Web Data Integration for Non-Expert Users

by

Ahmed El-Roby

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 2018

© Ahmed El-Roby 2018

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

External Examiner:	Renée J. Miller Professor Department of Computer Science University of Toronto
Supervisor:	Ashraf Aboulnaga Adjunct Associate Professor David R. Cheriton School of Computer Science University of Waterloo
Internal Member:	Tamer Özsu Professor David R. Cheriton School of Computer Science University of Waterloo
Internal Member:	Grant Weddell Associate Professor David R. Cheriton School of Computer Science University of Waterloo
Internal-External Member:	Wojciech Golab Assistant Professor Department of Electrical and Computer Engineering University of Waterloo

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Today, there is an abundance of structured data available on the web in the form of RDF graphs and relational (i.e., tabular) data. This data comes from heterogeneous sources, and realizing its full value requires integrating these sources so that they can be queried together. Due to the scale and heterogeneity of the data sources on the web, integrating them is typically an automatic process. However, automatic data integration approaches are not completely accurate since they infer semantics from syntax in data sources with a high degree of heterogeneity. Therefore, these automatic approaches can be considered as a first step to quickly get reasonable quality data integration output that can be used in issuing queries over the data sources. A second step is refining this output over time while it is being used. Interacting with the data sources through the output of the data integration system and refining this output requires expertise in data management, which limits the scope of this activity to power users and consequently limits the usability of data integration systems.

This thesis focuses on helping non-expert users to access heterogeneous data sources through data integration systems, without requiring the users to have prior knowledge of the queried data sources or exposing them to the details of the output of the data integration system. In addition, the users can provide feedback over the answers to their queries, which can then be used to refine and improve the quality of the data integration output. The thesis studies both RDF and relational data. For RDF data, the thesis focuses on helping non-expert users to query heterogeneous RDF data sources, and utilizing their feedback over query answers to improve the quality of the interlinking between these data sources. For relational data, the thesis focuses on improving the quality of the mediated schema for a set of relational data sources and the semantic mappings between these sources based on user feedback over query answers.

Acknowledgements

First, I would like to express my sincere gratitude to my supervisor, Prof. Ashraf Aboulnaga, whom I learned most of my research skills from. Ashraf has been always patient, flexible, and inspiring in his mentorship. I literally learned new things each time we met on both academic and personal levels. I will be eternally in debt to him.

I would also like to thank the people I collaborated with at the Qatar Computing Research Institute and Carnegie Mellon University. Specifically, I would like to thank Prof. Andy Pavlo who, despite me working with him for only three months, influenced me greatly. He has been a great manager who got the best out of his team despite their different backgrounds and skills. He has also been a pleasure to work and hang out with.

I am also very grateful to my thesis committee members, Prof. Tamer Özsu and Prof. Grant Weddell, who gave me great feedback and advice on how to improve my PhD research after Comp-II, and also Prof. Renée J. Miller and Prof. Wojciech Golab for taking the time and effort to read and offer suggestions to further improve this thesis.

I would like to thank my mother-in-law, Hala, who believed in me and was always there for our small family, supporting us in difficult times during my PhD. I would also like to thank my brothers-in-law, Marawan and Mohammed, for the fun times we had together, and my sister Sara for being such a great sister.

I would especially thank my parents without whom I would have not been where I am today. They gave me so much selflessly, expecting nothing in return. I am forever in debt for everything they did for me. I would also like to thank my late father-in-law who was the kindest person I have ever met. Looking up to him has inspired me on how to be a better person.

Finally, this dissertation would have not been possible without the love of my life, my wife, Yara. Yara has always encouraged me, motivated me, and supported me during the happy moments and especially during the sad and tough ones. I believe that having her by my side, achieving our dreams together is a great blessing. I especially thank my child Yahya, without whom, this thesis would have been defended a year earlier.

Dedication

This thesis is dedicated to my amazing child Yahya, my beloved wife Yara, my wonderful parents Hassan and Aida, and my late father-in-law, Moustafa.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

In recent years, there has been a big increase in the amount of structured data available on the web. This structured data exists in different formats like RDF and relational tables. Large RDF data sets are known as *knowledge bases*, and they span many domains. Some of these knowledge bases are general-purpose [15, 28, 63], and others focus on specific domains such as movies[1], geographic information[2], music[3], and city data[4]. Publishing these RDF data sets on the web was encouraged by the Semantic Web principle of making data on the web readable and processable directly by machines [22]. These data sets are graph-structured, and are interlinked via edges that point from one data set to another, forming a massive graph known as the *Linked Open Data (LOD) cloud* [7]. Today, the LOD cloud contains almost 150 billion triples from almost 1000 data sets[5].

Just like RDF data is expanding on the web, relational (i.e., table-structured) data on the web is also expanding. The web contains a massive amount of crawlable, relational data. This data is extracted from various sources, such as HTML tables, web forms, and on-line spreadsheets [1, 32, 61, 72, 121, 143]. Each of these data sources can be viewed as a *relational data source* (i.e., a table with a schema). The number of such sources is very large and continuously increasing. For example, 125M HTML tables were extracted in 2015 [55].

This wealth of structured information available in both RDF and relational formats can be extremely valuable to users and applications in diverse domains. To realize the full

[1]http://www.linkedmdb.org
[2]http://www.geonames.org
[3]http://musicbrainz.org
[4]http://www.data.gov/opendatasites
[5]http://stats.lod2.eu

value of the data, users should be able to query data from multiple sources at the same time, which requires some form of data integration. At web scale, most data integration techniques are fully automatic. They do not rely on a data architect to guide the data integration process, but rather infer the best way to integrate data sources based on the features of these sources. Fully automatic data integration techniques are best effort in nature and can have errors in their output. Thus, they could benefit from guidance or feedback from end users. This thesis is about the interaction between end users, specifically non-expert end users, and data integration systems.

At a high level, the focus of this thesis is on (a) helping non-expert users to easily query heterogeneous data sets which typically are the output of data integration techniques, and (b) using the feedback of non-expert users over the answers to their queries to improve the quality of the output of data integration systems. The thesis comprises three technical contributions, two dealing with RDF data and one dealing with relational data. The first contribution is improving the quality of links between heterogeneous RDF data sets based on user feedback. The second is facilitating the process of querying multiple heterogeneous RDF data sets. The third is improving the quality of the output of relational data integration systems based on user feedback. An overview of these three contributions is presented next.

1.1 Improving the Interlinking of RDF Data Sets

Publishing RDF data sets individually on the web is useful, but not sufficient to realize the full potential of linked open data. The true power of linked data is realized only when the data sets are linked to each other so that their semantic properties can be fully exploited [27]. Linking data sets is crucial for answering queries that cannot be answered using one RDF data set alone. For example, consider the query "Find all New York Times articles about the no. 1 squash player in the world". Articles about people are available from the New York Times RDF knowledge base. However, the identity of the "No. 1 squash player" is unknown, since this information cannot be found in the New York Times data set. Another data set like DBpedia could have the information that "Mohamed El Shorbagy" is the "No. 1 squash player". One can use the Web Ontology Language (OWL)[6] to define an *owl:sameAs* relation linking the two entities representing "Mohamed El Shorbagy" from both data sets. This relation indicates that the two entities refer to the same individual, and enables the system to return all articles about "Mohamed El Shorbagy" from the New York Times data set.

[6]http://www.w3.org/2001/sw/wiki/OWL

Some work has been done on aligning RDF schemas [16, 70] and automatically linking equivalent entities from different data sets [25, 65, 76]. That work aims to automatically introduce *owl:sameAs* links between two data sets. However, automatic linking approaches are best effort in nature, with no guarantees on the quality of the output. They try to automatically infer the semantics of the data based on its syntax, which is a difficult task in the absence of human guidance. As such, automatic linking of RDF data sets can greatly benefit from user feedback on the quality of the generated links.

This thesis introduces *ALEX* (Automatic Link EXploration in linked data) [56, 57], a system that improves the quality of links between linked RDF data sets by utilizing feedback that users provide on answers to their queries (helping users issue queries on linked RDF data sets will be discussed in the next section). ALEX allows users to issue queries over multiple RDF data sets that are linked using any automatic linking approach. These queries can be answered using one or more data sets. When the query answer is produced using links between multiple data sets, ALEX gives the user the opportunity to approve or reject the query answers (i.e., mark the answers as correct or incorrect). ALEX considers the approval/rejection of a query answer as an approval/rejection of the link(s) used to produce this answer, and it uses this feedback to improve the quality of links. ALEX uses a stochastic reinforcement learning technique that generalizes the feedback provided by the user and is resilient to errors in this feedback. Errors in user feedback (approving a wrong answer or rejecting a correct answer) can arise due to errors in the data or errors made by the user.

1.2 Answering Questions on RDF Data Sets

Utilizing user feedback over query answers to improve the interlinking of RDF data sets is useful, but it requires the user to be able to issue queries on these data sets, and querying RDF data is far from trivial. This section introduces some challenges to querying a large number of heterogeneous RDF data sets and gives a high-level overview of the approach proposed by the thesis to face these challenges.

Answering questions over RDF data generally follows one of two approaches: (a) natural language queries, and (b) structured querying using SPARQL [5], the standard query language for RDF. Natural language approaches rely on keyword search or more complex question answering techniques. These approaches are convenient and easy to use, and they find accurate answers for simple questions such as *"How many people live in New York?"*. Questions like this one that ask about a specific property of an entity (e.g., population of a city) are termed *factoid questions*. Such questions can be answered by a simple structured search that can be constructed effectively by natural language approaches.

However, the RDF data that makes up the LOD cloud is not limited to answering simple questions. This data can be used to answer complex questions that require complex structured searches. These complex structured searches are valuable in many applications such as searching for entities, business analytics, and recommendation systems. Natural language approaches are not effective at constructing such complex structured searches. Instead, complex structured searches are better expressed directly by the user using SPARQL queries. It is a common practice for data sources in the LOD cloud to provide *SPARQL endpoints* that allow users and applications to issue SPARQL queries on the RDF data that they contain[7].

To illustrate the need for SPARQL, consider the question *"How many scientists graduated from an Ivy League university?"* This question was used in the QALD-5 competition in 2015 [132]. QALD is an annual competition for Question Answering over linked data, and this question was not answered by any of the natural language systems that participated in QALD-5. This is not surprising since the question involves concepts such as "scientist", "graduated", and "Ivy League university" that are not easy to map to a structured search over the queried data set (DBpedia), in addition to requiring a count of the results. On the other hand, it is possible to construct a SPARQL query to answer the question. For example, the following query over the SPARQL endpoint of DBpedia will find the required answer:

```
PREFIX res: <http://dbpedia.org/resource/>
PREFIX dbo: <http://dbpedia.org/ontology/>
SELECT DISTINCT count (?uri) WHERE {
 ?uri rdf:type dbo:Scientist.
 ?uri dbo:almaMater ?university.
 ?university dbo:affiliation res:Ivy_League.
}
```

To be able to correctly compose a query such as this one, the user needs to know the structure of the data set, the vocabulary used to represent different concepts, and the literals used in the data set including their data types and format. For example, the user needs to know that "scientist" is an `rdf:type` associated with the persons thet need to be found, "graduated" is represented by the `almaMater` predicate that links a person to

[7]e.g., http://dbpedia.org/sparql for DBpedia.

the university they graduated from, and “Ivy League” is an `affiliation` of a university. Achieving this level of knowledge about a data set can be difficult even for experienced users given the massive scale and diverse vocabulary of the LOD cloud. Due to the size and heterogeneity of the data sets in the LOD cloud, there is a great diversity in the vocabularies used. To illustrate the diversity of the vocabulary in the LOD cloud, consider that DBpedia alone has over 3K distinct predicates. Thus, it is quite likely that a user would need to construct SPARQL queries on data whose structure and vocabulary she does not know in full, for example when querying a new data set. This is a challenging task, and one of the goals of this thesis is to help users with this task.

This thesis introduces *Sapphire* [59], an interactive tool aimed at helping users write syntactically and semantically correct SPARQL queries on RDF data sets they do not have any prior knowledge about. Sapphire is aimed at users who have a technical background but are not necessarily SPARQL experts, e.g., data scientists or application developers. Thus, Sapphire makes no attempt to “shield” its users from the syntax of SPARQL, but rather helps them construct valid SPARQL queries with ease.

1.3 Refining Relational Web Data Integration Systems

The third technical contribution of the thesis shifts the focus from RDF to relational data sources on the web. As mentioned earlier, the web has a massive amount of table-structured data sources that can be viewed as relations. Users and application programs can greatly benefit from having a unified interface to simultaneously query multiple heterogeneous relational data sources. Data integration systems for relational data provide such a unified interface by automatically building a relational mediated schema for the data sources along with semantic mappings between the schemas of the data sources and the mediated schema. Despite substantial research progress in the field of data integration, web data sources such as web tables still represent a significant challenge for traditional data integration. One reason is the scale of the problem, since web data integration typically deals with a large number of data sources. The more important reason is that these data sources are semantically heterogeneous to a high degree. Data sources on the web deal with many different real world domains (e.g., sports, movies, finance, etc.), and the representation of table names, attribute names, and data values is based on human language and can therefore be ambiguous, inconsistent, and varying. Even if the data sources are clustered into individual domains [73, 95], the semantic gap among the data sources can still be very high [67].

A good way to address the challenge of data integration on the web is to recognize that many applications do not require full integration of the data sources that they use in order to provide useful services. This encourages a *pay-as-you-go* approach to integrating web data sources [66]. The pay-as-you-go approach involves two phases: *setup* and *refinement*. In the setup phase, the system creates: (1) a mediated schema or possibly several schemas, and (2) mappings between the schemas of the data sources and the mediated schema(s). Since setup is done fully automatically, the mediated schema and mappings will likely contain semantic errors. The pay-as-you-go philosophy requires the mediated schema and mappings to be *refined* during use based on feedback from the user. The typical way a user would use the mediated schema is to issue queries against it and receive answers to these queries from the data sources. Thus, the natural way for a user to provide feedback is to indicate the correctness of the tuples that she sees in the answers to her queries. Surprisingly, most of the prior work on the refinement step has been decoupled from the querying process. Instead of requiring a user to provide feedback on the answers to her queries, most prior refinement approaches expose the user to the details of the mediated schema and the source schemas used to answer her queries, and enable her to directly modify the mediated schema and mappings. Thus, instead of the system automatically fixing its mistakes based on feedback from the user, the system presents these mistakes to the user and asks her to fix them. Such approaches implicitly assume that the user is a database expert, which may not be true in practice. In addition, the focus in prior work has been on refining the mappings, assuming that the target schema (mediated schema) is correct. While this assumption is valid when the mediated schema is manually created (and therefore of high quality), this is not a practical assumption for pay-as-you-go data integration, where the mediated schema is automatically generated.

This thesis introduces *UFeed* [58], a pay-as-you-go data integration system that addresses the problems with prior refinement approaches. To the best of my knowledge, UFeed is the first system that fixes both the mediated schema and mappings based on user feedback over query answers. UFeed accepts as input a mediated schema and mappings between each source schema and this mediated schema. UFeed does not make any assumptions about the techniques used to create the initial mediated schema and mappings (the setup phase), and can work with any technique for schema matching and mapping.

1.4 Thesis Statement

After presenting an overview of the individual technical contributions that make up the thesis, I now present my thesis statement:

Data integration systems can be made more useful to non-expert end users by facilitating the interaction of the users with the output of these systems, and by using feedback information learned from this interaction to improve the quality of this output.

1.5 Thesis Outline

The thesis is structured as follows:

- Chapter 2 presents the preliminaries about RDF and SPARQL needed for the rest of the thesis.
- Chapter 3 discusses the first contribution of the thesis, improving the quality of *owl:sameAs* links based on user feedback over query answers using [56, 57].
- Chapter 4 presents the second contribution the thesis, facilitating the querying of RDF data sets [59].
- Chapter 5 discusses the third contribution of the thesis, improving the quality of relational mediated schema and schema mappings based on user feedback over query answers [58].
- Chapter 6 concludes and discusses future research directions.

  The related work and experimental evaluation of Chapters 3, 4, and 5 are covered within each of the chapters.

Chapter 2

Preliminaries on the Resource Description Framework (RDF) and SPARQL

This chapter presents preliminary information about RDF and SPARQL, which is required for Chapters 3 and 4.

2.1 Resource Description Framework (RDF)

The Resource Description Framework (RDF) [2] is a framework for representing information on the web, which follows a simple graph-based data model. An RDF graph consists of a set of subject-predicate-object *triples*. Each triple represents a directed edge from a vertex representing the subject to a vertex representing the object. The label on the edge is the predicate, which represents the relationship between the subject and the object.

Resources (instances, classes, or relations) in the RDF graph are identified by Uniform Resource Identifiers (URIs). Literals in RDF are strings of characters, and only objects can be literals. Formally, assume two countably infinite disjoint sets, U for URIs and L for literals. Any RDF triple $(s, p, o) \in U \times U \times (U \cup L)$. The following is an example of a set of triples about the movie "The Godfather" from DBpedia in Turtle format [6]:

```
Prefix dbp: <http://dbpedia.org/property/> .
Prefix dbo: <http://dbpedia.org/ontology/> .
Prefix res: <http://dbpedia.org/resource/> .
res:The_Godfather
  dbp:name "The Godfather"@en ;
  dbo:director res:Francis_Ford_Coppola ;
  dbo:distributor res:Paramount_Pictures .
```

This example shows three triples. The subject is the URI representing the movie. The predicates are URIs representing relations to other entities or literals (name, director, and distributor). One object is a literal (the name of the movie "The Godfather"), and the other two are URIs of other entities.

This thesis focuses on RDF data that is published as linked data on the web. While it is true that the RDF format is very commonly used to publish linked data, note for completeness that linked data does not necessarily mandate RDF. Linked data can be published in any format as long as it follows the following very broad linked data principles [21]:

1. Use URIs as names for things.
2. Use HTTP URIs so that people can look-up those names.
3. When someone looks-up a URI, provide useful information.
4. Include links to other URIs, so that it is easier to navigate between resources.

2.2 The SPARQL Query Language

With the adoption of RDF as an abstraction to publish data comes the challenge of providing an interface or query language for accessing this data. SPARQL is the standard query language for RDF [5]. The basic building blocks of a SPARQL query are *graph patterns* and *conditions* applied over these patterns. Formally, assume another countably infinite set V for variables, which start with a question mark. This set is disjoint from U and L from the previous section.

A condition is inductively defined as follows:

- if $?x, ?y \in V$ and $a \in (U \cup L)$, then $?x \ op \ a$ and $?x \ op \ ?y$, where *op* is a binary operator, are conditions,
- if R_1 and R_2 are conditions, then $(\neg R_1)$, $(R_1 \vee R_2)$, and $(R_1 \wedge R_2)$ are conditions.

A graph pattern is inductively defined as follows:

- a triple from $(U \cup V) \times (U \cup V) \times (U \cup V \cup L)$ is a graph pattern,
- if P_1 and P_2 are graph patterns, then (P_1 AND P_2), (P_1 UNION P_2), and (P_1 MINUS P_2) are graph patterns (see [5] for a definition of these operators),
- if P is a graph pattern and R is a condition, then (P FILTER R) is a graph pattern.

The goal of SPARQL query processing is to match the graph patterns in the query to the queried data sets to find values that can be associated with the variables in the graph pattern. These values are the answers to the SPARQL query. If a pattern has a FILTER, then only the matches that satisfy the FILTER condition are retained. SPARQL also has other features such as solution modifiers (e.g., DISTINCT), aggregations (e.g., COUNT, GROUP BY, and HAVING), and optional pattern matching (the OPTIONAL keyword). See [5] for a full specification of the SPARQL query language.

Following is an example of a SPARQL query that finds the name of the director of the movie "The Godfather" and other movies by the same director, given that they were distributed by the same studio:

```
PREFIX dbp: <http://dbpedia.org/property/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dbo: <http://dbpedia.org/ontology/>
SELECT DISTINCT ?directorName ?movies WHERE {
  ?movie dbp:name "The Godfather"@en.
  ?movie dbp:director ?director.
  ?movie dbo:distributor ?godFatherDistr.
  ?director foaf:name ?directorName.
  ?movies dbp:director ?director.
  ?movies dbo:distributor ?otherDistr.
  FILTER (?godFatherDistr = ?otherDistr).
}
```

Chapter 3

ALEX: Automatic Link Exploration Based on User Feedback

This chapter presents the first contribution of the thesis, namely improving the quality of *owl:sameAs* links between RDF data sets in the LOD cloud based on user feedback over query answers. The LOD cloud contains a wealth of structured information that can be extremely useful to users and applications in diverse domains. However, for the LOD to be useful, the data sets in this cloud need to be (a) accurately and efficiently linked, and (b) easily usable to find answers to questions. This thesis addresses these two problems by allowing non-expert users to easily issue queries over the LOD cloud, and by utilizing their feedback over query answers to improve the quality of the links between RDF data sets. Helping the user to issue queries over the LOD cloud will be discussed in Chapter 4. In this chapter, the focus is on how ALEX improves the quality of *owl:sameAs* links between RDF data sets based on user feedback over query answers.

ALEX starts with two RDF data sets and a set of automatically generated links between these data sets. These links can be produced using any automatic linking algorithm, and they are referred to in this thesis as *candidate links*. When a user issues queries on the two data sets and receives answers, she can provide feedback on the answers that are based on the links. If an answer is marked correct (also referred to as *positive feedback* or *approving the link*), the link used to produce this answer is assumed to be correct. If an answer is marked incorrect (also referred to as *negative feedback* or *rejecting the link*), the link is assumed to be incorrect.

ALEX removes incorrect links rejected by the user. However, the main focus of ALEX is to discover new links that are *similar* to the links approved by the user. The way ALEX discovers similar links is as follows: An entity in an RDF data set is represented by a URI. Each entity has a set of attributes (RDF predicates), and values corresponding to these attributes (RDF objects). A link between two entities from different data sets is represented as a set of *features* made up of the predicates of the two entities. A feature is a pair of predicates where the first predicate comes from the first entity and the second comes from the second entity. Each feature has a value, which is the *similarity score* of the values of the two objects associated with the predicates. When a user approves a link by approving a query answer based on this link, ALEX chooses one feature and finds new candidate links for which the value of this feature (i.e., the similarity score) is within a (narrow) range around the value of the feature of the approved link. We refer to this as *exploring around the feature*.

An important question that ALEX needs to answer is: *Which feature to explore around for a given approved link?*. Exploring around a random feature is not effective since it incorrectly assumes that all features are of equal importance in determining whether the entities are equivalent. At the same time, ALEX has no prior knowledge of which features may be important, and the best feature to explore around can depend on the link being explored. For example, the title of the two entities may be a good feature to explore around for some data set, while the phone number may return better results for another. Thus, ALEX needs to (a) identify features that can distinctively differentiate between equivalent and non-equivalent entities, and (b) find the distinctive scores for the distinctive features such that this score differentiates between correct and incorrect links. To guide the answer to these questions, Section 3.4 presents an analysis of several pairs of real RDF linked data sets in the LOD cloud that aims to identify the characteristics of correct links. The data sets vary in size and represent diverse domains.

I propose that identifying the features to explore around can be solved using *Monte Carlo reinforcement learning* methods [123], where ALEX can *learn* which feature to explore around for different links. Using the terminology of reinforcement learning, ALEX aims at learning from interacting with the environment in order to learn the best action to take (feature to explore around) in order to maximize the reward (future positive user feedback over query answers). The feature is chosen by ALEX using a policy that is iteratively improved.

ALEX also includes several optimizations that help it to converge faster. These optimizations include reducing the search space of links, partitioning data to exploit parallelism, using a blacklist to prevent known incorrect links from being proposed, and rolling back actions to undo actions that result in exploring many incorrect links.

Experiments conducted over real-world data sets show that ALEX can improve the quality of the initial set of candidate links, while not exposing the end-user to a large number of incorrect answers. The experiments also show that ALEX converges in a reasonable amount of time and is robust to the changes in parameters values.

The contributions of this chapter are as follows:

- To the best of my knowledge, ALEX is the first system to bridge the gap between automatic linking of data sets on one side and querying linked data on the other side by leveraging user feedback to discover new links between entities without prior knowledge of the data sets or how they were originally linked.
- Analyzing *owl:sameAs* links between several pairs of linked data sets and identifying the common and distinctive characteristics of these links.
- Discovering new links based on user feedback over query answers using a reinforcement learning approach, while preserving link precision.
- Proving that this approach is sound in terms of finding an optimal policy for links exploration.
- Developing optimizations to reduce execution time and converge in fewer steps.
- Demonstrating the validity of this approach by running experiments on large, real-world, multi-domain data sets.

3.1 Related Work

3.1.1 Automatic Linking of RDF Data Sets

Reducing the semantic gap between different representations of the same information has been an active topic of research for a long time. The existence of such different representations is due to the natural human tendency for different people to have different perspectives and hence to model problems differently. Research on reducing such a semantic gap spans both ontology alignment (or ontology matching) and entity linking. Ontology alignment solves the problem of using different vocabularies among data sets representing the same real-world domain, while entity linking solves the problem of using different representations for the same real-world entity. There has been extensive work on both ontology alignment [16, 40, 49, 70, 82, 90, 122, 138, 152] and entity linking [76, 78, 80, 103, 105, 112, 113, 122, 135]. This thesis focuses on the latter problem with the goal of improving the quality of inter-linking between different linked RDF data sets on the web based on user feedback over query answers. The related work on this topic is presented next.

