ Σ^i_id
7      Σ_id ← Σ^i_id ∩ ... ∩ Σ^m_id
8      Σ ← Σ_id ∪ ⋃_{i=1...m} Σ^i_other
9      RecoveredLabels ← set of labels l s.t. (x, l, y) → (x, l, y) ∈ Σ_id
10     ToCheckConst ← Γ
11     if I_S ⊆ Σ then
12         Add Σ to T^Γ
13     else
14         repeat
15             Find one γ_i ∈ ToCheckConst s.t. all labels in the premises of γ_i are in
               RecoveredLabels or all labels in the premises of τ'_i are in RecoveredLabels
16             if γ_i exists then
17                 Remove γ_i from ToCheckConst
18                 If there is no other γ_j ∈ ToCheckConst whose relation label in its
                   conclusion is a_i, add a_i to RecoveredLabels
19             else
20                 Break and continue with next Σ
21         until ToCheckConst = ∅
22         Add Σ to T^Γ
23 return T^Γ
```

Theorem 4.6. *Given a schema S whose constraints set is Γ, Algorithm 4.4 returns all possible transformation induced by Γ over S.*

Proof. Since returned transformations are accompanied with an inverse such that the compositions between them are equivalent to a given set of constraints, Γ, they are transformation induced by Γ. Further, the algorithm iterates over all combinations of transformation rules obtained from each constraint in Γ. We must note that, because the composition of the inverse and the transformation must be equivalent to Γ, one cannot construct a transformation rule such that the atoms of its premise are not from a single constraint in Γ. That is, the algorithm explores all possible candidate transformation rules, and so it is complete.

Algorithm 4.4 calls Algorithm 4.3 for each constraint, and there are $O(\exp\{\exp\{|\gamma|\}\})$

possibilities for each set of transformations induced by a constraint. Suppose there are m constraints: γ_1, ..., and γ_m. The complexity of Algorithm 4.4 is dominated by the computation of $\mathbb{T}^{\gamma_1} \times ... \times \mathbb{T}^{\gamma_m}$. Hence, the complexity of Algorithm 4.4 is $O(\exp\{\exp\{|\gamma|\}\}^m)$ where m denotes the number of constraint in Γ and $|\gamma|$ denotes a maximum number of atoms in a constraint in Γ.

4.8 OPTIMIZING SR-PATHSIM

Our proposed algorithms in both Section 4.6 and Section 4.7 are theoretically inefficient. In particular, if the set of RRE patterns returned by Algorithm 4.1 is large, our system has to compute the similarity scores for many patterns. Therefore, it may *not* be efficient on a large database. In this section, we provides a method to reduce the set of RREs in order to improve the efficiency of SR-PathSim while ensuring its effectiveness and robustness.

According to our discussion in Section 4.4, to have an information-preserving variation of a database I, I must satisfy some tgd constraints. However, these constraints may be *trivial*, e.g., $(x, a, y) \rightarrow (x, a, y)$. Intuitively, it is *not* efficient to consider these trivial constraints over a database in our algorithms.

Proposition 4.3 provides that structural variations require database constraints. However, the proposition does not exclude the use of trivial constraints. A trivial constraint is a constraint such that its premise and its conclusion are logically equivalent, e.g., $\phi(\bar{x}) \rightarrow \phi(\bar{x})$. As we have simplified our analysis in Section 4.4.2 where the conclusion of a constraint is an atom of a single label, e.g., (x_i, a, x_j) where $x_i, x_j \in \bar{x}$ and a is a schema label. That is a trivial constraint is of the form $(x_i, a, x_j) \rightarrow (x_i, a, x_j)$. Clearly, every database in a schema S satisfies all trivial constraints for every label $a \in S$. In order to simplify our discussion, since trivial constraints are meaningless in term of putting restriction in a database, we ignore any occurrence of trivial constraints and treat them as if they do not exist.

We have shown in Theorem 4.3 that, for structural variations beyond renaming, either the source schema or the target schema must contain some non-trivial constraints. Similar to the idea of trivial constraint, there always exists a transformation from a schema S to schema T, $T \equiv S$, whose transformation constraints are also trivial. In Section 4.7, we define and call such transformation an *identity* transformation.

Assume that all associated constraints with a schema S are trivial. It is possible to have a non-identity transformation Σ_{ST} from S to a target schema T. For instance, consider a schema $S = \{a, b\}$ without any non-trival constraint. A transformation Σ_{ST} from S to a target schema $T = \{a, b, c\}$ described as $\{(x_1, a, x_2) \wedge (x_2, b, x_3) \rightarrow (x_1, c, x_3), (x_1, l, x_2) \rightarrow (x_1, l, x_2), l \in S\}$ is information preserving. In this case, T consists of a constraint

$(x_1, a, x_2) \wedge (x_2, b, x_3) \rightarrow (x_1, c, x_3)$. However, with only trivial constraints available in S, Algorithm 4.2 cannot produce an RRE $\lceil\lceil a \cdot b \rfloor\rfloor$ over S which is mapped to c over T. In fact, with only trivial constraints, our algorithms do not modify an input expression. Fortunately, our Theorem 4.4 in Section 4.7 provides that if a schema S contains no constraint, every information preserving transformation from S are easy.

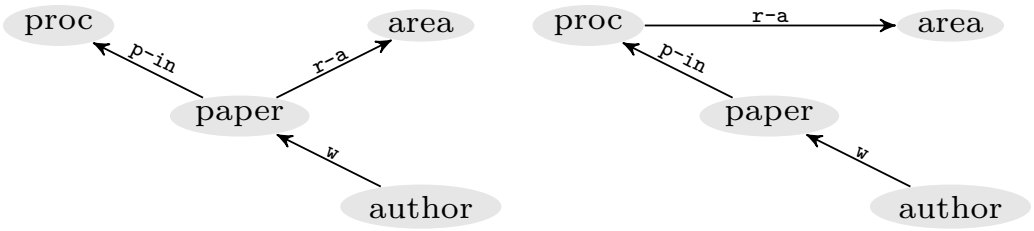
Based on Theorem 4.4 and Corollary 4.2, to avoid the described issues of trivial constraints, our proposed algorithms should ignore producing any expression over a transformation induced by any non-trivial constraint of the form of $\phi(\bar{x}) \rightarrow (x_1, l, x_2)$ where l does not appear in ϕ.

Consider a schema S whose constraint is $\gamma : \phi(\bar{x}) \rightarrow (x_1, l, x_2)$, where l does not appear in ϕ. S is information equivalent to a schema $T = S \setminus \{l\}$ under any transformation rule Σ of a transformation Σ^{γ}_{ST} induced by γ. We have that $v_1 \hookrightarrow_{\mathcal{G}(\Sigma)} v_2$ exists in both S and T. Also, following the proof of Theorem 4.1, we have that an expression l over S is mapped to an expression $r : \lceil\lceil v_1 \hookrightarrow_{\mathcal{G}(\phi_\Sigma)} v_2 \rfloor\rfloor$. However, r is not a simple expression and might not be easily discovered by a user. If we would like to ensure the robustness of SR-PathSim via the use of Algorithm 4.1, then either label l should be disallowed or every l should be replaced with $v_1 \hookrightarrow_{\mathcal{G}(\phi_\gamma)} x_2$ that does not contains a skip-operation.

Further, using Propositions 4.7, we may conclude that non-easy transformations are induced by some constraint $\phi(\bar{x}) \rightarrow (x_1, l, x_2)$ where l appears in $\phi(\bar{x})$. That is, an RRE that does *not* contain label l is obtained from some easy transformation.

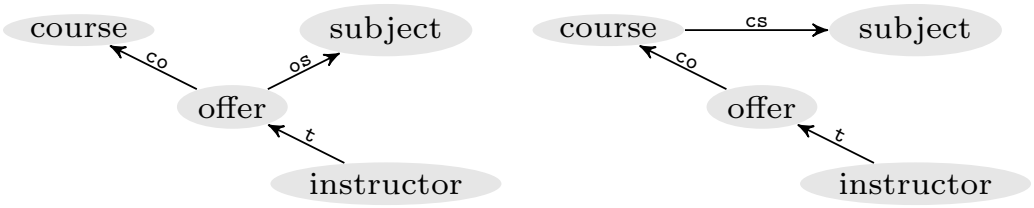
To this end, we should filter out all RREs returned by Algorithm 4.1 that are induced by any easy transformation. For instance, given a constraint $(x_1, \texttt{area}, x_3) \wedge (x_3, \texttt{published-in}, x_4) \wedge (x_2, \texttt{published-in}, x_4) \rightarrow (x_1, \texttt{area}, x_2)$ in Figure 4.1(a), the algorithm should ignore producing an RRE such as $\texttt{published-in} \cdot \texttt{published-in}^{-}$. However, an RRE such as $\texttt{area} \cdot \texttt{published-in}$ is valid because area appears in the conclusion of the constraint. Hence, this filtering helps reduce the space and running time of aggregate SR-PathSim over a set of relationship patterns returned by Algorithm 4.1.

Lastly, one may argue that Algorithm 4.2 does not consider transformations induced by multiple constraints. However, our findings in Section 4.7.3 imply that one can obtain any transformation induced by multiple constraints from a set of transformation rules induced by each of those constraints. Since each related relationship pattern in Algorithm 4.2 is obtained from a single rule specified by one of these transformations. Hence, it suffices to compute and find the patterns from transformations induced by a single constraint.



(a) DBLP (b) SIGMOD Record

Figure 4.3: Schema fragments of bibliographic databases. `p-in`, `r-a` and `w` denote edge labels `published-in`, `research-area` and `writes`, respectively.



(a) WSU (b) Alchemy UW-CSE

Figure 4.4: Variations for course databases. `cs`, `os`, `t` and `co` denote edge labels `course-subject`, `offering-subject`, `teach`, and `offering-course`, respectively.

4.9 EMPIRICAL EVALUATION

4.9.1 Experiment Settings

We use 4 datasets in our experiments: DBLP[1], Microsoft Academic Search[3] (MAS), WSU course dataset[4] and a Biomedical dataset (BioMed). *DBLP* consists of 1,227,602 nodes and 2,692,679 edges, which contains bibliographic information of publications in computer science. We add information about the research areas for each conference in DBLP from information extracted from MAS. Figure 4.3(a) depicts the schema of DBLP. We also use a subset of 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*. WSU course database contains information about courses, instructors and course offerings in the university. The dataset consists of 1,124 nodes and 1,959 edges. Figure 4.4(a) depicts the schema of WSU dataset. The Biomedical dataset, (BioMed), is made available to us as a part of an NIH funded project. The dataset contains information about genetic conditions, diseases, drugs, and their relationships. Figure 4.5 depicts a fragment of BioMed. It consists of 43,307 nodes and 1,742,970 edges.

[3] *academic.research.microsoft.com*

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

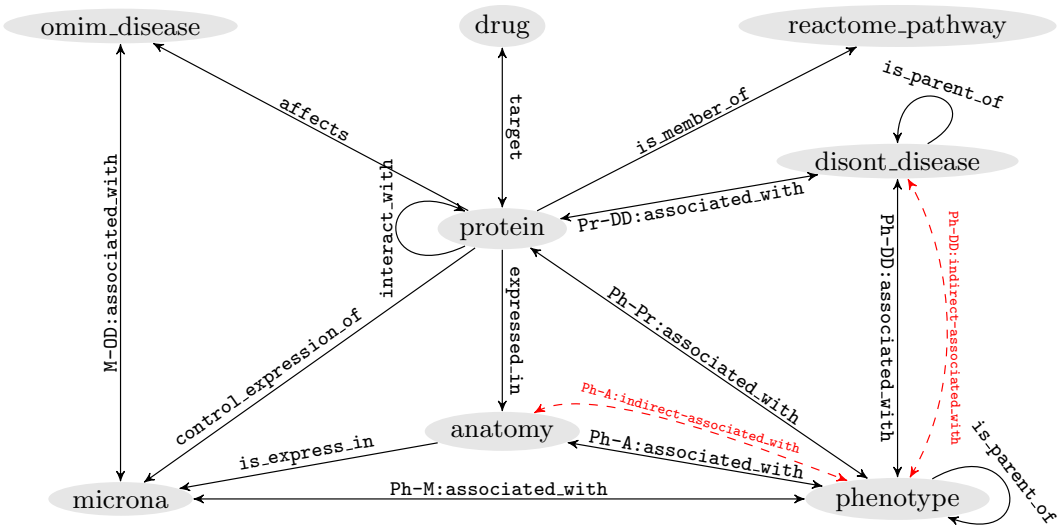


Figure 4.5: Schema fragment of BioMed dataset

We compare robustness, effectiveness and efficiency of SR-PathSim with RWR [5] using a restart probability of 0.8, SimRank [2] using a damping factor of 0.8, and PathSim [4]. Since previous study show that the PathSim similarity computation method is more effective than those of SimRank and RWR [4], we use SR-PathSim with the similarity computation score of PathSim , i.e., Equation 4.2. We implement all algorithms using MATLAB 8.5 on a Linux server with 64GB memory and two quad core processors.

4.9.2 Structural Robustness

We adopt normalized Kendall's tau measurement to compare two ranked lists. The value of normalized Kendall's tau varies between 0 and 1 where 0 means two lists are identical and 1 means one list is the reverse of another. Because users are interested in highly ranked answers, we compare only top 5 and top 10 answers.

We use DBLP, WSU and BioMed databases to evaluate the structural robustness of RWR, SimRank, PathSim and SR-PathSim. Because SimRank takes too long to finish on full DBLP dataset, we perform this evaluation using a subset of DBLP with 24,396 nodes and 98,731 edges.

DBLP dataset satisfies constraint $(paper_1, \texttt{r-a}, area) \wedge (paper_1, \texttt{p-in}, proc) \wedge (paper_2, \texttt{p-in}, proc) \rightarrow (paper_2, \texttt{r-a}, area)$. We transform this database to a database with the structure shown in Figure 4.3(b), which follows the style of SIGMOD Record database[2]. We call this transformation DBLP2SIGM. We randomly sample 100 proceedings based on their node degrees as our query workload over these datasets.

WSU course dataset satisfies the constraint $(offer_1, \texttt{os}, subject) \wedge (offer_1, \texttt{co}, course) \wedge (offer_2, \texttt{co}, course) \rightarrow (offer_2, \texttt{os}, subject)$. We transform WSU course database to the graph structure of Alchemy UW-CSE database[5] whose structure is shown in Figure 4.4(b). We call this transformation WSUC2ALCH. We also randomly sample 100 courses from WSU based on their degrees as our query workload for these datasets.

BioMed database satisfies the constraints: ($phenotype_1$, `is-parent-of`, $phenotype_2$) $\wedge$ ($phenotype_1$, `associated-with`, $anatomy$) $\rightarrow$ ($phenotype_2$, `indirect-associated-with`, $anatomy$) and ($phenotype_1$, `is-parent-of`, $phenotype_2$) $\wedge$ ($disease$, `associated-with`, $phenotype_1$) $\rightarrow$ ($disease$, `indirect-associated-with`, $phenotype_2$). We transform the BioMed dataset such that all edges of label `indirect-associated-with` are removed. We denote the transformation over BioMed dataset as BioMedT. The structure of the transformed BioMed dataset is also shown in Figure 4.5 excluding all dashed edges. A main goal of using this dataset in the NIH project is to discover the drugs that are closely related to queried diseases. Since we use this dataset to also evaluate the effectiveness of our algorithms, we have obtained a set of 30 diseases and their relevant drugs from experts in the domain of the data. Since paths between diseases and drugs are asymmetric, we cannot compute similarity scores using PathSim formula over this dataset. Instead, we evaluate the queries using HeteSim [44], which extends PathSim to support asymmetric paths, e.g., finding similarity between different entity types.

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

Table 4.1 shows the average ranking differences for top 5 and 10 answers returned by RWR, SimRank, PathSim and/or HeteSim. We do *not* report the results of SR-PathSim because it returns the same answers over all transformations. For PathSim, we use the expressions $\texttt{p-in}^- \cdot \texttt{r-a} \cdot \texttt{r-a}^- \cdot \texttt{p-in}$ and $\texttt{r-a} \cdot \texttt{r-a}^-$ over DBLP and SIGMOD Record structures, respectively. For SR-PathSim, we use the same expression as that for PathSim over DBLP and use an RRE $[\texttt{p-in}^-] \cdot \texttt{r-a} \cdot \texttt{r-a}^- \cdot [\texttt{p-in}^-]$ over SIGMOD. Over WSU and Alchemy UW-CSE, we use the simple patterns $\texttt{co}^- \cdot \texttt{os} \cdot \texttt{os}^- \cdot \texttt{co}$ and $\texttt{cs} \cdot \texttt{cs}^-$, respectively, for PathSim. For SR-PathSim, we use the same expression over WSU, but we use $[\texttt{co}^-] \cdot \texttt{cs} \cdot \texttt{cs}^- \cdot [\texttt{co}^-]$ over Alchemy UW-CSE. For BioMed dataset, we consult an expert and obtain an RRE

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

Table 4.1: Average ranking differences

	DBLP2SIGM		WSUC2ALCH		BioMedT	
	top 5	top 10	top 5	top 10	top 5	top 10
RWR	0.447	0.412	0.259	0.253	0.130	0.112
SimRank	0.455	0.410	0.387	0.341	0.405	0.385
PathSim	0.608	0.590	0.310	0.247	0.438	0.461

p_{ex} : `target·express-in·` (`Ph-A:associated-with` + `Ph-A:indirect-associated-with`) · (`Ph-DD:associated-with` + `Ph-DD:indirect-associated-with`). The corresponding RRE over the transformed BioMed dataset is p_{ex}^T : `target·express-in·` (`Ph-A:associated-with` + $\lceil\lceil$`is-parent-of`$^-$ · `Ph-A:associated-with`$\rfloor\rfloor$) · (`Ph-DD:associated-with` + $\lceil\lceil$`is-pareentof`$^-$ · `Ph-DD:associated-with`$\rfloor\rfloor$). Since HeteSim does *not* support RRE, we compute the similarity scores between a pair of drug dr and disease dd by averaging similarity scores computed over the following patterns: `target·express-in·` $L_1 \cdot L_2$, where L_1 is either `Ph-A:associated-with` or `Ph-A:indirect-associated-with` and L_2 is either `Ph-DD:associated-with` or `Ph-DD:indirect-associated-with`. Since edges of label `indirect-associated-with` do *not* exist in the transformed database, our best attempt to construct an equivalent expression to p_{ex}^T for HeteSim results in the following patterns: `target·express-in·` $L_1' \cdot L_2'$, where L_1' is either `Ph-A:associated-with` or `is-parent-of`$^-$·`Ph-A:associated-with` and L_2' is either `Ph-DD:associated-with` or `is-parent-of`$^-$ ·`Ph-DD:associated-with`. According to Table 4.1, the outputs of all algorithms, except SR-PathSim, are significantly different across databases under these information-preserving transformations.

4.9.3 Effectiveness

We evaluate the effectiveness of SR-PathSim over MAS and BioMed. For query workload over MAS, we randomly sample 100 conferences based on their degrees in the dataset. To provide the ground truth, for a given conference q, we manually label other conferences in three categories: *similar*, *quite-similar* and *least-similar*. A conference is considered similar to q when they share the same research area. A conference is considered quite-similar to q when they are connected to strongly related research area. Otherwise, the conference is considered least-similar to q. For example, *Data Mining* and *Databases* are strongly related, but *Databases* and *Computer Vision* are not. We use Normalized DCG (nDCG) to evaluate the effectiveness because it supports multiple levels of relevance for returned answers [60]. The value of nDCG varies between 0 and 1 where the high values indicate more effective

ranking. We compare the effectiveness of SR-PathSim with PathSim and report the values of nDCG for top 5 (nDCG@5) and top 10 (nDCG@10) answers. For BioMed, we obtain the query workload from an expert. We use 30 queries of diseases with their relevant drugs from the domain experts. Since each disease query relates only to a single drug, we use MRR to evaluate the effectiveness of the algorithms.

Over MAS, we compute similarities between conferences based on the keywords in their domains. We use the pattern $\mathtt{pc} \cdot \mathtt{pd} \cdot \mathtt{da} \cdot \mathtt{da}^- \cdot \mathtt{pd}^- \cdot \mathtt{pc}^-$ and $\lceil\lceil \mathtt{pc} \cdot \mathtt{pd} \rfloor\rfloor \cdot \mathtt{da} \cdot \mathtt{da}^- \cdot \lceil\lceil \mathtt{pd}^- \cdot \mathtt{pc}^- \rfloor\rfloor$ for PathSim and SR-PathSim, respectively. The average nDCG@5 (nDCG@10) for R-PathSim and PathSim are 1.0 (1.0) and 0.969 (0.901), respectively. SR-PathSim significantly outperforms PathSim. This is because, nodes about paper should not influence the similarity score between conferences based on the keywords of their domains. Since the language of relationship pattern used by PathSim is less expressive, the pattern between conferences and keywords always include papers. Hence, it deems conferences with more papers to be more similar although they may not have many common keywords. The RRE language used by SR-PathSim is more expressive, and so it could avoid this problem. For example, the top 5 answers returned by SR-PathSim for query *SIGKDD* are *ICDM*, *IDEAL*, *PAKDD*, *PJW* and *PKDD*. However, the the top 5 answers returned by PathSim for the same query are *ICOMP*, *IC-AI*, *ICAIL*, *ICALP* and *ICANN*.

Over original BioMed dataset, the average MRR of HeteSim, SimRank, RWR and SR-PathSim are 0.077, 0.062, 0.010, 0.077, respectively. Over BioMed under BioMedT, the average MRR of HeteSim, SimRank, RWR and SR-PathSim are 0.072, 0.062, 0.010, 0.077, respectively. According to our discussion with the experts, these queries are very hard to answer effectively by using only the structural patterns in the data set and without consulting external sources of knowledge. According to the experts, even a slight improvement in the accuracy of the returned answers may save a great deal of time and effort in their research. The overall results show that SR-PathSim are more effective than other algorithms.

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

4.9.4 Efficiency

We evaluate the query processing time of SR-PathSim and PathSim over DBLP and BioMed datasets using the query workloads reported in Section 4.9.2. First, we evaluate the query processing time of SR-PathSim and PathSim for the case where the user provides an exact relationship pattern (Section 4.5). All reported running times in this section assume

Table 4.2: Running times and a list of expressions used to measure query-processing times of PathSim and SR-PathSim, where `e-i`, `a-w`, `i-a-w` and `i-p-o` stand for `express-in`, `associated-with`, `indirect-associated-with` and `is-parent-of`, respectively.