Linked RDF data on the web has enabled seamless connections between open data sets [26]. The idea is to link entities from different data sets that are semantically equivalent to each other. There have been many works on semantic matching of entities, taking different approaches. The SILK framework [78, 135] uses manually defined mapping rules that are applied on input data sets. New data sets require new mapping rules. LIMES [103] also uses manual mapping rules in addition to a supervised learning approach. OBJECTCOREF [76] uses training data to learn how to link entities. However, this approach requires having good training data that captures most aspects of the input data sets, which is difficult in practice. LN2R [112, 113], CODI [105] and ASMOV [80] follow similar approaches to that of OBJECTCOREF.

PARIS [122] is an iterative ontology alignment/entity matching system, where the results of the current iteration are carefully utilized for improving the results in the next iteration. It is fully automatic and does not require any prior information or training. It also produces better quality links than other approaches. Due to its generality and superior quality, I use PARIS in this thesis as the automatic linking algorithm to produce the candidate links that are the starting point for ALEX. However, I emphasize that ALEX can work with any initial set of candidate links, regardless of how they were generated.

3.1.2 Incorporating Users in Automatic Linking

In the context of linking open data, ZenCrowd [45] utilizes the crowd by forming micro-tasks using a probabilistic model for manual matching. Its goal is to link traditional web content to the Linked Open Data (LOD) cloud. In contrast, the goal of this thesis is to utilize user feedback to improve the quality of links *in* the LOD cloud to which ZenCrowd tries to link traditional web pages.

The system in [34] summarizes descriptions of candidate entities by selecting and presenting only a subset of features to a human to confirm that the entities are equivalent. The selected features are expected to effectively substitute entire entity descriptions to avoid overloading human users with too much information. One goal of [9] that is related to ALEX is to refine links in DBpedia by removing incorrect links to external web pages

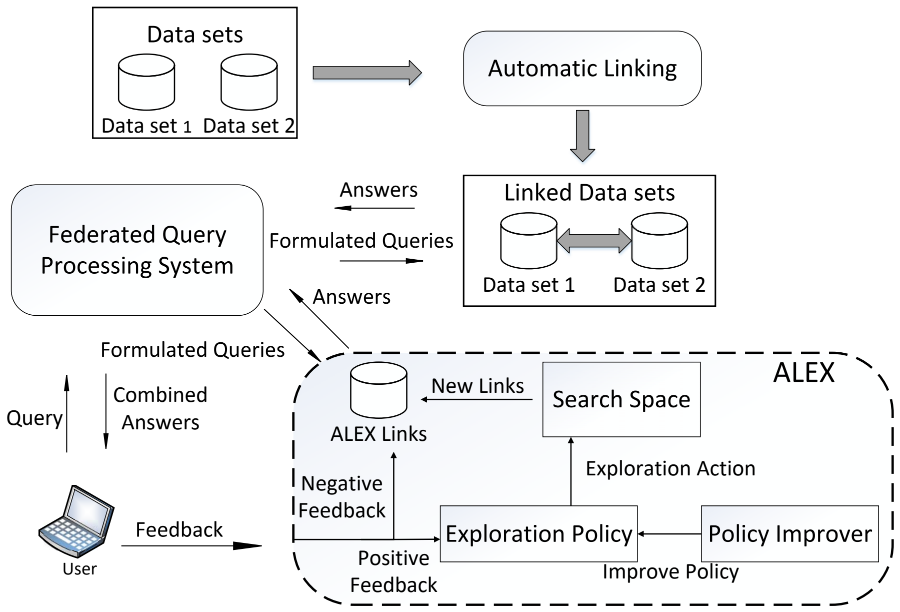


Figure 3.1: Architecture of a federated query system with ALEX.

or resources. Users are shown a caption of the external source and determine if it matches the entity in DBpedia or not.

In contrast to all the aforementioned approaches, the most distinct feature of ALEX is that it not only removes incorrect links from the set of candidate links, but also discovers new links that were not part of this set. In addition, ALEX does not expose the user to the ontology of the data sets or the details of the linked entities but rather directly improves the quality of links by utilizing user feedback on the answers to her queries.

3.2 Overview of ALEX

This section presents an end-to-end overview of the architecture of ALEX (shown in Figure 3.1). ALEX can be integrated in a system that answers queries over multiple linked RDF data sets, such as Sapphire, which is described in the next chapter. The data sets are linked using any automatic linking algorithm. They could be hosted anywhere on the web and be accessible via SPARQL endpoints. The user is able to issue queries over these data sets and receives answers using a federated query processor, such as FedX [116], which is used in Sapphire. If a query answer is generated based on a link between two data sets, the user is given a chance to evaluate this answer and provide feedback about which answers are correct and which are not. Feedback about an answer is interpreted as feedback on the link that is used to generate the answer. That is, if the answer is correct, then the link is correct, and if the answer is incorrect, the link is incorrect.

This feedback is sent to ALEX. If the feedback is positive (correct answer), ALEX uses its *exploration policy* (referred to as the *policy* for simplicity) to take an *exploration action* (referred to as simply an *action*). This action is defined as choosing a feature of the approved link and expanding its score value to be a band of values, then finding new links that have a feature score for this feature within this band. This search is done over a pre-computed search space of feature sets, where there is a feature set for every pair of entities in the two data sets. The newly discovered links are stored in a local RDF store that is accessible via a SPARQL endpoint. This endpoint is also used in answering queries just like the endpoints of the queried data sets. Thus, the federated query processor will automatically incorporate the links discovered by ALEX into query processing, since these links are stored at one of the endpoints being queried, and federated SPARQL query processing guarantees incorporating all data (including links) regardless of the endpoint at which this data is stored.

If the feedback is negative (incorrect answer), then there are two cases: 1. The incorrect link is in the local RDF store of ALEX, in which case, the link is deleted from the local store. 2. The incorrect link is in the original data set and ALEX does not have the authority to remove it. In this case, ALEX uses query rewriting to retrieve all the links used to compute the query answers. ALEX then looks up each of these links in a blacklist that it uses to hide links that are known to be incorrect based on past user feedback (more on the blacklist used in ALEX in Section 3.8). If a link used to find an answer to the query is found in the blacklist, the query is rewritten using additional FILTER statements to filter out this blacklisted link. Formally, if the query has any triple $?v_i \vee u_j$ *owl:sameAs* $?v_k \vee u_l$, where $?v_i$ and $?v_k$ are variables and u_j and u_l are URIs, ALEX rewrites the query to retrieve the entities connected by this *owl:sameAs* link. If this link is blacklisted, the original query is rewritten to add one (if there is one variable in the triple) or two (if there are two variables) FILTER statements for such a triple. Each filter states that the variable in the triple (subject or object) cannot be the (subject or object) URI in the blacklisted link.

It is worth noting that a user is not *required* to provide feedback on each query answer; if no feedback is provided on an answer, this answer will simply not trigger an action by ALEX. The policy in ALEX is improved after a number of feedback instances is received in order to have better exploration actions in the future. The details of the ALEX policy and actions will be discussed in Section 3.6.

Data Set	Version	Domain	Number of Triples
DBpedia	2016-10	Multi-domain	178M
OpenCyc	4.0	Multi-domain	1.2M
Geonames	2.1	Geography	169M
NYTimes	2010-01-13	Media	206K
Linkedmdb	2009-05	Media	6.1M
Drugbank	2014-07-25	Life Sciences	589K
Lexvo	2013-09	Linguistics	704K
UK Learning	2017	Education	3.6K

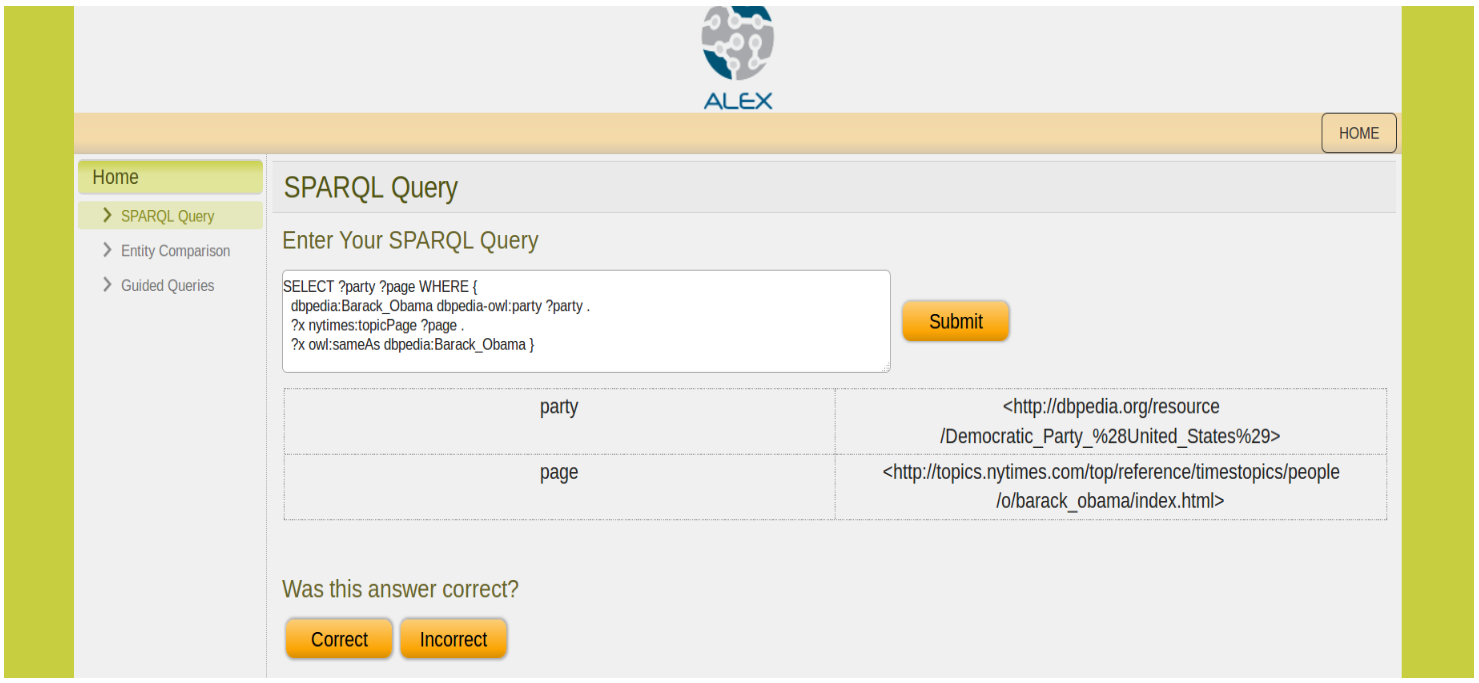
Table 3.1: Data sets used in the analysis and experiments.

3.3 User Interface

There are two methods to interact with ALEX [57]:

- The user can write a SPARQL query for ALEX to answer. ALEX can answer SPARQL queries that involve multiple data sets and return the answers to the user, who can provide feedback over any returned answer. Figure 3.2 shows an example of this method. The figure shows an example query that is interested in Barack Obama's party affiliation and the New York Times news pages about him. A user can click on any of the returned URIs to see the corresponding entity. After returning the answer, ALEX asks the user for her feedback about whether the answer is correct.

- ALEX can explicitly show the user current existing candidate links. The user can choose an entity and see the links to this entity one at a time, approving or rejecting each link. Alternately, ALEX can show the user one link at a time at random, regardless of the linked entities. The user can view the two entities from the different data sets and decide whether they represent the same real-world entity or not. Figure 3.3 shows an example of this method.



SPARQL Query

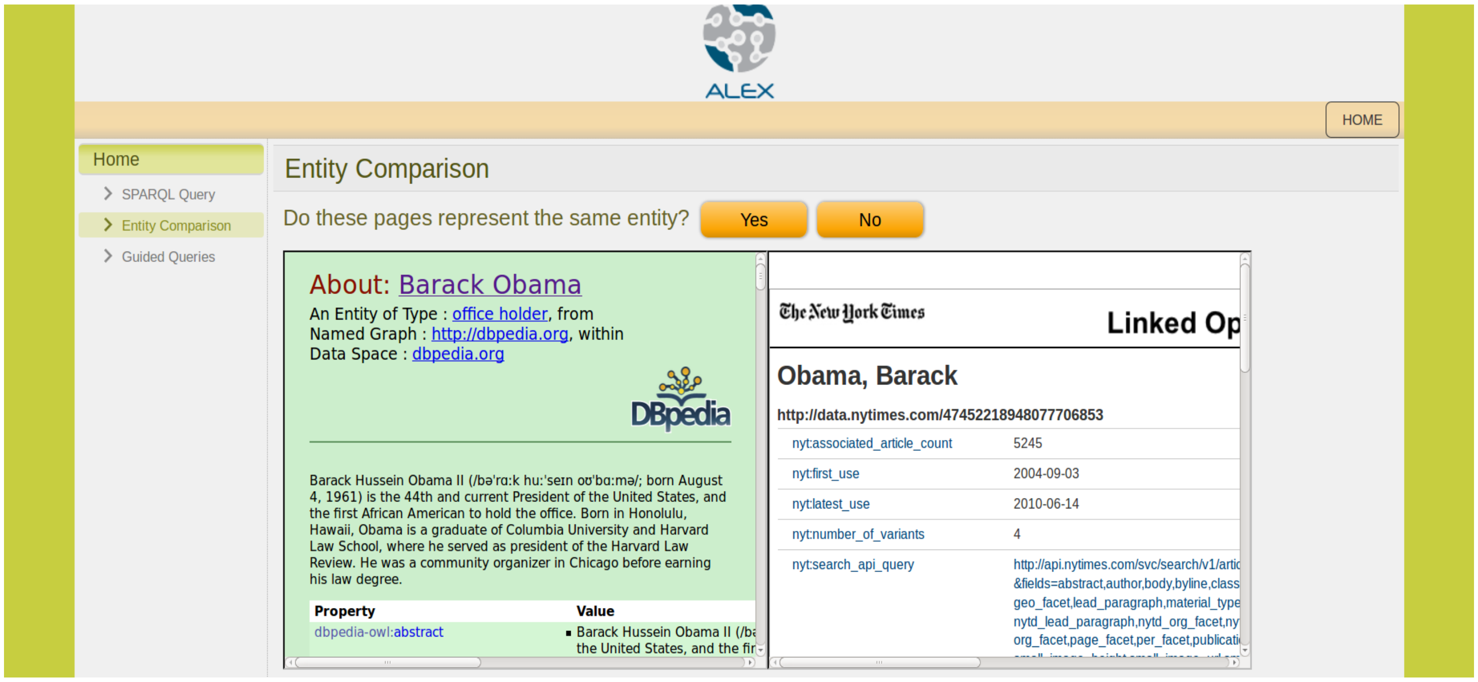
Enter Your SPARQL Query

```
SELECT ?party ?page WHERE {
 dbpedia:Barack_Obama dbpedia-owl:party ?party .
 ?x nytimes:topicPage ?page .
 ?x owl:sameAs dbpedia:Barack_Obama }
```

party	<http://dbpedia.org/resource /Democratic_Party_%28United_States%29>
page	<http://topics.nytimes.com/top/reference/timestopics/people /o/barack_obama/index.html>

Was this answer correct?

Figure 3.2: Issuing SPARQL queries and giving feedback over the returned answer.



Entity Comparison

Do these pages represent the same entity? Yes No

About: Barack Obama

An Entity of Type : office holder, from Named Graph : http://dbpedia.org, within Data Space : dbpedia.org

Barack Hussein Obama II (/bəˈrɑːk huːˈseɪn oʊˈbɑːmə/; born August 4, 1961) is the 44th and current President of the United States, and the first African American to hold the office. Born in Honolulu, Hawaii, Obama is a graduate of Columbia University and Harvard Law School, where he served as president of the Harvard Law Review. He was a community organizer in Chicago before earning his law degree.

Property	Value
dbpedia-owl:abstract	▪ Barack Hussein Obama II (/ba the United States, and the fir

The New York Times Linked Op

Obama, Barack

http://data.nytimes.com/47452218948077706853

nyt:associated_article_count	5245
nyt:first_use	2004-09-03
nyt:latest_use	2010-06-14
nyt:number_of_variants	4
nyt:search_api_query	http://api.nytimes.com/svc/search/v1/artic &fields=abstract,author,body,byline,class geo_facet,lead_paragraph,material_type nytd_lead_paragraph,nytd_org_facet,ny org_facet,page_facet,per_facet,publicati

Figure 3.3: Showing the entities that are connected with an *owl:sameAs* link. A user can explicitly approve or reject the link.

Data Set Pair	Number of Links
DBpedia - NYTimes	11289
DBpedia - OpenCyc	41040
DBpedia - Linkedmdb	29575
DBpedia - Lexvo	4365
DBpedia - Drugbank	3824
DBpedia - Geonames	430819
DBpedia - UK Learning	215
Geonames - NYTimes	1788
Geonames - Linkedmdb	247
Geonames - Lexvo	249
OpenCyc - Umbel	56195

Table 3.2: Pairs of data sets used in the analysis and experiments.

3.4 Analysis of the *owl:sameAs* Links in the LOD Cloud

This section presents an analysis of the *owl:sameAs* links between pairs of data sets from the LOD cloud. This analysis aims at understanding the characteristics of pairs of entities from two different data sets that are linked by an *owl:sameAs* link. This will help in choosing the right strategy for exploring and discovering new links between pairs of data sets as will be discussed in Section 3.6. First, the data sets used in the analysis are described. Then, all the links between the data set pairs are analyzed to find the features that can be used to identify any two entities as being the same. Finally, I discuss how the analysis findings can inform how new links are discovered.

3.4.1 Data Sets Used in the Analysis

The analysis uses heterogeneous data sets that span different domains from the LOD cloud. The data sets used are shown in Table 3.1. Two of the data sets are multi-domain (DBpedia and OpenCyc). DBpedia contains structured data extracted from Wikipedia, and OpenCyc contains parts of the Cyc knowledge base of everyday knowledge. Both of these data sets cover multiple domains, and are in the center of the Linked Open Data cloud, with many links to other data sets. Geonames is a geographical database that covers all countries and contains over eleven million place names. NYTimes contains data about locations, people, and organizations. Linkedmdb is an open semantic web database for movies. Drugbank is a repository of almost 5000 FDA-approved small molecule and biotech drugs. Lexvo contains data about human languages. UK Learning Providers is a data set about UK universities. Some pairs of data sets are already linked (shown in Table 3.2). These pairs of data sets are used in the analysis of existing *owl:sameAs* links in this section, and in the evaluation in Section 3.9.

3.4.2 Analysis of *owl:sameAs* Links

Features

Consider two entities $E_1 = \{(p_{11}, o_{11}), (p_{12}, o_{12}), \dots, (p_{1n}, o_{1n})\}$, and $E_2 = \{(p_{21}, o_{21}), (p_{22}, o_{22}), \dots, (p_{2m}, o_{2m})\}$, where E_1 has n pairs of (predicate, object), and E_2 has m pairs. An example entity is {(name, "Mohamed El Shorbagy"), (birth year, 1991), (age, 27)}. A feature is the pair (p_{1i}, p_{2j}), where p_{1i} is a predicate of one entity from the first data set, and p_{2j} is a predicate of another entity from the second data set. The feature means that there is a correspondence between p_{1i} and p_{2j}. The value of the feature measures the degree of correspondence.

Link Representation

If there is a link between entities E_1 and E_2, the features of this link are all the pairs (p_{1i}, p_{2j}) where p_{1i} is a predicate of E_1 and p_{2j} is a predicate of E_2. Every feature of a link is assigned a score in the range $[0, 1]$, which is the similarity score between the two objects associated with the predicates of the feature, p_{1i} and p_{2j}. Recall from Section 2.1 that the object in an RDF triple can be either a literal or a URI $(U \cup L)$. If both objects are literals, the similarity score for the feature (p_{1i}, p_{2j}) is $score = sim(o_{1i}, o_{2j})$. ALEX uses a generic similarity function that takes into account the type of the attributes to be compared (string, integer, float, date, etc.). This function is presented in Appendix A. If both objects are URIs, all objects associated with these URIs are retrieved. These objects are referred to as *second layer* objects. In this case, the similarity score for the feature is the maximum similarity score between all pairs of second layer objects. Formally, if the two objects are URIs u_1 and u_2, a SPARQL query is issued to find the set of literal objects associated with u_1 and u_2, termed l_1 and l_2, respectively. The score for the feature (p_{1i}, p_{2j}) is $score = MAX(sim(l_{1k}, l_{2m}))$. If only one object is a URI and the other is a literal, the score is $score = MAX(sim(o_{1i}, l_{2m}))$ or $score = MAX(sim(l_{1k}, o_{2j}))$. The justification for using MAX to choose among possible scores is presented in the next paragraph. Note that if the set l_1 or l_2 is empty, then any score involving computing a similarity with this set is defined to be zero; ALEX does not retrieve third or higher layer objects, that is, objects associated with URIs that are themselves second layer objects. The justification for using MAX to choose among possible scores is presented in the next paragraph. Note that if the set l_1 or l_2 is empty, then any score involving computing a similarity with this set is defined to be zero; ALEX does not retrieve third or higher layer objects, that is, objects associated with URIs that are themselves second layer objects.

For any link, if the first entity has n triples (i.e., there are n triples in the data with the URI of this entity as the subject, and with different predicates and objects that relate to this entity), and the second entity has m triples, an $n \times m$ similarity matrix is created for the link, where the values of the item at row i and column j is the similarity score between the ith predicate of the first entity and the jth predicate of the second entity, computed as explained earlier. If the similarity score is less than a threshold θ, the matrix item is set to 0. This similarity matrix is then reduced to a feature set by choosing the maximum value for each row in the similarity matrix if $n > m$ or each column if $m > n$. The reason for using the maximum value is that the goal is finding the pair of predicates that most predicates correspond to each other. For example, if the first entity has the predicates *{name, home phone, work phone, address, date of birth, email}* and the second entity has the predicates *{label, job title, date joined, phone, salary}*, it is desired that the *name* and *label* correspond to each other rather than to any other predicate. Using the maximum similarity value will ensure that this happens. If we are looking for the predicate to associate with *name*, using the maximum similarity value will ensure that we choose *label* since the similarity between *name* and *label* will be higher than than the similarity between *name* and any other predicate, say, *phone*.

It is worth noting that, according to our model of defining the feature set, it is not required that one predicate from one entity corresponds to exactly one predicate from another entity. Depending on the maximum score, it could happen that a predicate from one entity makes a feature with multiple predicates from another entity. In our example, we could see a feature *(home phone, phone)* and another feature *(work phone, phone)*. Figure 3.4 shows how the feature set is generated for any link.

Distinctive Features

In this section, the aim is to find the *distinctive features*, which are features that can be used to identify two entities as being the same. A feature is considered distinctive if it can be used to accurately classify the pairs of entities that correspond to it as being the same (i.e., an *owl:sameAs* link exists between the entities) or not the same (i.e., no link exists). To use a feature to classify a pair of entities as being the same or not the same

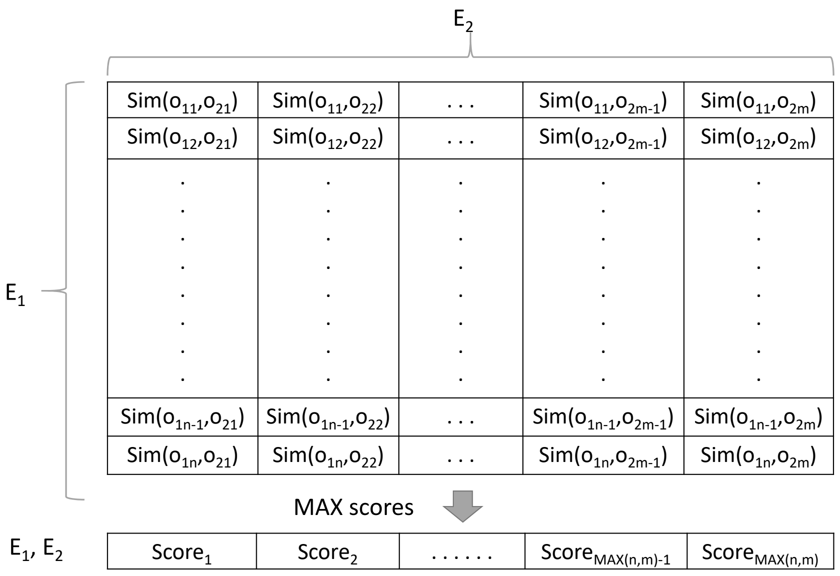


Figure 3.4: Generating a feature set for any two entities E_1 and E_2 by constructing a similarity matrix then reducing it to a feature set.

(i.e., having an *owl:sameAs* link or not), the score of this feature is computed. If the feature score is greater than a cutoff value γ, the entities are classified as the same. If the feature score is less than γ, the entities are classified as not the same. The value of γ that maximizes classification accuracy is found, and if the accuracy is above 90%, then the feature is identified as a distinctive feature.

This definition of distinctive features requires formalizing the notion of "maximizing classification accuracy". The classification accuracy is defined in terms of *true positives* tp (correct links identified as correct), *true negatives* tn (incorrect links identified as incorrect), *false positives* fp (incorrect links identified as correct), and *false negatives* fn (correct links identified as incorrect). These accuracy measures are functions of γ. Equation 3.1 defines the optimization problem for maximizing classification accuracy in terms of these accuracy measures. The γ that maximizes classification accuracy for a distinctive feature is called the *distinctive score*.

$$\begin{aligned} \underset{\gamma}{\text{maximize}} \quad & \frac{|tp| + |tn|}{|tp| + |tn| + |fp| + |fn|} \\ \text{subject to} \quad & \frac{|tp| + |tn|}{|tp| + |tn| + |fp| + |fn|} > 0.9, \\ & 1 \geq \gamma \geq 0 \end{aligned} \tag{3.1}$$

All distinctive features and their distinctive scores in the eleven pairs of data sets in Table 3.2 are found. Figure 3.5 shows the total number of features and the number of distinctive features (note that the y-axis of the figure is in log scale). The figure shows that the percentage of distinctive features ranges from 1.7% to 12.7% of the total number of features. Appendix B shows these distinctive features.

One may want to consider the following simple approach for finding the distinctive score: A static, high value of γ can be chosen. For example, set γ to 0.8, and classify any feature with a score above 0.8 as a correct link and any feature with a score below 0.8 as an incorrect link. The analysis shows that this simple approach does not yield high classification accuracy. Figure 3.6 shows the average distinctive score that solves the optimization problem in Equation 3.1 for all the distinctive features in Figure 3.5, and the average feature score for correct links for the distinctive features. The error bars show the standard deviation of the distribution of scores. The figure shows that approaches that rely only on high similarity scores of features to determine if two entities are the same will not achieve the best possible accuracy for finding correct links. Neither the average distinctive score nor the average correct link score are particularly high. The large error bars also show that there is a high variance in both the distinctive score and the feature score for correct links among different pairs of data sets, which shows that identifying the distinctive score automatically is a challenging task.

This can also be highlighted with a sample of different patterns of distinctive feature scores, as shown in Figure 3.7. Each graph in Figure 3.7 plots the feature scores for one distinctive feature connecting pairs of entities in two data sets. Each plot has one point for each link between entities in the two data sets, and an equal number of points for incorrect links. Each point represents the feature score for a specific pair of entities. Blue points are feature scores for correct links for the considered distinctive feature, and orange points are feature scores for incorrect links for the same feature. The incorrect links are randomly generated. The x-axis represents an arbitrary ordering of links and the y-axis represents the feature scores.

The patterns in Figure 3.7 show that using a high value of the feature score to indicate that two entities are the same may work in some cases. In Figures 3.7(a), 3.7(b), and 3.7(c),

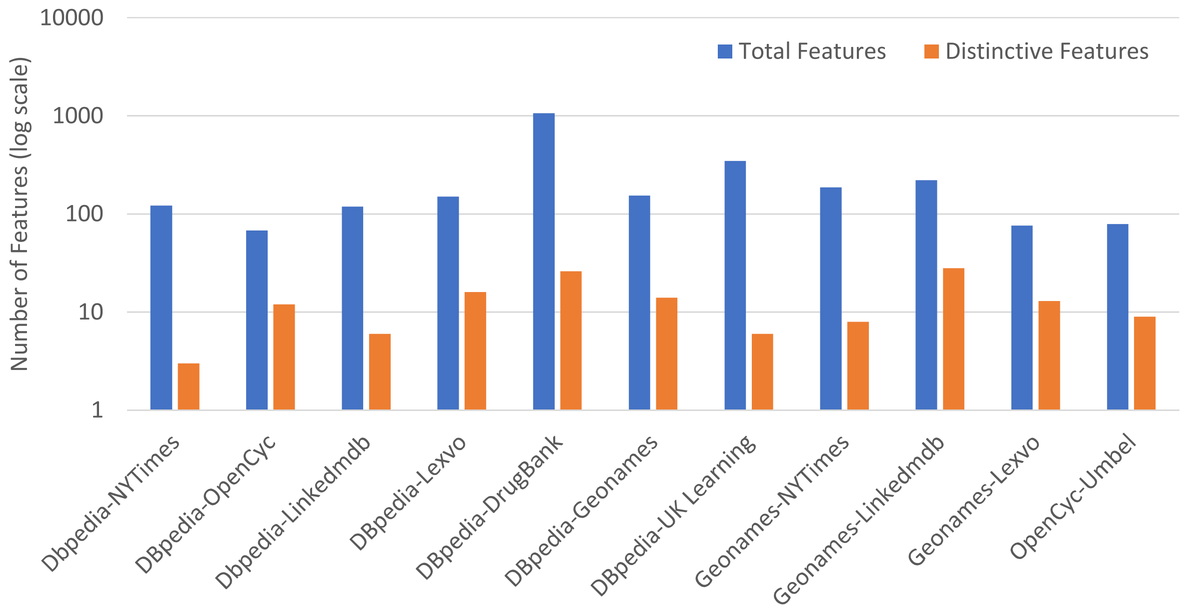


Figure 3.5: The total number of features and number of distinctive features for each pair of data set.

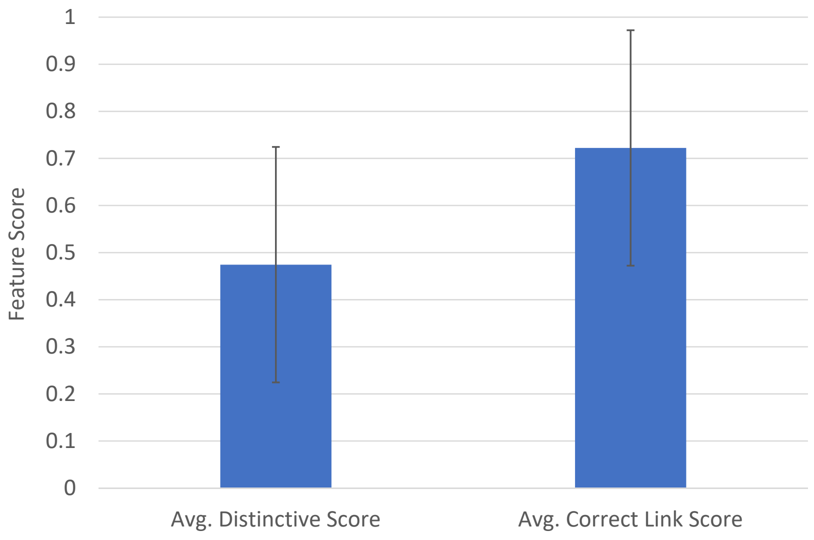


Figure 3.6: The average distinctive score and the average correct link score. The error bars represent the standard deviation.

it may be possible to set a threshold score around, say, 0.8 and get high classification accuracy. However, in Figures 3.7(d), 3.7(e), and 3.7(f), a value of 0.8 would be too high. For these plots, a static threshold value would need to be 0.5 to 0.6, which is too low for other plots such as Figure 3.7(b). Figures 3.7(g), 3.7(h), and 3.7(i) show cases in which it is not simple to find an accurate distinctive score, yet finding such a score is critical for classification accuracy. Figures 3.7(h) and 3.7(i) show that in some features, higher similarity scores do not necessarily mean that more correct links can be found. For example, if the linking approach chooses a similarity threshold of 0.8 to find correct links, the feature in Figure 3.7(i) will not be identified as a distinctive feature. Some other patterns show that there are *bands* of scores at which most correct links exist. Figure 3.7(j) shows that almost all correct links have a score that is around 0.6 for this feature. Figure 3.7(k) shows that there are two bands of score (around 0.6 and 0.75). Figure 3.7(l) also shows two bands. However, they are wider in range, the first being $[0.5, 0.7]$ and the second being $[0.9, 1]$.

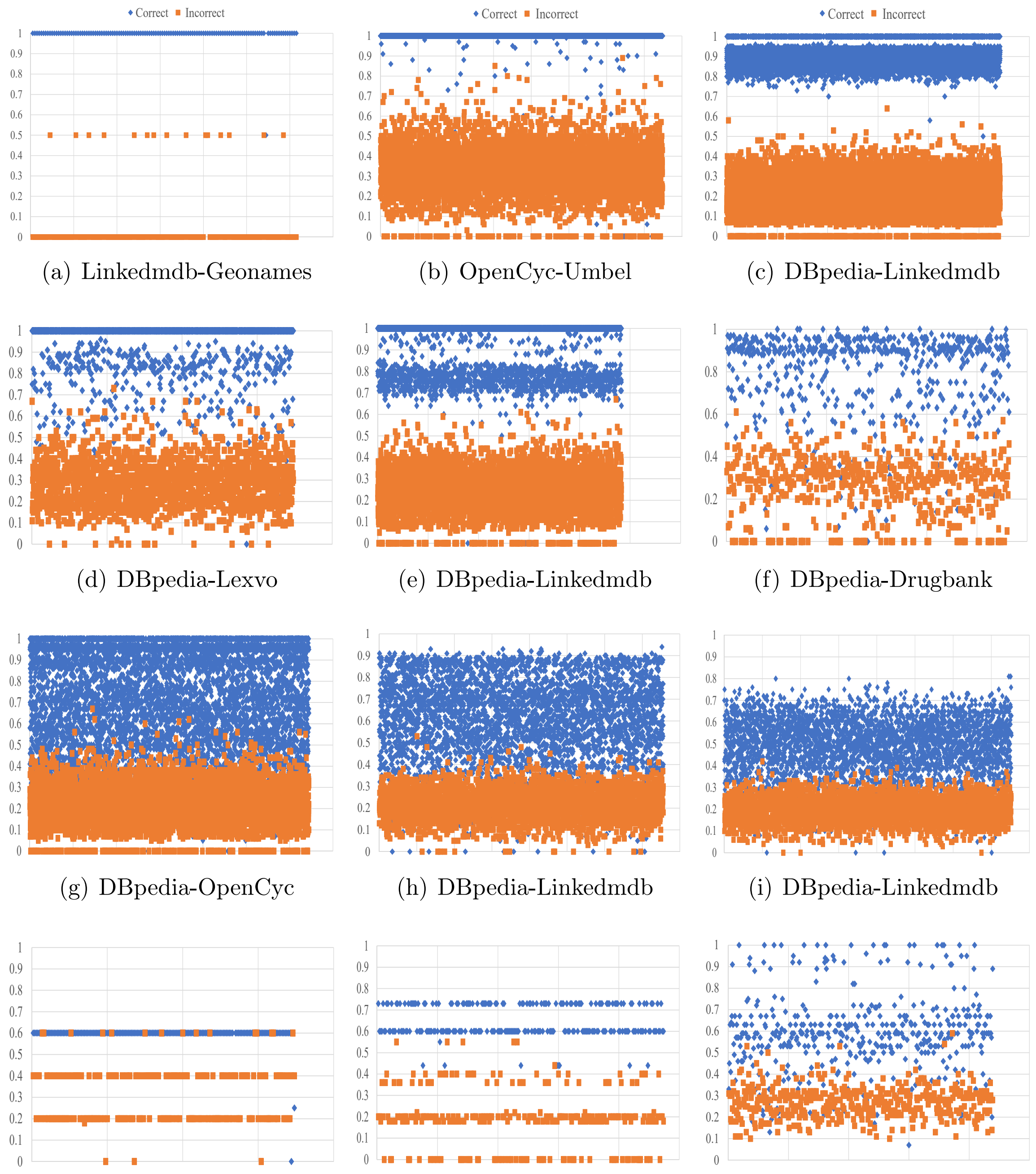
The observations in this section define the goals of ALEX: 1. Among all the features in a feature space of a pair of data sets, ALEX should identify the distinctive features. 2. ALEX should discover the correct links that have feature scores above the distinctive score for the distinctive features. Section 3.6 will show how ALEX discovers new links based on these two goals. But first, the next section presents some background on reinforcement learning.

3.5 Background on Reinforcement Learning

As explained in the previous section, ALEX needs to: (a) identify the distinctive features that can classify two entities as equivalent, and (b) identify the score of this feature that distinguishes between correct and incorrect links. This is modelled as a reinforcement learning problem, in which ALEX learns which features and scores to use to find correct links. This section gives an overview of reinforcement learning. The next section builds on this overview and presents the details of ALEX.

In reinforcement learning [123], the learner and decision maker is called the *agent*. Everything else outside of the agent is considered to be the *environment*. A reinforcement learning system consists of four main components:

- *Policy*: The policy defines how a reinforcement learning agent interacts with the environment at a given state. It can be viewed as the mapping from an environment *state* to an *action* taken by the agent. The policy can be as simple as a lookup table, or it can involve extensive computations. It also can be either de-



(a) Linkedmdb-Geonames (b) OpenCyc-Umbel (c) DBpedia-Linkedmdb

(d) DBpedia-Lexvo (e) DBpedia-Linkedmdb (f) DBpedia-Drugbank

(g) DBpedia-OpenCyc (h) DBpedia-Linkedmdb (i) DBpedia-Linkedmdb

(j) DBpedia-Drugbank (k) DBpedia-Drugbank (l) DBpedia-Lexvo

Figure 3.7: Patterns of feature scores for a sample of distinctive features. The x-axis is an arbitrary ordering of links and the y-axis is the similarity score value.

terministic or stochastic. In ALEX, a stochastic policy is used. For example, consider a user providing positive feedback on the link (E_1, E_2), where E_1, and E_2 are entities. The policy π might state that when this link is encountered, explore around the feature *(E_1.label, E_2.name)* with probability 0.8, and around the feature *(E_1.birth, E_2.birthDate)* with probability 0.2. Formally, $\pi((E_1, E_2), (E_1.label, E_2.name)) = 0.8$ and $\pi((E_1, E_2), (E_1.birth, E_2.birthDate)) = 0.2$. When an action is chosen, say the one that has higher probability, then $\pi\left((E_1, E_2)\right) = (E_1.label, E_2.name)$.

- *Reward Function*: The reward function defines the goal of the reinforcement learner. It can be viewed as a mapping from a state (or a state-action pair) to a *reward*. The goal of the agent is to maximize the total reward throughout its dynamic interactions with the environment. In ALEX, positive feedback results in positive reward (*pReward*), and negative feedback results in negative reward (*nReward*). For example, assume that positive feedback over the link (E_1, E_2) resulted in the exploration of 5 links $\{link_1, \ldots, link_5\}$. Assume that positive feedback comes over $link_1$ and $link_5$, negative feedback comes over $link_3$, and no feedback is received over $link_2$ and $link_4$. In this case, the reward is $R(E_1, E_2) = 2 \times pReward - nReward$.
- *Value Function*: The value function defines what is good in the long run. A *value* of a state can be viewed as the total reward that can be collected from this state taking into account the states that are likely to follow and the rewards available from those states. The major difference between the reward function and the value functions is that the first indicates what is good in an immediate sense (next reward), whereas the second indicates the long-term value of a state (total rewards that can be collected in the future starting from the current state). The reward of a state may be low but its value can be high because other states that yield high rewards can be reached from this state.
- *Environment Model* (optional): This model simulates the behaviour of the environment. For example, given a state and an action, the model can determine which state is next and the reward of the action. The model is used in planning because it enables the agent to determine which action to take without experiencing the state. However, there may be cases where a model is not available. In this case, the only way to determine the next state and reward given a state and action is by actually performing the action in the environment. In ALEX, the environment model is unknown because it is not possible to know in advance what feedback users will provide.

Reinforcement learning differs from other branches of machine learning in that the learning agent is not told what actions to take. The agent learns over time how to act by experiencing the return it gets from interacting with the dynamic environment. Reinforcement learning is more suitable than supervised learning [100] for interactive problems

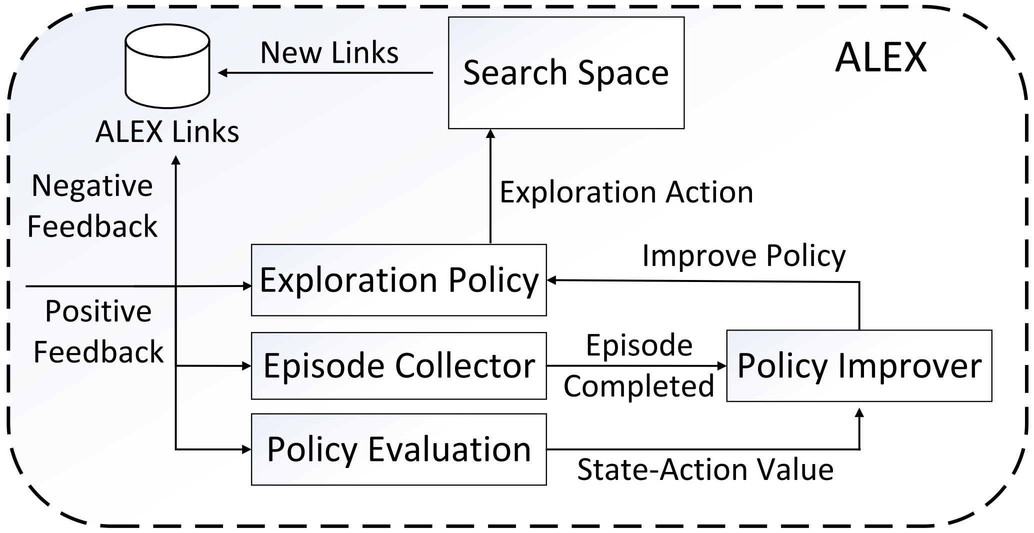


Figure 3.8: Detailed view of the components of ALEX.

where it is impractical to obtain examples of the desired behaviours to use as training data that are correct and representative of most situations in the environment. The reinforcement learning agent learns how to act by trying different actions at different states and aiming to maximize the expected return at those states [123]. The challenge that faces the learning agent is finding a balance between the need to explore as many states as possible while exploiting the current knowledge to maximize the total reward. Also, reinforcement learning needs a way to evaluate the policy used to take actions in order to improve it. In ALEX, a Monte Carlo method is used to evaluate the policy through returns from interactions with the environment. This is a suitable approach in situations like ours where the model of the environment is unknown.

3.6 Discovering New Links in ALEX

This section describes how ALEX models the link discovery problem as a reinforcement learning problem, takes actions to discover new links, and learns which features and feature scores result in finding more correct links.

Each link in ALEX represents a state, and ALEX takes an action starting from this state after receiving user feedback. The action, in the case of positive feedback, can be described as choosing a feature and using this feature to find new links based on their feature score for this feature. In the case of negative feedback, the incorrect link is removed from the set of candidate links. The given feedback is also translated into a reward in ALEX. This reward is positive in the case of an approved link (positive feedback), and negative in the case of a rejected one (negative feedback). The value of the reward can be the same for positive and negative feedback, or negative feedback can be penalized more.

ALEX works on a pair of data sets, and it explores links in a space of feature sets for the two data sets. This space is populated in a pre-processing step, with a feature set for every pair of entities in the two data sets. Initially, ALEX chooses arbitrary actions whenever a state is encountered because there is no prior knowledge of which actions are better. Rewards are collected for each state-action pair encountered, and are aggregated to estimate the value of the state-action pair. This is called *policy evaluation*. Policy evaluation takes place until sufficient feedback is collected, e.g., until 1000 feedback instances are collected. This is called a feedback *episode*. At the end of an episode, *policy improvement* takes place. Policy improvement modifies the policy so that actions that maximize the reward are taken most of the time, while assigning a low but non-zero probability to other actions in order to ensure continuous exploration.

These last two steps of policy evaluation and policy improvement are repeated until convergence. ALEX converges when the set of candidate links does not change after an episode of policy evaluation/policy improvement or when a maximum number of episodes is reached. ALEX can also use a more relaxed convergence condition and stop if the change in the set of candidate links is less than 5%. The details of the reinforcement learning approach used by ALEX are presented next.

3.6.1 States in ALEX

In a general reinforcement learning problem, the agent and the environment interact at every discrete time step, $t = 0, 1, 2, \dots$. At each time step t, the agent perceives the state of the environment $s_t \in S$, where S is the set of all states in the environment. The agent then takes an action $a_t \in A(s_t)$, where $A(s_t)$ is the set of all actions available at the state s_t. As a result of the action taken, the agent receives a reward r_{t+1} and a new state s_{t+1} is reached. The agent's action is determined by the policy π_t where $\pi_t(s, a)$ is the probability that $a_t = a$ at $s_t = s$. In ALEX, the state is defined by the link that was approved or rejected by a user in a feedback instance. The state is represented by the feature set consisting of pairs of attributes of the two entities linked by this link (recall Section 3.4.2).

3.6.2 Actions in ALEX

Given a state s_t, ALEX takes an action a_t that is based on a policy π_t. The environment (the user in our case) then responds with a reward r_{t+1}. The ultimate goal of ALEX is to improve the policy π_t so that the maximum total reward is collected.

The action of ALEX when it receives positive feedback can be viewed as exploring an area surrounding the current state (the link between two entities) in a particular direction (one feature of the feature set). Once ALEX has chosen a feature f of the approved link to use for exploration, an important question that needs to be answered is how to use the feature score of f to find new links. A feature set exists for every pair of entities in the data, and any pair of entities that has feature f in their feature set is a potential candidate for a new link. Whether or not ALEX creates a link between these two entities depends on their feature score of f. It was shown in Section 3.4 that it is not accurate to use a static distinctive score and assume that a link is correct if its feature score is above this distinctive score. It was also shown that feature scores for correct links sometimes occur in bands. Based on these observations, ALEX uses the following approach to find new links: A new link is created if its feature score on f falls within a band around the feature score of the approved link. That is, ALEX explores around feature f in some exploration band.

The simplest way to set the exploration band is to use a static step size α for the band. So if the exploration feature f has a score s, the exploration band would be $s \pm \alpha$. This is the approach used in [56]. In this case, given a state represented by a feature set sf of n features, the action a is also a feature set af of n features with a single non-zero feature that represents the step size by which ALEX should explore to discover new links. Formally, ALEX finds all the links that have similarity value between sf and $sf \pm af$. For example, consider the feature set $sf(E_1, E_2)$ ={((*label*, *name*), 0.8), ((*birth*, *year*), 0.6), ((*age*, *year*), 0.4)}. The first element of the feature set means that the predicate *label* from the first entity maps to the predicate *name* from the second entity, and the similarity between the object values is 0.8. A possible action can be represented by the action feature set $af(E_1, E_2)$ ={((*label*, *name*), 0.05), ((*birth*, *year*), 0), ((*age*, *year*), 0))}, which means that links that have a similarity score between attributes *label* and *name* in the range $[0.75, 0.85]$ should be added to the set of candidate links for future queries and possible feedback opportunities.

It was shown in Section 3.4 that different patterns exist for the distinctive scores of different distinctive features. With a static step size for the exploration band, ALEX explores the new links in these different patterns through sets of narrow (potentially interleaving) exploration bands. ALEX implicitly finds the distinctive score by learning to explore bands with more correct links in them, while avoiding bands with incorrect links. The width of these band is defined by the step size (the default value of the step size in the evaluation in this chapter is 0.05). However, this approach usually needs to explore a large number of bands to find all the correct links. To reduce the number of explorations required to discover correct links, ALEX uses *dynamic bands.* In dynamic bands, the step size increases from one iteration to the next. The step size α_i changes to $\alpha_{i+1} = MIN(1, 2 \times \alpha_i)$, where i is the episode number. The band size then changes to $[feature\_score - \alpha_0, feature\_score + \alpha_i]$. Note that the upper bound of the band changes after each episode, while the lower bound remains the same. This is based on the observations made in Section 3.4 that most links with feature scores higher than the distinctive score are correct. This means that if exploring around a feature score is estimated to discover more correct links, ALEX should look for more links with higher scores and still explore with a lower score, but with a narrow step to avoid discovering incorrect links.

ALEX can sometimes take an action that explores around a feature that has values that do not distinguish between entities. For example, it can decide to explore around the feature (*rdf:type*, *rdf:type*) which has a categorical value *owl:thing*. Exploring around this feature is expected to return a large number of incorrect links because a large number of different entities share this attribute and value. ALEX can *learn* that this feature is not distinctive and avoid exploring around it in the future.

3.6.3 Rewards and Feedback

The goal of ALEX is to maximize the expected return, R_t, defined as the sum of all future rewards:

$$R_t = r_{t+1} + r_{t+2} + \ldots + r_T \tag{3.2}$$

where T is the final time step. In ALEX, the final time step is when a feedback episode ends. After that, the policy used during the episode is refined through policy improvement, and a new episode is started using the new policy.

In ALEX, the reward is the feedback given by the user. The feedback could be positive (approving a link) or negative (rejecting a link). The value of the reward can be equal in both cases, or wrong links can be severely penalized by giving them a negative value that is larger than the positive value used for approved links. In this chapter the same reward value is used for both positive and negative feedback, since it was found in the experiments that increasing the penalty for incorrect link does not affect the quality of the discovered links.

The value of taking an action a at a state s under a policy π, which is called the *action-value function* $Q^\pi(s, a)$ is defined by:

$$
\begin{aligned}
Q^\pi(s,a) &= E_\pi\left\{R_t | s_t = s, a_t = a\right\} \\
&= E_\pi\left\{\sum_{k=t+1}^{T} r_k | s_t = s, a_t = a\right\}
\end{aligned}
\tag{3.3}
$$

where E_π is the expected value given that ALEX follows policy π. A fundamental property of the value function is that it follows a recursive relationship between the current and future state-action values:

$$
\begin{aligned}
Q^\pi(s,a) &= E_\pi\left\{\sum_{k=t+1}^{T} r_k | s_t = s, a_t = a\right\} \\
&= E_\pi\left\{r_{t+1} + \sum_{k=t+2}^{T} r_k | s_t = s, a_t = a\right\} \\
&= \sum_{s'} P^a_{ss'}\left[R^a_{ss'} + E_\pi\left\{\sum_{k=t+2}^{T} r_k | s_{t+1} = s', a_t = a\right\}\right]
\end{aligned}
\tag{3.4}
$$

where $P^a_{ss'}$ is the probability that the next state is s' when action a is taken at state s at time step t: $P^a_{ss'} = Pr\{s_{t+1} = s' | s_t = s, a_t = a\}$, and $R^a_{ss'}$ is the expected value of the next reward when taking action a at state s to move to state s': $R^a_{ss'} = E\{r_{t+1} | s_t = s, a_t = a, s_{t+1} = s'\}$.