		Patterns over transformed database	
Source	Pattern over source	PathSim (simple)	SR-PathSim (RRE)
DBLP	p-in·r-a	p-in·r-a	[p-in$^-$]·r-a
DBLP	w·r-a	w·p-in·r-a	w·⌈⌈p-in · r-a⌋⌋
BioMed	target·e-i· (Ph-A:a-w+Ph-A:i-a-w) ·(Ph-DD:a-w+Ph-DD:i-a-w)	target·e-i·Ph-A:a-w·Ph-DD:a-w	target·e-i ·(Ph-A:a-w+⌈⌈i-p-o$^-$ · Ph-A:a-w⌋⌋) ·(Ph-DD:a-w+⌈⌈i-p-o$^-$ · Ph-DD:a-w⌋⌋)

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 SR-PathSim and PathSim. Theoretically, both SR-PathSim and PathSim have the same time complexity. However, the expressiveness of RRE used in SR-PathSim 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 SR-PathSim 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 SR-PathSim 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 SR-PathSim using p_R and report the results. Table 4.2 shows lists of referenced patterns and the corresponding patterns used to measure query-processing time of PathSim and SR-PathSim. The average query processing time for a single pattern per query of SR-PathSim (PathSim) over DBLP and BioMed dataset are 0.035 (0.024) and 0.473 (0.267) seconds, respectively. The reason that SR-PathSim is slower than PathSim as SR-PathSim uses more complex and longer patterns than those used by PathSim. But, the running time is still relatively short over large datasets.

Next, we measure the efficiency of SR-PathSim that incorporates Algorithm 4.1 introduced in Section 4.6 and applied the filter discussed in Section 4.8. In this version, SR-PathSim takes a simple pattern as an input. Hence, we supply the same pattern to both SR-PathSim and PathSim, and compare their query processing times. Since a constraint is given per dataset, we have precomputed the set of predicted transformations induced by the constraint and its set of all transformation rules, e.g., $\mathbb{T}^\gamma$ and $\mathbb{R}$, respectively. We use the same relationship patterns over DBLP and BioMed as described in Section 4.9.2. The average query processing time per query of SR-PathSim (PathSim) over DBLP and BioMed dataset are 0.061 (0.024) and 0.730 (0.477) seconds, respectively. Overall, the running time of SR-PathSim is slightly slower than PathSim due to the procedure of Algorithm 4.1. This

result also shows that making SR-PathSim more usable does *not* increase its running time considerably.

CHAPTER 5: COST-EFFECTIVE CONCEPTUAL DESIGN

Ideally, we would like data analytics algorithms to achieve both structural robustness and effectiveness. Achieving such properties means that we have eliminated the cost of revising the structure of a database during the data preparation phase. However, it is also possible that the cost for design independence is too high and, in some case, infeasible to achieve. Some developers may also be willing to exchange some robustness and effectiveness for improved efficiency. In this chapter, we would like to provide a method to choose an effective structural design whose cost in construction and/or maintenance is affordable, while ensuring the imposed design helps to improve the effectiveness of an algorithm as much as possible.

5.1 BACKGROUND

Taxonomies provide shared understandings of concepts in domains of interests [81]. In particular, they facilitate query answering over unstructured and semi-structured datasets in these domains. For example, assume that a user likes to find information about types of pains caused by Trachoma over excerpts of the medical articles in Figure 5.1. In the absence of any structured data, she may explore this dataset using inherently ambiguous keyword queries and submit query Q_1:*Trachoma pain*. Unfortunately, the article about Trachoma in Figure 5.1 refers to this infection by its other name, *Granular conjunctivitis*. Because all articles contain the term *pain*, the query interface may return all articles, two of which do not have any information about Trachoma.

Given a taxonomy, we can annotate entities in an unstructured dataset by their concepts in the taxonomy. Users may also learn the taxonomy and use its concepts in their queries. Figure 5.4 depicts fragments of the Medical Subject Heading (MeSH) taxonomy, in which nodes denote concepts and edges show subclass relationships [82]. Figure 5.2 shows the medical article excerpts in Figure 5.1 whose entities are annotated by their concepts from MeSH taxonomy. Now, our user may mention the concept *Trachoma* in her query and query interface will return only the articles that contain entities from this concept.

Organizations often use available taxonomies to annotate their datasets so that more users can effectively search and explore their data. For example, the U.S. National Library of Medicine annotates the articles in MEDLINE/PubMED using concepts in MeSH taxonomy [82, 83]. Researchers have used the ProBase taxonomy to extract concepts from Web data [84]. The NIH funded Gene Ontology Consortium (*geneontology.org*) encourages re-

```
<article>
    Granular conjunctivitis causes pain in the outer surface or cornea. ...
</article>
<article>
    Stye may lead to pain on the eyelids. ...
</article>
<article>
    GAS caused infections cause pain in tissues. ...
</article>
```

Figure 5.1: Medical article excerpts

```
<article>
    <Trachoma>Granular conjunctivitis</Trachoma> causes pain in the outer
    surface or cornea. ...
</article>
<article>
    <Hordeolum>Stye</Hordeolum> may lead to pain  pain on the eyelids. ...
</article>
<article>
    <Ecthyma>GAS caused infections</Ecthyma> cause pain in tissues. ...
</article>
```

Figure 5.2: Annotated medical article excerpts

```
<article>
    <Eye-Infections>Granular conjunctivitis</Eye-Infections> causes pain in
    the outer surface or cornea. ...
</article>
<article>
    <Eye-Infections>Stye</Eye-Infections> may lead to pain on the eyelids. ...
</article>
<article>
    <Skin-Infections>GAS caused infections</Skin-Infections> cause pain in
    tissues. ...
</article>
```

Figure 5.3: Medical article excerpts annotated with more general concepts

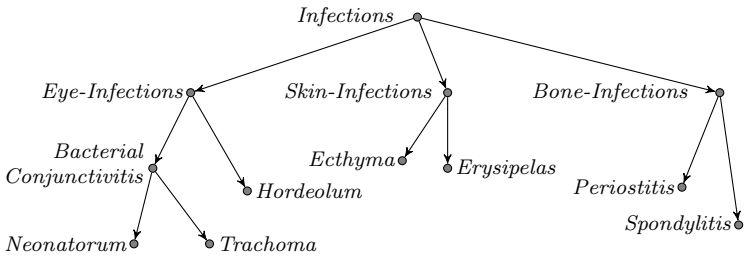


Figure 5.4: Fragments of MeSH taxonomy

searchers to annotate their datasets using a standard taxonomy so their datasets become more accessible to other researchers. We have been recently asked by some botanists to annotate a collection on plant biology by the concepts in Plant Ontology (*plantontology.org*). Organizations also use taxonomies to extract entities in general domains. For example, researchers have used *ProBase* taxonomy to extract concepts in general domains from Web data [84, 85]. Also, Google and Bing ask organizations to annotate their Web documents by concepts in Schema.org taxonomy, which is developed for datasets in general domains.

Ideally, one would like to annotate all relevant concepts in a given taxonomy from a data set to answer all queries effectively. However, an organization has to spend significant amounts of time, financial and computational resources, and manual labor to accurately extract entities of a concept in a large data set [23–25, 86–90]. The organization usually has to develop or obtain a complex program called a *concept annotator* to annotate instances of a concept from a collection of documents [91]. It is not uncommon for an annotator to have thousands of manually written programming rules, which takes a great deal of resources to write and debug [91]. One may also use machine learning algorithms to develop an extractor for a concept to avoid hand-tuned programming rules. In this approach, developers have to find a set of *relevant features* for the learning algorithm. As the specifications of relevant features *not* usually clear, developers have to find the relevant features through trial and error over numerous iterations, which takes a great deal of time and effort [86, 92]. Moreover, if concept annotators use supervised learning algorithms, developers have to also create training data, which require additional time and manual labor. It is more resource-intensive to develop annotators for concepts in specific domains, such as biology, as it requires expensive communication between domain experts and developers. Current studies indicate that these communications are not often successful and developers themselves have to go through the data to find the relevant features in these domains [86]. As most concept annotators perform complex text analysis, it may take them days to process a large dataset and produce an annotated collection [23, 24, 89, 90] Since concept annotators may not be sometimes sufficiently accurate, domain experts have to review and revise the results of the

annotations [82, 83] It is estimated that annotating each article in MEDLINE/PubMED collection using concepts in MeSH taxonomy costs about $9.4 [83].

Because the structure and content of underlying datasets evolve over time, annotators should be regularly rewritten and repaired. Many annotators need to be rewritten in average almost every two months [87]. Recent studies show that many concept annotators need to be rewritten in average about every two months [87]. Thus, enterprises often have to repeat the resource-intensive steps of developing a concept annotator to maintain an up-to-date annotated data set.

Because the financial or computational resources of an organization are limited, it may not be able to afford to develop and maintain annotators for all concepts in a taxonomy. Also, many users may need an annotated data set quickly and cannot wait days for an (updated) annotated collection [89]. For example, a reporter who pursues some breaking news and an epidemiologist that follows the pattern of a new potential pandemic on the Web and social media need relevant answers to their queries fast. They may not want to wait for organizations to (re-)write and (re-)deploy the annotators for all concepts in their domains of interests. Hence, an organization may be able to afford to annotate only a subset of concepts in a taxonomy. Similarly, many users may not have the time to learn all concepts in a large taxonomy and may prefer to learn and use a relatively small subset of the taxonomy in their queries. For example, an enterprise may annotate entities in Figure 5.1 with only concepts *Eye-Infection* and *Skin-Infection* from MeSH taxonomy and get the collection in Figure 5.3.

Intuitively, a query interface may provide less effective answers to queries over the dataset in Figure 5.3 than the one in Figure 5.2. Assume that a user wants to find information about the type of pain associated with Trachoma. She may mention the concept *Eye-Infection* in her query. The query interface may return the articles about Trachoma and the one about Hordeolum. Nevertheless, the annotation in Figure 5.3 still helps the query interface not to return the non-relevant article about the skin infection. Clearly, we would like to select a subset of concepts whose required time and/or resources for annotation do not exceed our budget and most improves the effectiveness of answering queries.

Currently, concept annotation experts use their intuitions to discover cost-effective conceptual designs from taxonomies. Because most taxonomies contain hundreds of concepts [93], this approach does not scale for real-world applications. We call this problem Cost-Effective Conceptual Design (CECD).

5.2 RELATED WORKS

Researchers have examined the problem of selecting cost effective designs from an unorganized set of concepts for annotation [25]. Nevertheless, the concepts in most real-world domains are maintained in taxonomies rather than unorganized sets. As the framework proposed in [25] does not consider superclass/subclass relationships between concepts, it cannot measure the effectiveness gained by designs over taxonomies. Because taxonomies have richer structures than unorganized sets of concepts, they provide new opportunities and challenges for finding cost-effective conceptual designs. We show in Section 5.3.3 that because the algorithms in [25] do not consider the structure of taxonomy, they select the designs that provide very ineffective answers. Our empirical results over real-world datasets in Section 5.9 also indicate that the algorithms in [25] deliver considerably less effective designs than the ones that take the structure of the taxonomy into account. Furthermore, the authors in [25] do not consider the cost dependencies between concepts. They also assume that each query refers to only a single concept.

There is a large body of work on building large-scale data management systems for annotating and extracting entities and relationships from unstructured data sources [23, 85, 94–97]. In particular, researchers have proposed several techniques to optimize the running time and/or required computational resources of concept annotation programs by processing only a subset of the underlying collection that is more likely to contain mentions to entities of a given concept [23, 24, 90, 98–100]. Our work complements these efforts by finding a cost-effective set of concepts in the design phase rather than a set of relevant documents in query time. Furthermore, our framework can handle other types of costs than computational overheads.

Taxonomies and ontologies have been used in some areas of data management, such as data integration and query refinement [101–103]. We extend this line of research by using taxonomies in schema design. Researchers have proposed methods to semi-automatically construct or expand taxonomies by discovering new concepts from large text collections [104]. We, however, focus on the problem of annotating instances of the concepts in a given taxonomy over an unstructured or semi-structured data set.

Researchers have considered selecting data sources for fusion such that the marginal cost of acquiring a data source does not exceed its marginal gain [22]. We, however, focus on finding cost-effective designs.

Similar to previous works, we do not rigorously define the notion of named entity [81]. We define a named entity (entity for short) as a unique name in some (possibly infinite) domain. A concept is a set of entities , i.e., its instances. We identify each concept with a unique name. Some examples of concepts are *person* and *country*. An entity of concept *person* is *Albert Einstein* and an entity of concept *country* is *Jordan*. Concept C is a *subclass* of concept D if and only if we have $C \subset D$. In this case, we call D a *superclass* of C. For example, *person* is a superclass of *scientist*. If an entity belongs to a concept C, it will belong to all its superclass's.

A taxonomy organizes concepts in a domain of interest [81]. We first investigate the properties of tree-shaped taxonomies and later in Section 5.8, we will explore the taxonomies that are directed acyclic graphs. Formally, we define *taxonomy* $\mathcal{X} = (R, \mathcal{C}, \mathcal{R})$ as a rooted tree, with a *root concept* R, a vertex set $\mathcal{C}$ and an edge set $\mathcal{R}$. $\mathcal{C}$ is a finite set of concepts. For $C, D \in \mathcal{C}$, we have $(C, D) \in \mathcal{R}$ if and only if D is a subclass of C. Every concept in $\mathcal{C}$ that is not a superclass of any other concept in $\mathcal{C}$ is a *leaf* concept. The leaf concepts are leaf nodes in taxonomy $\mathcal{X}$. For instance, concepts *Trachoma* and *Hordeolum* are leaf concepts in Figure 5.4. Let $ch(C)$ denote the children of concept C. For the sake of simplicity, we assume that $\cup_{D \in ch(C)} D = C$ for all concepts C in a taxonomy.

Each dataset is a set of documents. Dataset DS is in the domain of taxonomy $\mathcal{X}$ if and only if some entities of concepts in $\mathcal{X}$ appear in some documents in DS. For instance, the set of documents in Figure 5.1 is in the domain of the taxonomy shown in Figure 5.4. An entity in $\mathcal{X}$ may appear in several documents in a dataset. For brevity, we refer to the occurrences of entities of a concept in a dataset as the occurrences of the concept in the dataset.

Each dataset is a set of documents. Dataset DS is in the domain of taxonomy $\mathcal{X}$ if and only if some entities of concepts in $\mathcal{X}$ appear in some documents in DS. For instance, the set of documents in Figure 5.1 is in the domain of the taxonomy shown in Figure 5.4. An entity in $\mathcal{X}$ may appear in several documents in a dataset. For brevity, we refer to the occurrences of entities of a concept in a dataset as the occurrences of the concept in the dataset.

A query q over DS is a pair (C, T), where $C \in \mathcal{C}$ and T is a set of terms. Some example queries are *(person,* $\{$*Michael Jordan*$\}$*)* or *(location,* $\{$*Jordan*$\}$*)*. This type of queries has been widely used to search and explore annotated datasets [105–107]. Query (C, T) over dataset DS is answered by a function that maps T to a ranked list of documents such that each document in the list contains an occurrence of an entity in C. One may use any reasonable ranking function to answer the query [60]. Empirical studies on real world query logs indicate that the majority of entity centric queries refer to a single entity [108].

We first consider queries that refer to a single entity and extend our model to support queries with multiple concepts in Section 5.7.

5.3.1 Conceptual Design

A *conceptual design* $\mathcal{S}$ over taxonomy $\mathcal{X} = (R, \mathcal{C}, \mathcal{R})$ is a non-empty subset of $\mathcal{C} \setminus \{R\}$. We exclude the root of $\mathcal{X}$ from the designs over $\mathcal{X}$ for simplicity. Other settings are converted to this case by adding a dummy root to the taxonomy. For brevity, we refer to conceptual design as *design*.

A design divides the set of leaf nodes in $\mathcal{C}$ into some partitions. As we have illustrated in Section 5.1, annotating a concept in a dataset may potentially help answering queries over the dataset whose entities belong to the leaf concepts that are descendants of the annotated concept. Consider again the fragment of MeSH taxonomy shown in Figure 5.4. Assume a user wants to find information about the types of pain associated with Trachoma over a dataset and submits query *(Trachoma,* $\{$*pain*$\}$*)* to the query interface. Suppose we have annotated the dataset with the concept in the design $\{$*Eye-Infections*$\}$. Because the articles about *Trachoma* are within the ones annotated by *Eye-Infections*, the query interface may return only articles annotated with *Eye-Infections* to the user. This annotation helps the query interface not to return the non-relevant articles about skin or bone infections. We say that *Trachoma* is in the *partition* of *Eye-Infections*. Now, suppose we annotate the dataset with the concepts in the design $\{$*Bacterial Conjunctivitis*, *Eye-Infections*$\}$. The articles annotated by *Eye-Infections* and *not* annotated by *Bacterial Conjunctivitis* do *not* contain any information about the concept *Trachoma*. Hence, the query interface may return only articles annotated with *Bacterial Conjunctivitis* in the response of the query *(Trachoma,* $\{$*pain*$\}$*)*. Due to the annotation of *Bacterial Conjunctivitis*, annotating *Eye-Infections* does not help answering queries with concept *Trachoma*. Thus, *Trachoma* is in the partition of *Bacterial Conjunctivitis* for this design. We formally define the *partition* of a concept in a design as follows.

Definition 5.1. *Let* $\mathcal{S}$ *be a design over taxonomy* $\mathcal{X} = (R, \mathcal{C}, \mathcal{R})$*, and let* $C \in \mathcal{S}$*. The* **partition** *of* C*, denoted as* $\texttt{part}(C)$*, is a maximal subset of leaf nodes in* $\mathcal{C}$ *such that, for all* $D \in \texttt{part}(C)$*, we have either* $D = C$ *or* C *is the lowest ancestor of* D *in* $\mathcal{S}$*.*

We denote a set of all partitions induced by all concepts in a design $\mathcal{S}$ as $\texttt{part}(\mathcal{S})$.

Example 5.1. *Consider the taxonomy described in Figure 5.5. Let the design* $\mathcal{S}_1$ *be* $\{$*Eye-Infections, Skin-Infections*$\}$*. The lowest ancestor in* $\mathcal{S}_1$ *of Neonatorum, Trachoma and Horde-*

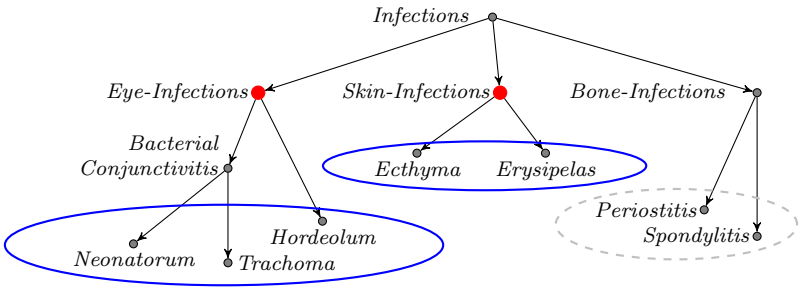


Figure 5.5: The concepts in red, *Eye-Infections* and *Skin-Infections*, denote the design. The blue curves denote the partitions created after annotating the design, and the dashed curved shows the free concepts of the selected design.

olum is Eye-Infections, and the lowest ancestor in $\mathcal{S}_1$ *of Ecthyma and Erysipelas is Skin-Infections. Hence, we have that* $\texttt{part}(\textit{Eye-Infections}) = \{\textit{Neonatorum, Trachoma, Hordeolum}\}$ *and* $\texttt{part}(\textit{Skin-Infection}) = \{\textit{Ecthyma, Erysipelas}\}$.

Example 5.2. *Consider the taxonomy described in Figure 5.6. Let the design* $\mathcal{S}_2$ *be* $\{\textit{Eye-Infections, Bacterial Conjunctivitis}\}$. *The lowest ancestor in* $\mathcal{S}_2$ *of Neonatorum and Trachoma is Bacterial Conjunctivitis, and the lowest ancestor in* $\mathcal{S}_2$ *of Hordeolum is Eye-Infections. Hence, we have that* $\texttt{part}(\textit{Bacterial Conjunctivitis}) = \{\textit{Neonatorum, Trachoma}\}$ *and* $\texttt{part}(\textit{Eye-Infections}) = \{\textit{Hordelum}\}$.

For each design $\mathcal{S}$, the set of *leaf concepts* that do not belong to any partition are called *free concepts* and denoted as $\texttt{free}(\mathcal{S})$. These concepts neither belong to $\mathcal{S}$ nor are descendant of any concept in $\mathcal{S}$.

Example 5.3. *Consider the design* $\mathcal{S}_1 = \{\textit{Eye-Infections, Skin-Infections}\}$ *over the taxonomy described in Figure 5.5. The free concepts of* $\mathcal{S}_1$, $\texttt{free}(\mathcal{S}_1)$, *are* $\{\textit{Periostitis, Spondylitis}\}$ *as they do not belong to any partition of* $\mathcal{S}_1$.

Example 5.4. *Consider the design* $\mathcal{S}_2 = \{\textit{Eye-Infections, Infections}\}$ *over the taxonomy described in Figure 5.6. The free concepts of* $\mathcal{S}_2$, $\texttt{free}(\mathcal{S}_2)$, *are* $\{\textit{Spondylitis, Periostitis, Ecthyma, Erysipelas}\}$.

Let DS be a dataset in the domain of taxonomy $\mathcal{X} = (R, \mathcal{C}, \mathcal{R})$ and $\mathcal{S}$ be a design over $\mathcal{X}$. $\mathcal{S}$ is the design of dataset DS if and only if for each concept $C \in \mathcal{S}$, all occurrences of concepts in the partition of C are annotated by C. In this case, we say DS is an *instance* of $\mathcal{S}$. For example, consider the design $\mathcal{T} = \{\textit{Eye-Infections, Skin-Infections}\}$ over the taxonomy in Figure 5.4. The dataset in Figure 5.3 is an instance of $\mathcal{T}$ as all instances of concepts *Trachoma* and *Hordeolum* that belong to the partition of *Eye-Infections*, are annotated by

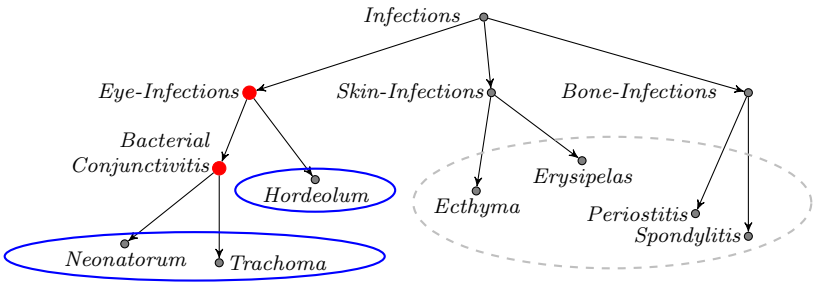


Figure 5.6: The concepts in red, *Eye-Infections* and *Bacterial Conjunctivitis*, denote the design. The blue curves denote the partitions created after annotating the design, and the dashed curved shows the free concepts of the selected design.