The relationship between the action-value of the current state and that of the next state can be obtained from Equation 3.4 as follows:

$$
\begin{aligned}
Q^\pi(s,a) = \sum_{s'} P^a_{ss'}\Bigg[R^a_{ss'} &+ \sum_{a'} \pi(s',a') \times \\
&E_\pi\left\{\sum_{k=t+2}^{T} r_k | s_{t+1} = s', a_{t+1} = a'\right\}\Bigg] \\
= \sum_{s'} P^a_{ss'}\left[R^a_{ss'} + \sum_{a'} \pi(s',a') Q^\pi(s',a')\right]
\end{aligned}
\tag{3.5}
$$

where $\pi(s',a')$ is the probability of choosing action a' at state s' according to policy π. In ALEX, when a positive feedback instance is received over a link (state) s, and an action a is taken based on a policy π with probability $\pi(s,a)$, a number of new links (states) is discovered and added to the set of candidate links. When one of these states s' is later visited and feedback (positive or negative) is received over it, the value of $R^a_{ss'}$ is then known. It is assumed that all states that are generated by an action a have equal probability of being visited. Thus, $P^a_{ss'} = \frac{1}{|s'|}$.

A policy π' is considered to dominate another policy π if and only if $Q^{\pi'}(s,a) \geq Q^{\pi}(s,a)$ for all $s \in S$. Policy π^* is considered to be an *optimal policy* if its value function dominates the value functions of all other policies. There may exist more than one optimal policy. An optimal policy implies an optimal action-value function:

$$Q^*(s,a) = max_{\pi}\, Q^{\pi}(s,a) \tag{3.6}$$

for all $s \in S$. Therefore, Equation 3.5 becomes:

$$Q^*(s,a) = \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + max_{a'} Q^*(s',a')\right] \tag{3.7}$$

3.6.4 Iterative Improvement

After an episode of feedback is collected, ALEX improves the policy based on the value function evaluated during the episode. A new episode is then started and the policy evaluation/policy improvement iterations continue until ALEX converges. Convergence is defined by the candidate links not changing in an episode of feedback. The details of evaluating a policy during an episode and improving it at the end of the episode are presented next.

Monte Carlo Policy Evaluation

The value function can only be evaluated through interactions between ALEX and the environment (the user and existing links). According to the definition of value in Equation 3.2, the value of a state or state-action pair can only be known if a user gives feedback on the current state and future states that follow. Since the value function needs to be evaluated at the present time without waiting for future feedback, the value function needs to be estimated according to the current state s, policy π, and action taken a. A Monte Carlo (MC) method is used for this purpose: to estimate the action-value of each state visited during each episode, while feedback is collected.

The existence of a state s in an episode is called a *visit*. ALEX uses a *first-visit* MC approach [123] to estimate the action-value function. In the first-visit MC approach, the average of returns following the first visit to s in which action a was taken is maintained. This means that if the state-action pair (s,a) is witnessed again during an episode, returns following that pair will not be considered. For example, if a state s_2 results from the state-action pair (s_1, a_1), and s_2 turns out to be a correct link, a positive reward is added to the return of state-action pair (s_1, a_1). Now, if from state s_2, action a_2 is taken and results in a wrong state s_3, a negative reward is added to (s_2, a_2) and (s_1, a_1). However, when state s_2 or s_3 is encountered again, no reward is added to the updated returns of the state-action pairs during the current episode. If a state, say s_2, is encountered in a future episode, that would be considered a new first visit. The first-visit MC approach converges asymptotically to $Q^{\pi}(s, a)$ [118].

The MC method requires π to be probabilistic. If π is deterministic rather than probabilistic at a state s, the same action a will always be taken. Thus, many relevant state-action pairs may never be visited. In such a case, there would be no need for learning how to choose among actions at any state. To compare the alternatives, the value of almost all actions from all witnessed states needs to be estimated. ALEX ensures continuous exploration to avoid this problem.

ALEX gives itself the option of choosing any action at any state. This means that at any state $s \in S$, and for all actions available in that state $a \in A(s)$, $\pi(s, a) > 0$. ALEX achieves this non-zero probability by using an ϵ-greedy policy so that it mostly chooses a greedy action that has the maximal estimated action value, but chooses a random action with low probability $\epsilon > 0$. In other words, with probability $1 - \epsilon + \frac{\epsilon}{|A(s)|}$ a greedy action is taken, and with probability $\epsilon - \frac{\epsilon}{|A(s)|}$, a non-greedy action is taken. This satisfies $1 \geq \pi(s, a) \geq \frac{\epsilon}{|A(s)|} > 0$, which means that no action has zero probability of being selected by the current policy, thereby ensuring continuous exploration.

Policy Improvement

If the rewards, $R^a_{ss'}$, and the probabilities of moving to states given an action, $P^a_{ss'}$, are known in advance, Equation 3.7 has a unique solution since it is a system of N equations where N is the number of states in the environment. If the optimal value function is known, it is straightforward to determine an optimal policy: At any state s, choose the action that yields the maximum value of $Q^*(s, a)$ in Equation 3.7. In other words, a greedy policy with respect to the optimal evaluation function is the optimal policy. However, as explained earlier, the feedback on links and which states can be visited next are unknown. This means that the reward of the current action will not be known until the user gives feedback on the links discovered after the current action is taken. Also, the next state visited is not known in advance.

The previous section explained how a Monte Carlo method can be used to estimate Q^{π_k} for arbitrary probabilistic π_k, where k is the iteration number in the cycle of policy evaluation/policy improvement. Policy improvement is done by changing the policy to a greedy policy with respect to the current value function. For any action-value function Q, the greedy policy is the one that, for all $s \in S$, chooses the action with the maximal Q value:

$$\pi(s) = argmax_a Q(s, a) \tag{3.8}$$

Equation 3.8 can be used as the basis for policy improvement as follows:

$$\begin{aligned} &Q^{\pi_k}(s, \pi_{k+1}(s)) = Q^{\pi_k}(s, argmax_a Q^{\pi_k}(s, a)) \\ &= max_a Q^{\pi_k}(s, a) \\ &\geq Q^{\pi_k}(s, \pi_k(s)) \end{aligned} \tag{3.9}$$

where, as explained above, π_{k+1} is the greedy policy with respect to Q^{π_k}. In Section 3.7, it is proven that π_{k+1} dominates π_k.

3.6.5 Interaction Between Policy Evaluation and Improvement

The value function is repeatedly updated to approximate the actual value of the current state with respect to the current policy. Also, the policy is repeatedly improved with respect to the current value function. Iteratively, these two processes cause the policy to approach optimality, and the value function to approach its actual value.

As discussed in Section 3.6.4, each evaluation step moves the value function Q^{π_k} towards its actual value. The value function converges to its actual value over many steps, at which point policy improvement can terminate. However, this convergence would require many feedback episodes. ALEX does not wait for complete policy evaluation before returning to policy improvement. In fact, policy improvement can start as soon as preliminary evaluation is done during the first episode. As state-action values become more accurate by receiving more feedback during future episodes, the policy can be further improved after each episode of feedback.

Algorithm 1 shows how ALEX alternates between policy evaluation and policy improvement on an episode-by-episode basis. While collecting feedback in an episode, policy evaluation is done by estimating the action-value function $Q(s, a)$ (lines 11 to 22), the policy is then improved at all states visited in the episode by choosing the greedy action (line 25).

Algorithm 1 shows how policy improvement is done using an ϵ-greedy policy. When ALEX starts, it chooses arbitrary actions for new states visited for the first time or before the first policy improvement cycle (lines 2 to 8). Lines 24 to 33 show how policy improvement takes place after an episode by assigning the greedy action a probability of $1-\epsilon+\frac{\epsilon}{|A(s)|}$ while non-greedy actions are assigned a probability $\frac{\epsilon}{|A(s)|}$ each (taking a non-greedy action

Algorithm 1: ALEX with ϵ-greedy policy

```
input  : set of states $S$, set of actions $A$
output: action-value function $Q(s,a)$, Policy $\pi(s)$
1  // Initialize
2  for all $s \in S$ do
3      for all $a \in A(s)$ do
4          $Q(s,a)$ = undefined;
5          $\pi(s)$ = arbitrary action;
6          $Returns(s,a)$ = empty list;
7      end
8  end
9  while set of candidate links different from last iteration do
10     // Policy Evaluation
11     while episode not complete do
12         receive feedback on a state $s'$;
13         if first visit of $s'$ then
14             append feedback value to all $Returns(s,a)$ that led to $s'$;
15         end
16         $Q(s,a) = AVG(Returns(s,a))$;
17         if positive feedback then
18             $a' = \pi(s')$;
19         else
20             remove link;
21         end
22     end
23     // Policy Improvement
24     for all states s in episode do
25         $a^* = argmax_a Q(s,a)$;
26         for all $a \in A(s)$ do
27             if $a = a^*$ then
28                 $\pi(s,a) = 1 - \epsilon + \frac{\epsilon}{|A(s)|}$;
29             else
30                 $\pi(s,a) = \frac{\epsilon}{|A(s)|}$;
31             end
32         end
33     end
34 end
```

has a total probability of $\epsilon - \frac{\epsilon}{|A(s)|}$). This is to ensure continuous exploration while at the same time moving towards a greedy policy. During the next episode, when a state that exists in the policy is encountered, a greedy action is taken with high probability, while other actions are explored with low probability.

3.7 Soundness of ALEX

This section presents a proof that: (1) Policy improvement always yields a better policy unless the policy is already optimal. (2) This property applies for the ϵ-greedy policy used in ALEX.

The value function discussed thus far is the action-value function $Q^\pi(s,a)$, which defines the expected return at a state s when choosing an action a following some policy π. Another value function, which is needed in this proof, defines the expected return, $V^\pi(s)$, for a state s under policy π:

$$\begin{aligned} V^\pi(s) &= E_\pi\left\{R_t | s_t = s\right\} \\ &= E_\pi\left\{\sum_{k=t+1}^{T} r_k | s_t = s\right\} \end{aligned} \tag{3.10}$$

This value function is called the *state-value function* for policy π. It defines the expected value given that ALEX follows policy π at state s. Similar to Equation 3.5, the value function of state s can be represented as a recursive relationship with the value function of the next state, s':

$$\begin{aligned} V^\pi(s) &= E_\pi\left\{\sum_{k=t+1}^{T} r_k | s_t = s\right\} \\ &= E_\pi\left\{r_{t+1} + \sum_{k=t+2}^{T} r_k | s_t = s\right\} \\ &= \sum_a \pi(s,a) \sum_{s'} P^a_{ss'}\left[R^a_{ss'} + E_\pi\left\{\sum_{k=t+2}^{T} r_k | s_{t+1} = s'\right\}\right] \\ &= \sum_a \pi(s,a) \sum_{s'} P^a_{ss'}\left[R^a_{ss'} + V^\pi(s')\right] \end{aligned} \tag{3.11}$$

Also, similar to Equation 3.7, the optimality equation for state-value function V^* is given by:

$$\begin{aligned}
V^*(s) &= max_a Q^{\pi^*}(s,a) \\
&= max_a E_{\pi^*}\left\{R_t | s_t = s, a_t = a\right\} \\
&= max_{a\in A(s)} E_{\pi^*}\left\{\sum_{k=t+1}^{T} r_k | s_t = s, a_t = a\right\} \\
&= max_{a\in A(s)} E_{\pi^*}\left\{r_{t+1} + \sum_{k=t+2}^{T} r_k | s_t = s, a_t = a\right\} \\
&= max_{a\in A(s)} E_{\pi^*}\left\{r_{t+1} + V^*(s_{t+1}) | s_t = s, a_t = a\right\} \\
&= max_{a\in A(s)} \sum_{s'} P^a_{ss'}\left[R^a_{ss'} + V^*(s')\right]
\end{aligned} \tag{3.12}$$

I now turn to proving that policy improvement always yields a better policy. The proof will use the state-value function that was just defined. For ease of explanation, assume that the policy is deterministic. However, the concepts discussed here can be applied to probabilistic policies like the one used by ALEX.

The approach to proving that policy improvement always yields a better policy can be illustrated with an example: Assume that the policy currently being used is π, and that for some state s the policy needs to be changed to choose an action $\pi'(s) = a \neq \pi(s)$. The value of the state, given that the policy π is followed, is given by $V^\pi(s)$. The question is whether it would be better or worse to change the policy so that it always chooses a at state s. The value of choosing a at s and then continuing for future states following the original policy π can be given by:

$$\begin{aligned}
Q^\pi(s,a) &= E_\pi\left\{r_{t+1} + V^\pi(s_{t+1}) | s_t = s, a_t = a\right\} \\
&= \sum_{s'} P^a_{ss'}\left[R^a_{ss'} + V^\pi(s')\right]
\end{aligned} \tag{3.13}$$

If it turns out that following policy π' only at state s (i.e., choosing action a), and then following policy π for future states gives greater state-value than the value of the state when following policy π, this situation can be represented by the following inequality:

$$Q^\pi(s,\pi'(s)) \geq V^\pi(s) \tag{3.14}$$

I want to show that if Equation 3.14 holds, then always following policy π' at state s yields a greater value than following policy π. That is, I want to prove that

$$V^{\pi'}(s) \geq V^\pi(s) \tag{3.15}$$

This can be proved by starting from Equation 3.14 and expanding $Q^\pi(s, \pi'(s))$ using Equation 3.13:

$$\begin{aligned}
&V^\pi(s) \leq Q^\pi(s, \pi'(s)) \\
&V^\pi(s) \leq E_{\pi'}\left\{r_{t+1} + V^\pi(s_{t+1}) | s_t = s\right\} \\
&V^\pi(s) \leq E_{\pi'}\left\{r_{t+1} + Q^\pi(s_{t+1}, \pi'(s_{t+1})) | s_t = s\right\} \\
&V^\pi(s) \leq E_{\pi'}\left\{r_{t+1} + E_{\pi'}\{r_{t+2} + V^\pi(s_{t+2})\} | s_t = s\right\} \\
&V^\pi(s) \leq E_{\pi'}\left\{r_{t+1} + r_{t+2} + V^\pi(s_{t+2}) | s_t = s\right\} \\
&V^\pi(s) \leq E_{\pi'}\left\{r_{t+1} + r_{t+2} + r_{t+3} + V^\pi(s_{t+3}) | s_t = s\right\} \\
&V^\pi(s) \leq V^{\pi'}(s)
\end{aligned} \tag{3.16}$$

So far, it was shown that a change in the policy at a single state to a particular action affects the state-value of this state, given the policy and the evaluation function. It was also shown that greedily choosing a policy that increases the action-value function at state s improves the overall policy. This reasoning can be extended to all states and all possible actions by selecting the action a that yields the highest $Q^\pi(s, a)$ at each state s. This is the greedy policy π' defined by:

$$\begin{aligned}
\pi'(s) &= argmax_a Q^\pi(s, a) \\
&= argmax_a E\left\{r_{t+1} + V^\pi(s_{t+1}) | s_t = s, a_t = a\right\} \\
&= argmax_a \sum_{s'} P^a_{ss'}\left[R^a_{ss'} + V^\pi(s')\right]
\end{aligned} \tag{3.17}$$

Thus far, it was proved that a greedy policy π' is as good as, or better than, an arbitrary policy π. If it is assumed that the greedy policy π' is as good as, but not better than, policy π (i.e.,$V^{\pi'} = V^\pi$), this can be expressed as:

$$\begin{aligned}
V^{\pi'}(s) &= max_a E\left\{r_{t+1} + V^{\pi'}(s_{t+1}) | s_t = s, a_t = a\right\} \\
&= max_a \sum_{s'} P^a_{ss'}\left[R^a_{ss'} + V^{\pi'}(s')\right]
\end{aligned} \tag{3.18}$$

This equation is the same as the optimality Equation 3.12. Thus, $V^{\pi'}$ must be V^*, and π and π' must be the optimal policies. This means that policy improvement must give a better policy unless the policy is already optimal. This also proves Equation 3.9 because each π_{k+1} is uniformly better than π_k, or equal to it if both are optimal policies:

$$\begin{aligned}
Q^{\pi_k}(s, \pi_{k+1}(s)) &\geq Q^{\pi_k}(s, \pi_k(s)) \\
&\geq V^{\pi_k}(s)
\end{aligned} \tag{3.19}$$

To show that policy improvement is sound for the ϵ-greedy probabilistic policy used by ALEX, let π' be the ϵ-greedy policy. It must be proved that the inequality $Q^\pi(s, \pi'(s)) \geq$

$V^{\pi}(s)$ holds. This can be done by expanding the left-hand side of the inequality as follows:

$$\begin{aligned} Q^{\pi}(s,\pi'(s)) &= \sum_a \pi'(s,a)Q^{\pi}(s,a) \\ &= \frac{\epsilon}{|A(s)|}\sum_a Q^{\pi}(s,a) + (1-\epsilon)max_a Q^{\pi}(s,a) \\ &\geq \frac{\epsilon}{|A(s)|}\sum_a Q^{\pi}(s,a) + \\ (1-\epsilon)\sum_a &\frac{\pi(s,a) - \frac{\epsilon}{|A(s)|}}{1-\epsilon} Q^{\pi}(s,a) \end{aligned} \tag{3.20}$$

The transition from the equality to the inequality is because the sum of the second term is a weighted average with non-negative weights summing to one ($\sum_a \frac{\pi(s,a)-\frac{\epsilon}{|A(s)|}}{1-\epsilon}$), and therefore must be less than or equal to the largest number averaged ($max_a Q^{\pi}(s,a)$).

$$\begin{aligned} Q^{\pi}(s,\pi'(s)) &\geq \frac{\epsilon}{|A(s)|}\sum_a Q^{\pi}(s,a) + \sum_a \pi(s,a)Q^{\pi}(s,a) \\ -\frac{\epsilon}{|A(s)|}&\sum_a Q^{\pi}(s,a) \\ Q^{\pi}(s,\pi'(s)) &\geq \sum_a \pi(s,a)Q^{\pi}(s,a) \\ Q^{\pi}(s,\pi'(s)) &\geq V^{\pi}(s) \end{aligned} \tag{3.21}$$

Equation 3.21 shows that policy improvement is sound for the ϵ-greedy policy. Using a greedy policy guarantees improvement in every step except when an optimal policy is reached. This analysis is independent of how the action-value functions are determined at each iteration.

3.8 Optimizations to ALEX

This section describes some optimizations incorporated into ALEX to improve execution time and reduce the number of iterations required for convergence.

3.8.1 Filtering to Reduce the Search Space

ALEX searches in the space of all possible links between entities in two data sets. It is computationally expensive to: (1) Construct the space of feature sets for each pair of entities from the two data sets. (2) Search for candidate links in this space. It is important to reduce the search space to eliminate unlikely links since the number of correct links is considerably small compared to the number of all possible links.

When computing a value for a feature (i.e., a similarity score between two objects associated with two predicates), ALEX keeps only features whose value is above a certain threshold θ. Feature values less than θ are set to zero. Feature sets that do not have any positive values are dropped. The experiments in this chapter use a value of $\theta = 0.3$.

3.8.2 Partitioning the Search Space

The search conducted by ALEX in the space of possible links can be parallelized by partitioning the space into independent partitions that do not require communication. In order to achieve this, the larger data set is partitioned and feature sets between each partition and all entities in the smaller data set are generated. Assume the first data set Ds_1 is larger than the second data set Ds_2. Ds_1 is partitioned into $\{Ds_{11} \cup Ds_{12} \cup \cdots \cup Ds_{1n}\}$. Feature sets are generated for each pair $\{(Ds_{11}, Ds_2), (Ds_{12}, Ds_2), \cdots, (Ds_{1n}, Ds_2)\}$. Feedback can then be directed to all partitions so that ALEX can take actions and explore new links in each partition. The different partitions can be independently explored in parallel, either on different CPU cores of the same machine or on multiple machines in a distributed setting.

ALEX uses a simple partitioning technique that is called *equal-size partitioning*. Equal-size partitioning divides the larger data set into equal-sized partitions in a round-robin fashion. That is, the ith entity is in partition $i \; mod \; n$, where n is the number of partitions. Equal-size partitioning enables parallelism that significantly reduces execution time without sacrificing the quality of candidate links.

3.8.3 Optimizations for Handling Incorrect Links

The actions taken by ALEX (exploring links that did not exist in the set of candidate links) lead to fast improvement in recall. However, ALEX can also generate incorrect links, which reduces precision. Negative feedback would eventually correct these errors by removing incorrect links. Based on negative feedback, ALEX learns that some action resulted in worse returns. During policy improvement, ALEX would change the policy so that this action is chosen only with small probability. However, relying only on policy improvement to remove incorrect links may result in slow convergence. In order to speed up convergence, ALEX uses two optimizations to improve precision without waiting for policy improvement: *blacklist* and *rollback*.

Blacklist: When negative feedback is received over a link, it is now known that the link is incorrect, so it is added to a list of incorrect links, which is called the blacklist. The blacklist is used to prevent links that are known to be incorrect from being returned by ALEX when exploring links at any state in the future.

Rollback: The probabilistic nature of the ϵ-greedy policy used by ALEX allows it to choose incorrect actions at any state to learn how to make better choices when choosing future actions. Some actions may result in the discovery of a large number of incorrect links. When this happens, it is a wise choice to rollback and remove the links generated by such actions. ALEX traces feedback on links to know which state-action pair was used to generate these links. When a sufficient number of negative feedback instances is received over links generated by a specific state-action pair, a rollback process is initiated, and all links generated by this state-action pair are removed. However, links removed without being marked with negative feedback are not added to the blacklist since they may include correct links. These links can be discovered later by another state-action pair with a better average return.

The rollback optimization is particularly useful for handling incorrect feedback due to errors in the data or errors by the user. It may be possible to refine the feedback so that ALEX uses only high quality feedback obtained from a large number of users (e.g., using techniques from [98]). However, feedback is not expected to be 100% correct, regardless of the measures taken to improve its quality. When incorrect feedback is received by ALEX, it will take an action that can be rolled-back if future feedback contradicts the incorrect feedback.

3.9 Experimental Evaluation

3.9.1 Experimental Setup

Data sets: The real data sets shown in Table 3.1 in Section 3.4 are used in the experiments. The linked pairs of data sets are shown in Table 3.2 along with the number of *owl:sameAs* links between each pair. These are the pairs used in the experiments.

Initial Set of Links: PARIS [122] is used as the automatic linking algorithm for generating the initial set of candidate links for ALEX. PARIS was shown to outperform other techniques, and it is not domain specific. PARIS produces links where each link is associated with a score. In order to find better quality links from PARIS, only links with score greater than 0.95 are used. Lowering this threshold does not improve recall, but it lowers precision.

Ground Truth: The data sets used in the experiments in this chapter are part of the LOD cloud, which means that they are already linked. In these experiments, all existing links between the data sets are removed and used as the ground truth. In addition, I randomly inspect samples from the ground truth to remove any incorrect links. PARIS is then run over the pair of data sets to be linked to discover candidate links, which ALEX uses as initial candidate links. I manually inspect the initial candidate links generated by PARIS to find correct links that do not exist in the ground truth. If such links are found, they are added to the ground truth.

Generating Feedback: A link is randomly chosen out of the set of candidate links and compared to the ground truth. If the link exists in the ground truth, a positive feedback instance is returned to ALEX. If the link is incorrect (i.e., does not exist in the ground truth), a negative feedback instance is returned.

Evaluation Metrics: The efficiency of ALEX is evaluated by comparing the candidate links it generates to the ground truth. This comparison is performed after each policy evaluation/policy improvement iteration, i.e., after each episode of feedback. The quality of candidate links is evaluated using precision $P = \frac{|C \cap G|}{|C|}$, recall $R = \frac{|C \cap G|}{|G|}$, and F-measure $F = \frac{2PR}{P+R}$, where C is the set of candidate links after each episode, and G is the ground truth. The execution time that ALEX requires to converge is also measured. The execution time includes the exploration of new candidate links and improving the policy after each episode.

Default Settings: The initial step size for defining the exploration band is set to 0.05. Thus, if a feature has a score of 0.8, ALEX will find links whose feature score is in the range [0.75, 0.85]. By default, dynamic bands are used for exploration, so the upper bound of the band increases as explained in Section 3.6. When using a static exploration band, the step size remains at 0.05. The default episode size is 1000. All the experiments use equal-size partitioning to partition the space of feature sets into 27 partitions.

Execution Environment: ALEX is implemented in Java. The experiments are run on a shared server running Linux Ubuntu 16.04.3 with 64 Intel Xeon CPU at 2.6 GHz and 256 GB of memory. The memory usage for the largest data sets never exceeded 60 GB.

Section 3.9.2 evaluates the quality of the links discovered by ALEX. Section 3.9.3 evaluates the efficiency of ALEX and the effect of the optimizations it employs. Section 3.9.4 evaluates the effect of incorrect feedback on the quality of links. Section 3.9.5 evaluates the sensitivity of ALEX to change in parameter values.

3.9.2 Quality of Links

I envision ALEX being used in one of two settings: 1. Batch Mode: A service provider can give users the ability to query multiple, large linked RDF data sets. In this setting, the service provider collects feedback from many users on a large number of links between different parts of the data sets. ALEX applies the feedback in batches, using a large episode size, in order to ensure that there is sufficient feedback on different parts of the data sets. In this setting, a default episode size of 1000 (e.g., 1000 users providing 1 feedback instance each) is used. 2. Specific Domains: Individual users can develop applications that target more specific domains, either small data sets or subsets of large data sets. The user feedback is focused on a specific domain (e.g., a small part of the DBpedia data set), and she expects to see quick improvement in link quality based on her feedback. In this setting, a small episode size of 10 is used. In both settings, a strict rule for convergence is used: ALEX stops when there is no change at all in the set of candidate links between episodes. The episode at which fewer than 5% of the links change compared to the previous episode is also shown. This can be used as a more relaxed convergence rule.