Eye-Infections and all instances of concepts *Ecthyma* and *Erysipelas* that are in the partition of *Skin-Infections*, are annotated by *Skin-Infections* in the dataset.

Intuitively, different designs may improve the effectiveness of answering queries differently. For example, consider a dataset in the domain of the taxonomy in Figure 5.5 in which almost all queries seek information about skin infections. If the query interface uses the design {*Eye-Infections, Skin-Infections*}, it can process queries that are about skin infection only over the documents that contain information about skin infections. But, if the query interface uses the design {*Eye-Infections, Bacterial Conjunctivitis*}, it has to process queries about skin infections over all documents in the dataset many of which do not contain any information about skin infections. Hence, the query interface may return more non-relevant answers for most queries than the case where it uses {*Eye-Infections, Skin-Infections*}. Moreover, as explained in Section 5.1, a design with more specific concepts, e.g., leafs in the taxonomy, helps the query interface to pinpoint the relevant documents more effectively than the designs with more general concepts. Because annotating documents and instances of different concepts may take different amounts of time and/or financial and computational resources, each design may have a distinct cost. Hence, finding the most effective design becomes an optimization problem that seeks the design that improves the effectiveness of answering queries the most and satisfies certain cost constraints. To formalize this problem, we first precisely quantify the amount by which a design improves the effectiveness of answering queries in Section 5.3.2. Then, in Section 5.3.3, we present a cost model for building and maintain annotations for a design and formally state the problem of finding the most effective design.

5.3.2 Design Queriability

Let $\mathcal{Q}$ be a set of queries over dataset DS. Given design $\mathcal{S}$ over taxonomy $\mathcal{X} = (R, \mathcal{C}, \mathcal{R})$, we would like to measure the degree by which $\mathcal{S}$ improves the effectiveness of answering

queries in $\mathcal{Q}$ over DS. The value of this function should be larger for the designs that help the query interface to answer a larger number of queries in $\mathcal{Q}$ more effectively. Let the query interface return k candidate answers for query Q in $\mathcal{Q}$ over the unannotated dataset. The effectiveness of the returned list of answers is usually measured using standard metrics of *precision* and *recall* [60]. The precision of the returned list of answers is the fraction of relevant answers for Q in the returned list. The recall of a returned list of answers is the ratio of the returned relevant answers to the number of total relevant answers for Q in the dataset. It has been shown that most information needs over annotated data sources are precision-oriented [106, 109]. Hence, we measure the effectiveness of the returned answers using precision-oriented metrics. More precisely, we use the standard metric of precision at k ($p@k$ for short), which is the precision of the top-k answers, to measure the ranking quality of the query result [60]. If a design helps the query interface to replace some non-relevant answers with relevant ones in the returned list for query Q, it improves the precision of Q in the top k returned answers. Hence, we estimate the amount by which a design increases the fraction of relevant answers in the top k returned answers for Q.

Let $Q : (C, T)$ be a query in $\mathcal{Q}$ such that C belongs to the partition of $P \in \mathcal{S}$. The query interface may consider only the documents that contain information about entities annotated by P to answer Q. For instance, consider query $Q_1 =$ *(Trachoma,* {*pain*}*)* over the dataset in Figure 5.3 whose design is {*Eye-Infections, Skin-Infections*}. The query interface may examine only the entities annotated by *Eye-Infections* in this dataset to answer Q_1. Thus, the query interface will avoid non-relevant results that otherwise may have been placed in the top k answers for Q. The query interface may further rank these answers according to a ranking function, such as the traditional TF-IDF scoring methods [60]. Our model is orthogonal to the ranking scheme of the candidate answers.

Nevertheless, only a fraction of documents with entities annotated by concept P contain information about entities in C. For instance, to answer query *(Trachoma,* {*pain*}*)* over the dataset in Figure 5.3, the query interface has to examine all documents that contain instances of concept *Eye-Infections*. Some documents in this set do not have any entity of concept *Trachoma*. We like to estimate the fraction of the results for $Q : (C, T)$ that contains entities of concept C. Given all other conditions are the same, the larger this fraction is, the more likely it is that the query interface delivers more relevant answers in the top k results for Q.

Let $d(C)$ denote the fraction of documents that contain entities of concept C in dataset DS. More precisely, $d(C)$ is the number of documents that contain entities of C in DS divided by the total number of documents in DS. We call $d(C)$ the *frequency* of C over DS. Let $d(P)$ be the total frequency of concepts in the partition of P, i.e., $d(P) = \sum_{C \in \texttt{part}(P)} d(C)$.

The fraction of the documents that contain information about entities in C amongst those that contain information of concept P is $\frac{d(C)}{d(P)}$. For example, assume that the mentions to entities of concept *Trachoma* appear more frequently in dataset DS than the ones of concept *Hordeolum*. Also, assume that we annotate only *Eye-Infections* in DS. Given query *(Hordeolum,* $\{pain\}$*)*, it is more likely for articles about Trachoma to appear in the top k answers than the ones about Hordeolum. That is, for each concept $C \in \texttt{part}(P)$, the contribution of C in improving the precision of answering queries with concepts C is $\frac{d(C)}{d(P)}$. Hence, the total contribution of partition P in improving the precision of answering queries is $\sum_{C \in \texttt{part}(P)} \frac{d(C)}{d(P)}$.

We call the fraction of queries in $\mathcal{Q}$ whose concept is C the *popularity* of C in $\mathcal{Q}$. Let $u_{\mathcal{Q}}$ be the function that maps concept C to its popularity in $\mathcal{Q}$. When $\mathcal{Q}$ is clear from the context, we simply use u instead of $u_{\mathcal{Q}}$. Assume that design $\mathcal{S}_1$ and $\mathcal{S}_2$ equally improve the values of precision for queries of all concepts except for $C_1, C_2 \in \mathcal{C}$. Also, let the precision of C_1 be improved more by $\mathcal{S}_1$ than by $\mathcal{S}_2$. Similarly, assume that the precision of C_2 is increased more by $\mathcal{S}_2$ than by $\mathcal{S}_1$. Given all other conditions are the same, if we have $u(C_1) > u(C_2)$, we have that $\mathcal{S}_1$ improves the total precision of queries in $\mathcal{Q}$ more than $\mathcal{S}_2$. Hence, we should take into account the popularities of concepts to compute the amount of improvement achieved by a design over $\mathcal{Q}$. Therefore, we modify the formula to estimate the contribution of partition P in improving precision of answering queries in $\mathcal{Q}$ as $\sum_{C \in \texttt{part}(P)} \frac{u(C)d(C)}{d(P)}$. Given all other conditions are the same, the larger this value is, the more likely it is that the query interface will achieve a larger precision in top k answers over queries in $\mathcal{Q}$.

Annotators may make mistakes in identifying the correct concepts of entities in a collection [91]. An annotator may recognize some instances of concepts that are not P as ones in P. For example, the annotator of concept *person* may identify *Lincoln*, the movie, as a person. The *accuracy* of annotating concept P over DS is the number of correct annotations of P divided by the number of all annotations of P in DS. We denote the accuracy of annotating concept P over DS as $\texttt{pr}_{DS}(P)$. When DS is clear from the context, we show $\texttt{pr}_{DS}(P)$ as $\texttt{pr}(P)$. Hence, we refine our estimate to

$$\sum_{C \in \texttt{part}(P)} \frac{u(C)\ d(C)}{d(P)}\ \texttt{pr}(P). \tag{5.1}$$

So far, we have estimated the relative improvement gained by $\mathcal{S}$ for queries whose concepts belong to some partitions in $\mathcal{S}$. Consider query $Q : (C, T)$ such that C does not belong to any partition in $\mathcal{S}$, i.e., C is a free concept. The query interface has to examine all documents in the dataset to answer Q. Thus, the fraction of returned answers for Q that contains some

instance of C is $d(C)$. The more instances of C appear in the dataset DS, the more likely it is that the returned answers to Q refer to entities in C. Hence, it is more likely that they contain some relevant answers for Q. Using a similar argument as the one used for non-free concepts, the total contribution of the free concepts of design $\mathcal{S}$ is

$$\sum_{C \in \texttt{free}(\mathcal{S})} u(C)d(C). \tag{5.2}$$

Following Formulas 5.1 and 5.2, we define the function that estimates the relative improvement in the value of precision in the top k answers for all concepts as follows.

Definition 5.2. *The queriability of design $\mathcal{S}$ from taxonomy $\mathcal{X}$ over dataset DS and query workload $\mathcal{Q}$ is*

$$QU(\mathcal{S}) = \sum_{P \in \mathcal{S}} \sum_{C \in \mathit{part}(P)} \frac{u(C)\ d(C)\ \mathit{pr}(P)}{d(P)} + \sum_{C \in \mathit{free}(\mathcal{S})} u(C)d(C).$$

Similar to other optimization problems in data management, such as query optimization [43], the complete information about the parameters of the objective function, i.e. frequencies and popularities of concepts, may not be available at the design-time. Nevertheless, our empirical results in Section 5.9 indicate that one can effectively estimate these parameters using a small sample of the full dataset. For instance, we show that the frequencies of concepts over a dataset of more than a million documents can be effectively estimated using a sample of about four hundred documents. In this work, we assume that a random sample of the dataset represents the relative frequencies of the concepts in the dataset reasonably accurately. A more principled approach to parameter estimation is an interesting subject for future work.

5.3.3 Cost-Effective Conceptual Design Problem

We have reviewed the literature on concept annotation and information extraction and talked to the experts to build a reasonable and general cost model for concept annotation. The types of costs for creating annotated datasets vary based on the methodology used for developing concept annotators. One may categorize the available methodologies to rule-based methods and approaches based on machine-learning techniques [88, 91, 95, 110–114]. In rule-based annotation, developers write a set of rules for each concept to detect and extract its instances in a dataset [97, 115, 116]. Rule-based approach is the dominating method in commercial information and entity extraction systems [117, 118]. This is mainly

attributed to the fact that rules are effective, interpretable, and easier to customize by non-experts than the methods based on machine learning [118]. Furthermore, rules-based systems perform better than state-of-the-art machine learning methods in some specialized domains [119, 120].

If one adapts a machine-learning approach to annotate the entities of a concept, he has to provide a set of training examples for the concept, which may be costly and time-consuming [118]. This stage is particularly resource-intensive in specific domains, such as medicine. Researchers have proposed the idea of distant supervision to reduce the overhead of providing training data for concept extraction [121]. However, distant supervision typically requires knowledge-bases in the domain of extraction with instances of the extracted concepts which is not always available, particularly in a specific domain such as biology. One may also generate training data for a concept by coding how the concept appears in the unstructured dataset in a programming language [122]. This method, however, needs a domain expert to learn a programming language and code her knowledge in a piece of program. Furthermore, the developer has to distinguish and select relevant features for each concept because many or all concepts may share some general features. For example, concept annotators may use the surrounding words of entities in text documents as features for all concepts. However, developers have to also engineer considerable number of features specific to each concept [123, 124]. For example, an informative feature for *zip-code* is that its instances have 5 digits, and a helpful feature for *person* is that its entities start with a capital letter. These features may be given to a classifier [123] or a probabilistic graphical model [125] or coded as first order logic formulas in a Markov Logic Network [124]. After developing the concept annotator, domain experts may review and evaluate the annotation of each concept [82, 83]. This process may repeat multiple times to generate accurate annotations of the concept. As most underlying datasets frequently evolve, the aforementioned steps have to be redone after a while for each concept in both approaches [87, 126].

Hence, given taxonomy $\mathcal{X} = (R, \mathcal{C}, \mathcal{R})$ and dataset DS, one may assign a real number to each concept $C \in \mathcal{C}$ that reflects the amount of resources required to annotate and maintain the annotations of C in DS. Let function $w : \mathcal{C} \to \mathbb{R}^+$ map each concept $C \in \mathcal{C}$ to a real number that reflects the cost of annotating C in DS. Developers also create and maintain some preprocessing modules to tokenize the input documents and separate the (potential) named entities from other tokens, e.g., adjectives, in both rule-based method and the methods based on machine learning techniques. This cost is generally independent of the number of extracted concepts and can be viewed as a fixed cost for annotating a dataset. This model assumes that annotating certain concepts does not affect the cost and accuracies of annotating other concepts. We later relax this restriction in Section 5.6 and consider

some dependencies between the costs of concepts in the taxonomy. Nevertheless, it usually takes significant amount of resources to develop and maintain a concept annotator even after pairing it with other annotators. Thus, it is still of interest to solve the problem where there is no dependency between costs of different concepts. The organization may predict the costs of development, deployment and maintenance of annotation programs using available methods for predicting costs of software development and maintenance [127].

The cost of annotating a dataset under design $\mathcal{S}$ is the sum of the costs the concepts in $\mathcal{S}$. Budget B is a positive real number that represents the amount of available resources for annotating the dataset. We define Cost-Effective Conceptual Design problem (CECD) as follows.

Problem 5.1. *Given taxonomy $\mathcal{X}$, dataset DS in the domain of $\mathcal{X}$, query workload $\mathcal{Q}$ and budget B, find design $\mathcal{S}$ over $\mathcal{X}$ that has the maximum queriability and $\sum_{C \in \mathcal{S}} w(C) \leq B$.*

Unfortunately, the CECD problem cannot be solved in polynomial time in terms of input size unless $\mathbf{P} = \mathbf{NP}$.

Theorem 5.1. *The problem of* CECD *is* **NP***-hard.*

Proof. The problem of CECD can be reduced to the problem of choosing a cost-effective design from a set of concepts by creating a taxonomy $\mathcal{X} = (R, \mathcal{C}, \mathcal{R})$ where all nodes except for R are leaf concepts. Since the problem of choosing cost-effective concepts from a set of concepts is **NP**-hard [25], CECD is also **NP**-hard.

Because the approximation algorithms proposed in [25] do not consider the superclass/subclass relationships between concepts, they do not effectively solve the CECD problem for tree taxonomies. In particular, these algorithms generally choose designs with more popular concepts, i.e., concepts with larger $u(.)$ values. Because each node has generally more $u(.)$ value than its descendant concepts in the input taxonomy, these algorithms spend the budget on picking concepts in higher levels, while it may worth including concepts in lower levels of the taxonomy in the design. Our empirical studies in Section 5.9 confirm that these algorithms do not generally find accurate solutions to CECD over tree taxonomies. We also show that the algorithms in [25] have a worst-case approximation ratio of $O(|\mathcal{C}|)$ for the problem, which is a significantly inaccurate approximation even for taxonomies with a modest number of concepts, as follows.

Consider the taxonomy shown in Figure 5.7 where each green leaf concept C_g has $w(C_g) = 1$, $u(C_g) = (1 + \epsilon)v$ and $d(C_g) = p$. For each red leaf concept C_r, $w(C_r) = B + 1$, $u(C_r) = Bv$ and $d(C_r) = \frac{1}{B^2}p$. For the orange leaf concept C_0, $w(C_o) = 1$, $u(C_o) = v$ and $d(C_o)$

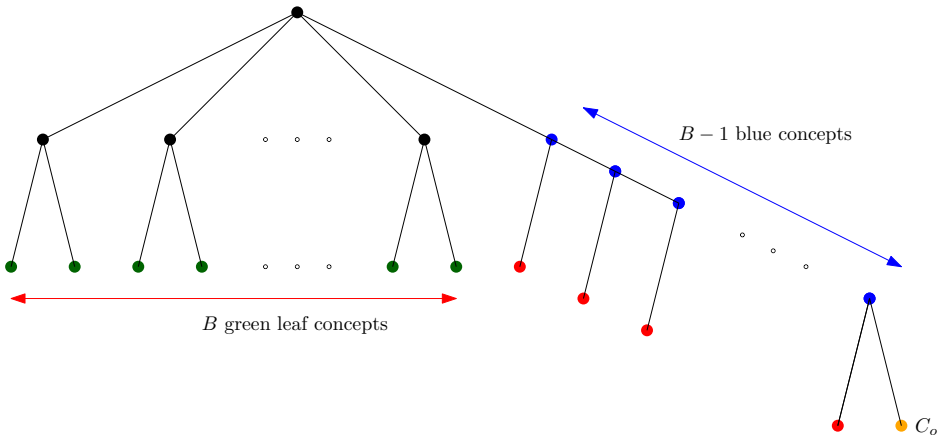


Figure 5.7: An example to analyze algorithms that ignore the tree structure.

$= p$. v and p are constants such that $\sum_{C\in\mathcal{C}} u(C) = \sum_{C\in\mathcal{C}} d(C) = 1$. For each non-root black concept C, $w(C) = B$ and for each blue concept C_b, $w(C_b) = 1$. Assume that the total available budget is B. The optimal solution of any algorithm that ignores the tree structure and works only with leaf concepts, is to pick exactly the green leaf nodes, which delivers the queriability of almost $B(1+\epsilon)v$. Now, assume that an algorithm considers both leaf and non-leaf concepts but not their relationships. For instance, the approximate popularity maximization (APM) algorithm in [25] selects concepts C with the largest $\frac{u(C)}{w(C)}$ values. APM picks exactly B green leaf concepts in our example and returns queriability of almost $B(1+\epsilon)v$. However, we can achieve the queriability of almost B^2v by picking all blue and orange concepts. The gap between optimal solution and any solution returned by an APM algorithm is B, which is $\frac{1}{3.5}|\mathcal{C}|$. It is an ineffective solution for taxonomies with modest numbers of concepts.

5.4 APPROXIMATION ALGORITHM

We propose an approximation algorithm called LEVEL-WISE algorithm to solve the problem of CECD using a greedy approach. It returns a design whose concepts are all from a same level of the input taxonomy. Our algorithm finds the design with maximum queriability for each level using the APM algorithm proposed in [25], which find the cost-effective subset of concepts over a set of concepts. Our algorithm eventually delivers the design with largest queriability across all levels in the taxonomy. Algorithm 5.1 illustrates the LEVEL-WISE algorithm.

Precisely, let $\mathcal{C}[i]$ be the set of all concepts of level i in $\mathcal{X} = (R, \mathcal{C}, \mathcal{R})$. For any concept $E \in \mathcal{C}[i]$, we define its popularity $u_i(E)$ to be the total popularity of its descendant leaves

Algorithm 5.1: LEVEL-WISE algorithm.

input : a taxonomy $\mathcal{X} = (R, \mathcal{C}, \mathcal{R})$
output: a conceptual design

$\texttt{sol}_{\text{level}} \leftarrow 0$ and $\texttt{sol}_{\max} \leftarrow 0$;
for $i = 0$ *to* h **do**
  for each concept C in distance i from the root **do**
    $\mathcal{C}[i] \leftarrow \mathcal{C}[i] \cup \{C\}$;
  end
  for each leaf concept C in distance less than i from the root **do**
    $\mathcal{C}[i] \leftarrow \mathcal{C}[i] \cup \{C\}$;
  end
  $\texttt{sol}_i \leftarrow$ approximate solution over $\mathcal{C}[i]$;
  $\texttt{sol}_{\text{level}} \leftarrow \max(\texttt{sol}_{\text{level}}, \texttt{sol}_i)$;
end
$C_{\max} \leftarrow$ the leaf concept with the largest u value;
$\texttt{sol}_{\max} \leftarrow u(\mathcal{C}_{\max}) + \sum_{C \in \texttt{free}(C_{\max})} u(C)d(C)$;
return the best of $\texttt{sol}_{\text{level}}$ and $\texttt{sol}_{\max}$;

in $\mathcal{X}$. LEVEL-WISE algorithm calls the APM algorithm to find the cost-effective subset of concepts for every $\mathcal{C}[i]$. It provides APM with the popularities and costs of concept in $\mathcal{C}[i]$. LEVEL-WISE algorithm then compares various selected designs across $\mathcal{C}[i]$s and keeps the answer with maximum queriability, denoted as $\text{SOL}_{\texttt{level}}$. The algorithm also computes the queriability delivered by picking only the leaf node with maximum popularity in $\mathcal{X}$ called $\text{SOL}_{\texttt{max}}$. The algorithm returns the solution with largest queriability amongst $\text{SOL}_{\texttt{level}}$ and $\text{SOL}_{\texttt{max}}$. The APM algorithm runs in $O(|\mathcal{C}| \log |\mathcal{C}|)$, where $|\mathcal{C}|$ is the size of its input set of concepts. Thus, the time complexity of the LEVEL-WISE algorithm is $O(h|\mathcal{C}| \log |\mathcal{C}|)$ over taxonomy $\mathcal{X} = (R, \mathcal{C}, \mathcal{R})$, where h is the number of levels in $\mathcal{X}$. If the taxonomy is not balanced, the popularities of all concepts of level i may not sum up to 1. Hence, the algorithm does not consider leaf concepts that are not descendant of any concept in $\mathcal{C}[i]$. To resolve this problem, when running APM algorithm over $\mathcal{C}[i]$, we consider leaf concepts that are not descendant of any concept of level i as members of $\mathcal{C}[i]$ (lines 6-8 in Algorithm 5.1).

Sometimes, it may be easier to use and manage designs whose concepts are not subclass/ superclass of each other. We call such a design a *disjoint design*. One may restrict the feasible solutions in the CECD problem to disjoint designs. Our empirical results in Section 5.9 show that this strategy returns effective designs when the available budget is relatively small. We call this case of CECD, *disjoint* CECD. Recent empirical results suggest that the distribution of concept frequencies over a large collection generally follows a *power law* distribution [85]. We show that the LEVEL-WISE algorithm has a bounded and reasonably small worst-

case approximation ratio for CECD with disjoint solution given that the distribution of concept frequencies follows a power law distribution.