ALEX in Batch Mode

Figure 3.9 shows how ALEX performs in batch mode. The figure shows the quality of links after each episode (i.e., iteration of policy evaluation/policy improvement). The figure shows that in most cases, the recall value starts from a low value (i.e., most ground truth links are not included). This recall value is significantly improved after the first episode. This means that a large number of links have been discovered and added to the set of candidate links after only one episode. In some iterations, the precision is hurt by adding some incorrect links to the set of candidate links. However, ALEX recovers fast and keeps improving both precision and recall until it converges. The vertical green line in this and subsequent figures shows the episode at which the number of changed links from the previous episode is less than 5%. For example, in Figure 3.9(a), this relaxed convergence happens after 5 episodes in this experiment, while full convergence (i.e., no change in links) happens after 8 episodes. An exception to this pattern is Figure 3.9(e), where PARIS is able to generate a set of initial candidate links with relatively good recall but with bad precision. The figure shows how ALEX performs in this case. The figure shows that the automatically generated links have a low starting precision value (less than 0.3), and high recall value (0.95). ALEX is able to significantly improve the precision value after three episodes. The recall value is also improved. ALEX converges after 4 episodes in this experiment (3 episodes with the relaxed condition), reaching an F-measure of 0.97. Figures 3.9(g) and 3.9(h) shows the case when ALEX starts with an output with higher precision. After the first episode, the precision drops before recovering in later episodes until convergence.

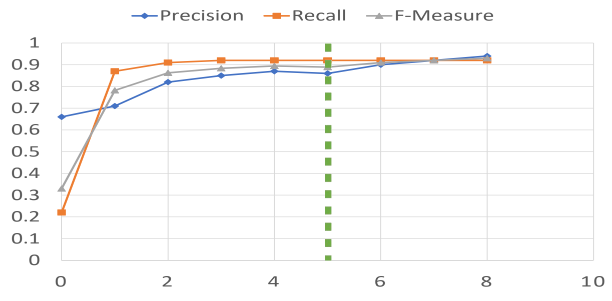
ALEX for Specific Domains

For this setting, the following pairs of data sets are used: (DBpedia, UK Learning Providers), (Geonames, Linkedmdb), and (Geonames, Lexvo). In addition, two subsets of DBpedia and OpenCyc about NBA basketball players are extracted and used with NYTimes. These pairs are specific domain data sets and are linked by a small number of *owl:sameAs* links.

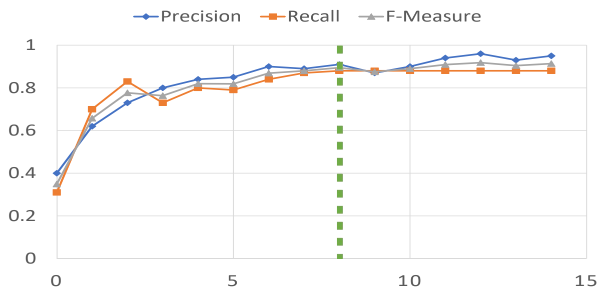
Figure 3.10(a) shows the performance of ALEX between DBpedia and the UK Learning Providers data set. The figure shows that ALEX achieves very good quality of links and converges in 3 episodes (i.e., significant improvement after 10 feedback instance, and full convergence after 30). Figures 3.10(b) and 3.10(c) show similar behaviour where ALEX converges after only 2 iterations. There are 93 links in the ground truth of DBpedia - NYTimes (NBA basketball players), and 35 for OpenCyc - NYTimes (NBA basketball players). Figures 3.10(d) and 3.10(e) show that ALEX significantly improves the quality of links for these data sets after one iteration and converges after just two iteration.

Finding Distinctive Features

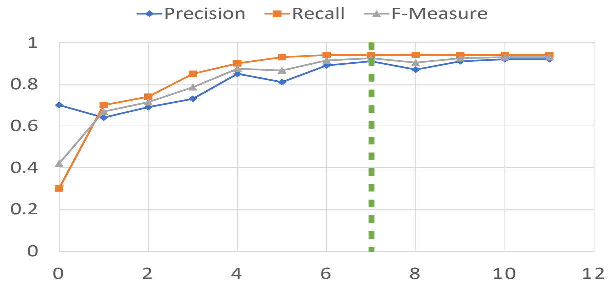
One question that relates to the quality of the links discovered by ALEX is: Does ALEX identify distinctive features? Recall that all the distinctive features were identified in Section 3.4. It is possible to check the policy of ALEX to see if it explores around these distinctive features. Specifically, when ALEX converges, if a distinctive feature is a feature that ALEX chooses with high probability for any state, then this means that ALEX has identified this feature as a distinctive one. This is desirable since it should improve the



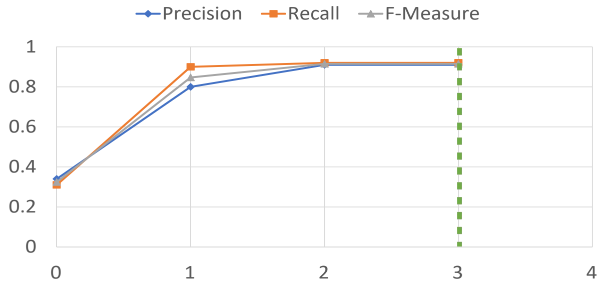
(a) DBpedia - NYTimes



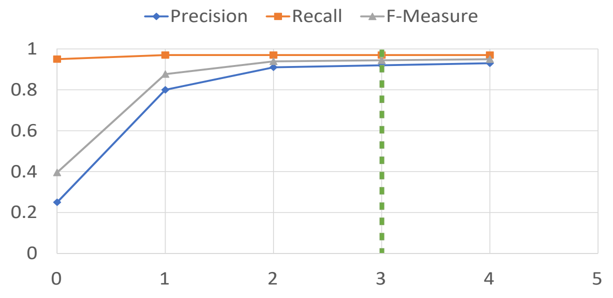
(b) DBpedia - OpenCyc



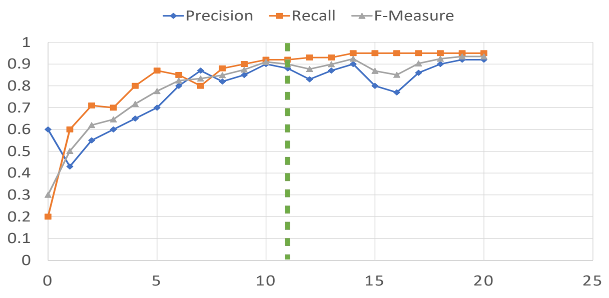
(c) DBpedia - Linkedmdb



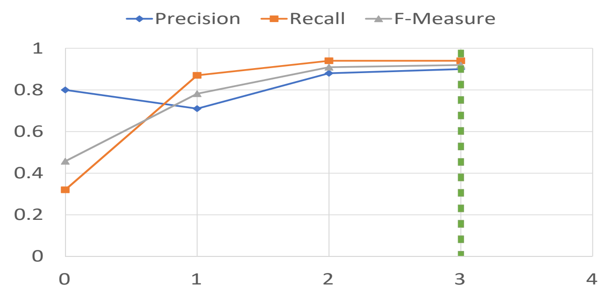
(d) DBpedia - Lexvo



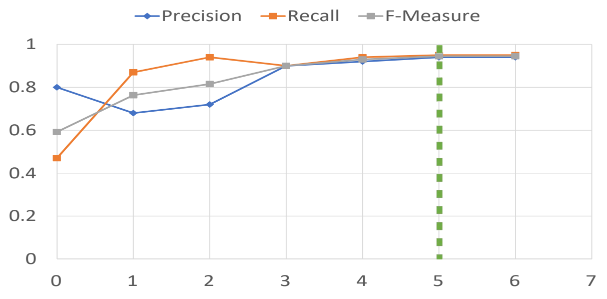
(e) DBpedia - Drugbank



(f) DBpedia - Geonames

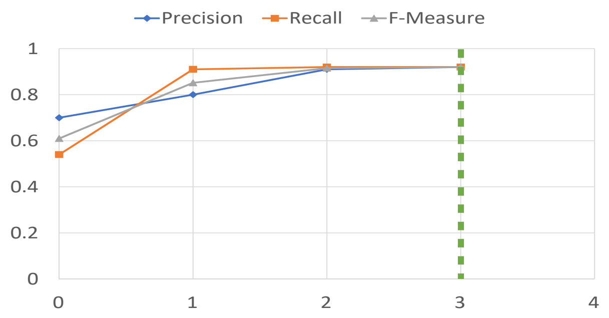


(g) Geonames - NYTimes

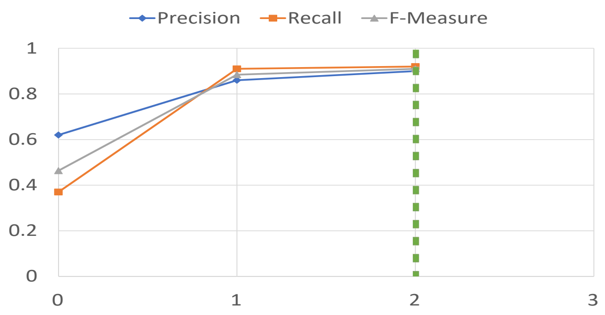


(h) OpenCyc - Umbel

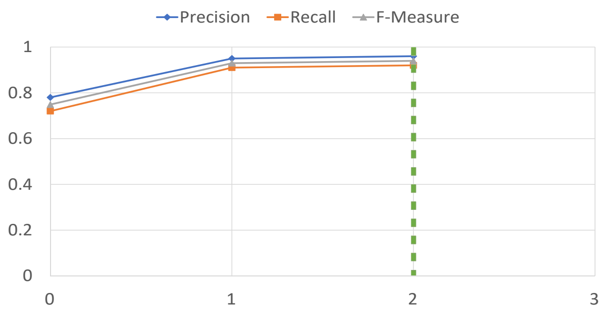
Figure 3.9: Quality of links for the batch mode setting. The x-axis is the episode number and the y-axis is the quality value.



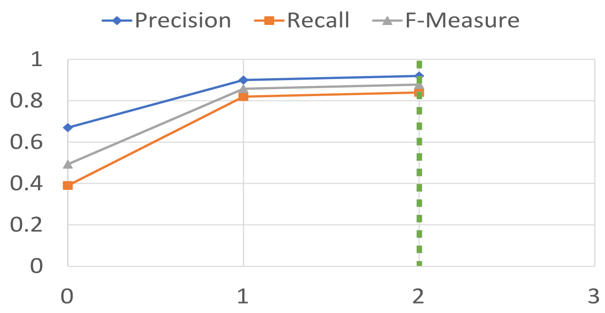
(a) DBpedia - UK Learning Providers



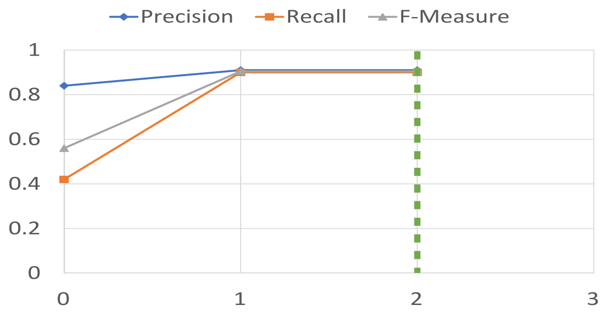
(b) Geonames - Linkedmdb



(c) Geonames - Lexvo



(d) DBpedia (NBA) - NYTimes



(e) OpenCyc (NBA) - NYTimes

Figure 3.10: Quality of links for the specific domain setting. The x-axis is the episode number and the y-axis is the quality value.

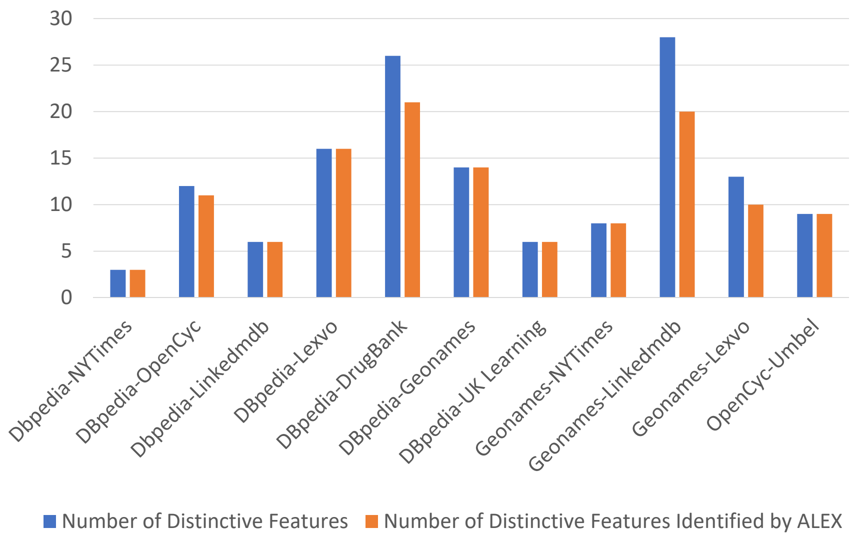


Figure 3.11: Number of distinctive features identified by ALEX.

quality of the discovered links. Formally, if a' is a distinctive feature, then ALEX is said to identify it if $\exists s \in S|\pi(s) = a'$. Figure 3.11 shows the number of distinctive features identified by ALEX. The figure shows that ALEX is able to identify almost 87% of the distinctive features in the data set pairs, which is one of the reasons ALEX identifies high quality links and converges fast.

3.9.3 Efficiency of ALEX

This section investigates the efficiency (i.e., running time) of ALEX, and the effect of the different optimizations to speed up convergence.

An important factor affecting convergence is having dynamic exploration bands compared to the static bands used in [56]. Table 3.3 shows each data set pair with the number of episodes required for ALEX to converge under two settings: (1) using a static band with step size 0.05, as in [56], and (2) using a dynamic band that grows as described in Section 3.6. The table also shows the saving as a percentage. The table shows that using a dynamic exploration band leads to convergence with 25% to 60% less feedback, with an average improvement of 39.3%.

Next, the optimizations proposed in Section 3.8 are evaluated. For the following experiments, the DBpedia and NYTimes data sets are used. These data sets are challenging for

Data Set Pair	No. of Episodes (Static Band)	No. of Episodes (Dynamic Band)	Feedback Saved
DBpedia - NYTimes	14	8	**43%**
DBpedia - OpenCyc	20	14	**30%**
DBpedia - Linkedmdb	18	11	**39%**
DBpedia - Lexvo	5	3	**40%**
DBpedia - Drugbank	10	4	**60%**
DBpedia - Geonames	45	20	**56%**
DBpedia - UK Learning	4	3	**25%**
Geonames - NYTimes	5	3	**40%**
Geonames - Linkedmdb	3	2	**33%**
Geonames - Lexvo	3	2	**33%**
OpenCyc - Umbel	9	6	**33%**

Table 3.3: Savings due to using a dynamic exploration band compared to a static band.

ALEX since they contain data from more heterogeneous domains than the rest of the data sets. The default episode size is similar to ALEX in batch mode (1000 feedback instances per episode).

Filtering to Reduce the Search Space: Figure 3.12 shows the total number of links that can be generated between the first partition of the DBpedia data set and the whole data set of NYTimes, the number of links after filtering using a threshold $\theta = 0.3$, and the number of ground truth links in this partition (note that the y-axis is in log scale). The figure shows that filtering is highly effective and reduces the search space by 95%. Ground truth links represent only 0.2% of the filtered links, demonstrating the efficiency of ALEX, which is able to discover correct links in such a large space.

Blacklist: Figure 3.13 shows a comparison between ALEX with and without the blacklist optimization. Figure 3.13(a) shows that using a blacklist gives a slight improvement in F-measure over not using it. Figure 3.13(b) shows that using a blacklist significantly decreases the fraction of negative feedback that the user provides on incorrect links between the data sets. Using a blacklist does not affect the execution time of ALEX. Intuitively, a black list is useful because when a user provides negative feedback on a link she should not need to provide this feedback again.

Rollback: ALEX learns by interacting with the environment. This means that it can sometimes make wrong decisions. These wrong decisions can result in exploring a large number of incorrect candidate links, significantly reducing the quality of the discovered

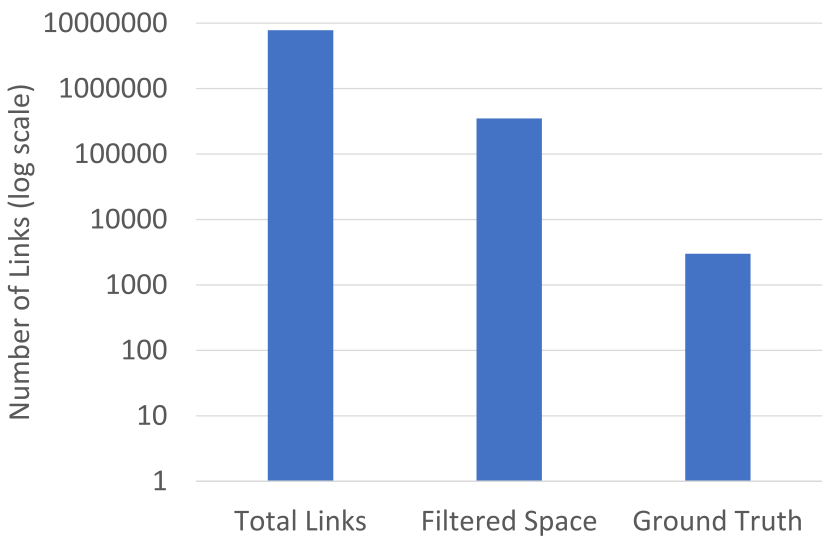
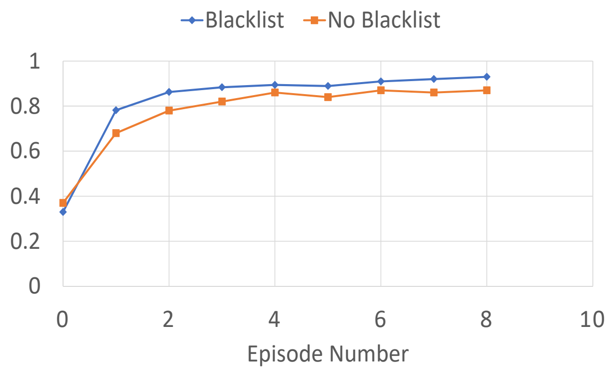
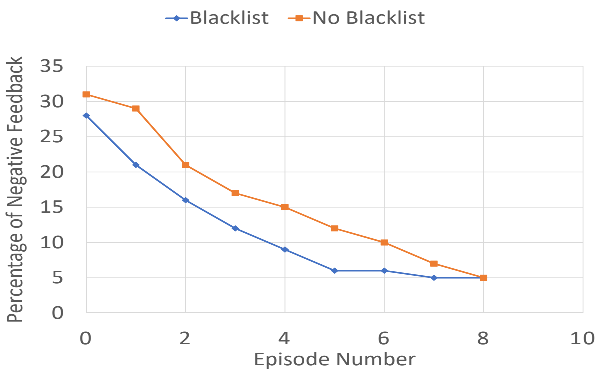


Figure 3.12: Comparing number of links: total possible links vs. filtered search space vs. ground truth.



(a) F-measure



(b) Negative feedback

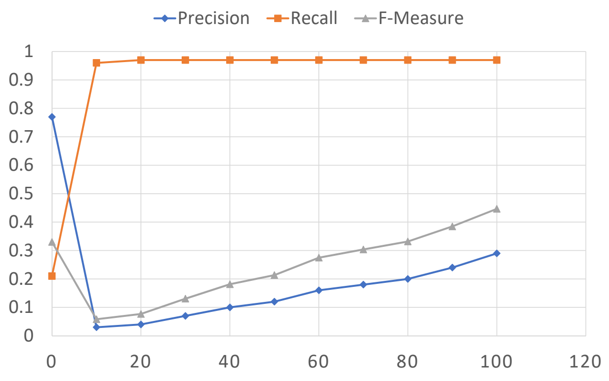
Figure 3.13: Effect of the blacklist: (a) F-measure, and (b) negative feedback.

links. If rollback is not used, ALEX can recover from wrong decisions only very slowly. Figure 3.14 shows the importance of the rollback optimization. Figure 3.14(a) shows the quality measures of ALEX without using the rollback optimization. This figure should be contrasted to Figure 3.9(a), which shows ALEX with rollback (the default). The figure shows that after the first episode, precision drops to a very low value. The figure also shows that it is hard to recover from the wrong decisions made during the first episode. After 100 episodes, which is the maximum number of iterations allowed by ALEX, precision is a little over 0.3. Figure 3.14(a) shows the overall quality of all partitions. If the partitions are examined independently, some partitions are found to be able to recover from wrong decisions made by ALEX, while others are not. Figure 3.14(b) shows an example of a partition that is able to recover from the wrong decisions and converges in 40 episodes. The same partition converges in 7 episodes when rollback is applied. However, another partition, shown in Figure 3.14(c), cannot recover from wrong decisions.

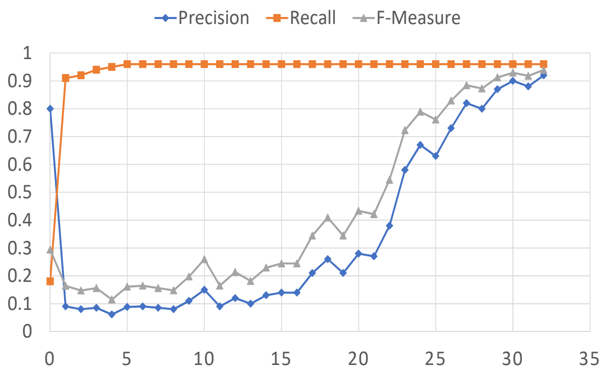
Execution Time: In the experiment with DBpedia and NYTimes shown in Figure 3.9(a), ALEX finishes execution in 84 minutes, which is the execution time of the slowest partition. This is approximately 10 minutes per episode. The average execution time of all partitions is approximately 55 minutes. In the specific domain experiment shown in Figure 3.10(a), ALEX finishes in approximately 4 seconds. This is approximately 1.3 second per episode (10 feedback instances). The faster convergence in the specific domain setting is because the data sets and the amount of feedback are smaller. Thus, in batch mode ALEX takes a few minutes per episode, while in interactive mode it takes a few seconds. This is acceptable efficiency for the application scenarios envisioned for ALEX.

3.9.4 Effect of Incorrect Feedback

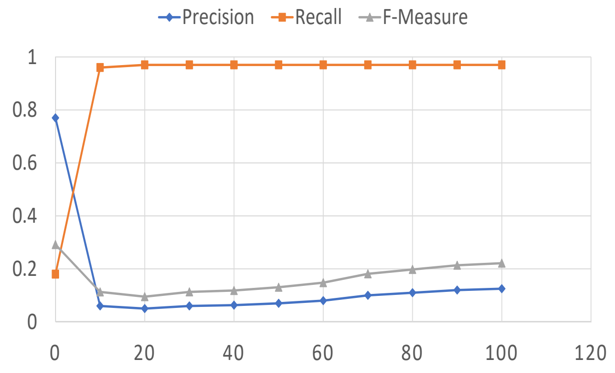
In this chapter, it is assumed that user feedback is always correct. However, in real-life scenarios, users may not agree on the correctness of query answers. Therefore, incorrect feedback items may be encountered. In order to evaluate ALEX in this context, random incorrect feedback items are generated such that 10% of the feedback items received by ALEX are incorrect. ALEX is evaluated with 10% incorrect feedback on the DBpedia - NYTimes data sets, using the default episode size of 1000. Figure 3.15 shows the precision, recall, and F-measure for ALEX with 10% incorrect feedback, and the corresponding values when all feedback is correct (from Figure 3.9(a)). The recall value in Figure 3.15(b) does not vary much, demonstrating that the reinforcement learning techniques used by ALEX are robust to incorrect feedback. The precision values shown in Figure 3.15(a) are slightly worse when there is incorrect feedback, and this also affects the F-measure in Figure 3.15(c). ALEX improves precision by removing incorrect links from the set of candidate links, and



(a) Quality without rollback

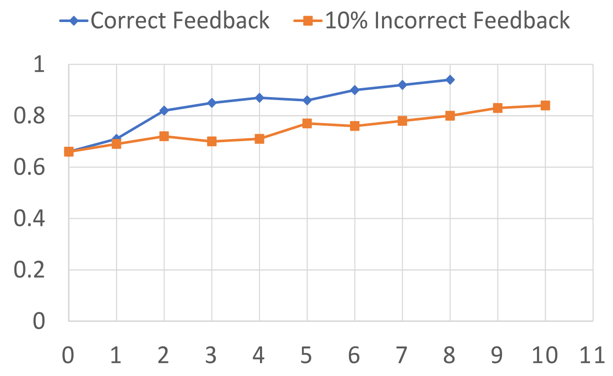


(b) A partition that converges

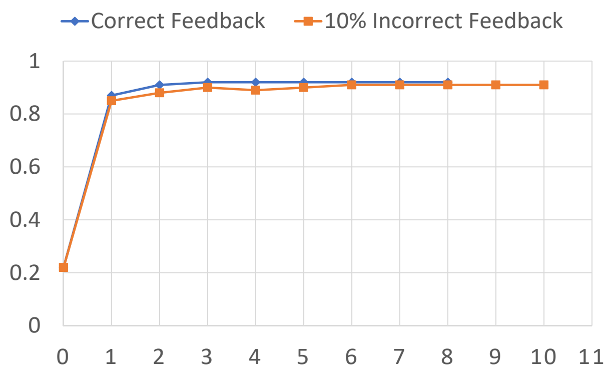


(c) A partition that does not converge

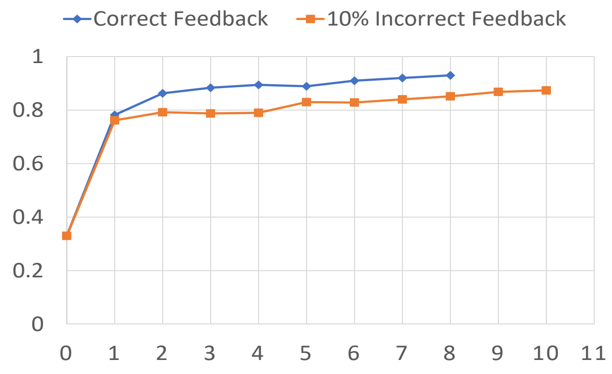
Figure 3.14: Effect of rollback: (a) quality without rollback, (b) a partition that converges, and (c) a partition that does not converge.



(a) Precision



(b) Recall



(c) F-measure

Figure 3.15: ALEX with correct feedback and with 10% incorrect feedback.

these incorrect links can be removed only when negative feedback is received over them. When there is incorrect feedback, a constant stream of positive feedback is received over incorrect links, so these incorrect links stay in the set of candidate links. Nevertheless, the degradation in precision is relatively small, and ALEX is able to produce good results even in the presence of incorrect feedback.

3.9.5 Sensitivity of ALEX to Parameter Values

Step Size: ALEX uses a dynamic step size, which doubles the upper bound of the exploration band after each episode (termed the *double step* approach). In addition to comparing a static step size to using a dynamic step size, another approach to dynamic

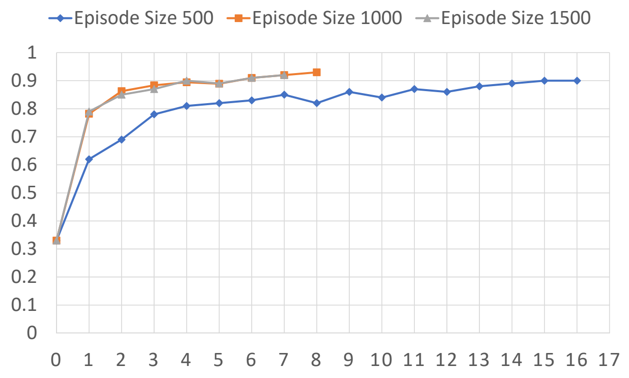


Figure 3.16: F-measure for ALEX with different episode sizes.

step sizing is also evaluated. In this approach, the step size extends the upper bound with more conservative increases. Specifically, the step size α_i changes according to $\alpha_{i+1} = MIN(1, \alpha_i + \alpha_0)$, where i is the episode number. The band size then changes to $[feature\_score - \alpha_0, feature\_score + \alpha_i]$. This approach is termed the *incremental step* approach. An experiment on the DBpedia and NYTimes data sets shows that both the incremental step and double step approaches achieve slightly better quality of links compared to the static step approach. More importantly, they converge in fewer episodes: the double step, incremental step, and static step approaches converged in 8, 10, and 14 episodes, respectively. This validates the choice of using the double step approach.

Episode Size: Like changing the step size, changing the episode size slightly affects the quality of the discovered links. Figure 3.16 shows a comparison of ALEX using an episode size of 500, 1000 (default), and 1500 on the DBpedia and NYTimes data sets. During each episode, policy evaluation uses the current policy to take action after every feedback item. At the end of an episode, the policy is improved for the next episode. Figure 3.16 shows the F-measure for the three episode sizes. The F-measures are very close to each other, with episode sizes 1000 and 1500 slightly outperforming episode size 500. A larger episode size results in ALEX taking fewer episodes to converge, since each episode has more feedback. ALEX converges in 16, 8, and 7 episodes for episode size 500, 1000, and 1500, respectively.

The experiments in this section show that ALEX is sensitive to changes in the values of its parameters. That is, the parameters have a noticeable effect on performance. However, the sensitivity is not so high as to make ALEX unstable and reliant on highly accurate parameter settings: the difference between the best case and the worst case performance is not too high, and reasonable choices of parameter values work well.

3.10 Conclusion and Future Work

This chapter presented ALEX, a system that utilizes user feedback on queries over linked data to remove incorrect links and discover new links that did not exist between the data sets. When a user provides positive feedback on a link, ALEX finds new links that are similar to this link. This exploration is conducted by taking one pair of attributes from the two data sets and exploring around the similarity value between these two attributes. ALEX uses a probabilistic policy to choose the attributes to explore around. This policy is learned and improved using a Monte Carlo reinforcement learning approach. Using this approach, ALEX learns how to find new links as the user continues giving feedback on query answers. ALEX uses several optimizations to speed up convergence. The experiments with real world data sets show the effectiveness and efficiency of ALEX.

ALEX uses syntactic approaches that are heavily guided by user feedback to find new correct links. When working with RDF data, it is possible to use semantic approaches that reason about the relationships between entities. Thus, a promising direction for future work is to investigate using semantic approaches to infer new correct links based on existing links and user feedback. For example, it may be possible to represent links as sets of axioms and functional dependencies, and to use logical inference on these sets to find new links while being guided by user feedback. Another direction for future work is to investigate using statistical or machine learning approaches to find new links. It may also be possible to extend ALEX to treat feedback from different users differently, for example, to give higher weight to feedback from users who are known to be credible and reliable. An interesting dimension for future work relates to the types of links that are explored. ALEX focuses on the links between individual entities, specifically *owl:sameAs*. Another (less commonly used) type of link between individual entities is *owl:differentFrom*. Thus, a possible direction for future work is to extend ALEX to find *owl:differentFrom* links based on user feedback. It is also possible to investigate a completely new direction for finding links, namely finding links between RDF classes. There are many types of links between RDF classes such as *rdfs:subClassOf*, *owl:equivalentClass*, *owl:disjointWith*, *rdfs:subPropertyOf*, *owl:equivalentProperty*, *owl:inverseOf*, *owl:FunctionalProperty*, and *owl:InverseFunctionalProperty*. An interesting direction for future work is to find these types of links based on user feedback, probably using a semantic (i.e., reasoning) rather than syntactic approach.

I conclude by noting that this chapter did not elaborate on how the user issues queries over multiple data sets from the LOD cloud. The problem of issuing such queries is far from trivial because it requires the user to have full knowledge of the structure and values of the queried data sets. This level of knowledge is not easy in practice. In the next chapter, I further discuss the problem and its challenges, and present Sapphire as a solution to this problem.

Chapter 4

Sapphire: Querying RDF Data Made Simple

As discussed earlier, querying multiple linked RDF data sets can be very useful to answer complex queries. Chapter 3 discussed utilizing user feedback over query answers to improve the quality of *owl:sameAs* links between the queried data sets. The assumption was that the user can compose SPARQL queries on the data as needed, which is often not an easy task. This chapter discusses the challenges of querying heterogeneous RDF data sets and describes how these challenges can be overcome using Sapphire [59], a tool that helps non-expert users to write syntactically and semantically correct SPARQL queries.

Sapphire targets technically knowledgeable users and enables them to query an RDF data set without requiring detailed prior knowledge of this data set. For example, programmers who can use Sapphire to gain knowledge of a new data set used in their work. Another example is a biologist who can use Sapphire to quickly and easily query heterogeneous RDF data sets about diseases, their pathophysiology, and drug-drug interactions. Sapphire provides an interface that allows users to compose SPARQL queries against the SPARQL endpoints of one or more RDF data source. Through this interface, Sapphire guides the user towards the SPARQL query that satisfies her information needs. Sapphire achieves this in two ways that both rely on a *predictive user model* that is built in an initialization phase. First, while a user is typing a query, Sapphire interactively provides her with data-driven suggestions to complete the predicates and literals in the query, similar to the auto-complete capability in many user interfaces. Second, when a user completes the query and submits it for execution, Sapphire suggests ways to modify the query into one that may be better suited to the needs of the user. For example, if the user query returns no answers, Sapphire would attempt to modify it into a query that does return answers.

Recall from Chapter 1 that answering questions over RDF data generally follows one of two approaches: (a) natural language queries, and (b) structured querying using SPARQL. Sapphire's query completion and query suggestion modules rely on natural language techniques. Thus, in the spectrum of approaches for querying RDF, Sapphire bridges the gap between the simple but ambiguous natural language approaches on the one hand, and the powerful but cumbersome SPARQL on the other. The novelty of Sapphire comes from the need to balance multiple conflicting goals: Sapphire must provide high quality recommendations that actually help the user to find the information that she needs, it must have fast response time since it is interactive, it must run on a reasonably sized machine without placing excessive demands on the machine's resources, and it must not overload the SPARQL endpoints that it queries. These design goals require judicious design choices which are presented in this chapter.

The contributions of this chapter are as follows:

- Summarizing the queried endpoints to collect concise, important data that is utilized by Sapphire.
- The predictive user model which is at the heart of Sapphire and includes two modules: query completion and query suggestion. This predictive user model helps the user while writing the SPARQL query by providing suggestions to complete the query, and after the query is issued by suggesting changes that can be made to the query to get answers that the user may be interested in.
- An extensive evaluation of Sapphire based on performance experiments and a user study. The evaluation demonstrates that Sapphire is significantly more effective than competing approaches at finding the answers to user queries, and it achieves interactive performance.

4.1 Related Work

4.1.1 Approaches to Querying RDF Data

The goal of Sapphire is to help users construct structured queries on RDF data without having full knowledge of the queried data sets. The prior work on this problem generally falls into three categories:

1. Natural language approaches, where a query is described using natural language.
2. Approximate structured queries, where the query posed by the user does not have to be exactly matched with the queried data sets.
3. Query by example approaches that construct structured queries based on examples of answers provided by the user.

Following is a discussion of the related work for these three approaches.

4.1.2 Natural Language Approaches to Querying RDF Data

Natural language approaches are simple for end-users and easy to use [85]. There has been a significant amount of research on querying information using keywords in relational databases [10, 20, 24, 36, 74, 75, 115, 147]. For RDF data, several prior works create structured queries based on natural language approaches [30, 41, 52, 64, 86, 91, 129, 142, 148, 151]. Each of these works focuses on one or more specific query templates, and uses keyword search or natural language questions to construct these templates and fill in the placeholders they contain. All of these approaches suffer from two limitations compared to Sapphire: (1) Their expressiveness is limited to specific query templates, and (2) inferring query structure, predicates, and literals based only on natural language is inherently ambiguous. In contrast, Sapphire can construct any SPARQL query, and it removes ambiguity by involving the user directly in query composition.

In this chapter, Sapphire is compared to QAKiS [30] and KBQA [41] as representatives of the state of the art in natural language approaches. QAKiS [30] is a question answering system over RDF that automatically extracts from Wikipedia different ways of expressing relations in natural language (e.g., “a bridge spans a river” and “a bridge crosses a river” express the same relation). These equivalent expressions are used to match fragments of a natural language question and construct the equivalent SPARQL query. KBQA [41] is a more recent question answering system that focuses on factoid questions. KBQA learns question templates from a large Q&A corpus (e.g., Yahoo! Answers), and learns mappings from these templates to RDF predicates in the queried data set. The templates and mappings are then used to answer user questions.

4.1.3 Approximate Structured Queries on RDF Data

This line of work goes beyond the fixed query templates used by natural language approaches, enabling the user to express approximate structured queries. That is, the query posed by the user does not have to be exactly matched with the queried RDF data [88, 87, 145]. These approaches are still limited in the query structure that they support, and they require the user to know the vocabulary and the approximate schema of the queried data sets. In contrast, Sapphire enables the user to compose any SPARQL query without prior knowledge of the queried data sets.

In this chapter, Sapphire is compared to S^4 [150], a recent system that was shown to outperform other approximate query approaches. S^4 summarizes and indexes the queried data set based on the RDF entity types. It maintains a graph of the relationships between RDF entity types based on the relationships between instances of these types. Queries are rewritten based on this graph. S^4 assumes that the user can issue queries using correct predicates and instance URIs in the data set, but possibly not with the correct query structure.

4.1.4 Querying By Example

In AIDE [48], the user selects a relational database and a set of attributes that are of interest to her. AIDE shows the user sample tuples and expects positive or negative feedback of whether the tuples are interesting or not. Based on this user feedback, AIDE predicts what other tuples may be interesting. For RDF data, [89] requires the user to specify examples of data that should be in the query answer and creates a SPARQL query based on these examples. This approach requires the user to know enough about the data to specify example answers, and can only create SPARQL queries of limited complexity. Similarly, SPARQLByE [14, 47] infers the SPARQL query that best suits the user's needs based on a set of example answers she provides. A key limitation of this approach is that the user needs to know a set of examples that satisfy her query, which is often not practical. For example, to answer the query *"How many people live in New York?"*, the user should know the precise population of some cities to provide as examples, which can be impractical. In contrast, Sapphire helps the user directly construct and refine a SPARQL query rather than indirectly inferring the query from example answers. This chapter compares Sapphire to SPARQLByE and shows that Sapphire is more expressive.

4.2 Sapphire Architecture and Challenges

This section presents the overall architecture of Sapphire, and an overview of the different design choices and challenges that must be addressed in order to implement a useful and

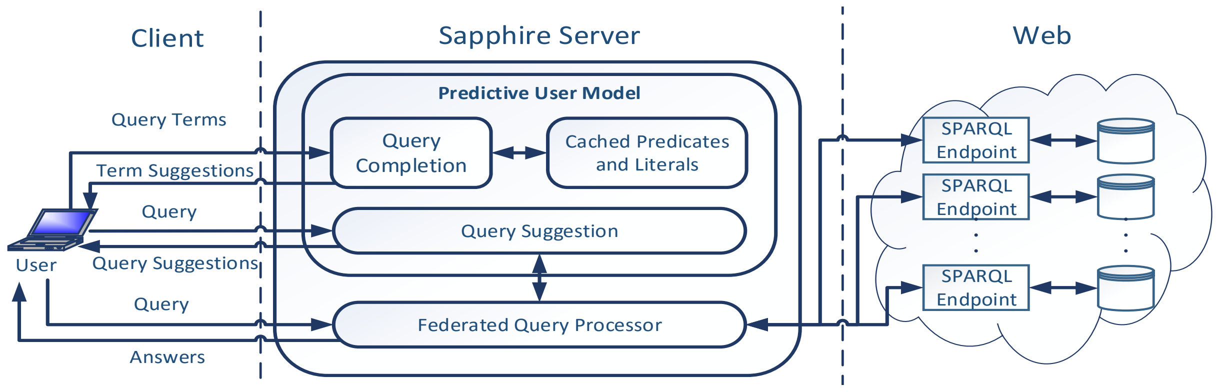


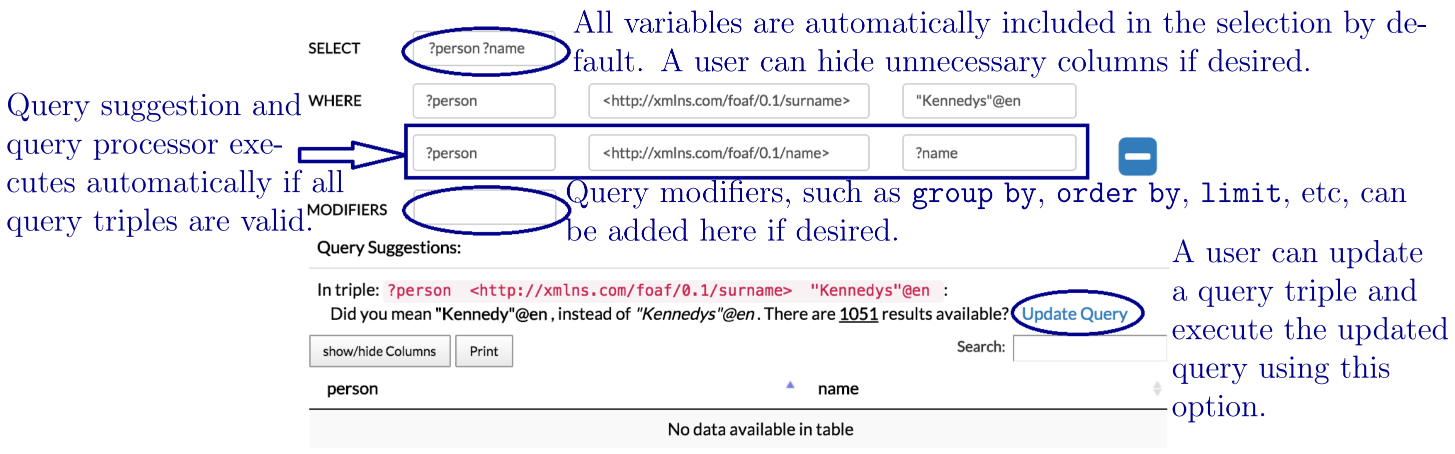
Figure 4.1: Architecture of Sapphire.

efficient system. Figure 4.1 shows the architecture of Sapphire. Sapphire runs as a server that sits between the user and the SPARQL endpoints for one or more RDF data sets on the web. Sapphire accesses the endpoints through a federated query processor (e.g., [109] or FedX [116]). Sapphire uses FedX [116], a widely-used federated query processor, but any other federated query processor can be used.

The core of Sapphire is the Predictive User Model (PUM), which helps the user express her information needs using SPARQL queries. The PUM relies on information about the data sets being queried. Before querying an endpoint, the user must register this endpoint with the Sapphire server, and the server goes through an initialization step in which it caches important data from this endpoint. One challenge that must be addressed by Sapphire is which data from an endpoint to cache, and how to retrieve this data without overloading the endpoint.

While the user is composing a query, the query terms are forwarded to the Query Completion Module (QCM) as they are typed by the user. The QCM interactively provides the user with suggestions to complete the terms in her query based on the data cached during initialization. A question that must be answered when designing the QCM is how to provide interactive response time even for the large scale of data in the LOD cloud.

After composing a syntactically correct query, the federated query processor executes the query and returns answers. Simultaneously, the Query Suggestion Module (QSM) suggests changes to the query to help the user find the answers she is looking for. The goal of the QSM is to suggest queries that are similar to the one issued by the user, but different enough to present her with useful alternatives that may help her satisfy her information needs. These suggestions span two directions: 1. Finding alternative literals and predicates to the ones used in the query. 2. Relaxing the structure of the issued query to *approximately*


All variables are automatically included in the selection by default. A user can hide unnecessary columns if desired.

Query suggestion and query processor executes automatically if all query triples are valid.

Query modifiers, such as `group by`, `order by`, `limit`, etc, can be added here if desired.

A user can update a query triple and execute the updated query using this option.

```
SELECT     ?person ?name
WHERE      ?person   <http://xmlns.com/foaf/0.1/surname>   "Kennedys"@en
           ?person   <http://xmlns.com/foaf/0.1/name>      ?name
MODIFIERS
Query Suggestions:
In triple: ?person <http://xmlns.com/foaf/0.1/surname> "Kennedys"@en :
  Did you mean "Kennedy"@en , instead of "Kennedys"@en . There are 1051 results available? Update Query
show/hide Columns   Print                                         Search:
person                                     name
                     No data available in table
```

Figure 4.2: User interface showing a suggestion to modify the current query which returned to answers.

match the issued query with candidate patterns in the data set. Query suggestions are provided for all queries, and it is up to the user to accept these suggestions if the returned answers do not satisfy her information needs. The QSM poses several interesting research questions, such as which literals and predicates to replace in the query and how to find replacement terms. Also, what does it mean to relax the structure of a query and how to find the relaxed structure efficiently. The way the different requirements are addressed and challenges in Sapphire is described in the next three sections. The user interface is discussed in Section 4.3, then how initialization happens for a new endpoint is presented in Section 4.5, and the PUM is described in Section 4.6.

4.3 User Interface of Sapphire

Sapphire has a web-based user interface that was demonstrated in [59]. This interface is shown in Figure 4.2. The interface presents a text box for each part of a SPARQL query. While the user is typing query terms, the QCM provides suggestions to complete these terms as shown in Figure 4.3. After the user inputs a query, the query is validated and executed. Whenever a query is executed, the QSM tries to find alternatives to the query that was constructed by the user. Figure 4.2 shows an example of how the QSM suggests changes to the executed query. In this example, the user wants to find all people with the surname "Kennedys" (in plural form). However, no answers were found using this surname. The QSM suggests a modification that will result in finding 1,051 answers

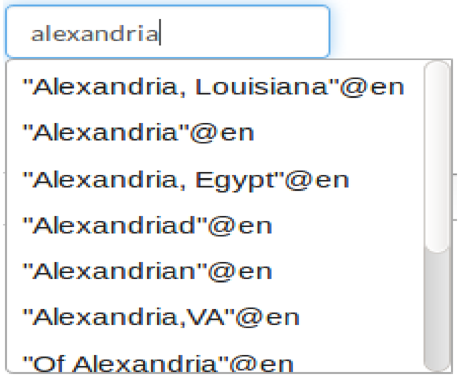
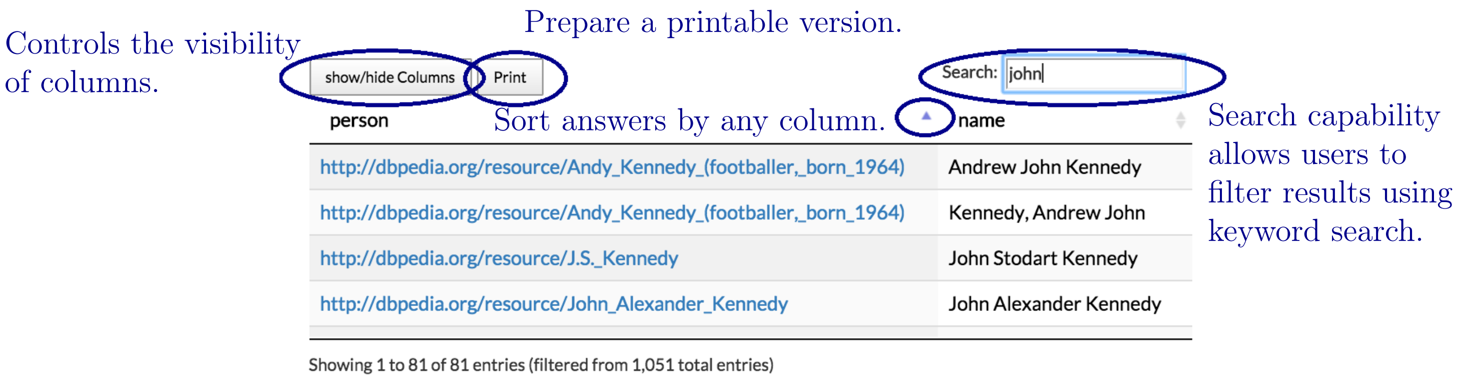

"Alexandria, Louisiana"@en
"Alexandria"@en
"Alexandria, Egypt"@en
"Alexandriad"@en
"Alexandrian"@en
"Alexandria,VA"@en
"Of Alexandria"@en

Figure 4.3: Auto-complete suggestions using the QCM.


Controls the visibility of columns.
Prepare a printable version.
show/hide Columns
Print
Search: john
Sort answers by any column.
Search capability allows users to filter results using keyword search.

person	name
http://dbpedia.org/resource/Andy_Kennedy_(footballer,_born_1964)	Andrew John Kennedy
http://dbpedia.org/resource/Andy_Kennedy_(footballer,_born_1964)	Kennedy, Andrew John
http://dbpedia.org/resource/J.S._Kennedy	John Stodart Kennedy
http://dbpedia.org/resource/John_Alexander_Kennedy	John Alexander Kennedy

Showing 1 to 81 of 81 entries (filtered from 1,051 total entries)

Figure 4.4: The answer table after applying the query suggestion in Figure 4.2. In this example, the 1,051 answers to the query are filtered via a keyword search on "john", and the filtered answers are ordered by the "person" column.

(at the time of writing), by changing “Kennedys” to “Kennedy”. If the user accepts this suggestion and updates the query, the answers to the new query (already executed) are displayed in the answer table (Figure 4.4). New suggestions are now displayed to the user in case these answers still do not satisfy her information needs. The query alternatives are shown to the user in the form of suggestions to change one term at a time. For example, one suggestion could be “In the triple `(subject, predicateX, object)`, did you mean `predicateY`, instead of `predicateX`? There are N answers available.”. This approach avoids showing the user a completely rewritten SPARQL query in one step, which would make the suggestions difficult to understand, especially for large and complex queries. The only exception is when the QSM suggests queries that are different in structure than the issued query. This specific type of query suggestion is discussed in detail in Section 4.6.2.

The suggested queries are executed in the background using the federated query processor and answers are prefetched so that when the user decides to choose one of the alternatives, the query is not re-executed, and the answers are displayed almost instantaneously. When the answers to a query are displayed to the user, she has the ability to manipulate them in the answer table, as shown in Figure 4.4. Supported operations include the following: the user can search the answer table using a keyword search box, order the answers by any column, show and hide columns, and drag and drop answers from the answer table to the query text boxes for additional queries.

4.4 SPARQL Features Not Supported by Sapphire

Sapphire supports commonly used SPARQL features. A list of the main SPARQL features not supported by Sapphire is as follows:

1. Group Graph Patterns: In this type of patterns, the query consists of multiple basic graph patterns. This also limits Sapphire in using other keywords that are based on multiple basic graph patterns in one query, such as OPTIONAL, UNION, EXISTS, NOT EXISTS, MINUS. Supporting this type of patterns requires primarily changes in the Sapphire user interface. The changes to the predictive user model are expected to be minimal.