Lemma 5.1. *Let $C_{\max}$ be the leaf concept in taxonomy $\mathcal{X} = (R, \mathcal{C}, \mathcal{R})$ with maximum u value and let assume that distribution of u over leaf concepts follows a* power law *distribution. For every schema $\mathcal{S}$ of $\mathcal{X}$, $QU(\texttt{free}(\mathcal{S})) \leq 2u(\mathcal{C}_{\max}) \log |\mathcal{C}|$.*

Proof. We have

$$\sum_{C \in \texttt{free}(\mathcal{S})} u(C)d(C) \leq u(C_{\max}) \sum_{C \in \texttt{free}(\mathcal{S})} d(C).$$

Since the frequencies of leaf concepts in $\mathcal{X}$ follow a power-law distribution, we have that

$$\sum_{C \in \texttt{leaf}(\mathcal{C})} d(C) \leq 1 + \log(|\texttt{leaf}(\mathcal{C})|)$$

where $\texttt{leaf}(\mathcal{C})$ is the set leaf concepts in $\mathcal{C}$ and $|\texttt{leaf}(\mathcal{C})|$ is the number of such concepts. Since $|\texttt{leaf}(\mathcal{C})| \leq |\mathcal{C}|$, then

$$QU(\texttt{free}(\mathcal{S})) \leq \sum_{C \in \texttt{free}(\mathcal{S})} u(C)d(C) \leq (1 + \log |\mathcal{C}|)\ u(C_{\max}) \leq 2u(\mathcal{C}_{\max}) \log |\mathcal{C}|.$$

Theorem 5.2. *Let $\mathcal{X} = (R, \mathcal{C}, \mathcal{R})$ be a taxonomy with height h and the minimum accuracy of $\boldsymbol{pr}_{\min} = \min_{C \in \mathcal{C}} \boldsymbol{pr}(C)$. The* Level-wise *algorithm is a $O(\frac{h + \log |\mathcal{C}|}{\boldsymbol{pr}_{\min}})$-approximation for the* CECD *problem with disjoint solution on $\mathcal{X}$ and budget B given that the distribution of frequencies in $\mathcal{C}$ follows a power law distribution.*

Proof. Let $\mathcal{S}^*$ be a disjoint design over $\mathcal{X}$ with total cost at most B that maximizes QU function. Let $\mathcal{S}^*[i]$ be the set of concepts in S^* of level i, i.e., $\mathcal{S}^*[i] = \mathcal{S}^* \cap \mathcal{C}[i]$. Since $\mathcal{S}^*$ is a disjoint design, for all $1 \leq i, j \leq h$, $\texttt{part}(\mathcal{S}^*[i]) \cap \texttt{part}(\mathcal{S}^*[j]) = \emptyset$. It follows that $QU(\mathcal{S}^*) = \sum_{1 \leq i \leq h} QU(\mathcal{S}^*[i]) + QU(\texttt{free}(\mathcal{S}^*))$. Next, we consider the two possible cases. First, assume that $\sum_{i=1}^{h} QU(\mathcal{S}^*[i]) \geq QU(\texttt{free}(\mathcal{S}^*))$. It follows that the Level-wise algorithm returns a $(\frac{2h}{\texttt{pr}_{\min}})$-approximate solution of the disjoint CECD defined over $\mathcal{X}$.

In the other case in which $QU(\texttt{free}(\mathcal{S}^*))$ is larger than the other term, by Lemma 5.1, extracting the concept with the maximum u value gives a $(\frac{4 \log(|\mathcal{C}|)}{\texttt{pr}_{\min}})$-approximation. These two cases together imply that we have an $O(\frac{h + \log |\mathcal{C}|}{\texttt{pr}_{\min}})$-approximation.

Because concept annotation algorithms are reasonably accurate, $\texttt{pr}_{\min}$ is generally close to one [88, 91, 128].

5.5 EXACT ALGORITHM

This section describes an exact pseudo-polynomial time dynamic programming algorithm for the CECD over taxonomy $\mathcal{X} = (R, \mathcal{C}, \mathcal{R})$. We assume that $u(C)$, $d(C)$, and $w(C)$ are positive integers for each $C \in \mathcal{C}$. In Section 5.9, we show that the algorithm can handle real values with scaling techniques at the expense of reporting near optimal solution instead of an optimal one. We refer to the second part of the queriability in Definition 5.2 as *free partition*. Given a concept $C \in \mathcal{X}$, the subtree rooted at C in $\mathcal{X}$ is denoted as $\mathcal{X}_C$, and the set of the children of C in $\mathcal{X}$ is denoted as $\texttt{child}(C)$. Let $Q[C, B, N]$ denote the maximum queriability over all designs in $\mathcal{X}_C$ such that the amount of queriability that these designs obtain from their free parts are exactly N, and they are selected by spending at most B units of the budget. Our algorithm computes $Q[C, B, N]$ for all $C \in \mathcal{X}$, all integers $0 \leq B \leq B_{\texttt{total}}$ and all integers $0 \leq N \leq N_{\texttt{total}}$, where $N_{\texttt{total}} = \sum_{C \in \texttt{leaf}(\mathcal{C})} u(C)d(C)$ and $\texttt{leaf}(\mathcal{C})$ is the set leaf nodes in $\mathcal{C}$. Our algorithm returns

$$\max_{0 \leq N \leq N_{\texttt{total}}} (Q[R, B_{\texttt{total}}, N] + N) \tag{5.3}$$

where R is the root of $\mathcal{X}$ and $B_{\texttt{total}}$ is the total budget. We try all possible queriabilities that one can obtain from the free part in $\mathcal{X}$ and N, to find out the maximum queriability over all designs in $\mathcal{X}$.

For a non-leaf concept C, we obtain a recursive description for $Q[C, B, N]$ according to the value of Q for the children of C. We consider the following two cases for C:

C is in the optimal design: If C belongs to the optimal design in $\mathcal{X}_C$, we have $N = 0$. This is because if C is picked, every concept in $\mathcal{X}_C$ belongs to a non-free partition in the selected design. In this case, we spend $w(C)$ of the budget to annotate C and the leftover budget $BL = B - w(C)$ should be assigned to the $\texttt{child}(C)$. Our algorithm tries all possible ways of assigning the leftover budget among the children of C. If C is selected, the total queriability obtained at C can be divided into the total non-free queriability of its children and the queriability of the partition of C. Consider the leaf nodes in $\mathcal{X}_C$ that belong to the free partitions in the subtrees rooted at the children of C. If C is selected, these nodes will be in the partition of C. Thus, the queriability of the partition of C is the total free queriability of the children of C times $\texttt{pr}(C)/d(C)$. Our algorithm tries all possible total free queriabilities for the children of C as well as all possible assignment of this free queriability to the set $\texttt{child}(C)$. Thus, we obtain the following recursion. We use Q_I to indicate the

inclusion of C in the solution.

$$Q_I[C,B,N] = \max_{\mathcal{B},\mathcal{N}}\Big(\sum_{\texttt{Ch}\in\texttt{child}(C)} Q[\texttt{Ch},\mathcal{B}(\texttt{Ch}),\mathcal{N}(\texttt{Ch})] + \frac{\texttt{pr}(C)}{d(C)}\sum_{\texttt{Ch}\in\texttt{child}(C)}\mathcal{N}(\texttt{Ch})\Big). \tag{5.4}$$

$\mathcal{B}$ includes all possible assignments of BL units of the budget to the children of C, and $\mathcal{N}$ includes all possible assignments of at most $N_{\texttt{total}}$ queriability to the children of C. If $N \neq 0$, we set $Q_I[C,B,N] = -\infty$ to indicate the infeasibility of a the case.

C **is not in the optimal design.** In this case, we want to obtain exactly N units of free queriability, while we maximize the total queriability. Our algorithm tries all possible ways of assigning this N units into the children of C. Since C is not picked, the entire budget B can be spent on the subtrees rooted at the children of C. Hence, our algorithm tries all possible assignments of this B units of the budget among the children of C. Thus, we obtain the following recursions. We use Q_E to indicate the exclusion of C from the solution.

$$Q_E[C,B,N] = \max_{\mathcal{B},\mathcal{N}}\sum_{Ch\in\texttt{child}(C)} Q[Ch,\mathcal{B}(Ch),\mathcal{N}(Ch)] \tag{5.5}$$

$\mathcal{B}$ includes all possible assignments of B units of budget among the children of C, and $\mathcal{N}$ includes all possible assignments of exactly N unit of benefit from the queriability of free partitions among the children of C.

Our algorithm considers both cases above and returns the maximum queriability that can be obtained by either including or excluding C from the optimal solution:

$$Q[C,B,N] = \max(Q_I[C,B,N], Q_E[C,B,N]). \tag{5.6}$$

The leaf concepts are the base cases for our recursion. Let C be a leaf concept, and suppose that we want to optimize $Q[C,B,N]$. If $N = 0$, we must select C. Otherwise, we cannot select C. Thus, there are two cases to consider:
$\mathbf{N \neq 0}$: If $N = u(C)d(C)$, we set $Q[C,B,N] = 0$. Otherwise, we set $Q[C,B,N]$ to $-\infty$.
$\mathbf{N = 0}$: If $B \geq w(C)$, we set $Q[C,B,N] = \texttt{pr}(C)u(C)$. Otherwise, we set $Q[C,B,N]$ to $-\infty$.

This completes the explanation of our recursive algorithm. We turn this recursive algorithm into a dynamic programming to avoid redundant computations. To this end, we build a $|\mathcal{C}| \times B_{\texttt{total}} \times N_{\texttt{total}}$ table, and fill it according to our recursions. The running time of such a dynamic programming would be $O(|\mathcal{C}|B_{\texttt{total}}N_{\texttt{total}}) \cdot T_{\texttt{cell}}$, where $T_{\texttt{cell}}$ is the maximum time required to compute the value of a single cell disregarding recursive calls. $T_{\texttt{cell}}$ exponentially depends on the maximum degree of concepts in the taxonomy. Hence, as detailed below, we

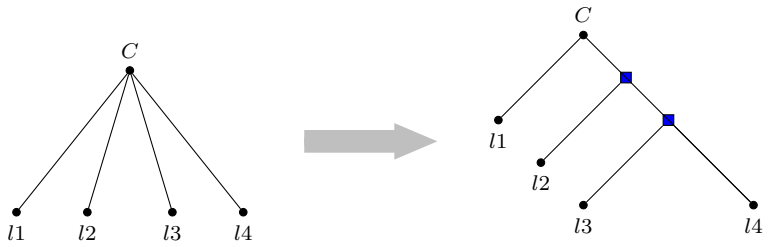


Figure 5.8: Transforming an input taxonomy. Dummy nodes are in blue squares.

further optimize the running time of the algorithm by adding a preprocessing step to modify our input taxonomy to a binary tree (i.e. the maximum number of children of each concept is two). Assuming that our input taxonomy is a binary tree, for each $Q[C, B, N]$, we have $O(B_{\texttt{total}})$ ways of dividing B and $O(N_{\texttt{total}})$ ways of dividing the expected free queriability between the children of C. So, $T_{\texttt{cell}} = O(B_{\texttt{total}} N_{\texttt{total}})$. Let $D = \sum_{C \in \texttt{leaf}(\mathcal{C})} d(C)$ and $U = \sum_{C \in \texttt{leaf}(\mathcal{C})} u(C)$. By Cauchy-Schwartz inequality, we have $N_{\texttt{total}} \leq UD$. We conclude that the running time is $O(|\mathcal{C}|(B_{\texttt{total}} N_{\texttt{total}})^2) = O(|\mathcal{C}|(B_{\texttt{total}} UD)^2)$.

5.5.1 Transforming a taxonomy into binary taxonomies

We explain how to transform an arbitrary taxonomy to a binary taxonomy. Let C be a non-leaf concept in $\mathcal{X}$. We replace the induced subtree of $C \cup \texttt{child}(C)$ with a full binary tree $\mathcal{X}'_C$ whose root is C and whose leaves are $\texttt{child}(C)$ as shown in Figure 5.8. Some internal nodes of $\mathcal{X}'_C$ do not correspond to any node in $\mathcal{X}$. We refer to such internal nodes as *dummy nodes*, and set their cost to $B_{\texttt{total}} + 1$ to make sure that our algorithm does not include them in the output design.

Applying the aforementioned transformation to all nodes of $\mathcal{X}$, we obtain a binary taxonomy $\mathcal{X}' = (R, \mathcal{C}', \mathcal{R}')$. The number of nodes in $\mathcal{C}'$ is at most twice the number of nodes in $\mathcal{C}$. Therefore, we can find the optimal design for $\mathcal{X}'$ in the same asymptotic running time, $O(|\mathcal{C}|(B_{\texttt{total}} UD)^2)$. Since this transformation does not change the subset of leaf concepts in the subtree rooted in any internal node, any internal node in $\mathcal{X}$ corresponds to a solution in $\mathcal{X}'$ with the same cost and queriability. Since dummy nodes are too costly, they do not introduce any new solution to the set of feasible solutions.

Theorem 5.3. *There is an exact algorithm to solve the* CECD *problem over taxonomy* $\mathcal{X} = (R, \mathcal{C}, \mathcal{R})$ *with budget* B_{total} *in* $O(|\mathcal{C}| B_{total}^2 U^2 D^2)$.

Algorithm	Approximation ratio	Running time
Level-wise (Disjoint CECD)	$O((h + \log \lvert\mathcal{C}\rvert)/\mathtt{pr}_{\min})$	$O(h\lvert\mathcal{C}\rvert \log(\lvert\mathcal{C}\rvert))$
Dynamic Programming	Pseudo-polynomial	$O(\lvert\mathcal{C}\rvert B^2U^2D^2)$

Table 5.1: Algorithms for the CECD problem.

5.5.2 Scaling technique to handle real values

In some applications, u and d values are positive real number. Following the scaling techniques described in [129], we define

$$\hat{u}(C) = \left\lfloor \frac{u(C)}{\varepsilon u_{\max}/n} \right\rfloor \tag{5.7}$$

and

$$\hat{d}(C) = \left\lfloor \frac{d(C)}{\varepsilon d_{\max}/n} \right\rfloor \tag{5.8}$$

where $u_{\max}$ and $d_{\max}$ are the maximum popularity and frequency of all leaf concepts in the taxonomy, respectively. The accuracy of the dynamic programming algorithm depends on values of ε_d and ε_u. If these values are sufficiently small, the algorithm finds the optimal schema. Small values of these parameters, however, may result in large u and d values and increase the running time of the algorithm considerably. Hence, one may choose relatively larger values for ε_d and ε_u to obtain a design whose queriability is close to the one of the optimal design in a shorter time.

Table 5.1 presents a summary of proposed algorithms for the CECD problem.

5.6 CECD WITH COST DEPENDENCY

The cost of extracting a concept may vary if its ancestors in the taxonomy have been already annotated. For example, if the instances of *person* have been already annotated in a collection, the extractor of *artist* will examine only a subset of the collection that are annotated by concept *person*, which may result in a faster extraction. Also, the extractors for *artist* and *person* may share some annotation rules and/or features. We generalize the CECD problem so that the cost of annotating a concept in a taxonomy may change depending on whether its ancestors in the taxonomy are part of the design. One may think of more complex cost functions that represent dependencies between arbitrary concepts in the taxonomy. However, it requires a space and time exponential in terms of the number of concepts in the taxonomy to express and estimate such functions. One may reduce the amount of space

and effort to store and estimate the cost values by enforcing some restrictions. On the other hand, finding such restrictions requires extensive empirical studies and more space than a single paper and is a subject of future work.

Assume that multiple ancestors of a concept C in $\mathcal{X} = (R, \mathcal{C}, \mathcal{R})$ are in the selected design. The annotator of C may examine only the documents annotated by the closest ancestor of C in the design. Moreover, the closest ancestor of C may share more annotation rules/features with C than other ancestors of C. Hence, we consider only the dependency between cost of C and its closest selected ancestor in $\mathcal{X}$. More formally, for each concept C, let $\texttt{anc}(C)$ be a positive integer value indicating the number of edges between C and its closest ancestor in the design. We redefine the cost w as a real-valued function over pairs of concepts and positive integer values. The total cost of design S over $\mathcal{X}$ is $\sum_C w(C, \texttt{anc}(C))$. We define the problem of CECD *with cost dependency* exactly the same as Problem 5.1, by only replacing its cost function with the redefined cost function. Because CECD is a special instance of the problem of CECD with cost dependency, CECD with cost dependency is also **NP**-hard.

We extend the dynamic programming algorithm in Section 5.5 to design a pseudo-polynomial time algorithm for CECD with cost dependency. As before, we assume that the values of functions u, d and w are positive integers. We define $\mathcal{X}_C$ and $\texttt{leaf}()$ the same way that they are defined in Section 5.5. Let $Q[C, B, N, \texttt{anc}]$ denote the maximum queriability we can obtain constraint to budget B from the partitions in $\mathcal{X}_C$ such that the queriability of the free parts is N and $\texttt{anc}$ indicates the closest selected ancestor of C in the design. The algorithm computes values of Q for all budgets up to total budget $B_{\texttt{total}}$ and all free queriability N up to $N_{\texttt{total}} = \sum_{C \in \texttt{leaf}(\mathcal{C})} u(C)d(C)$ and returns

$$\max_{0 \leq N \leq N_{\texttt{total}}} (Q[R, B_{\texttt{total}}, N, 1] + N) \tag{5.9}$$

where R is the root of $\mathcal{X}$ and $B_{\texttt{total}}$ is the total budget. Note that for the root node, R, the exact value of $\texttt{anc}$ does not matter because R has no ancestor. We use the technique in Section 5.5 to transform the taxonomy to a binary tree. We extend the recursive formulas in Section 5.5 for $Q[C, B, N, \texttt{anc}]$ in the following two cases. Q_I and Q_E are defined similar to the ones in Section 5.5.

C is in the optimal design.

$$Q_I[C, B, N, \texttt{anc}] = \max_{\mathcal{B}, \mathcal{N}} \Big(\sum_{\texttt{Ch} \in \texttt{child}(C)} Q[\texttt{Ch}, \mathcal{B}(\texttt{Ch}), \mathcal{N}(\texttt{Ch}), 1] + \frac{\texttt{pr}(C)}{d(C)} \sum_{\texttt{Ch} \in \texttt{child}(C)} \mathcal{N}(\texttt{Ch}) \Big). \tag{5.10}$$

$\mathcal{B}$ includes all possible assignments of $B - w(C, \texttt{anc})$ units of the budget, and $\mathcal{N}$ includes all possible assignments of at most N free queriability between children of C. Because concept

C is in the optimal design, it resets the value of *anc* to 1 for its children.

C **is not in the optimal design.** If C is *not* a dummy node:

$$Q_E[C, B, N, \texttt{anc}] = \max_{\mathcal{B},\mathcal{N}} \sum_{Ch \in \texttt{child}(C)} Q[Ch, \mathcal{B}(Ch), \mathcal{N}(Ch), \texttt{anc} + 1]. \tag{5.11}$$

Since C is not selected, it updates and transfers the information about its closest selected ancestor to its children. If C is a dummy node:

$$Q_E[C, B, N, \texttt{anc}] = \max_{\mathcal{B},\mathcal{N}} \sum_{Ch \in \texttt{child}(C)} Q[Ch, \mathcal{B}(Ch), \mathcal{N}(Ch), \texttt{anc}]. \tag{5.12}$$

The dummy nodes are not selected and do not change the value of `anc`. In these formulas, $\mathcal{B}$ includes all possible assignments of B units of the budget and $\mathcal{N}$ includes all possible assignments of exactly N free queriability between children of C.

The leaf concepts make the base cases for our recursion and their equations are similar to the bases cases in Section 5.5. Because the information about the closest selected ancestor of a leaf node is available in Q, the algorithm can compute the cost of leaf node.

Let L be the height of $\mathcal{X}$. To compute the running time of this algorithm, we need to give an upper bound on the number of cells in Q and the time required to compute the value for each cell. The time to compute a single cell in Q is exponential in terms of the maximum degree of the taxonomy, which is 2 after transforming $\mathcal{X}$ to a binary tree. Because we have $N \leq UD$, and the maximum value of `anc` is L, the time for computing all cells in Q will be $O(|\mathcal{C}|(B_{\texttt{total}}UD)^2L)$. Generally, L for most real-world taxonomy is considerably smaller than $|\mathcal{C}|$ and closer to $O(\log|\mathcal{C}|)$.

5.7 QUERIES WITH MULTIPLE CONCEPTS

In this section, we propose fast algorithms for the problem of CECD where queries may refer to multiple concepts. We call this problem, Multiple-Concept CECD (MC-CECD).

5.7.1 MC-CECD Over a Set of Concepts

We assume that each query refers to at most two different concepts. The choice of two is only for the sake of simplicity and our algorithm extends to any fixed number of concepts. We also assume that the annotation each (leaf) concept over a dataset is independent of the annotation of any other (leaf) concept in the dataset to simplify our analysis. Our empirical

studies in Section 5.9 indicate that generally this assumption does *not* significantly reduce the accuracy of our models and algorithms. Let $QU_n(\mathcal{S})$ be the queriability of $\mathcal{S}$ from a set of concepts $\mathcal{C}$ over queries with exactly n concepts. The queriability of $\mathcal{S}$ for queries with one concept, $QU_1(\mathcal{S})$, is $\sum_{C\in\mathcal{S}} u(C)\mathtt{pr}(C) + \sum_{C\in\mathtt{free}(\mathcal{S})} u(C)d(C)$ [25]. Next, we compute $QU_2(\mathcal{S})$. Assume that query q refers to entities of concepts C_1 and C_2. The query interface should select the answers to q from the set of documents that contain instances of both concepts. If the instances of both concepts are annotated in the dataset, the query interface will correctly identify such documents. If only C_1 is annotated, the query interface will return documents annotated as C_1, of which only about $d(C_2)$ may contain the desired answers. If none of these concepts are annotated, the query interface has to explore all documents in the collection of which only about $d(C_1 \cap C_2)$ may have the desired answers. We have:

$$\begin{aligned} QU_2(\mathcal{S}) = & \sum_{\substack{C_1\in\mathcal{S}\\ C_2\in\mathcal{S}}} u(C_1\cap C_2)\mathtt{pr}(C_1)\mathtt{pr}(C_2) \\ & + \sum_{\substack{C_1\in\mathcal{S}\\ C_2\in\mathtt{free}(\mathcal{S})}} u(C_1\cap C_2)d(C_2)\mathtt{pr}(C_1) \\ & + \sum_{\substack{C_1\in\mathtt{free}(\mathcal{S})\\ C_2\in\mathtt{free}(\mathcal{S})}} u(C_1\cap C_2)d(C_1\cap C_2) \end{aligned} \tag{5.13}$$

The total queriability of $\mathcal{S}$, $QU(\mathcal{S})$, is $QU_1(\mathcal{S}) + QU_2(\mathcal{S})$. In MC-CECD, given a set of concepts $\mathcal{C}$, dataset DS in the domain of $\mathcal{C}$ and budget B, we like to find design $\mathcal{S}$ over $\mathcal{C}$ such that $\sum_{C\in\mathcal{S}} w(C) \leq B$ and $\mathcal{S}$ has the maximum queriability. We reduce MC-CECD to CECD over a set of concepts by considering instances in which $u(C_i \cap C_j) = 0$ for all C_i, C_j. Hence, MC-CECD over a set of concepts is **NP**-hard.