2. Property Paths: A property path is a possible route through a graph between two graph nodes. A trivial case is a property path of length exactly 1, which is a triple pattern. The ends of the path may be RDF terms or variables. Variables cannot be used as part of the path itself, only the ends. The problem with property path

queries is that they assume that the user has prior knowledge of the vocabulary of the queried data sets, and enough knowledge of regular expressions. Adding support for this type of queries is part of the future work of Sapphire.

3. Binding: The BIND keyword allows a value to be assigned to a variable from a basic graph pattern or property path expression. Sapphire can support this by extending the user interface.

4. Subqueries: Subqueries are a way to embed SPARQL queries within other queries, for example, to limit the number of results from some sub-expression within the query. Supporting such queries requires extending the user interface and the query suggestion module of Sapphire to accommodate simultaneous multi-level suggestions (from innermost query to the outermost query). This is left as future work.

4.5 Initialization for a New Endpoint

This section describes the initialization step in which Sapphire retrieves data from a newly registered SPARQL endpoint representing a new RDF data set. Specifically, the section discusses: 1. Which data from the endpoint to cache? 2. How is this data retrieved? 3. How is it indexed for efficient access by the PUM?

4.5.1 Caching Data from a New Endpoint

The data cached by Sapphire from the endpoints plays a significant role in helping the user write a query that describes her information needs. The design of Sapphire assumes that it is simpler and more intuitive for users to express their information needs using keywords rather than using URIs. Therefore, the focus of the Sapphire PUM is on mapping keywords entered by the user in her query to RDF predicates (represented by URIs) and literals in the data set.

Thus, Sapphire needs to cache RDF predicates and literals from a data set so that these predicates and literals can subsequently be matched to keywords in the user query. Which predicates and literals to cache is a challenging question. The choice of data to cache cannot rely on statistical knowledge of the queried data sets or the query logs, since such knowledge is not available. Sapphire, therefore, relies on heuristics based on common characteristics of RDF data sets and SPARQL queries.

The first heuristic is based on the observation that the number of distinct predicates in a data set is typically much smaller than the number of distinct literals. For example, at the time of writing, DBpedia has approximately 3K distinct predicates compared to 70M distinct literals. Therefore, Sapphire caches all the predicates in a data set.

Given the typically large number of literals in a data set, Sapphire uses heuristics to limit the number of literals that it caches. First, Sapphire assumes that very long literals are not likely to be used in queries. Thus, Sapphire only caches literals below a certain length (in this chapter, 80 characters is used as the limit). Second, Sapphire assumes that the user is interested only in a certain language and allows the user to restrict the language of the cached literals (in this chapter, only English literals are cached).

Following the aforementioned heuristics reduces the number of cached literals. However, the number of literals that satisfy these heuristics will likely be too large to retrieve from the endpoint using a single SPARQL query. Such a query would be a long-running query, and most endpoints impose a timeout limit on queries to avoid overloading their computing resources. Thus, this query is decomposed into multiple queries that are each within the timeout limit. Furthermore, the entire initialization process is supposed to complete within a reasonable amount of time. The design point in Sapphire is for the initialization time to be on the order of hours, which is reasonable since the initialization process happens only once for each endpoint. The queries that Sapphire uses to retrieve literals from an endpoint for caching are discussed next.

Sapphire divides the data set based on the predicates and the class hierarchy defined by RDF schema (RDFS) [3]. RDFS defines classes that serve as data types for different entities, and organizes the classes into a hierarchy based on the `subClassOf` relation. For example, `MovieDirector` and `Politician` are two classes that are both subclasses of `Person`. Sapphire issues a SPARQL query to retrieve all classes and their subclasses from the endpoint. It also issues a query to retrieve all RDF predicates associated with literals, ordered by the numbers of literals associated with each predicate. These are short queries that are not expected to time out. Sapphire then iterates through the predicates associated with literals, from most frequent to least frequent. For each predicate, Sapphire navigates through the class hierarchy from root to leaves. At each class of the hierarchy, Sapphire creates a query to retrieve literals associated with the current predicate and current class, and that are below the threshold length and in the target language. To increase the likelihood that this query will succeed, it is decomposed into multiple queries using SPARQL pagination techniques (OFFSET and LIMIT). If this query succeeds, Sapphire moves to the next sibling in the class hierarchy. If this query times out, Sapphire navigates down to the next level of the class hierarchy, which contains smaller classes, and issues the query. This process continues until all the literals are retrieved. Sapphire allows the user to set a limit on the number of queries to issue to an endpoint and stops when this limit is reached. Since Sapphire orders predicates by frequency, it prioritizes caching the literals associated with frequent predicates.

For the uncommon case of data sets that do not use the class hierarchy of RDFS (about 75% of the data sets in the LOD cloud use RDFS[1]), Sapphire issues a query to retrieve the entity types that occur frequently in the data set. Sapphire then issues queries to retrieve the literals associated with each predicate and each of these entity types, iterating through the predicates and types from most frequent to least frequent. If there is a limit on the number of queries, Sapphire stops if this limit is reached. The complete list of queries that are sent to an endpoint during initialization can be found in Appendix C.

4.5.2 Indexing Cached Data

As discussed earlier, one of the key challenges facing Sapphire is providing suggestions to the user interactively. These suggestions come from the cached data, so this data must be indexed in a way that supports fast lookup.

The basic lookup operation for suggesting completions to the user is as follows: given a string t entered by the user, what strings in the data contain t? I observe that a *suffix tree* [139] is ideally suited for this type of lookup, so it is used as an index in Sapphire. The advantage of a suffix tree is that lookup operations depend only on the size of the lookup string t and the number of times z that this string occurs in the input, with a time complexity $O(|t| + z)$. Therefore, the suffix tree has diverse applications in many domains, e.g., bioinformatics [97]. The disadvantage of a suffix tree is that it can grow very large, sometimes over an order of magnitude larger than the size of the input [96].

Given the space consumption of suffix trees, only a subset of the cached data can be indexed in this tree. Since the number of RDF predicates is relatively small, all predicates are indexed. The more challenging question is which subset of the literals to index? To answer this question, the notion of *most significant* literals is introduced, and Sapphire indexes only these literals in the suffix tree. A literal is considered significant when the entity it is associated with occurs frequently in the data set. That is, there are many incoming edges in the RDF graph pointing to this entity, indicating the entity's importance.

Definition 1 *The significance score of a literal* l *is* $S(l) = |\{s|(s, p_1, o) \wedge (o, p_2, l)\}|$, *where* (s, p_i, o) *is an RDF triple.*

[1] http://stats.lod2.eu

For example, the literal "New York" is associated with the entity representing this city. Since this entity is pointed to by many other entities (i.e., occurs as an object in many RDF triples), the literal "New York" is significant. This definition of significance captures important classes in the RDF class hierarchy, and also captures important instances (people, locations, etc.). To identify the significant literals, Sapphire issues queries along the class hierarchy as it did for retrieving the literals (The queries used to retrieve most significant literals can be found in Appendix C).

The final issue related to initialization is how to lookup in cached literals not in the suffix tree. These are called the *residual literals* in this chapter. Lookup on the residual literals requires a sequential search, and this may be too slow for interactive response. To speed up this sequential search, Sapphire organizes the literals into *bins of residual literals*, or *residual bins* for short, where each bin has all the literals of a given length (i.e., $bin(literal) = |literal|$). As discussed in the next section, the PUM always searches for strings within a certain range of lengths, so its sequential search will be limited to a few bins. In addition, the search can be parallelized, with multiple threads simultaneously scanning the bins. The experiments in Section 4.7 show that this simple organization is effective at guaranteeing interactive performance.

To illustrate the cost of initialization, it is worth noting that initialization for DBpedia, one of the largest data sets in the LOD cloud, took 17 hours. In the process, Sapphire issued approximately 800 SPARQL queries to retrieve literals and 3000 to identify significant literals, in addition to the few queries that retrieve predicates and the class hierarchy. Approximately 200 queries timed out. The suffix tree for DBpedia contains 43K strings (3K predicates and 40K literals) and is 400MB in size. There are around 21M literals not in the suffix tree, divided among 80 bins. Section 4.7 shows that having even a small fraction of the literals in the suffix tree benefits performance.

4.6 Predictive User Model

The Predictive User Model (PUM) uses the data cached during initialization to help the user compose her SPARQL query. The user inputs a query to Sapphire by entering the triple patterns that describe the structure of the query. As the user types a subject, predicate, or object in a triple pattern, the PUM invokes the Query Completion Module (QCM) to provide suggestions for the user to complete the term being typed. When the user composes a full query and clicks "Run" in the Sapphire user interface, the PUM passes the query to the federated query processor for execution and also invokes the Query Suggestion Module (QSM) to suggest changes to the query. The QSM suggests changes to the query based on the structure of potential candidate answers in the data set, in order to bring the user closer to the query that finds the answer she is looking for. The user can choose one of the suggestions of the QSM and update the query, and possibly continue editing it. Editing the query would invoke the QCM again, and the process repeats as many times as needed by the user. The QCM is presented next, followed by the QSM.

4.6.1 Query Completion Module

The Sapphire user interface is organized so that the user inputs each subject, predicate, or object of a triple pattern in a separate text box. As the user types a string in one of these text boxes, the QCM is invoked every time the user types a character in order to provide auto-complete suggestions for the string being typed. An example is shown in Figure 4.3. The only exception is if the user enters a variable (i.e., a string starting with '?'), in which case Sapphire makes no suggestions.

Specifically, the problem solved by the QCM is as follows: Given the string t entered thus far by the user, find k strings in the data that contain t to suggest to the user. In this chapter, the value $k = 10$ is used. Figure 4.5 shows how the QCM finds the required k strings. The term t is looked up in both the suffix tree and the residual bins. Matches in the suffix tree are returned to the user as soon as they are found. If the search in the suffix tree returns fewer than k matches, the remaining matches come from the residual bins. Sapphire assumes that the auto-complete suggestions are most useful if they are not much longer than the current input string t. Therefore, the QCM only searches bins containing literals of length $|t|$ to $|t| + \gamma$, which reduces the cost of the sequential search. In this chapter, we use $\gamma = 10$. When the search in residual bins completes, the shortest result literals are returned as part of the k auto-complete suggestions.

To ensure interactive response time, the QCM's sequential search in the residual bins is parallelized, utilizing P parallel processes (threads). Typically, P would equal the number of available cores on the Sapphire server. Each process searches one or more bins, and the QCM assigns work to processes in a way that balances load, with each process scanning an equal number of literals. Algorithm 2 shows the details of task assignment.

4.6.2 Query Suggestion Module

The QSM suggests alternative queries that are *semantically close* to the query issued by the user. The suggestions of the QSM are particularly important if the query issued by the user returns no answers, but they can be useful even if the query returns answers. Defining

Algorithm 2: Assign Tasks to Processes

```
input  : Bins to Search $bins'$, Number of Processes $P$
output: Assigned Task for Each Process
1  Number of literals to search $n = \sum_{i=1}^{|bins'|} |bins'_i|$;
2  Process capacity $d = \frac{n}{P}$;
3  Process id $pid = 0$;
4  for $i = 1$ to $|bins'|$ do
5      Number of literals remaining in bin $i$ $j = |bins'_i|$;
6      while $j > 0$ do
7          if $j < Process_{pid}.d$ then
8              // Process $pid$ assigned all literals in bin
9              $Assign(Process_{pid}, [bins'_i[0], bins'_i[|bins'_i| - 1]])$;
10             $Process_{pid}.d = Process_{pid}.d - |bins'_i|$;
11             $j = 0$;
12         else
13             // Process $pid$ assigned remaining capacity
                 $Assign(Process_{pid}, [bins'_i[|bins'_i| - j], bins'_i[|bins'_i| - j + Process_{pid}.d]])$;
14             $j = j - Process_{pid}.d$;
15             $Process_{pid}.d = 0$;
16             $pid = pid + 1$;
17         end
18     end
19 end
```

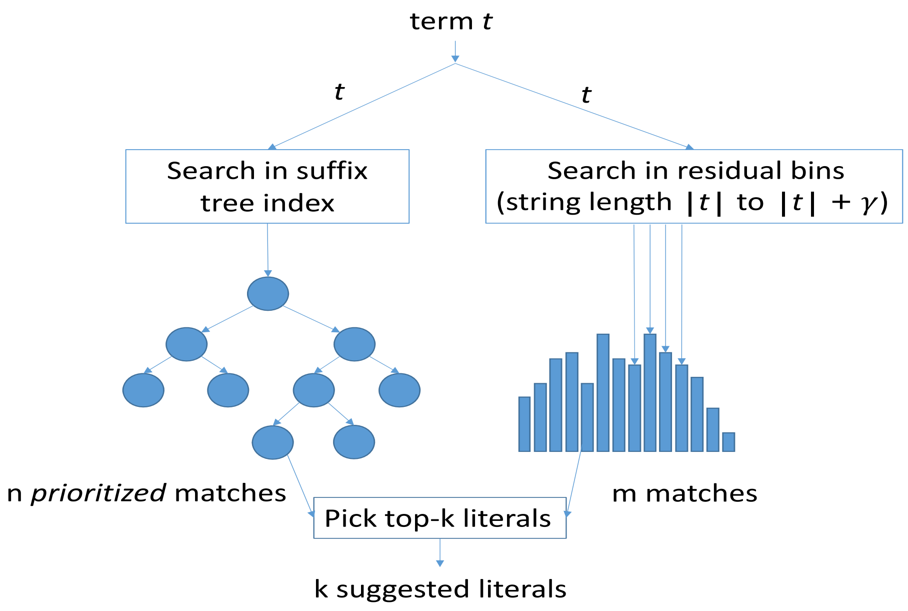

term t
t
t
Search in suffix tree index
Search in residual bins (string length $|t|$ to $|t| + \gamma$)
n *prioritized* matches
m matches
Pick top-k literals
k suggested literals

Figure 4.5: Completing a query term in the QCM.

semantic closeness is an interesting question. In Sapphire, the QSM suggests changes to the query in two directions: (1) suggesting alternatives to the terms (predicates and literals) used in the query, and (2) relaxing the structure of the query.

Alternative Query Terms

Algorithm 3 shows how the QSM finds alternatives for predicates and literals in the user query. The basic idea is to find predicates and literals in the data set that are similar to the ones in the query or to their synonyms. The synonyms provide knowledge about how terms are expressed in natural language. For example, "wife" or "husband" can be expressed as "spouse". The QSM examines the predicates and literals used in the triple patterns of the query one at a time. For each predicate p, the QSM first finds the synonyms of the predicate (line 4). The DBpedia Lemon Lexicon [37, 133] is used to provide these synonyms. The QSM then finds alternative predicates in the data set whose similarity score with the original predicate p or its synonyms exceeds a similarity threshold θ (in this chapter, the value $\theta = 0.7$ is used). In Sapphire, the Jaro-Winkler (JW) similarity [38] is used to calculate the similarity between strings. JW similarity is based on the minimum number of single-character transpositions required to change one string into the other, while giving a higher score to strings that match from the beginning. This similarity measure outperforms other similarity measures in this context. For each literal l, the QSM considers the bins containing literals of length in the range $[|l| - \alpha, |l| + \beta]$ (termed $bins'$ in Algorithm 3). A search operation over these bins is conducted, similar to the search over bins in the QCM. The difference is that the search to find alternative literals is based on the JW similarity. All literals that have a similarity score $\geq \theta$ are considered to be matches. This chapter uses $\alpha = 2$ and $\beta = 3$. The lists of alternative predicates and literals are sorted based on the JW similarity score. Similar to the QCM, the QSM can parallelize finding alternative terms among P processes.

A new SPARQL query is constructed for each of the alternative predicates and literals found by the QSM. Sapphire uses the federated query processor to execute the alternative queries and suggests the first queries that return answers.

Relaxing Query Structure

If the structure of the graph pattern specified by the user in the query is different from the structure of the queried data set, the user will not find the desired answer, even if the predicates and literals in the query match the desired answer in the data set. Therefore, the QSM suggests changes to relax the structure of the query (i.e., make it less constrained) based on the structure of the data set.

Figure 4.6 shows a motivating example. The query in this example is syntactically correct (top left box), and it aims to find books by "Jack Kerouac" that were published by "Viking Press". The figure shows part of the graph of the queried data set. The predicates and literals of the query can be found in the data set, and the matches are shown in the figure as dotted lines and rectangles. The figure also shows two answers that satisfy the query requirements, and the path that connects them in bold ("Door Wide Open" and "On the Road"). These answers will not be found by the query as posed by the user since the query structure does not match the structure of the data (the dotted matches are not connected). Relaxing the query structure can solve this problem by bringing the structure of the query closer to the structure of the data set.

In Sapphire, it is assumed that it is easier for the user to identify correct literals than to identify correct query structure. Thus, the goal of query relaxation can be defined as connecting literals in the query (or similar literals found by the JW similarity search) through valid paths in the graph of the data set. Ideally, the paths should be short and the algorithm should prefer paths that include the predicates entered by the user as part of the query. I observe that connecting the literals in the query can be formulated as a

Algorithm 3: Suggesting Alternative Query Terms

```
input  : Query $q$, Predicate Set $PR$, Literal Bins to Search $bins'$, Number of
         Processes $P$
output: Alternative Queries $Q'$
for each triple tr in q do
    for each non-variable element e in tr do
        if e is a predicate then
            Synonyms of term $S = Lemon.getLexica(e)$;
            for Each element s in S do
                Predicate alternatives $pa.add$(FindPredicateAlternatives($s$, $PR$, $P$));
            end
            for For each alternative a in pa do
                Construct a new query $q'$;
                Alternative queries for predicates $PQ.add(q')$;
            end
        else
            Literal alternatives $la(e)$ = FindLiteralAlternatives($e$, $bins'$, $P$);
            for For each alternative a in la do
                Construct a new query $q'$;
                Alternative queries for literals $LQ.add(q')$;
            end
        end
    end
end
SortBySimilarityScore($PQ$);
SortBySimilarityScore($LQ$);
$Q'.add$(TopQueriesWithAnswer($PQ, k$));
$Q'.add$(TopQueriesWithAnswer($LQ, k$));
return $Q'$;
```

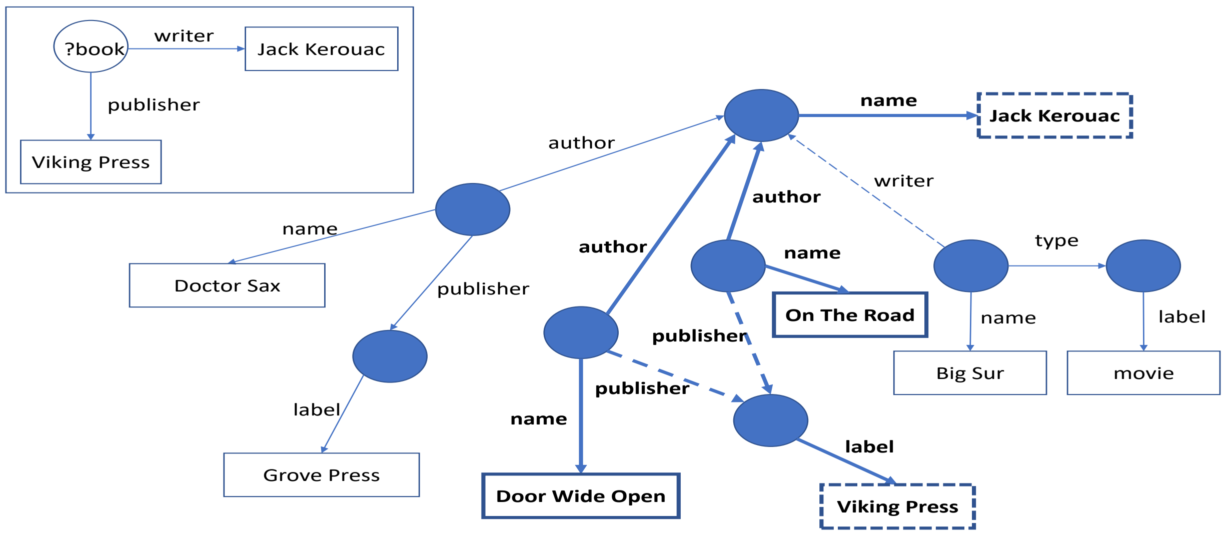


Figure 4.6: Example query and the subgraph from the data set that can be used to answer this query.

Steiner tree problem [84]. Favouring paths that include certain predicates can be achieved by modifying the weights on the edges of the graph.

The Steiner tree problem is defined as follows. In any undirected graph $G = (V, E)$, where V is the set of vertices and E is the set of edges, and each edge e_{ij} connecting vertices (i, j) has a weight w_{ij}, the Steiner tree problem is defined as finding a minimum weight tree that spans a subset of terminal vertices (literals in this case) $T \subset V$. If $T = V$, the problem is reduced to a minimum spanning tree problem. If $|T| = 2$, the problem is reduced to a shortest path problem. However, when $2 < |T| < |V|$, finding a minimum weight tree is NP hard.

A weight is associated with each edge in the graph representing the RDF data. These weights can be inferred by the algorithm and do not need to be materialized. For an edge representing a predicate that matches one of the predicates in the query, or one of the predicates identified by the QSM as an alternative for a predicate in the query, this weight is set to a value w_q. For any other edge, the weight is set to a value $w_{default} > w_q$. Since the Steiner tree algorithm aims to find the tree with the minimum overall weight, assigning weights in this manner favors matching the predicates in the query (or alternatives to these predicates) over simply finding a tree with a small number of edges.

Since finding the Steiner tree is an NP hard problem, an efficient approximate algorithm needs to be used. Moreover, traditional Steiner tree algorithms, whether exact or approximate, require fast access to any vertex or edge in the graph, whereas in the case of Sapphire the graph exists on remote endpoints and can be accessed only through SPARQL queries, which have a non-negligible cost. The algorithm must minimize the number of such queries. I describe next, (1) the literals to be connected via the Steiner tree algorithm, and (2) the algorithm used to connect these literals.

The QSM generates alternative query terms for the literals in the query as described earlier (line 13 in Algorithm 3). Each literal in the query and the alternative terms generated for it form a *group*, and the vertices representing these literals in the RDF graph are referred to as *seeds* for the QSM to explore the graph. For example, "Viking Press", "The Viking Press", and "The Viking" are all seeds in the same group. The goal of the QSM query relaxation algorithm is to create a Steiner tree that connects one literal from each group. It is not useful to connect multiple literals from the same group since these literals are alternatives to each other and not meant to be used together in the same query.

To connect the literals efficiently, the algorithm expands the graph starting from the seeds until the groups are all connected, and it attempts to minimize the number of vertices visited in this expansion. A known Steiner tree approximation algorithm [77] is used and adopted for this problem (Algorithm 4). The algorithm consists of the following two steps:

1. Connecting seeds: The goal of this step is to find a tree, not necessarily minimal, that connects all groups. Initially, each seed is a *candidate subgraph* of the RDF graph. Following the approximate Steiner tree algorithm in [77], the candidate subgraphs are expanded using the *bi-directional Dijkstra shortest path algorithm* [69]. In this algorithm, seeds from different groups take turns in expansion, in contrast to the regular Dijkstra shortest path algorithm in which chooses a single source seed from which to start the expansion. In practice, the bi-directional Dijkstra algorithm visits (expands) fewer vertices than the regular Dijkstra algorithm, which means fewer SPARQL queries. The expansion continues until paths are found that connect seeds from all groups.

In the expansion, each vertex v in a candidate subgraph is expanded into a subgraph $subG$ defined as follows: (1) $subG = \{(?s, ?p, ?o) | ?o = v\}$ if v is a literal, and (2) $subG = \{(?s, ?p, ?o) | ?s = v \vee ?o = v\}$ if v is a URI. That is, if the vertex is a literal (initially, all vertices are literals), the subgraph is expanded by finding all triples that have this literal as an object since literals can only be objects. Each of these triples introduces a new edge (the predicate) and vertex (the subject) to the candidate subgraph. If a vertex is a URI, the subgraph is expanded by finding all triples that have this vertex as a subject or an object. As in the case of literal vertices, each of these triples introduces a new vertex to the candidate subgraph (the subject of the triple if the expanded vertex is the object, and the object if the expanded vertex is the subject). The edge connecting the new vertex to

Algorithm 4: Relaxing Query Structure

```
   input  : Query $q$
   output: Matching Graphs $G_{suggested}$
1  Literals in query $L = q$.extractLiterals();
2  for Each literal $l$ in $L$ do
3      Seed group $seeds(l) = l \cup$ Top $k-1$ literals from $la(l)$;
4  end
5  Start with empty graph $g$;
6  while $g$ does not span terminals from all seed groups do
7      Scan vertices using Dijkstra's bi-directional shortest path algorithm;
8      Select a terminal $x$ not in $g$ that is closest to a vertex in $g$ (initially any literal
        from the query);
9      Add to $g$ the shortest path that connects $x$ with $g$;
10 end
11 // There can be several $g$ subgraphs spanning terminals if
12 // multiple paths with the same weight cost exist
13 for Each $g$ found while connecting seeds do
14     Construct subgraph $g'$ induced by $g$ in $G$;
15     Construct minimum spanning tree(s) of $g'$;
16     while There exist non-terminals of degree 1 from spanning tree(s) do
17         remove non-terminals of degree 1 from this spanning tree;
18     end
19     Add minimum spanning tree(s) to $G_{suggested}$;
20 end
21 Return $G_{suggested}$;
```

the expanded vertex is the predicate of the triple. These expansion steps are expressed as SPARQL queries executed on the endpoint of the data set.