Our algorithm for MC-CECD is based on the algorithm of Set Union Knapsack (SU-Knapsack) problem [130]. In SU-Knapsack, we have a collection of elements $\mathcal{E} = \{e_1, ..., e_n\}$, each with a *positive weight* denoted by $w(e_i)$. Furthermore, we have a collection of *items* $\mathcal{I} = \{i_1, ..., i_m\}$ such that each item is a subset of $\mathcal{E}$. Each item i has *profit* $p(i)$. Given budget B, the goal of SU-Knapsack$(\mathcal{E}, \mathcal{I})$ is to find a set of items $\mathcal{I}_{\text{SOL}}$ such that the total cost of the elements in $\mathcal{I}_{\text{SOL}}$, $\sum_{e\in\bigcup_{i\in\mathcal{I}_{\text{SOL}}}} w(e)$, is not more than B, and the total profit of $\mathcal{I}_{\text{SOL}}$, $\sum_{i\in\mathcal{I}_{\text{SOL}}} p(i)$, is maximized. The algorithm proposed in [130] starts with all possible pair of items as its initial set. Then, for each set, it goes through several iterations and at each iteration picks an item i from the remaining items with the maximum value of $\frac{p(i)}{w'(i)}$, where $w'(i) = \sum_{e\in i} \frac{w(e)}{\mathtt{freq}(e)}$ and $\mathtt{freq}(e)$ denotes the number of occurrences of element e in items of $\mathcal{I}$. The iterations will be performed until the budget is used up. The algorithm keeps the solution with maximum benefit amongst all constructed solutions. Then, it compares

this solution with the solution that has only the item with maximum profit and returns the one with the maximum profit amongst them.

Consider an instance of MC-CECD over concepts $\{C_1, ..., C_n\}$ with budget B. To solve this instance, we make an instance $\mathcal{M}$ of SU-Knapsack as follows. Let $C_1, ..., C_n$ be the input set of elements such that $w(C_i)$ is the annotation cost of concept C_i. We define the input items to be all subsets of $\{C_1, ..., C_n\}$ of size at most two (including empty set). The profit of each item is:

$$p(\emptyset) = \sum_{C \in \mathcal{C}} u(C)d(C) + \sum_{C_1, C_2 \in \mathcal{C}} u(C_1 \cap C_2)d(C_1 \cap C_2) \tag{5.14}$$

$$p(\{C_i\}) = u(C_i)(\mathtt{pr}(C_i) - d(C_i)) + \sum_{C \in \mathcal{C}} u(C_i \cap C)(d(C)\mathtt{pr}(C_i) - d(C \cap C_i)) \tag{5.15}$$

$$\begin{aligned} p(\{C_i, C_j\}) = {} & u(C_i \cap C_j)\mathtt{pr}(C_i)\mathtt{pr}(C_j) + u(C_i \cap C_j)d(C_i \cap C_j) \\ & - u(C_i \cap C_j)d(C_i)\mathtt{pr}(C_j) - u(C_i \cap C_j)\mathtt{pr}(C_i)d(C_j) \end{aligned} \tag{5.16}$$

The item corresponding to the empty set has cost zero and is picked by every solution for $\mathcal{M}$. Because annotators work better than a random selection, for each concept C, $\mathtt{pr}(C) \geq d(C)$ [91]. Hence, the profits of items in $\mathcal{M}$ are positive.

Theorem 5.4. *There exists a* $(1 - 1/e^{\frac{1}{k+1}})$*-approximation algorithm for* MC-CECD *instances in which each concept queried with at most* k *different other concepts.*

Proof. As described above, construct an instance $\mathcal{M}$ of SU-Knapsack from the MC-CECD instance.

For each pair of concepts C_i, C_j, the total cost of $\mathcal{I}_1 := (\{C_i\}, \{C_j\})$ is equal to the total cost of $\mathcal{I}_2 := (\{C_i\}, \{C_j\}, \{C_i, C_j\})$, and all elements have positive profits. If items $\{C_i\}, \{C_j\}$ are selected in an optimal solution of $\mathcal{M}$, item $\{C_i, C_j\}$ will also be picked. Thus, an optimal solution of $\mathcal{M}$ corresponds to a design $\mathcal{S}$ in MC-CECD: $\mathcal{I}_\mathcal{S} = \emptyset \cup \bigcup_{C \in \mathcal{S}} \{C\} \cup \bigcup_{C_1, C_2 \in \mathcal{S}} \{C_1, C_2\}$. Furthermore, the profit of $\mathcal{I}_\mathcal{S}$ is $p(\mathcal{I}_\mathcal{S}) = p(\emptyset) + \sum_{C \in \mathcal{S}} p(\{C\}) + \sum_{C_1, C_2 \in \mathcal{S}} p(\{C_1, C_2\}) = QU(\mathcal{S})$. Hence, the algorithm of [130] return a design whose queribility is at least $(1 - 1/e^{\frac{1}{k+1}})$ time the queribility of an optimal design.

5.7.2 MC-CECD Over Tree Taxonomies

Given design $\mathcal{S}$ over tree taxonomy $\mathcal{X} = (\mathcal{C}, \mathcal{R}, R)$, we denote the queriability of $\mathcal{S}$ for queries with exactly one concept and exactly two concepts by $QU_1(\mathcal{S})$ and $QU_2(\mathcal{S})$, respectively. Definition 5.2 computes $QU_1(\mathcal{S})$ over $\mathcal{X}$. Next, we compute the value of $QU_2(\mathcal{S})$.

Let query q refers to entities of concepts C_1 and C_2. Similar to computing $QU_2(\mathcal{S})$ over set of concepts, there are four cases to consider. First, if C_1 and C_2 both belong to the same partition P in $\mathcal{S}$, the query interface may examine only the documents annotated with P to find answers for q. The documents that contain instances of both C_1 and C_2 contain relevant answers to q. Hence, the query interface will find the desired answers to q with the probability of $\frac{d(C_1 \cap C_2)}{d(P)}$. In the second case, C_1 and C_2 belong to different partitions P_1 and P_2 in $\mathcal{S}$, respectively. The query interface should search the documents annotated under both P_1 and P_2, which are $d(P_1 \cap P_2)$ portion of the collection. Because the desired answers should contain instances of both concepts, the query interface finds the desired answers to q with probability of $\frac{d(C_1 \cap C_2)}{d(P_1 \cap P_2)}$. Now, assume that only one of C_1 or C_2, e.g., C_1, is in a partition P of $\mathcal{S}$. The query interface may search the documents that contain annotations of P. Because the desired answers are documents with instances of both C_1 and C_2, the query interface may return the desired answer with the probability of $\frac{d(C_1 \cap C_2)}{d(P)}$. Finally, if neither C_1 nor C_2 are in any partition, i.e., they both belong to $\texttt{free}(\mathcal{S})$, the query interface will search the whole collection and may find the desired answer in $d(C_1 \cap C_2)$ portion of the collection. We have:

$$
\begin{aligned}
QU_2(\mathcal{S}) = & \sum_{P \in \mathcal{S}} \sum_{\substack{C_1 \in \texttt{part}(P) \\ C_2 \in \texttt{part}(P)}} \frac{u(C_1 \cap C_2)\ d(C_1 \cap C_2)}{d(P)} \texttt{pr}(P) \\
& + \sum_{\substack{P_1 \in \mathcal{S} \\ P_2 \in \mathcal{S}}} \sum_{\substack{C_1 \in \texttt{part}(P_1) \\ C_2 \in \texttt{part}(P_2)}} \frac{u(C_1 \cap C_2) d(C_1 \cap C_2)}{d(P_1 \cap P_2)} \texttt{pr}(P_1) \texttt{pr}(P_2) \\
& + \sum_{P \in \mathcal{S}} \sum_{\substack{C_1 \in \texttt{part}(P) \\ C_2 \in \texttt{free}(\mathcal{S})}} \frac{u(C_1 \cap C_2) d(C_1 \cap C_2)}{d(P)} \texttt{pr}(P) \\
& + \sum_{\substack{C_1 \in \texttt{free}(\mathcal{S}) \\ C_2 \in \texttt{free}(\mathcal{S})}} u(C_1 \cap C_2) d(C_1 \cap C_2)
\end{aligned}
\tag{5.17}
$$

Let $QU(\mathcal{S})$ denote the queriability of $\mathcal{S}$ over queries with at most two concepts, i.e., $QU(\mathcal{S}) = QU_1(\mathcal{S}) + QU_2(\mathcal{S})$. We define the problem of MC-CECD over tree taxonomies similar to Problem 5.1 with the new formula for $QU(\mathcal{S})$. Since CECD over tree taxonomies is **NP**-hard, MC-CECD is **NP**-hard (consider instances with $u(C_i \cap C_j) = 0$ for all C_i, C_j).

We extend the Level-wise algorithm for the problem of MC-CECD over *tree* taxonomy. To this end, we combine the Level-wise algorithm and the algorithm for SU-Knapsack described in Section 5.7.1. Consider a level r of the tree taxonomy and let $C_1^r, \cdots, C_p^r$ be the concepts of level r. We compute the profits as defined in Section 5.7.1 for all subsets of size

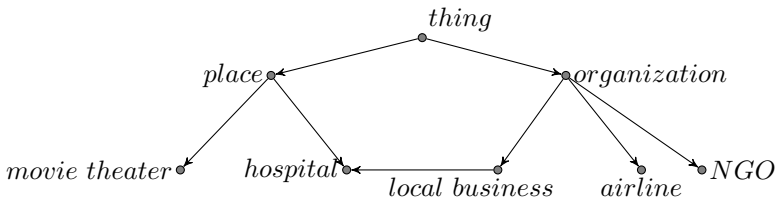


Figure 5.9: Fragments of *schema.org* taxonomy.

at most two (including the empty set) of the concepts in level r. Then, we find the solution with maximum queriability in level r using the approximation algorithm of SU-Knapsack. Our algorithm returns the solution with the maximum queriability over all levels of the taxonomy. The running time of the presented heuristic is $O(|\mathcal{C}|^3 L) \sim O(|\mathcal{C}|^3 \log |\mathcal{C}|)$ where L is the height of the tree taxonomy.

5.8 COST-EFFECTIVE DESIGN FOR DAG TAXONOMIES

5.8.1 Directed Acyclic Graph Taxonomies

Many taxonomies are in form of *directed acyclic graphs* (DAGs). Figure 5.9 shows fragments of *schema.org* taxonomy. Some concepts in this taxonomy are included in multiple superclasses. For example, a *hospital* is both a *place* and an *organization*. Therefore, a tree structure is not able to represent these relationships.

Formally, a *directed acyclic graph taxonomy* $\mathcal{X} = (R, \mathcal{C}, \mathcal{R})$ (*DAG taxonomy* for short), is a DAG, with vertex set $\mathcal{C}$, edge set $\mathcal{R}$ and root R where $\mathcal{C}$ is a set of concepts, $(D, C) \in \mathcal{R}$ if and only if $D, C \in \mathcal{C}$ and C is a subclass of D. The root concept $R \in \mathcal{X}$ is a concept without superclass. A concept $C \in \mathcal{C}$ is a *leaf concept* if and only if it has no subclass in $\mathcal{X}$. The definitions of *child*, *ancestor* and *descendant* over tree taxonomies naturally extends to DAGs.

5.8.2 Design Queriability

Design $\mathcal{S}$ over DAG taxonomy $\mathcal{X} = (R, \mathcal{C}, \mathcal{R})$ is a non-empty subset of $\mathcal{C} \setminus \{R\}$. Because of the richer structure of DAG taxonomies, designs over DAG taxonomies may improve the effectiveness of answering queries in more ways than the ones over tree taxonomies. For example, let dataset DS be in the domain of DAG taxonomy in Figure 5.9, and $\mathcal{S}_1 = \{$*place*, *organization*$\}$ be a design over this taxonomy. Because concept *hospital* is a subclass of both *place* and *organization*, its entities in DS are annotated by both these concepts.

By examining the entities that are annotated by both *place* and *organization*, the query interface is able to identify the instances of *hospital* in DS. Generally, the query interface may pinpoint instances of some concepts in the dataset by considering the intersections of multiple concepts in a design over a DAG taxonomy. Hence, subsets of a design may create partitions in a DAG taxonomy.

Definition 5.3. *Let $\mathcal{S}$ be a design over the DAG taxonomy $\mathcal{X} = (R, \mathcal{C}, \mathcal{R})$, and let $C \in \mathcal{C}$ be a leaf concept. An ancestor A of C in $\mathcal{S}$ is C's direct ancestor if and only if one of the followings hold.*

- *$A = C$.*
- *For each $D \in \mathcal{S}$, if D is an ancestor of C then D is not a descendant of A.*

The *full-ancestor-set* of C is the set of *all its direct ancestors*.

Example 5.5. *Consider a DAG taxonomy shown in Figure 5.9. The set {place, organization} is the full-ancestor-set of the concept hospital in design $\mathcal{S}_1$ = {place, organization}, and the set {local business, place} is the full-ancestor-set of the concept hospital in design $\mathcal{S}_2$ = {organization, local business, place} over the taxonomy in Figure 5.9.*

Definition 5.4. *Given design $\mathcal{S}$ over the DAG taxonomy $\mathcal{X} = (R, \mathcal{C}, \mathcal{R})$, the partition of a set of concepts $\mathcal{D} \subseteq \mathcal{S}$ is a set of leaf concepts $\mathcal{L} \subseteq \mathcal{C}$ such that for every leaf concept $L \in \mathcal{L}$, $\mathcal{D}$ is the full-ancestor-set of L.*

Example 5.6. *Let $\mathcal{S}_1$ = {organization, place} be a design over the DAG taxonomy shown in Figure 5.9. The concept hospital belongs to the partition of {organization, place} in $\mathcal{S}_1$. However, it does not belong to the partition of {place} because {place} is not the full-ancestor-set of hospital.*

The definitions of functions `part` and `free` over tree taxonomies naturally extend to DAG taxonomies. We define the frequency of partition P, denoted by $d(P)$, as the frequency of the intersection of concepts in its root. Using a similar analysis to the one in Section 5.3.2, we define the queriability of design S over DAG taxonomy $\mathcal{X} = (R, \mathcal{C}, \mathcal{R})$ as $\sum_{P \in \texttt{all-parts}(\mathcal{S})} \frac{\sum_{C \in P} u(C)d(C)}{d(P)} + \sum_{C \in \texttt{free}(\mathcal{S})} u(C)d(C)$ where the function `all-parts`($\mathcal{S}$) returns the collection of all full-ancestor-sets of $\mathcal{S}$ in $\mathcal{X}$.

5.8.3 Hardness of CECD over DAG Taxonomies

We define the CECD problem over DAG taxonomies similar to the CECD over tree taxonomies. Following from the **NP**-hardness results for CECD problem over tree taxonomy,

CECD problem over DAG taxonomies is NP-hard. We prove that finding an approximation algorithm with a reasonably small bound on its approximation ratio for the problem CECD over DAG taxonomies is significantly hard. Unfortunately, this is true even for the special cases where concepts in the taxonomy have equal costs or the desired design is disjoint. More precisely, we show that the CECD problem over a DAG taxonomy generalizes a hard problem in the approximation algorithms literature: DENSEST-k-SUBGRAPH. Given a graph $G = (V, E)$, in the DENSEST-k-SUBGRAPH problem, the goal is to compute a subset $U \in V$ of size k that maximizes the number of edges in the induced subgraph of U. It is known that, unless $\mathbf{P} = \mathbf{NP}$, no polynomial time approximation scheme, i.e., PTAS, exists to compute the densest subgraph [131]. Also, there are strong evidences that DENSEST-k-SUBGRAPH does not admit any approximation guarantee better than polylogarithmic factor [132, 133]. Recently, it has been shown that, assuming exponential time hypothesis (ETH), there is no polynomial time algorithm that approximates DENSEST-k-SUBGRAPH within a factor of $n^{1/(\log\log n)^c}$ [134]. The following theorem shows that approximating the k-densest subgraph reduces to approximating CECD.

Theorem 5.5. *An α-approximation algorithm for the* CECD *problem over DAG taxonomy with m concepts implies that there is an algorithm for the* DENSEST-k-SUBGRAPH *problem on $G = (V, E)$ with n vertices that returns a $O(\alpha)$-approximate solution.*

Proof. We use the following construction to reduce DENSEST-k-SUBGRAPH to CECD problem over DAG. Given a graph $G = (V, E)$ and a number k, we build an instance of the CECD over a DAG taxonomy as follows. For each edge $e \in E$, we introduce a leaf concept a_e, and for each vertex $v \in V$, we introduce a leaf concept a_v and a non-leaf concept S_v such that S_v is an ancestor of a_v and all a_e corresponding to the incident edges of v in G. Further, we set the budget B to k, the cost of each non-leaf concept to 1, and the cost of leaf concepts to $k+1$. If we select S_v and S_u in the design and $(u, v) \in E$, a_e will be a singleton partition. We set the popularities and frequencies of all concepts in the taxonomy respectively to the same fixed values u and d. Let m be the number of edges in G and n be the number of vertices in G. For each partition $p \in \texttt{part}(\mathcal{S})$, we set $d(p) = 1/\beta$ if p is a singleton edge concept and $d(p) = 1$ otherwise. β is a parameter which we determine later in the proof. Since leaf concepts are not affordable, there is an optimal design with exactly k non-leaf concepts. In each design $\mathcal{S}$ of size k, the contribution of every leaf concept in a non-singleton edge concept partition is exactly ud. Let $H_{\mathcal{S}}$ be the set of vertices in G whose corresponding non-leaf concepts in $\mathcal{C}$ are in $\mathcal{S}$. $E(H_{\mathcal{S}})$ denotes the set of edges with both endpoints in $H_{\mathcal{S}}$. It corresponds to the set of edge-concepts of $\mathcal{C}$ whose both non-leaf concepts associated with their endpoints are in $\mathcal{S}$. Let $\mathcal{S}_{\text{OPT}}$ be the solution of CECD corresponding to an optimal solution of the

DENSEST-k-SUBGRAPH. We have $QU(\mathcal{S}_{\text{OPT}}) = ud(\beta \cdot t + m + n - t)$ where t denotes the number of edges in $H_{\mathcal{S}_{\text{OPT}}}$. Let $\mathcal{A}$ be an α-approximation algorithm of CECD, and $\mathcal{S}_{\mathcal{A}}$ be the α-approximate design returned by $\mathcal{A}$. Let $QU(\mathcal{S}_{\mathcal{A}}) = ud(t' \cdot \beta + m + n - t')$ where t' is the number of edges whose both endpoint are in $\mathcal{S}_{\mathcal{A}}$. Since $\mathcal{A}$ is an α-approximation algorithm of CECD, $t \cdot \beta + m + n - t \leq \alpha(t' \cdot \beta + m + n - t')$. Thus, $t \leq t' \cdot \alpha + (m+n) \cdot \frac{\alpha-1}{\beta-1}$. Setting $\beta = (m+n)(\alpha-1) + 1$ leads to $O(\alpha)$-approximation for the DENSEST-k-SUBGRAPH.

Because the concepts in the instance of CECD in the proof of Theorem 5.5 have equal costs and its optimal solution is disjoint, i.e., there is no directed path between any two of concepts in the design, the hardness results of Theorem 5.5 is true for the special cases of CECD over DAG taxonomies where the concepts are equally costly and the problem has disjoint optimal solution.

The following corollaries result from Theorem 5.5.

Corollary 5.1. *If* **NP** $\not\subseteq \cap_{\epsilon>0}$**BPTIME**$(2^{n^\epsilon})$*, there is no polynomial time approximation scheme,* **PTAS***, for* CECD *over DAG taxonomies.*

Proof. By setting $\beta = \frac{(m+n)(\alpha-1)}{\epsilon} + 1$ in proof of Theorem 5.5 and using the hardness result of [131], no polynomial time approximation scheme algorithm exists for CECD unless **NP** $\subseteq$ $\cap_{\epsilon>0}$**BPTIME**(2^{n^ϵ}).

Corollary 5.2. *Assuming Exponential Time Hypothesis (ETH), the problem of* CECD *over DAG taxonomies does not admit an approximation guarantee better than* $n^{1/(\log\log n)^c}$.

Proof. It follows from the hardness result of [134] and Theorem 5.5.

5.9 EXPERIMENTS

5.9.1 Experiment Settings

Taxonomies and datasets

We have extracted eight taxonomies from YAGO ontology version 2008-w40-2 [93] to validate our model and evaluate effectiveness and efficiency of our proposed algorithms. YAGO organizes its concepts using subclass relationships in a DAG with a single root. We have created the breath-first tree of YAGO and randomly selected the concepts from the tree for our taxonomies. To validate our model, we have to compute and compare the effectiveness of answering queries using every feasible design over a taxonomy. Thus, we need

Table 5.2: The sizes and heights of taxonomies and the sizes of corresponding datasets and query workloads.

Taxonomy	T1	T2	T3	T4	T5	T6	T7	T8
#Concept	10	17	17	28	63	185	279	2269
#Height	2	2	3	7	6	8	8	9
#Documents	68982	267653	88479	1470661	1470661	1470661	1470661	1470661
#queries	648	256	146	4219	4888	4728	5216	11156

tree taxonomies with relatively small number of concepts for our validation experiments. We have extracted three tree taxonomies with relatively small numbers of nodes, called *T1*, *T2* and *T3*, to use in our validation experiments. The selected concepts for T1, T2 and T3 are from level 3 to 6 of the full YAGO tree which has a total of 9 levels. We have further picked five other tree taxonomies with larger numbers of concepts, denoted as *T4*, *T5*, *T6*, *T7* and *T8*. The concepts for T4 and T5 are selected from level 3 to 6 of the full YAGO tree. The concepts for T6 and T7 are randomly selected from levels 2 to 9 of the full YAGO tree. T8 contains all concepts from the original YAGO tree that appear at least once in the collection of English Wikipedia articles. We use all tree taxonomies to evaluate the effectiveness and efficiency of our proposed algorithms. Table 5.2 shows the properties of these taxonomies.

Query Workload

We use a subset of Bing[1] query log whose relevant answers are Wikipedia articles [135]. The relevant answers for each query in this query workload have been determined using the click-through information by eliminating noisy clicks. Each query contains between one to six keywords and has between one to two relevant answers with most queries having one relevant answer. Because the query log does not have the concepts behind its queries, we adapt an automatic approach to find the leaf concept from the taxonomy associated with each query. We label each query by the concept of the matching instance in its relevant answer(s). Then, we select queries labeled with a *single* concept. Using this method, we create a query workload per each of our datasets. It is well known that the effectiveness of answering some queries may not be improved by annotating the dataset [108]. For instance, all candidate answers for a query may contain mentions to the entities of the query concept. To reasonably evaluate our algorithms, we have ignored the queries whose rankings remain the same over the unannotated version and the version of the dataset where all concepts in the taxonomy are annotated. Table 5.2 shows the information about our query workloads.

We estimate the popularities of concepts for each taxonomy by sampling a small subset

[1] *www.bing.com*

of randomly selected queries from their corresponding query workloads. We compute the popularity of each concept using estimation error rate of 5% under the 95% confidence level. The number of sampled queries are between 100 and 500 for each taxonomy. Because some concepts in a taxonomy may not appear in its query workload, we smooth the popularity of a concept C, $u(C)$, using the Bayesian m-estimate method [60]: $\hat{u}(C) = \frac{\hat{P}(C|QW)+mp}{m+\sum_C \hat{P}(C|QW)}$, where $\hat{P}(C|QW)$ is the probability that C occurs in the query workload QW and p denotes the prior probability. We set the value of the smoothing parameter, m, to 1 and use a uniform distribution for all the prior probabilities.

Query Interface

We index our datasets using Lucene[2]. Given a query, we rank its candidate answers using BM25 ranking formula, which is shown to be more effective than other similar document ranking methods [60]. Then, we apply the information about the concepts in the query and documents to return the answers whose matching instances have the same concept as the concept of the query. If the concept in the query has not been annotated in the collection, the query interface returns the list of documents ranked by BM25 method without any modification. We have implemented our query interface and algorithms in Java 1.7 and performed our experiments on a Linux server with 100 GB of main memory and two quad core processors.

Effectiveness Metric

Because all queries in our query workloads have one or two relevant answers, we measure the ranking quality of answering queries over a dataset using mean reciprocal rank (MRR) [60]. MRR is an inverse of the rank of the first relevant answer in the returned list of answers. MRR is used to measure the effectiveness of results for queries that have a small number of relevant answers [60]. Since most queries in our query workload have a single relevant answer, MRR is an appropriate metric to measure the effectiveness of their results. We measure the statistical significance of our results using the paired-t-test at a significant level of 0.05.

[2] *lucene.apache.org*

Cost Models

We use three models for generating costs of concept annotation. First, we assign a randomly generated cost to each concept in a taxonomy. The results reported for this model are averaged over 5 sets of random cost assignments per budget. We call this model *random cost* model. Second, when there is not any reliable estimation available for the cost of annotating concepts, an organization may assume that all concepts are equally costly. Hence, in our second cost model, we assume that all concepts in the input taxonomy have equal cost. We name this model *uniform cost* model. These two cost models have also been used to model the costs of large-scale data curation [22, 25, 136]. Lastly, an organization may also assume that the cost of a concept depends on how likely the concept appears in a collection. Hence, in our third cost model, we assume that a cost of a concept is proportional to its frequency. Specifically, the cost of a concept c is $d(c)/\sum_{l \text{ is leaf}} d(l)$. We call this model *frequency-based cost* model.

We use a range of budgets between 0 and 1 with a step size of 0.1 where 1 means sufficient budget to annotate all leaf concepts in a taxonomy and 0 means no budget is available.

5.9.2 Validating Queriability Function

Oracle: Given a fixed budget, Oracle enumerates all feasible designs over the input taxonomy. Given an effectiveness metric, such as MRR, for each design, it computes the average the effectiveness metric for all queries in the query workload over the dataset annotated by the design. It then returns the design with maximum value of the average effectiveness metric.

Popularity Maximization (*PM*): Following the traditional approach towards conceptual design for databases, one may select concepts in a design that are *more important* for users [43]. Hence, we implement an algorithm, called *PM*, that enumerates all feasible designs, such as $\mathcal{S}$, in a taxonomy and returns the one with the maximum value of $\sum_{p \in \texttt{part}(\mathcal{S})} \sum_{C \in p} u(C)\texttt{pr}(p)$. This design contains the concepts that are more frequently queried by users and also annotated more accurately.

Queriability Maximization (*QM*): QM enumerates all feasible designs over the input taxonomy and returns the one with the maximum queriability as computed in Section 5.3.2.

Because we would like to explore how accurately PM and QM predict the amount of improvement in the effectiveness of answering queries by a design, we assume that these algorithms have complete information about the popularities and frequencies of concepts. Since Oracle, PM and QM algorithms enumerate all feasible designs, it is not possible to

run them over large taxonomies. Hence, we run these algorithms over small taxonomies, i.e. T1, T2 and T3.

Table 5.3 illustrates the average MRR achieved by Oracle, QM and PM over taxonomies T1, T2 and T3 for various budgets. The values of MRR show at budget 0.0 is the one achieved by BM25 ranking without annotating any concept in the datasets. We note that there is no improvement in average MRR over taxonomy T1 for budget 0.1 under uniform cost because each concept in the taxonomy costs more than 0.1.

Over all taxonomies and cost models, the designs selected by QM deliver close MRR values to the ones selected by Oracle. There are a few cases where the results of QM are significantly worse than the results of Oracle. For instance, consider the results of QM and Oracle for budget 0.3 over taxonomy T2. In T2, concept *writing* is the parent of a leaf concept *dramatic composition* and a couple other leaf concepts whose popularities are much less than that of *dramatic composition*. QM picks a design that contains *dramatic composition*. This design will deliver the highest values of MRR for queries with concept *dramatic composition*, but it does *not* help improving the values of MRR for queries whose concepts are other children of *writing* over the unannotated dataset. That is, QM picks a relatively less popular concept, but maximizes the improvement of the effectiveness for queries with this concept. On the other hand, Oracle selects *writing* instead of *dramatic composition*. Intuitively, this design improves the values of MRR for queries with concept *dramatic composition* less than the design selected by QM. However, this design will improve the values of MRR for queries with other child concepts of *writing*. For this dataset, selecting *writing* helps improving the values of MRR for queries with concept *dramatic composition* as equal as selecting *dramatic composition*. Because, the design selected by QM is not able to improve the effectiveness of answering queries whose concepts are other children of *writing*, QM is less effective than Oracle. We have observed a similar behavior for other cases when the results of QM are significantly worse than Oracle. This observation suggests that if the budget is relatively small, it is sometimes better to annotate rather general concepts.

For frequency-based cost, Oracle, QM and PM are equally effective. Under this cost model, the leaf concepts are cheaper than any internal concept node. Hence, every algorithm chooses leaf concepts and returns the same design. Furthermore, Oracle, QM and PM return the same values of average MRR over T1 for all budgets. This is because these algorithms can select 8 out of 9 leaf concepts using budget 0.1, and the returned designs help to answer queries with those concepts effectively. However, the remaining leaf concept, e.g., *person*, costs more than 0.9 because of its 92% frequency in T1. Because each internal concept either costs more than *person* or has all of their leaf descendants included in a design, these algorithms choose all leaf concepts except *person* and do not include any internal

concept. Therefore, over T1, the algorithms are equally effective across all test budgets under frequency-based cost model.

Nevertheless, the results from Table 5.3 indicate that QM delivers designs that improve the average MRR of answering queries more than those delivered by PM. Overall, PM annotates more general concepts from the taxonomy to improve the effectiveness of larger number of queries. Hence, to answer a query, the query interface often has to examine the documents annotated by an ancestor of the query concept. As this set of documents contain many answers whose concepts are different from the query concept, the query interface is usually not able to improve the value of MRR for a query significantly. QM selects the designs with relatively less general concepts. Although its designs may not improve the ranking quality of every query, the designs significantly improve the ranking quality of queries whose concepts belong to the selected designs.

5.9.3 Effectiveness of the Proposed Algorithm

Queriability formula needs the value of the frequency for each concept in the input taxonomy over the dataset. However, it is not possible to find the exact frequencies of concepts without annotating the mentions to their entities in the dataset. Similar to [25], we estimate the concept frequencies by sampling a small subset of randomly selected documents from the dataset. We compute the frequency of each concept using an estimation error rate of 5% under the 95% confidence level, which is about 400 documents for all datasets. We also smooth the sampled frequencies using Bayesian m-estimates with smoothing parameter of 1 and uniform priors. We denote the Level-wise algorithm as *LW* and the dynamic programming algorithm as *DP* for brevity. We also compare LW and DP with the *APM* algorithm from [25] which finds a design over a set of concepts. We use all concepts in the taxonomy as a set of concepts for an input to APM. APM uses a technique to convert popularities and costs to positive integers [25]. We set the ϵ value of the scaling for APM to 0.01. As we have mentioned in Section 5.4, LW uses APM to find the optimal design in each level. We set the scaling factor of the APM algorithm used by LW to 0.01. In addition, we also perform APM algorithm only over a set of leaf concepts in a taxonomy, and denote this modification of APM as *APM-L*.

Since DP also assumes popularity (u), frequency (d) and cost (w) to be positive integers, we also use scaling to convert the values of popularity, frequency and cost of all concept in the input taxonomy to positive integers [129]. The details of the scaling techniques is described in Section 5.5. For simplicity, we set both ε_U and ε_D to be the same and denote them as ε. Intuitively, the smaller the value of ε is, the more exact result DP will deliver.

However, DP takes longer to run for smaller values of ε as the range of U, D and $B_{\texttt{total}}$ will become larger. Since we cannot use a very small value of ε, our preliminary results of using this scaling technique shows that many concepts have u or d values equals to 0 after scaling. Hence, DP may not explore all feasible designs and may miss some popular concepts whose d value are small. Thus, we modify the aforementioned scaling technique by adding a constant value of 1 to both $\hat{u}(C)$ and $\hat{d}(C)$. This addition ensures that each concept, after the scaling, contributes in the computation of queriability, and DP will explore all feasible designs. We set the value of ε for DP to 0.05 for the experiments in this section.

Table 5.4 shows the values of average MRR for APM, APM-L, LW and DP over T1, T2, T3, T4 and T5. Overall, the designs returned by LW and DP improve the effectiveness of answering queries for all taxonomies more than the designs returned by APM. This is because APM does not consider the structural information of the taxonomy. APM often picks many popular concepts that are ancestor or descendant of each other, LW and DP use structural information of the taxonomy when selecting a design and thus avoid this problem.

Although APM-L is shown to be more effective than APM, LW is generally more effective than APM-L. Since APM-L returns designs only over leaf concepts of a taxonomy, it does not have the same drawback as that of APM and so the average MRR values of APM-L is higher than APM. In many cases, the designs returned APM-L and LW are equally effective. This is because there are only two levels of concepts in T1 and T2, and the given budgets are usually enough to select popular concepts in the leaf level. Hence, both LW and APM-L returns the same designs. In addition, although the height of T4 and T5 are higher than T1, T2 and T3, the taxonomy trees are also unbalanced, and for each internal concept, the distribution of the popularity and frequency of its child concepts are extremely skewed. For instance, *organism* has only two children in T4, and both children are leaves. One of the children, namely *person*, has almost the same popularity as that of *organism*. Hence, a design with concepts from a leaf level and a design with concepts from the same level as *organism* are equally effective. Nevertheless, LW returns different designs than APM-L in some cases, and they are more effective than those of APM-L.

DP is also generally more effective than APM-L. This is because designs returned by DP can contain both leaf and non-leaf concepts. For instance, given a budget of 0.3 over T2, APM-L picks *dramatic composition* and *literary composition* while DP picks *writing* instead of *dramatic composition*. Since *writing* is a parent of both *dramatic composition* and *literary composition*, DP can answer queries of *literary composition* and queries of other concepts that are children of *writing* more effectively.

The results shown in Table 5.4 indicate that the designs returned by DP are more effective than those returned by LW for small budgets and small taxonomies such as T1, T2 and

T3. However, the designs returned by LW are more effective than those delivered by DP for moderate to large budgets, e.g., 0.4-0.7. Because the frequencies and popularities of concepts in most taxonomies follow a power-law distribution, most of all concepts have frequency and popularity close to zero. When a given budget is very small, neither methods have a sufficient budget to select the concepts with medium or small frequencies and popularities. However, over sufficiently large budgets, both algorithms are able to select some of these concepts. If the DP algorithm does not use sufficiently small values for ϵ, concepts with considerably different frequencies and popularities may end up with equal values of scaled frequencies and popularities. Since it takes a very long time to run DP with an ϵ value less than 0.05, one cannot use a sufficiently small value of ϵ to preserve the differences in popularities and frequencies for all concepts As the DP algorithm is not able to distinguish these concepts, it may not be able to find a design with the most queriability. Since we are able to run LW using sufficiently small scaling factors in its APM component, LW often returns more effective designs than DP over relatively large budgets.

Table 5.5 shows the values of average MRR for APM and LW over T6 and T8. The results of budgets greater than 0.6 are omitted because they are the same as the previous budgets. We do not report any result for DP over T6, T7 and T8 because the algorithm does not terminate for almost all budgets after several days. The results shown in Table 5.5 indicate that the designs returned by LW are more effective than the ones delivered by APM.

Because of a very skewed distribution of concept popularity in T6, T7 and T8, budgets of 0.4 for T6 and T7, and 0.2 for T8 are usually sufficient to create a design that includes all leaf concepts that appear in the query workloads. Hence, APM, APM-L and LW deliver almost equally effective designs for relatively large budgets. We further evaluate APM, APM-L and LW using budgets 0.01 and 0.05 for T6, T7 and T8 and budgets 0.001 and 0.005 for T8 as shown in Table 5.5. Overall, LW is significantly more effective than APM-L and APM. This is because, with small budget, it is more preferable to choose a concept in a higher level instead of multiple leaf concepts that are children or descendants of that one concept. Then an algorithm can spend the remaining budget on other concepts to help answer other queries more effectively.

Dynamic Programming with Cost-Dependency

Table 5.6 show the values of MRR for Queriability Maximization (QM) and dynamic programming algorithm with cost dependencies (*DPC*) over T1, T2 and T3 taxonomies. The cost for each concept in these taxonomies has been randomly generated and depends on which ancestors of the concept have been selected in the design. The budgets that cover

all leaf nodes in T1, T2 and T3 are 1, and the budget that cover all nodes are 1.25, 1.1 and 1.1, respectively. QM is a brute force algorithm that explores all feasible solutions and finds the design with maximum queriability. Because the space of the possible solutions for this problem is larger than the original CECD problem, it takes much longer to run QM for this problem. Hence, we have covered a smaller range of budgets in this set of experiments. The results shown in Table 5.6 indicate that the smaller the value of ϵ, the closer the average MRR of the designs returned by DPC are to the ones delivered by QM.

5.9.4 Efficiency of Proposed Algorithms

We measure the running times of LW and DP over moderate and large taxonomies, i.e., T4, T5, T6, T7 and T8, and set the available main memory of Java Virtual Machine to 64GB. Table 5.7 shows the average running times of APM, LW and DP for T4, T5, T6, T7 and T8 over budgets 0.1 to 0.9. Some results of DP are not reported because the algorithms did not finish after a day. Overall, LW is as efficient as APM, and it is more efficient than DP. Because the size of the table required in the DP algorithm is substantially large for $\epsilon = 0.05$, it occupies most of the available main memory. Thus, the running time of DP is longer than APM and LW. Therefore, LW scales for large taxonomy and is efficient for a design-time task. On the other hand, DP has a reasonable running time for T4 and T5, but it does not scale for large taxonomy such as T6, T7 and T8.

Dynamic Programming with Cost-Dependency

Table 5.7 shows the average running times of DPC for T4, T5, T6, T7 and T8 over budgets 0.25, 0.5, 0.75 and 1 using the scaling factor, ϵ, of 0.1 and 0.2. We do not report the running time of DPC using $\epsilon = 0.1$ for T6, T7 and T8 and DPC using $\epsilon = 0.2$ for T7 and T8 because the algorithm did not finish after a day. Because DPC requires a larger table than the one required for DP, the running time of DPC is longer than that of DP for the same scaling factor, i.e., ε. Hence, we run DPC using only ϵ values of 0.1 and 0.2. Overall, DPC with $\epsilon = 0.1$ is reasonably efficient to perform a design-time task for a taxonomy of up to size 70, and DPC with $\varepsilon = 0.2$ is reasonably efficient for a taxonomy of up to size 200.

5.9.5 Queries With Multiple Concepts

Validation and Effectiveness

We have selected all queries with multiple concepts which belong to T1, T2 and T3 from our query workload and filtered out the queries whose ranking quality is not improved by annotating all concepts in the corresponding taxonomy. This results in 6, 37 and 8 queries over T1, T2 and T3, respectively. The number of concepts for each query is 2. Because there are not enough queries with multiple concepts for T1 and T3, we do not evaluate our models over T1 and T3. Since the number of queries with one concept over T2 is seven times larger than the number of multiple concepts, we randomly select a subset of queries with one concept from the original query workload and combine them with the queries with multiple concepts over T2. The new query workload contains 106 queries. We run Oracle, the queriability maximization over queries with multiple concepts (MQM), the Level-wise algorithm for multiple concepts (MLW), and the Level-wise algorithm (LW) over the query workload. Oracle is described in Section 5.9.2. MQM enumerates all feasible designs over the input taxonomy and returns the one with maximum queriability as computed in Section 5.7. Table 5.9 shows the values of average MRR for Oracle, MQM, MLW and LW algorithms over T2. MQM generally returns the designs similar to those of Oracle. The designs returned by MLW also deliver close average MRR the ones delivered by MQM in most cases. Overall, MLW significantly outperforms LW over T2.

We have also evaluated the effectiveness of MLW and LW over taxonomies T4, T5, T6, T7 and T8. We have selected all queries with multiple concepts over T4, T5, T6, T7 and T8 from our query workload. Table 5.8 shows the numbers of queries with multiple concepts and the minimum, average and maximum numbers of concepts per query for T4, T5, T6, T7 and T8. Table 5.10 shows the values of MRR for MLW and LW algorithms over T4, T5, T6, T7 and T8. Overall, the designs returned by MLW deliver significantly higher average MRR than the designs selected by LW over all taxonomies.

We have observed less difference between the results of MLW and LW when the distribution of concept popularity in the taxonomy is less skewed. This is because, when the popularity distribution is very skewed, both algorithms select all or most relatively popular concepts. Hence, they selected designs are almost equally effective. For example, the distribution of concept frequencies in T6 is considerably more skewed than those of the concepts in T7, thus, the designs delivered by MLW and LW over T6 are significantly more different than the ones these algorithms return over T7.

Efficiency

We measure the average running times of MLW over moderate and large taxonomies, i.e., T4, T5, T6, T7 and T8, and set the available main memory of the Java Virtual Machine to 64GB. The average running times of MLW for T4, T5, T6, T7 and T8 over budgets 0.1 to 0.9 are 3, 3, 4, 4 and 8 minutes, respectively, which are reasonable for a design-time task.

Table 5.3: Average MRR for Oracle, PM and QM over T1, T2 and T3. Statistically significant differences between PM and QM and between Oracle and QM are marked in bold and italic, respectively. B denotes a given budget.

		Uniform Cost			Random Cost			Frequency-based Cost		
	B	Oracle	QM	PM	Oracle	QM	PM	Oracle	QM	PM
T1	0.0					0.187				
	0.1	0.187	0.187	0.187	0.259	0.255	0.239	0.475	0.475	0.475
	0.2	0.362	**0.362**	0.197	0.385	**0.384**	0.226	0.475	0.475	0.475
	0.3	0.415	**0.406**	0.203	0.424	**0.417**	0.212	0.475	0.475	0.475
	0.4	0.459	**0.459**	0.227	0.461	**0.461**	0.262	0.475	0.475	0.475
	0.5	0.492	**0.492**	0.400	0.492	**0.492**	0.341	0.475	0.475	0.475
	0.6	0.501	**0.501**	0.444	0.501	**0.499**	0.426	0.475	0.475	0.475
	0.7	0.507	0.507	0.497	0.507	**0.504**	0.476	0.475	0.475	0.475
	0.8	0.507	0.507	0.507	0.507	0.507	0.505	0.475	0.475	0.475
	0.9	0.507	0.507	0.507	0.507	0.507	0.507	0.475	0.475	0.475
T2	0.0					0.342				
	0.1	0.504	0.479	0.504	0.515	0.471	0.482	0.598	0.598	0.598
	0.2	0.577	0.543	0.551	0.616	0.586	0.559	0.662	0.662	0.662
	0.3	*0.641*	**0.629**	0.574	0.677	**0.661**	0.576	0.677	0.674	0.674
	0.4	0.729	**0.729**	0.586	0.725	**0.725**	0.616	0.741	0.741	0.741
	0.5	0.745	**0.745**	0.615	0.744	**0.742**	0.652	0.747	0.747	0.747
	0.6	0.751	**0.751**	0.647	0.754	0.754	0.695	0.748	0.748	0.748
	0.7	0.763	0.763	0.759	0.763	0.758	0.732	0.748	0.748	0.748
	0.8	0.764	0.764	0.764	0.764	0.763	0.763	0.751	0.748	0.751
	0.9	0.764	0.764	0.764	0.764	0.764	0.764	0.763	0.763	0.763
T3	0.0					0.322				
	0.1	0.469	0.469	0.453	0.528	0.521	0.514	0.447	0.447	0.447
	0.2	0.595	0.594	0.579	0.646	**0.629**	0.587	0.531	0.531	0.531
	0.3	0.680	**0.679**	0.600	0.714	**0.712**	0.660	0.594	0.594	0.594
	0.4	0.745	**0.734**	0.685	0.747	0.739	0.711	0.725	0.725	0.725
	0.5	0.754	0.754	0.741	0.758	0.757	0.748	0.734	0.734	0.734
	0.6	0.760	0.760	0.760	0.760	0.760	0.760	0.734	0.734	0.734
	0.7	0.760	0.760	0.760	0.760	0.760	0.760	0.739	0.739	0.739
	0.8	0.760	0.760	0.760	0.760	0.760	0.760	0.745	0.739	0.739
	0.9	0.760	0.760	0.760	0.760	0.760	0.760	0.754	0.754	0.754

Table 5.4: Average MRR for APM, APM-L, LW and $DP_{\epsilon=0.05}$ over T1, T2, T3, T4 and T5. Statistically significant difference between APM and LW, between APM and DP, between DP and LW, between APM-L and LW, and between DP and APM-L are in italic, bold, underline, apostrophe(') and star(*), respectively. B denotes a given budget.

		Uniform Cost				Random Cost				Frequency-based Cost			
	B	APM	APM-L	LW	DP	APM	APM-L	LW	DP	APM	APM-L	LW	DP
	0.1	0.187	0.187	0.187	0.187	0.239	0.250	0.250	0.255	0.475	0.475	0.475	0.475
	0.2	0.197	0.220	*0.220*	**0.362***	0.204	0.318	*0.318*	**0.384**	0.475	0.475	0.475	0.475
	0.3	0.221	0.394	*0.394*	**0.406**	0.288	0.390	*0.390*	**0.417***	0.475	0.475	0.475	0.475
	0.4	0.394	0.438	*0.438*	**0.459***	0.357	0.440	*0.440*	**0.460***	0.475	0.475	0.475	0.475
T1	0.5	0.438	0.492	*0.492*	**0.492**	0.421	0.492	*0.492*	**0.486**	0.475	0.475	0.475	0.475
	0.6	0.492	0.501	0.501	0.492	0.471	0.496	*0.496*	0.488	0.475	0.475	0.475	0.475
	0.7	0.501	0.502	0.502	0.493	0.494	0.502	*0.502*	0.494	0.475	0.475	0.475	0.475
	0.8	0.502	0.507	0.507	0.507	0.502	0.503	0.503	0.502	0.475	0.475	0.475	0.475
	0.9	0.507	0.507	0.507	0.507	0.502	0.506	0.506	0.506	0.475	0.475	0.475	0.475
	0.1	0.504	0.479	0.479	0.479	0.481	0.466	0.466	0.476	0.589	0.598	0.589	0.598
	0.2	0.534	0.566	*0.566*	**0.566**	0.564	0.567	0.571'	**0.591***	0.595	0.662	*0.662*	**0.662**
	0.3	0.580	0.629	*0.629*	**0.629**	0.607	0.638	*0.638*	**0.652***	0.667	0.668	0.668	0.671
	0.4	0.651	0.718	*0.718*	**0.729**	0.666	0.686	*0.686*	**0.713***	0.668	0.741	*0.741*	**0.739**
T2	0.5	0.673	0.745	*0.745*	**0.734**	0.673	0.738*	*0.738*	0.728	0.703	0.747	*0.747*	**0.747**
	0.6	0.746	0.751	0.751	0.750	0.702	0.747	0.748	0.746	0.709	0.748	*0.748*	**0.748**
	0.7	0.751	0.758	0.758	0.763	0.747	0.755	0.755	0.751	0.747	0.748	0.748	0.748
	0.8	0.757	0.764	0.764	0.764	0.755	0.760	0.761	0.759	0.748	0.748	0.748	0.748
	0.9	0.757	0.764	0.764	0.764	0.758	0.763	0.764	0.761	0.748	0.748	0.748	0.756
	0.1	0.453	0.469	0.469	0.469	0.458	0.475	0.499'	0.524*	0.407	0.407	0.407	0.447
	0.2	0.453	0.594	*0.594*	**0.594**	0.545	0.608	*0.615*'	**0.629***	0.531	0.531	0.531	0.531
	0.3	0.579	0.679	*0.679*	**0.679**	0.616	0.682	*0.698*'	**0.701***	0.577	0.577	0.577	0.594
	0.4	0.664	0.734	*0.734*	**0.734**	0.648	0.732	*0.732*	**0.734**	0.626	0.679	*0.679*	**0.688***
T3	0.5	0.719	0.739	*0.739*	**0.739**	0.685	0.738	*0.738*	**0.744**	0.669	0.734	*0.734*	**0.734**
	0.6	0.737	0.760	*0.760*	**0.760**	0.730	0.752	*0.752*	0.751	0.715	0.734	0.734	0.734
	0.7	0.758	0.760	0.760	0.760	0.750	0.760	0.760	0.752	0.676	0.739	0.739	0.739
	0.8	0.760	0.760	0.760	0.760	0.759	0.760	0.760	0.760	0.701	0.739	0.739	0.739
	0.9	0.760	0.760	0.760	0.760	0.759	0.760	0.760	0.760	0.739	0.749	0.749	0.754
	0.1	0.343	0.413	*0.413*	**0.413**	0.408	0.423	*0.439*'	**0.426**	0.344	0.344	0.344	**0.363***
	0.2	0.363	0.456	*0.456*	**0.456**	0.422	0.457	*0.467*'	**0.462***	0.364	0.364	0.365	**0.394***
	0.3	0.433	0.491	*0.491*	**0.496***	0.440	0.516	*0.518*	**0.508**	0.389	0.395	0.391	**0.488***
	0.4	0.435	0.563*	*0.563*	**0.544**	0.441	0.540	*0.548*'	**0.547***	0.525	0.556*	*0.556*	0.544
T4	0.5	0.456	0.573	*0.573*	**0.601***	0.464	0.587	*0.587*	**0.588**	0.544	0.584	*0.581*	**0.584**
	0.6	0.480	0.608	*0.608*	**0.612**	0.503	0.606	*0.606*	**0.597**	0.548	0.609	*0.609*	**0.609**
	0.7	0.547	0.627*	*0.627*	**0.621**	0.539	0.622	*0.622*	**0.621**	0.525	0.609	*0.609*	**0.609**
	0.8	0.550	0.628	*0.628*	**0.627**	0.556	0.627	*0.627*	**0.627**	0.539	0.619	*0.619*	**0.619**
	0.9	0.555	0.629	*0.629*	**0.629**	0.563	0.629	*0.629*	**0.629**	0.559	0.628	*0.628*	**0.628**
	0.1	0.376	0.442	*0.442*	**0.448**	0.431	0.471	*0.475*'	**0.467**	0.373	0.374	*0.379*'	**0.379***
	0.2	0.450	0.516	*0.516*	**0.510**	0.458	0.540*	*0.540*	**0.528**	0.394	0.394	*0.409*'	**0.415***
	0.3	0.501	0.571	*0.571*	**0.571**	0.523	0.577	*0.580*'	**0.580***	0.416	0.545*	*0.545*	**0.461**
	0.4	0.543	0.608*	*0.608*	**0.574**	0.561	0.605	*0.605*	**0.600**	0.557	0.608	*0.608*	**0.608**
T5	0.5	0.572	0.621*	*0.621*	**0.608**	0.581	0.624	*0.624*	**0.617**	0.579	0.624	*0.624*	**0.624**
	0.6	0.607	0.638	*0.638*	**0.624**	0.608	0.637	*0.637*	**0.625**	0.600	0.626	*0.625*	**0.627**
	0.7	0.613	0.647	*0.647*	**0.647**	0.621	0.648*	*0.648*	**0.638**	0.556	0.648	*0.648*	**0.648**
	0.8	0.633	0.653	*0.653*	**0.651**	0.631	0.653	*0.653*	**0.651**	0.581	0.652	*0.652*	**0.652**
	0.9	0.642	0.656	*0.656*	**0.656**	0.643	0.655	*0.655*	**0.655**	0.569	0.655	*0.655*	**0.655**

Table 5.5: Average MRR for APM, APM-L and LW over T6 and T8. Statistically significant differences between APM and LW, between APM and APM-L, and between APM-LW and LW are marked in bold, italic and underline, respectively.

		Uniform Cost			Random Cost			Frequency-based Cost		
	B	APM	APM-L	LW	APM	APM-L	LW	APM	APM-L	LW
T6	0.01	0.235	0.235	0.235	*0.375*	0.291	**0.403**	0.259	0.259	0.259
	0.05	0.411	*0.474*	**0.474**	0.412	*0.491*	**0.497**	0.311	*0.327*	**0.327**
	0.1	0.461	*0.564*	**0.564**	0.466	*0.559*	**0.563**	0.333	*0.380*	**0.380**
	0.2	0.583	*0.625*	**0.625**	0.589	*0.624*	**0.625**	0.392	0.397	**0.404**
	0.3	0.627	*0.648*	**0.648**	0.623	*0.647*	**0.648**	0.403	*0.410*	**0.417**
	0.4	0.631	*0.652*	**0.652**	0.638	*0.652*	**0.652**	0.418	*0.589*	**0.589**
	0.5	0.642	0.653	0.653	0.642	*0.653*	**0.653**	0.573	*0.626*	**0.626**
	0.6	0.646	0.653	0.653	0.646	*0.653*	**0.653**	0.577	*0.632*	**0.632**
	0.7	0.650	0.653	0.653	0.650	0.653	0.653	0.578	*0.643*	**0.643**
	0.8	0.651	0.653	0.653	0.651	0.653	0.653	0.588	*0.648*	**0.648**
	0.9	0.652	0.653	0.653	0.652	0.653	0.653	0.590	*0.651*	**0.651**
T7	0.01	0.344	*0.380*	**0.380**	0.385	0.391	**0.406**	0.286	0.285	**0.310**
	0.05	0.448	*0.525*	**0.525**	0.451	*0.517*	**0.530**	0.371	*0.403*	**0.403**
	0.1	0.545	*0.609*	**0.609**	0.550	*0.612*	**0.616**	0.410	*0.460*	**0.460**
	0.2	0.637	*0.669*	**0.669**	0.633	*0.671*	**0.671**	0.464	0.463	0.461
	0.3	0.659	*0.682*	**0.682**	0.667	*0.683*	**0.683**	0.464	*0.602*	**0.602**
	0.4	0.677	0.686	0.686	0.675	*0.686*	**0.686**	0.606	*0.642*	**0.642**
	0.5	0.683	0.686	0.686	0.681	0.686	0.686	0.609	*0.646*	**0.646**
	0.6	0.684	0.686	0.686	0.684	0.686	0.686	0.611	*0.662*	**0.662**
	0.7	0.684	0.686	0.686	0.684	0.686	0.686	0.615	*0.672*	**0.672**
	0.8	0.685	0.686	0.686	0.685	0.686	0.686	0.609	*0.679*	**0.679**
	0.9	0.685	0.686	0.686	0.685	0.686	0.686	0.607	*0.682*	**0.682**
T8	0.001	0.357	0.357	0.357	*0.365*	0.334	**0.407**	0.277	0.277	0.277
	0.005	0.433	*0.452*	**0.452**	0.462	0.470	**0.473**	0.316	*0.336*	**0.334**
	0.01	**0.537**	0.517	0.517	0.535	0.535	0.535	0.355	*0.372*	**0.379**
	0.05	0.686	*0.707*	**0.707**	0.712	0.712	0.712	0.479	*0.561*	**0.561**
	0.1	0.730	*0.738*	**0.738**	0.738	0.738	0.738	0.541	*0.598*	**0.598**
	0.2	0.740	0.743	0.743	0.742	0.742	0.742	0.570	*0.704*	**0.704**
	0.3	0.742	0.743	0.743	0.743	0.743	0.743	0.665	*0.716*	**0.716**
	0.4	0.743	0.743	0.743	0.743	0.743	0.743	0.680	*0.737*	**0.737**
	0.5	0.743	0.743	0.743	0.743	0.743	0.743	0.651	*0.737*	**0.737**
	0.6	0.743	0.743	0.743	0.743	0.743	0.743	0.668	*0.737*	**0.737**
	0.7	0.743	0.743	0.743	0.743	0.743	0.743	0.696	*0.737*	**0.737**
	0.8	0.743	0.743	0.743	0.743	0.743	0.743	0.683	*0.741*	**0.741**
	0.9	0.743	0.743	0.743	0.743	0.743	0.743	0.691	*0.743*	**0.743**

Table 5.6: Average MRR of QM and DPC using different values of ε over T1, T2 and T3. Statistically significant difference between QM and DPC are marked in bold. B denotes a given budget.

	B	QM	DPC$_{\varepsilon=0.05}$	DPC$_{\varepsilon=0.1}$	DPC$_{\varepsilon=0.2}$
T1	0.25	0.426	**0.299**	**0.364**	0.413
	0.50	0.488	0.464	0.464	0.464
	0.75	0.507	0.507	0.507	0.507
	1	0.507	0.507	0.507	0.507
T2	0.25	0.609	0.596	0.594	0.594
	0.50	0.747	0.743	0.742	**0.695**
	0.75	0.763	0.759	0.755	0.750
	1	0.764	0.764	0.764	0.764
T3	0.25	0.707	0.662	**0.655**	**0.655**
	0.50	0.756	0.747	0.741	**0.729**
	0.75	0.760	0.760	0.758	0.749
	1	0.760	0.760	0.760	0.760

Table 5.7: Average running time (minute) of APM, LW, DP and DPC.

	APM	LW	$\text{DP}_{\varepsilon=0.05}$	$\text{DP}_{\varepsilon=0.1}$	$\text{DPC}_{\varepsilon=0.1}$	$\text{DPC}_{\varepsilon=0.2}$
T4	1	1	127	11	151	12
T5	1	1	184	15	778	77
T6	1	1	-	-	-	520
T7	1	1	-	-	-	-
T8	1	1	-	-	-	-

Table 5.8: The numbers of queries with multiple concepts with the minimum, average and maximum numbers of concepts per query for T4, T5, T6, T7 and T8.

Taxonomy	T4	T5	T6	T7	T8
#queries	687	1578	1403	1882	2603
minimum #concepts	2	2	2	2	2
maximum #concepts	4	4	5	5	5
average #concepts	2.2	2.1	2.2	2.2	2.2

Table 5.9: Average MRR for Oracle, MQM, MLW and LW over T2. Statistically significant difference between MQM and Oracle, MQM and MLW, and MLW and LW are marked in italic, bold, and underline, respectively. B denotes a given budget.

	Uniform Cost				Random Cost				Frequency-based Cost			
B	Oracle	MQM	MLW	LW	Oracle	MQM	MLW	LW	Oracle	MQM	MLW	LW
0.1	*0.517*	0.430	0.491	0.491	*0.577*	0.491	0.465	0.454	0.569	0.569	0.569	0.569
0.2	0.593	**0.577**	0.516	0.523	0.651	0.596	0.569	0.529	0.657	0.657	0.657	0.657
0.3	*0.747*	0.602	0.600	0.526	*0.785*	0.657	0.670	0.577	*0.758*	0.682	0.682	0.682
0.4	0.826	0.796	0.796	0.563	0.826	0.791	0.791	0.618	0.829	0.829	0.829	0.829
0.5	0.850	0.821	0.821	0.576	0.846	0.832	0.832	0.626	0.832	0.832	0.832	0.829
0.6	0.859	0.852	0.852	0.576	0.862	0.837	0.837	0.663	0.832	0.832	0.832	0.832
0.7	0.865	0.865	0.865	0.823	0.862	0.852	0.852	0.768	0.832	0.832	0.832	0.832
0.8	0.865	0.865	0.865	0.832	0.862	0.862	0.862	0.835	0.832	0.832	0.832	0.832
0.9	0.865	0.865	0.865	0.865	0.862	0.862	0.862	0.858	0.861	0.861	0.861	0.832

Table 5.10: Average MRR for MLW and LW over T4, T5, T6, T7 and T8. Statistically significant differences between MLW and LW are marked in bold. B denotes a given budget.

		Uniform Cost		Random Cost		Frequency-based Cost	
	B	MLW	LW	MLW	LW	MLW	LW
T4	0.1	**0.334**	0.274	**0.448**	0.368	**0.574**	0.487
	0.2	**0.622**	0.488	**0.638**	0.487	**0.721**	0.629
	0.3	**0.748**	0.509	**0.827**	0.601	0.770	0.770
	0.4	**0.873**	0.712	**0.880**	0.712	0.773	0.784
	0.5	**0.873**	0.733	**0.887**	0.749	0.815	0.799
	0.6	0.873	0.833	**0.890**	0.813	0.845	0.799
	0.7	0.901	0.872	0.901	0.868	0.845	0.862
	0.8	0.901	0.862	0.912	0.894	0.902	0.872
	0.9	0.929	0.930	0.924	0.930	0.900	0.873
T5	0.1	**0.466**	0.430	**0.619**	0.466	**0.674**	0.536
	0.2	**0.799**	0.673	**0.803**	0.645	0.708	0.695
	0.3	0.807	0.810	**0.819**	0.729	0.719	0.728
	0.4	**0.826**	0.743	**0.828**	0.784	**0.827**	0.745
	0.5	0.834	0.811	0.833	0.819	**0.838**	0.773
	0.6	0.848	0.848	0.866	0.845	**0.852**	0.800
	0.7	0.883	0.840	0.883	0.840	0.863	0.851
	0.8	0.883	0.846	0.883	0.850	0.874	0.856
	0.9	0.883	0.869	0.883	0.862	0.883	0.876
T6	0.1	**0.794**	0.473	**0.796**	0.545	0.765	0.763
	0.2	**0.813**	0.508	**0.840**	0.561	**0.827**	0.777
	0.3	**0.883**	0.667	**0.882**	0.690	**0.853**	0.799
	0.4	**0.884**	0.795	**0.884**	0.810	**0.857**	0.813
	0.5	0.890	0.887	0.890	0.887	**0.871**	0.820
	0.6	0.892	**0.920**	0.894	0.897	0.888	0.887
	0.7	0.901	0.838	**0.912**	0.833	**0.908**	0.896
	0.8	0.932	0.906	0.932	0.913	**0.932**	0.903
	0.9	0.932	0.906	0.932	0.913	0.939	0.926
T7	0.1	0.764	0.778	0.771	0.778	**0.723**	0.605
	0.2	**0.858**	0.805	**0.856**	0.552	**0.762**	0.717
	0.3	0.860	0.848	0.860	0.852	**0.779**	0.731
	0.4	0.872	0.876	0.872	0.876	**0.826**	0.791
	0.5	**0.875**	0.768	**0.876**	0.767	0.845	0.846
	0.6	0.880	0.867	0.882	0.861	0.855	0.855
	0.7	0.889	0.895	0.889	0.895	0.866	0.866
	0.8	0.889	0.895	0.889	0.845	0.875	0.874
	0.9	0.890	0.896	0.901	0.896	0.895	0.895
T8	0.1	**0.882**	0.783	**0.880**	0.793	**0.776**	0.548
	0.2	**0.883**	0.848	**0.890**	0.844	**0.785**	0.511
	0.3	**0.887**	0.848	**0.893**	0.859	**0.791**	0.589
	0.4	**0.889**	0.867	0.895	0.876	**0.798**	0.654
	0.5	**0.894**	0.887	0.896	0.888	**0.802**	0.736
	0.6	0.896	0.896	0.898	0.892	**0.846**	0.750
	0.7	0.896	0.896	0.900	0.892	**0.881**	0.754
	0.8	0.901	0.901	0.901	0.892	0.891	0.894
	0.9	0.901	0.901	0.901	0.894	0.900	0.900

CHAPTER 6: CONCLUSIONS

6.1 SUMMARY OF CONTRIBUTIONS

In this dissertation, we have examined and addressed the problem of reducing the cost of data preparation for database analytics through two approaches: achieving design independence and selecting cost-effective conceptual designs. The summary of our contributions is as follows.

6.1.1 Design Independence

We examined a variety of issues in design independence and demonstrated the effectiveness of our proposed approaches using examples drawn from graph databases and the analytics task of similarity search. Our contributions are as follows.

- We created, implemented and evaluated a version of similarity search, namely, R-PathSim, that is design independent across a fairly general set of structural variations that preserve information content, namely, relationship-reorganizing transformations and entity-rearranging transformations. A relationship-reorganizing transformation is one in which intermediate nodes that only materialize existing relationship between pairs of entities can be added to or remove the database. For example, in an ancestry database, one could add cousin nodes between every pair of people who are already known to be cousins. An entity-rearranging transformation is one in which entities are connected through different sets of relation edges. For example, in a bibliographic database, for each paper published in a proceedings, one may present this relationship as a graph where a paper node is adjacent to a proceedings node, and the paper is adjacent to a (proceedings) year node. However, one may also represent this database by connecting the paper node to the proceedings node, and connecting the proceedings node to the year node.

- We observed that if a dataset is guaranteed to have equivalent information content when represented in two different structures, then there must be an underlying constraint that guarantees the equivalence. A simple example of this is traditional relational database normalization and denormalization, which transforms the structure of a database but preserves its content. In this case, tuple-generating dependencies or equality-generating dependencies guarantee the equivalence of the content across

structural transformations. We proved that this observation is also true for graph databases.

- For graph databases with constraints that can be expressed as conjunctive regular path queries, the current language that is used to specify relationship patterns over the data graphs is insufficient, in the sense that a relationship pattern between entities in one structural representation may have no equivalent in another representation of the same information. For instance, over a bibliographic database where all paths between authors and conferences must pass through some publications, the strength of the connection between an author and a conference depends on the number of publications the author published in that conference. The current language cannot express the relationship pattern between an author and a conference without considering the publications. However, it is possible that, in an equivalent database, authors may have an edge directly connected to a conference where they have a publication. With this limitation, some relationship patterns cannot be properly expressed over certain representations of a database. Therefore, we have presented a new relationship expressing language, namely *rich-relationship expression* (RRE), to address this issue. RRE ensures that a similarity search task expressed over a particular representation can also be expressed over other equivalent representations.

- We presented a similarity search algorithm, SR-PathSim, which extends the PathSim algorithm so that it could accept a relationship pattern in RRE when computing similarity scores. We proved that the use of a relationship pattern language with at least the expressiveness of RRE is a necessary condition for SR-PathSim to be design independent, and RRE is also sufficient when given an acyclic constraint.

- Since users are familiar with the current relationship expressing language in the form of simple paths, RRE may be hard to use for average users. We presented an algorithm that, given a simple path provided by a user, finds a set of related RREs that can be supplied to SR-PathSim. Our algorithm guarantees that SR-PathSim computed over this set of RREs will return the same answers across different structural variants, i.e., it is design independent.

- We evaluated the degree of design independence for previously existing similarity search algorithms over graph databases, focusing on a set of fairly general structural variations that preserve information content, and determined that no previously existing algorithm was design independent. In particular, the algorithms gave different answers for minor structural variants of existing databases.

- Through empirical evaluations with existing databases, we showed that SR-PathSim is as effective as or more effective than previously existing similarity search algorithms in terms of ranking quality.

6.1.2 Conceptual Design

We also addressed the case where design independence is not achievable, e.g., due to the high computational cost of producing a design independent algorithm. In this case, we would like to find the conceptual design for an unstructured dataset in the form of annotations, such that an algorithm that runs over the annotated data is effective. For instance, given a set of documents, we may decide to annotate all mentions of particular individuals with the concept *person*. These annotations can help analytics algorithms to run faster and provide better answers, particularly over queries about *person*.

Unfortunately, in general, it is too expensive to completely annotate a large set of documents with the entire set of concepts that might be helpful when answering future queries. Thus we focused on the problem of cost-effective conceptual design: given a taxonomy of relevant concepts and a fixed budget, how can we find the conceptual design (i.e., choose which concepts to annotate) that is most helpful for a future workload, without exceeding the budget? Our contributions are as follows.

- We formalized the problem of cost-effective conceptual design using taxonomies of concepts, where given a taxonomy, one would like to choose a subset of concepts in the taxonomy whose annotation will maximize the improvement in the effectiveness of query answering without exceeding the annotation budget.

- We proved that the cost-effective conceptual design problem is **NP**-hard.

- We proposed an efficient approximation algorithm (LW), and proved that it is a $O(\frac{h+\log|C|}{pr_{\min}})$-approximation for the problem with certain restrictions, e.g., a disjoint solution.

- We proposed an exact algorithm (DP) for the cost-effective conceptual design problem, and proved that it has pseudo polynomial running time.

- We proved that it is not possible to find any approximation algorithm with reasonably small approximation ratio or any pseudo-polynomial time exact algorithm for the problem when the taxonomy is a directed acyclic graph. This is unfortunate, since many real-world taxonomies are directed acyclic graphs.

- We showed that our formalization framework effectively estimates the amount by which a conceptual design improves the effectiveness of answering queries, through experiments over real-world datasets, taxonomies and queries.

- Our empirical studies also indicated that both DP and LW are effective in most cases. Overall, DP is more accurate than LW over small taxonomies and can handle the case when the cost of a concept depends on whether other concepts have also been annotated. However, LW is generally more scalable than DP and can be extended to handle queries that involve multiple concepts.

6.2 FUTURE DIRECTIONS

As directions for future work, we suggest the following open problems.

- Our work on design independence assumes that constraints are tuple-generating dependencies written in conjunctive regular path queries. It is interesting to explore and analyze the problem for the case where the languages of schema mappings and constraints are more complex than regular path queries. For instance, one may write a constraint using nested regular expressions.

- Our work focused on the problem of ensuring that analytics algorithms return the same answers across different structural variations of a given set of content. Related questions would be what guarantees we could give if the structural variation does not exactly preserve content. For example, a dataset that provides a field *name* is not exactly equivalent to one that instead provides *family name* and *given name*.

- A third interesting open problem is the question of how to determine the sensitivity of a particular analytics algorithm to structural choices. Ideally, we would like to have a metric applicable to all analytics algorithms, which could serve to quantify how design independent an algorithm is.

REFERENCES

[1] D. Pyle, *Data preparation for data mining.* morgan kaufmann, 1999.

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

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

[4] 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.

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

[6] L. Holder, D. Cook, and S. Djoko, "Substructure Discovery in the Subdue System," in *KDD*, 1994.

[7] H. Tong, B. Gallagher, C. Faloutsos, and T. Eliassi-Rad, "Fast Best-effort Pattern Matching in Large Attributed Graphs," in *KDD*, 2007.

[8] C. Faloutsos, K. S. McCurley, and A. Tomkins, "Fast Discovery of Connection Subgraphs," in *KDD*, 2004.

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

[10] X. Yan, H. Cheng, J. Han, and P. S. Yu, "Mining Significant Graph Patterns by Leap Search," in *SIGMOD*, 2008.

[11] M. Zaki, "Efficiently Mining Frequent Trees in a Forest," *TKDE*, vol. 17, no. 8, 2005.

[12] D. Chakrabarti and C. Faloutsos, "Graph mining: Laws, Generators, and Algorithms," *ACM Computeing Surveys*, vol. 38, no. 1, 2006.

[13] F. Geerts, H. Mannila, and E. Terzi, "Relational Link-based Ranking," in *VLDB*, 2004.

[14] J. M. Hellerstein, C. Ré, F. Schoppmann, D. Z. Wang, E. Fratkin, A. Gorajek, K. S. Ng, C. Welton, X. Feng, K. Li et al., "The madlib analytics library: or mad skills, the sql," *Proceedings of the VLDB Endowment*, vol. 5, no. 12, pp. 1700–1711, 2012.

[15] J. Han, J. Pei, and M. Kamber, *Data mining: concepts and techniques.* Elsevier, 2011.

[16] A. Abouzied, D. Angluin, C. Papadimitriou, J. M. Hellerstein, and A. Silberschatz, "Learning and verifying quantified boolean queries by example," in *Proceedings of the 32Nd ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, ser. PODS '13. New York, NY, USA: ACM, 2013. [Online]. Available: http://doi.acm.org/10.1145/2463664.2465220 pp. 49–60.

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

[18] W. Fan and P. Bohannon, "Information Preserving XML Schema Embedding," *TODS*, vol. 33, no. 1, 2008.

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

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

[21] J. Picado, A. Termehchy, A. Fern, and P. Ataei, "Schema independent relational learning," in *Proceedings of the 2017 ACM International Conference on Management of Data*, ser. SIGMOD '17. New York, NY, USA: ACM, 2017. [Online]. Available: http://doi.acm.org/10.1145/3035918.3035923 pp. 929–944.

[22] X. L. Dong, B. Saha, and D. Srivastava, "Less is More: Selecting Sources Wisely for Integration," *PVLDB*, vol. 6, no. 2, pp. 37–48, 2013.

[23] P. Kanani et al., "Selecting actions for resource-bounded information extraction using reinforcement learning," in *WSDM*, 2012, pp. 253–262.

[24] A. Jain, A. Doan, and L. Gravano, "Optimizing SQL Queries over Text Databases," in *ICDE*, 2008, pp. 636–645.

[25] A. Termehchy, A. Vakilian, Y. Chodpathumwan, and M. Winslett, "Which concepts are worth extracting?" in *SIGMOD*, 2014, pp. 779–790.

[26] N. C. Shu, B. C. Housel, R. W. Taylor, S. P. Ghosh, and V. Y. Lum, "Express: a data extraction, processing, and restructuring system," *ACM Transactions on Database Systems (TODS)*, vol. 2, no. 2, pp. 134–174, 1977.

[27] R. J. Miller, L. M. Haas, and M. A. Hernández, "Schema mapping as query discovery," in *VLDB*, vol. 2000, 2000, pp. 77–88.

[28] L. Popa, Y. Velegrakis, M. A. Hernández, R. J. Miller, and R. Fagin, "Translating web data," in *Proceedings of the 28th international conference on Very Large Data Bases*. VLDB Endowment, 2002, pp. 598–609.

[29] R. Fagina, P. Kolaitis, R. Miller, and L. Popa, "Data exchange: semantics and query answering," *TCS*, vol. 336, no. 1, 2005.

[30] M. Arenas and L. Libkin, "Xml data exchange: consistency and query answering," *Journal of the ACM (JACM)*, vol. 55, no. 2, p. 7, 2008.

[31] M. Arenas, P. Barcelo, L. Libkin, and F. Murlak, "Relational and xml data exchange," *Synthesis Lectures on Data Management*, vol. 2, no. 1, pp. 1–112, 2010.

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

[33] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa, "Data exchange: Semantics and query answering," in *International conference on database theory*. Springer, 2003, pp. 207–224.

[34] D. Maier, A. O. Mendelzon, and Y. Sagiv, "Testing implications of data dependencies," *ACM Transactions on Database Systems (TODS)*, vol. 4, no. 4, pp. 455–469, 1979.

[35] C. Beeri and M. Y. Vardi, "A proof procedure for data dependencies," *Journal of the ACM (JACM)*, vol. 31, no. 4, pp. 718–741, 1984.

[36] R. Fagin, P. G. Kolaitis, and L. Popa, "Data exchange: getting to the core," *ACM Transactions on Database Systems (TODS)*, vol. 30, no. 1, pp. 174–210, 2005.

[37] S. Abiteboul and O. M. Duschka, "Complexity of answering queries using materialized views," in *Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*. ACM, 1998, pp. 254–263.

[38] M. A. Hernández, R. J. Miller, and L. M. Haas, "Clio: A semi-automatic tool for schema mapping," *ACM SIGMOD Record*, vol. 30, no. 2, p. 607, 2001.

[39] D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Y. Vardi, "Rewriting of regular expressions and regular path queries," in *Proceedings of the eighteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. ACM, 1999, pp. 194–204.

[40] D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Y. Vardi, "Simplifying schema mappings," in *Proceedings of the 14th International Conference on Database Theory*. ACM, 2011, pp. 114–125.

[41] R. Hull, "Relative Information Capacity of Simple Relational Database Schemata," 1984.

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

[43] H. GarciaMolina, J. Ullman, and J. Widom, *Database Systems: The Complete Book*. Prentice Hall, 2008.

[44] C. Shi, X. Kong, Y. Huang, S. Y. Philip, and B. Wu, "Hetesim: A general framework for relevance measure in heterogeneous networks," *TKDE*, no. 10, 2014.

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

[46] A. Khan, N. Li, Z. Guan, S. Chakraborty, and S. Tao, "Neighborhood Based Fast Graph Search in Large Networks," in *SIGMOD*, 2011.

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

[48] A. Ng, A. Zheng, and M. Jordan, " Stable Algorithms for Link Analysis," in *SIGIR*, 2001.

[49] G. Ghoshal and A. Barbasi, "Ranking Stability and Super-stable Nodes in Complex Networks," *Nature Communications*, vol. 2, no. 394, 2011.

[50] A. Borodin, G. Roberts, J. S. Rosenthal, and P. Tsaparas, "Link Analysis Ranking: Algorithms, Theory, and Experiments," *ACM Trans. Inter. Tech.*, vol. 5, no. 1, 2005.

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

[52] J. Heer, J. Hellerstein, and S. Kandel, "Predictive interaction for data transformation," in *CIDR*, 2015.

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

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

[55] M. Arenas and L. Libkin, "A Normal Form for XML Documents," *TODS*, vol. 29, no. 1, 2004.

[56] C. Yu and H. V. Jagadish, "Efficient Discovery of XML Data Redundancy," in *VLDB*, 2006.

[57] C. Gutierrez, C. Hurtado, A. Mendelzon, and J. Perez, "Foundations of Semantic Web Databases," *JCSS*, vol. 77, no. 3, 2010.

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

[59] J. Hayes and C. Gutierrez, "Bipartite Graphs as Intermediate Model for RDF," in *ISWC*, 2004.

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

[61] A. Halevy, A. Rajaraman, and J. Ordille, "Data Integration: The Teenage Years," in *VLDB*, 2006.

[62] R. Hull, "Managing semantic heterogeneity in databases: A theoretical perspective," in *PODS*, 1997.

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

[64] R. J. Miller, Y. E. Ioannidis, and R. Ramakrishnan, "The Use of Information Capacity in Schema Integration and Translation," in *VLDB*, 1993.

[65] M. Arenas, "Normalization theory for xml," *SIGMOD Rec.*, vol. 35, no. 4, pp. 57–64, Dec. 2006. [Online]. Available: http://doi.acm.org/10.1145/1228268.1228284

[66] C. E. Dyreson and S. S. Bhowmick, "Querying xml data: As you shape it," in *2012 IEEE 28th International Conference on Data Engineering*, April 2012, pp. 642–653.

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

[68] Y. Chodpathumwan, A. Aleyasen, A. Termehchy, and Y. Sun, "Towards representation independent similarity search over graph databases," in *CIKM*, 2016. [Online]. Available: http://doi.acm.org/10.1145/2983323.2983673

[69] I. Boneva, A. Bonifati, and R. Ciucanu, "Graph data exchange with target constraints," in *EDBT/ICDT Workshop GraphQ*, ser. PODS '17, 2015, pp. 171–176.

[70] C. Beeri and M. Y. Vardi, "A proof procedure for data dependencies," *J. ACM*, vol. 31, no. 4, Sep. 1984. [Online]. Available: http://doi.acm.org/10.1145/1634.1636

[71] N. Francis and L. Libkin, "Schema mappings for data graphs," in *PODS*, ser. PODS '17. New York, NY, USA: ACM, 2017. [Online]. Available: http://doi.acm.org/10.1145/3034786.3056113 pp. 389–401.

[72] I. Cruz, A. Mendelzon, and P. Wood, "A graphical query language supporting recursion," in *SIGMOD*, 1987, p. 323–330.

[73] P. T. Wood, "Query languages for graph databases," *SIGMOD Rec.*, vol. 41, no. 1, Apr. 2012. [Online]. Available: http://doi.acm.org/10.1145/2206869.2206879

[74] R. Fagin, "Inverting schema mappings," *ACM Trans. Database Syst.*, vol. 32, no. 2, 2007.

[75] A. Hernich, L. Libkin, and N. Schweikardt, "Closed world data exchange," *ACM Trans. Database Syst.*, vol. 36, no. 2, pp. 14:1–14:40, June 2011. [Online]. Available: http://doi.acm.org/10.1145/1966385.1966392

[76] M. Y. Vardi, "On decomposition of relational databases," in *FOCS*, Nov 1982, pp. 176–185.

[77] L. Katz, "A new status index derived from sociometric analysis," *Psychometrika*, vol. 18, no. 1, pp. 39–43, 1953.

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

[79] R. Ramakrishnan and J. Gehrke, *Database management systems*. McGraw Hill, 2000.

[80] R. Fagin, P. G. Kolaitis, L. Popa, and W.-C. Tan, "Schema mapping evolution through composition and inversion," in *Schema matching and mapping*. Springer, 2011, pp. 191–222.

[81] S. Abiteboul, I. Manolescu, P. Rigaux, M. Rousset, and P. Senellart, *Web Data Management*. Cambridge University Press, 2011.

[82] J. Mork, D. Demner-Fushman, S. Schmidt, and A. Aronson, "Recent enhancements to the nlm medical text indexer," in *CLEF (Working Notes)*, 2014, pp. 1328–1336.

[83] K. Liu et al., "Meshlabeler: improving the accuracy of large-scale mesh indexing by integrating diverse evidence," *Bioinformatics*, vol. 31, no. 13, pp. i339–i347, 2015.

[84] W. Hua, Z. Wang, H. Wang, K. Zheng, and X. Zhou, "Short text understanding through lexical-semantic analysis," in *ICDE*, 2015, pp. 495–506.

[85] W. Wu, H. Li, H. Wang, and K. Zhu, "Probase: A probabilistic taxonomy for text understanding," in *SIGMOD*, 2012, pp. 481–492.

[86] M. Anderson et al., "Brainwash: A data system for feature engineering," in *CIDR*, 2013.

[87] P. Gulhane et al., "Web-scale information extraction with vertex," in *ICDE*, 2011, pp. 1209–1220.

[88] L. Chiticariu, Y. Li, S. Raghavan, and F. R. Reiss, "Enterprise information extraction: Recent developments and open challenges," in *SIGMOD*, 2010, pp. 1257–1258.

[89] W. Shen, P. DeRose, R. McCann, A. Doan, and R. Ramakrishnan, "Toward best-effort information extraction," in *SIGMOD*, 2008, pp. 1031–1042.

[90] J. Huang and C. Yu, "Prioritization of Domain-Specific Web Information Extraction," in *AAAI*, 2010.

[91] A. McCallum, "Information Extraction: Distilling Structured Data From Unstructured Text," *ACM Queue*, pp. 48–57, 2005.

[92] M. Anderson, M. Cafarella, Y. Jiang, G. Wang, and B. Zhang, "An Integrated Development Environment for Faster Feature Engineering," *PVLDB*, vol. 7, no. 13, pp. 1657–1660, 2014.

[93] F. Suchanek et al., "Yago: A core of semantic knowledge unifying wordnet and wikipedia," in *WWW*, 2007, pp. 697–706.

[94] N. Dalvi, R. Kumar, B. Pang, R. Ramakrishnan, A. Tomkins, P. Bohannon, S. Keerthi, and S. Merugu, "A web of concepts," in *PODS*, 2009, pp. 1–12.

[95] A. Doan, R. Ramakrishnan, and S. Vaithyanathan, "Managing information extraction: state of the art and research directions," in *SIGMOD*, 2006, pp. 799–800.

[96] C. De Sa, A. Ratner, C. Ré, J. Shin, F. Wang, S. Wu, and C. Zhang, "Deepdive: Declarative knowledge base construction," *SIGMOD Record*, vol. 45, no. 1, pp. 60–67, 2016.

[97] R. Fagin, B. Kimelfeld, F. Reiss, and S. Vansummeren, "Spanners: A formal framework for information extraction," in *PODS*, 2013, pp. 37–48.

[98] P. Ipeirotis, E. Agichtein, P. Jain, and L. Gravano, "To Search or to Crawl? Towards a Query Optimizer for TextCentric Tasks," in *SIGMOD*, 2006, pp. 265–276.

[99] E. Agichtein and L. Gravano, "Querying Text Databases for Efficient Information Extraction," in *ICDE*, 2003, pp. 113–124.

[100] A. Jain, P. Ipeirotis, and L. Gravano, "Building Query Optimizers for Information Extraction: The SQoUT Project," *SIGMOD Record*, vol. 37, no. 4, pp. 28–34, 2008.

[101] D. Calvanese, G. D. Giacomo, D. Lembo, M. Lenzerini, and R. Rosati, "Conceptual Modeling for Data Integration," in *Conceptual Modeling: Foundations and Applications*, 2009.

[102] B. Ding, H. Wang, R. Jin, J. Han, and Z. Wang, "Optimizing Index for Taxonomy Keyword Search," in *SIGMOD*, 2012.

[103] M. Lenzerini, "Ontology-based Data Management," in *CIKM*, 2011.

[104] M. Clarka et al., "Automatically Structuring Domain Knowledge from Text: An Overview of Current Research," *Information Processing & Management*, vol. 48, no. 3, pp. 552–568, 2012.

[105] S. Chakrabarti, K. Puniyani, and S. Das, "Optimizing Scoring Functions and Indexes for Proximity Search in Type-annotated Corpora," in *WWW*, 2007, pp. 717–726.

[106] J. Chu-Carroll et al., "Semantic Search via XML Fragments: a High-Precision Approach to IR," in *SIGIR*, 2006, pp. 445–452.

[107] J. Pound, I. Ilyas, and G. Weddell, "Expressive and Flexible Access to Web-Extracted Data: A Keyword-based Structured Query Language," in *SIGMOD*, 2010.

[108] M. Sanderson, "Ambiguous queries: Test collections need more sense," in *SIGIR*, 2008, pp. 499–506.

[109] S. Dill et al., "Semtag and seeker: Bootstrapping the semantic web via automated semantic annotation," in *WWW*, 2003, pp. 178–186.

[110] B. Kimelfeld, "Database principles in information extraction," in *PODS*, 2014, pp. 156–163.

[111] S. Sarawagi, "Information extraction," *Foundations and Trends® in Databases*, vol. 1, pp. 261–377, 2008.

[112] C.-H. Chang, M. Kayed, M. R. Girgis, and K. F. Shaalan, "A survey of web information extraction systems," *TKDE*, vol. 18, pp. 1411–1428, 2006.

[113] O. Deshpande, D. Lamba, M. Tourn, S. Das, S. Subramaniam, A. Rajaraman, V. Harinarayan, and A. Doan, "Building, Maintaining, and Using Knowledge Bases: A Report from the Trenches," in *SIGMOD*, 2013, pp. 1209–1220.

[114] A. Doan, J. F. Naughton, R. Ramakrishnan, A. Baid, X. Chai, F. Chen, T. Chen, E. Chu, P. DeRose, B. J. Gao, C. Gokhale, J. Huang, W. Shen, and B. Vuong, "Information extraction challenges in managing unstructured data," *SIGMOD Record*, vol. 37, no. 4, pp. 14–20, 2008.

[115] L. Chiticariu, R. Krishnamurthy, Y. Li, S. Raghavan, F. R. Reiss, and S. Vaithyanathan, "Systemt: An algebraic approach to declarative information extraction," in *ACL*, 2010, pp. 128–137.

[116] W. Shen, A. Doan, J. F. Naughton, and R. Ramakrishnan, "Declarative information extraction using datalog with embedded extraction predicates," *PVLDB*, pp. 1033–1044, 2007.

[117] L. Chiticariu, Y. Li, and F. R. Reiss, "Rule-based information extraction is dead! long live rule-based information extraction systems!" in *EMNLP*, 2013, pp. 827–832.

[118] S. Gupta and C. D. Manning, "Improved pattern learning for bootstrapped entity extraction," in *CoNLL*, 2014, pp. 98–108.

[119] S. Gupta, D. L. MacLean, J. Heer, and C. D. Manning, "Research and applications: Induced lexico-syntactic patterns improve information extraction from online medical forums," *JAMIA*, vol. 21, no. 5, pp. 902–909, 2014.

[120] R. Nallapati and C. D. Manning, "Legal docket-entry classification: Where machine learning stumbles," in *EMNLP*, 2008, pp. 438–446.

[121] M. Mintz, S. Bills, R. Snow, and D. Jurafsky, "Distant supervision for relation extraction without labeled data," in *ACL*, 2009, pp. 1003–1011.

[122] A. Ratner et al., "Data programming: Creating large training sets, quickly," *arXiv:1605.07723*, 2016.

[123] H. Isozaki and H. Kazawa, "Efficient support vector classifiers for named entity recognition," in *COLING*, 2002, pp. 1–7.

[124] S. Satpal, S. Bhadra, S. Sellamanickam, R. Rastogi, and P. Sen, "Web information extraction using markov logic networks," in *KDD*, 2011, pp. 1406–1414.

[125] D. Z. Wang, M. J. Franklin, M. Garofalakis, J. M. Hellerstein, and M. L. Wick, "Hybrid in-database inference for declarative information extraction," in *SIGMOD*, 2011, pp. 517–528.

[126] T. Furche, J. Guo, S. Maneth, and C. Schallhart, "Robust and noise resistant wrapper induction," in *SIGMOD*, 2016, pp. 773–784.

[127] B. Boehm, C. Abts, and S. Chulan, "Software Development Cost Estimation Approaches: A Survey," *Annals of Software Engineering*, vol. 10, pp. 177–205, 2000.

[128] D. Downey, O. Etzioni, and S. Soderland, "A probabilistic model of redundancy in information extraction," in *IJCAI*, 2005.

[129] V. V. Vazirani, *Approximation algorithms*. Springer Science & Business Media, 2013.

[130] A. Arulselvan, "A note on the set union knapsack problem," *Discrete Applied Mathematics*, vol. 169, pp. 214–218, 2014.

[131] S. Khot, "Ruling out PTAS for Graph Min-bisection, Densest Subgraph and Bipartite Clique," in *FOCS*, 2004, pp. 136–145.

[132] A. Bhaskara, M. Charikar, A. Vijayaraghavan, V. Guruswami, and Y. Zhou, "Polynomial Integrality Gaps for Strong SDP Relaxations of Densest K-subgraph," in *SODA*, 2012, pp. 388–405.

[133] S. Arora, R. Manokaran, D. Moshkovitz, and O. Weinstein, "Inapproximability of densest κ-subgraph from average-case hardness," people.csail.mit.edu/dmoshkov/papers, 2011.

[134] P. Manurangsi, "Almost-polynomial ratio ETH-hardness of approximating densest k-subgraph," *CoRR*, vol. abs/1611.05991, 2016. [Online]. Available: http://arxiv.org/abs/1611.05991

[135] E. Demidova, X. Zhou, I. Oelze, and W. Nejdl, "Evaluating evidences for keyword query disambiguation in entity centric database search," in *DEXA*, 2010, pp. 240–247.

[136] T. Rekatsinas, X. L. Dong, and D. Srivastava, "Characterizing and selecting fresh data sources," in *SIGMOD*, 2014, pp. 919–930.