The algorithm expands candidate subgraphs according to the bi-directional Dijkstra algorithm until it finds a shortest path that connects two seeds from different groups. Following the approximate Steiner tree algorithm [77], this path becomes the graph g that will be used to find the tree connecting all the groups. The expansion of other candidate subgraphs continues according to the bi-directional Dijkstra algorithm, and whenever the expansion of a candidate subgraph results in connecting to g a seed from a group that is not yet part of g, the path that connects this seed to g is added to g. The expansion stops when there is a set of connected seeds, one from each group. Recall that lower weights are assigned to the edges that match predicates in the query or similar predicates. This guides the bi-directional Dijkstra algorithm towards expanding paths that match query predicates first, and consequently reduces the number of SPARQL queries required to find a tree that matches the query predicates.

The expansion algorithm has budget for the number of queries that can be used. In order to remain within the budget, the expansion of sibling vertices that are chosen for expansion does not start if the number of siblings is larger than the remaining query budget. This restriction discourages the expansion of vertices with a high degree branching factor in the hope that this candidate subgraph's seed can be reached by another seed from a different group. Sapphire uses a budget of 100 SPARQL queries for graph expansion, which resulted in a good response time for query suggestion in the experiments. While expanding the candidate subgraphs, the results of the expansion are memoized so that if a vertex is encountered more than once during expansion, the results will be obtained from the memoized data structure without issuing a new SPARQL query.

Figure 4.7 shows how the vertices in the example are expanded starting from the seeds in the query. Common vertices between candidate subgraphs are lightly shaded. All the edges have a cost of $w_{default}$ except for "writer" and "publisher", which have a cost of w_q. Therefore, "writer" is chosen to be expanded in Step a.3. However, this vertex will not be further expanded because the expansion did not result in any common vertices with the subgraph of the other literal in the query. Therefore, it is not possible that further expansion will help finding a shorter path than the one already found.

2. Constructing the minimum tree: After the expansion step, a graph G consisting of the union of all expansions is constructed. Following the approximate Steiner tree algorithm [77], for each g found during expansion, a subgraph g' is constructed. This subgraph is the graph induced by g in G. That is, g' is a graph whose vertices are the same as g and whose edges are the edges in G such that both ends of the edge are vertices

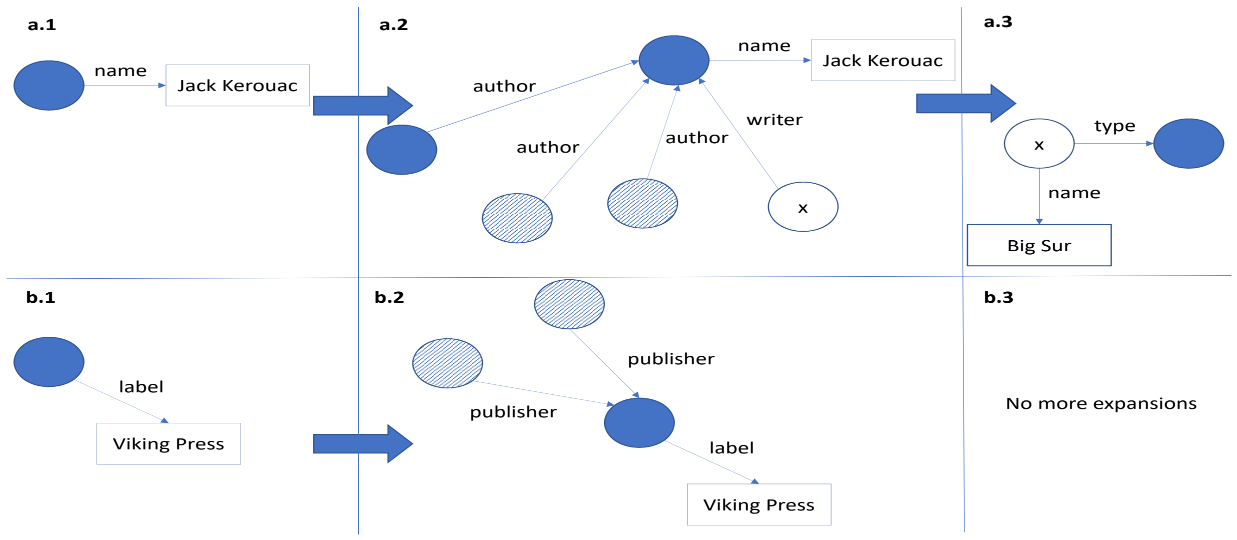


Figure 4.7: The expansion steps in the process of relaxing query structure.

in g. Next, a minimum spanning tree is constructed for subgraph g'. Multiple minimum spanning trees may exist and be generated in this step. Finally, all non-terminal vertices that have a degree of 1 are repeatedly deleted from the minimum spanning tree(s) since they cannot be part of the Steiner tree. There could be several Steiner trees with the same total edge weight. Each tree is an alternative query suggested to the user. The approximation ratio of this algorithm is known to be $2 - 2/s$ [77], where s is the number of seeds in the query.

Performance: Unlike the QCM, which should have sub-second latency to provide suggestions while the user types, the QSM can have a latency of a few seconds. That is, after the user submits a query, she will see alternative, complete, and syntactically correct suggested SPARQL queries after waiting a few seconds. In querying the LOD cloud using SPARQL, a query will likely have a small number of literals (in the user study, the maximum number of literals in a query was 3). The algorithm is fast enough for such problem sizes to have a QSM response time of less than 10 seconds on average.

4.7 Experimental Evaluation

Sapphire is evaluated along the following dimensions: 1. A user study in which participants answer questions using a natural language QA system and Sapphire (Section 4.7.1). 2. A quantitative comparison with recent natural language, approximate query, and query-by-example systems (Section 4.7.2). 3. Analyzing the response time of the QCM and QSM modules (Section 4.7.3).

Sapphire is implemented in Java. It runs as a web application over a web server. The user interacts with Sapphire through a web browser as described in [59]. A publicly available implementation of the suffix tree construction algorithm [128] is used, FedX [116] is used as the federated query processor, and the Lemon Lexicon for DBpedia[2] is used to find synonyms of literals and predicates. This lexicon can also be used for data sets other than DBpedia. DBpedia is the data set used in all the experiments in this section, and Sapphire interacts with it via its SPARQL endpoint[3]. DBpedia is a good evaluation data set because it is large and it is the central and most connected multi-domain data set in the LOD cloud[4]. The experiments are run on a machine with an 8-core Intel i7 CPU at 2.6 GHz and 8GB of memory. The memory usage of Sapphire to query DBpedia never exceeds 4GB.

4.7.1 User Study

User Study Setup

The most important question related to Sapphire is whether it actually helps users find answers in RDF data sets. To answer this question, I conducted a user study in which users are presented with a set of questions they need to answer using both Sapphire and QAKiS [30], a natural language question answering system that performs well compared to the other natural language systems (see Section 4.7.2).

The questions in the study are a subset of the query set from the Schema-agnostic Queries Semantic Web Challenge [4]. These queries are questions over DBpedia derived from the Question Answering over Linked Data (QALD) competition[5]. 35 questions were chosen and the four authors of [59] independently labeled each question as easy, medium, or difficult. Out of the 35 questions, the authors of [59] all agreed on the difficulty level of 27 questions. These questions are the ones used in the user study, and are presented in Appendix D.

[2]http://github.com/ag-sc/lemon.dbpedia
[3]http://dbpedia.org/sparql
[4]http://lod-cloud.net
[5]http://qald.sebastianwalter.org

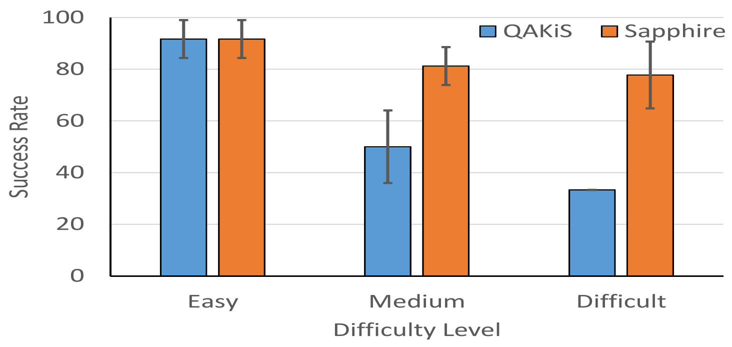


Figure 4.8: Success rate of answering questions.

The participants in the user study were 16 users who have a computer science background but are not familiar with RDF or SPARQL. A standard tutorial was given to the participants by two of the authors of [59]. The participants were first introduced to the basic concept of describing their information needs in triples. Then they were introduced to Sapphire's user interface and shown the fields they will be using to type in the query terms (refer to Figure 4.2). A sample query is issued by the tutorial instructor using both QAKiS and Sapphire to show the participants how both systems are used. This tutorial was approximately 15 minutes long. Each participant was then given 10 questions (4 easy, 3 medium, and 3 difficult). The questions were randomly assigned to participants per category. The participants were asked to find answers to all the questions using both Sapphire and QAKiS. Since the two systems are fundamentally different in the way they are used, using one system to find an answer should have minimal effect on how the other system is used. However, the system the user used first for every question is alternated. For example, if the participant answers one question using Sapphire first then QAKiS, the next question is answered using QAKiS first then Sapphire. One question from the easy category was used in a tutorial prior to the study to demonstrate the two systems to the users (the same question for all participants). During the study, the first question a participant tried (from the easy category) was used as a warm-up question to familiarize the user with the two systems. The data collected for this first question is dropped from the results. Screen recording was used to capture the sessions of all participants.

Quantitative Results

I first investigate whether Sapphire helped users find answers to their assigned questions, and how it compares to QAKiS. A total of 48 questions in each category were given to

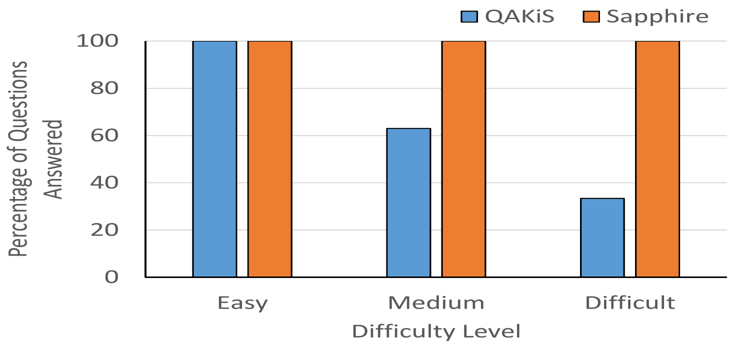


Figure 4.9: Percentage of questions answered by at least one participant.

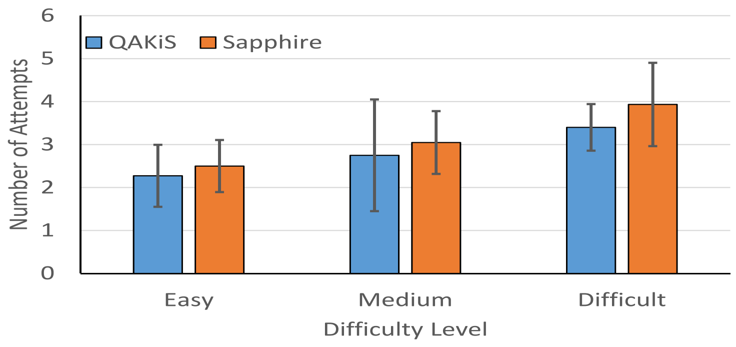


Figure 4.10: Average number of attempts before finding an answer.

the participants in this study (16 participants × 3 questions per category, excluding the first warm-up question). First, the success rate in answering these questions is evaluated. That is, of the questions given to a user, what fraction was answered correctly? Figure 4.8 shows the success rate of finding answers for the 48 questions in each category using Sapphire and QAKiS. The bars in the figure show the average success rate, averaged over the 16 participants. The 95% confidence interval is reported to demonstrate that the findings are consistent among all participants. In this experiment and all subsequent experiments, whenever a noticeable difference between Sapphire and QAKiS was observed, the p-value was calculated and in all cases it was found to be less than the significance level (0.05), which indicates that these differences are statistically significant. The figure shows that Sapphire is superior to QAKiS in the medium and difficult categories, while both systems perform the same for the easy category. Participants found answers for over 80% of the medium difficulty questions using Sapphire, compared to around 50% using QAKiS. The gap widens for the difficult category, where participants answered almost 80% of the

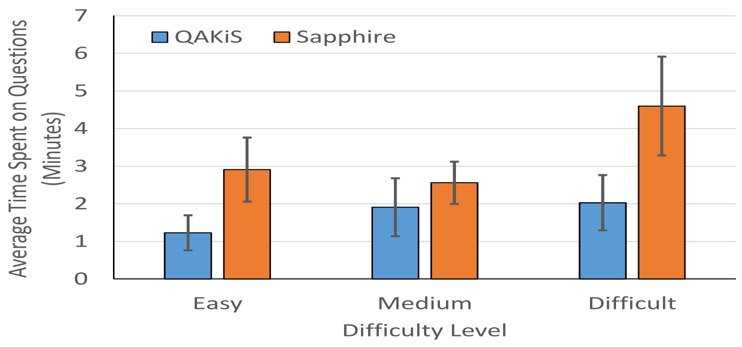


Figure 4.11: Average time spent on answered queries.

questions using Sapphire and only 33% using QAKiS.

The success rate does not tell the full story since some questions are easier than others and some users are better at answering questions than others, regardless of the difficulty category or the system used. Another way to compare the two systems is to see, for every question, whether that question was answered by any participant. Figure 4.9 shows the percentage of questions answered by at least one participant using both systems. The figure shows that every question was answered by at least one participant using Sapphire, while QAKiS could find answers for only 63% of the questions in the medium category and 33% in the difficult category.

Figure 4.10 shows the average number of attempts the participants went through before finding an answer in each category. An attempt is counted when a participant clicks "Run" to issue a query. Sapphire requires slightly more attempts than QAKiS, but the numbers are comparable and not statistically significant (p-value > 0.05). This demonstrates that Sapphire is not overly difficult to use despite the need to describe a query in a structured format. Note that attempts are counted only for the questions that were answered correctly. Participants gave up on finding an answer for a question after 3 to 4 attempts when using QAKiS, and after 3 to 5 attempts when using Sapphire.

Sapphire does require more time to use than QAKiS, as demonstrated in Figure 4.11, which shows the time the participants spent on questions from each category. The figure only shows the time spent on questions that were answered successfully. The figure shows that participants spent more time using Sapphire than QAKiS for all difficulty categories. This is expected due to the fundamentally different approach of describing a question in Sapphire. A participant spends more time to describe the question as a set of triple patterns, and to examine Sapphire's suggestions and choose from them. This additional effort is justified by Sapphire's ability to find answers to more questions.

In summary, the user study shows that Sapphire is more effective than QAKiS at answering medium and difficult questions. The cost of this effectiveness is more time spent in answering the questions.

Qualitative Results

After each session, the participants are surveyed about their experience using Sapphire and how it compares to QAKiS. The comments received are consistent across participants: At first, they find it difficult to express the question using triple patterns (due to the lack of experience in RDF) but are still able to answer the questions. However, when they get used to this style of querying, Sapphire becomes much easier to use. They also agree that Sapphire is much more helpful than QAKiS in answering more difficult questions.

Another observation from viewing the recorded sessions is that different participants answering the same question sometimes take different approaches and use different terms, but end up with the same SPARQL query. In other cases, different participants end up with different queries to find the same answer. For example, some participants rank results by a condition and select the correct answers while others include the condition in the triple patterns of the query. This demonstrates the flexibility and effectiveness of Sapphire.

The logs of the user study indicate that participants used the suggestions of the QSM in over 90% of the questions. Users utilized alternative predicates in 28% of the questions, alternative literals in 17% of the questions, and relaxed query structure in 67% of the questions. This demonstrates the crucial role the QSM plays in guiding the user towards correctly describing her questions.

For another qualitative perspective on Sapphire, two SPARQL experts were recruited, one with no experience in querying DBpedia and the other with three years of experience. The two participants were asked to write SPARQL queries to find answers to the 48 questions used in the user study, with and without Sapphire. Without Sapphire, i.e., interacting directly with the SPARQL endpoint of DBpedia, the first participant was unable to answer any of the questions because he did not know how the DBpedia URIs are represented and what kind of vocabulary is used in it. When using Sapphire, he was able to find answers to most questions. The participant with three years experience in DBpedia answered most questions. Sapphire did help him answer the questions he failed to find answers for when using DBpedia's SPARQL endpoint. Both experts agreed on Sapphire's value in helping users to write SPARQL queries against data sources they are less familiar with and expressed interest in using Sapphire for their future projects.

	$\#pro$	%	$\#cor$	$\#par$	R	R^*	P	P^*	F_1	$F_1{}^*$
Xser [141]	42	84%	26	7	0.52	0.66	0.62	0.79	0.57	0.72
APEQ [132]	26	52%	8	5	0.16	0.26	0.31	0.50	0.21	0.34
QAnswer [111]	37	74%	9	4	0.18	0.26	0.24	0.35	0.21	0.30
SemGraphQA [17]	31	62%	7	3	0.14	0.20	0.23	0.32	0.17	0.25
YodaQA [132]	33	40%	8	2	0.16	0.20	0.24	0.30	0.19	0.24
QAKiS [30]	40	80%	14	9	0.28	0.46	0.35	0.58	0.31	0.51
KBQA [41]	8	16%	8	0	0.16	0.16	**1.0**	**1.0**	0.28	0.28
S^4 [150]	26	52%	16	5	0.32	0.42	0.62	0.81	0.42	0.55
SPARQLByE [47]	7	14%	4	0	0.08	0.08	0.57	0.57	0.14	0.14
Sapphire	**43**	**86%**	43	0	**0.86**	**0.86**	**1.0**	**1.0**	**0.92**	**0.92**

Table 4.1: Comparing systems using questions from QALD-5.

4.7.2 Comparison to Other Systems

In this section, Sapphire is compared to other state of the art systems for querying RDF data. Sapphire is compared to the systems participating in the QALD-5 competition [132]. Sapphire is also compared using the same questions to (a) QAKiS, which is used in the user study, (b) the more recent natural language QA system KBQA [41], (c) the recent approximate query matching system S^4 [150], and (d) the recent query-by-example system SPARQLByE [47]. The performance numbers of the systems that participated in QALD-5 are from [132] and the numbers for KBQA are from [41]. The QAKiS and SPARQLByE systems are publicly available, so I ran them and obtained their performance numbers for this experiment. I implemented S^4 to obtain its performance numbers for the experiment.

QAKiS is a natural language QA system, and in this experiment, up to 3 attempts are allowed for each question. In these attempts, I do not change the query terms using my knowledge of the vocabulary. For example, the question “What is the revenue of IBM?” can be paraphrased in a different attempt as “IBM’s revenue”, but would not be changed to “IBM’s income”.

S^4 constructs a summary graph of the data in on offline step, and accepts SPARQL queries that it rewrites to match the structure of the data based on the summary graph. S^4 expects the predicates and literals to be correct, so I use Sapphire to find predicates and literals that exist in DBpedia when constructing the SPARQL query for S^4. I compose the SPARQL query for S^4 based on the question in QALD-5, restricting myself to the terms in the question. S^4 rewrites the query and I execute the rewritten query using FedX.

SPARQLByE requires the user to provide example answers. The system attempts to learn the commonalities between these answers and capture them in a SPARQL query. The answers of this SPARQL query are presented to the user as additional candidate answers, and the user can mark them as correct or incorrect. SPARQLByE requires at least two sample answers so it is used in this experiment for questions that have three answers or more in their gold standard result. Two answers from the gold standard result are used as inputs to SPARQLByE, and I provide feedback to the system until it finds the correct query or cannot learn any more (i.e., cannot modify the query).

When using Sapphire, I only used terms from the question to enter the query, as I did with other systems. I then use Sapphire's suggestions to complete and modify the query until an answer is found. I do not use my knowledge of the vocabulary to change the terms or query structure.

The systems are evaluated using the following performance measures [41, 132]: 1. The number of questions that are processed and for which answers are found ($\#pro$). 2. The number of questions whose answers are correct ($\#cor$, referred to in [41, 132] as $\#ri$) 3. The number of questions whose answers are partially correct ($\#par$). In addition, the following recall and precision measures are computed, where $\#total$ is the total number of questions in the question set: *Recall* defined as $R = \frac{\#cor}{\#total}$, *partial recall* defined as $R^* = \frac{\#cor+\#par}{\#total}$, *precision* defined as $P = \frac{\#cor}{\#pro}$, *partial precision* defined as $P^* = \frac{\#cor+\#par}{\#pro}$, F_1 defined as $2.\frac{P.R}{P+R}$, and $F_1{}^*$ defined as $2.\frac{P^*.R^*}{P^*+R^*}$. The value of $\#total$ is 45 in this experiment, since this is the number of questions in QALD-5.

Table 4.1 shows the performance of the different systems. The table shows that Sapphire outperforms all other systems on all measures. Natural language QA systems suffer from low precision due to the challenge of inferring the structure and terms of a SPARQL query from the natural language formulation of the question. This challenge is not faced by Sapphire, which helps the user to directly construct SPARQL queries. Therefore, Sapphire has a precision of 1.0 for the questions it is able to answer. Among the natural language systems, KBQA has precision of 1.0 like Sapphire, but it has much lower recall. This is because KBQA focuses only on factoid questions. If only the factoid questions are considered, KBQA achieves a recall of 0.67, still lower than Sapphire. S^4, while lower in performance than Sapphire, performs better than other systems. SPARQLByE has much lower recall than other systems because it cannot answer most of the questions.

The table justifies the choice of QAKiS as a representative QA system in the user study. Other than Xser and S^4, QAKiS is the best performing system after Sapphire in terms of recall and F-measure. Xser is not publicly available. S^4 requires exact knowledge of the literals and URIs in the queried data set, which is deemed too difficult for a user study.

4.7.3 Sapphire Response Time

The performance of the QCM is studied first. It is important for the QCM to provide auto-complete suggestions with very low response time in order to guarantee an interactive experience for users. Two components contribute to the response time of the QCM: the lookup in the suffix tree, and the sequential search in the bins of literals (the residual bins). The total response time of these two components is, on average, 0.16 seconds when including 40K significant literals in the suffix tree and using 8 cores for the sequential search in the residual bins. This response time is low enough to provide a good interactive experience.

The two components of this response time are studied in more detail. A lookup operation in the suffix tree takes approximately 0.25 milliseconds, regardless of the number of literals that are indexed. This response time is certainly low enough for an interactive user experience. Recall that matches in the suffix tree are returned immediately to the user before the search in the bins of literals begins. Thus, having a hit (match) in the suffix tree greatly enhances the interactive experience, since the user sees auto-complete suggestions very quickly. Even if these suggestions are not chosen by the user, they still give an impression of a responsive system. Therefore, a higher hit ratio in the suffix tree is better for interactive response of the QCM. The hit ratio (fraction of query terms for which a match is found in the suffix tree) depends on the number of literals included in the suffix tree. The experiments show that even with only 40K literals in the suffix tree, a hit ratio of 50% is achieved.

The second component of the QCM response time is the sequential search in the literal bins. Recall that the bins to be searched are filtered based on the length of the term entered by the user. It was found that, on average, this filtering eliminates 46% of the literals to be searched. The search in the residual bins takes 0.6 seconds when using 1 core, and 0.16 seconds when using 8 cores. The takeaway of this experiment is that the QCM can provide interactive response time by utilizing more cores.

The discussion now turns to the performance of the QSM. The QSM spends around 10 seconds on average before returning suggestions to the user. This is acceptable since the QSM does not interact with the user while she is typing. Instead, the user waits for suggestions from the QSM, and a 10-second wait is reasonable.

4.8 Conclusion and Future Work

This chapter introduced Sapphire, a tool that helps users construct SPARQL queries that find the answers they need in RDF data sets. Sapphire caches data from the data sets to be queried and uses this cached data to suggest completions for SPARQL queries as the user is entering them, and modifications to these queries after they are executed. Sapphire was found to be effective at helping users with no prior knowledge of the queried data sets to answer complex questions that other systems fail to answer. As such, Sapphire is a valuable tool for querying the LOD cloud.

As future work, it is possible to extend Sapphire to cover more SPARQL keywords (address the limitations presented in Section 4.4). It is also possible to explore new UI models that make Sapphire more interactive and usable. This includes summarizing the queried schemas and showing suggestions to the user prior to typing the queries, and continuous user-guided filtering of the query answers of the query composed thus far.

In the next chapter, the focus is shifted from RDF data to relational data. The focus is still on involving users in data integration, and specifically on improving the quality of automatically generated mediated schemas and mappings based on user feedback over query answers.

Chapter 5

UFeed: Refining Relational Web Data Integration Based on User Feedback

The previous two chapters focused on RDF data. This chapter shifts the focus to integrating relational (i.e., table structured) data on the web. Recall from Chapter 1 that there is a large number of relational data sources on the web, such as web forms, web tables, and online spreadsheets. Users can benefit from the ability to simultaneously query several of these data sources, so it is useful to have a unified interface to these data sources in the form of a mediated schema that represents the attributes in the data sources, along with semantic mappings between the schemas of the data sources and the mediated schema.

Given the scale and semantic heterogeneity of web data, a pay-as-you-go approach to data integration is suitable. Pay-as-you-go data integration involves two phases: setup and refinement. In the setup phase, the system creates: (1) a mediated schema or possibly several schemas, and (2) mappings between the schemas of the data sources and the mediated schema(s). Since setup is done fully automatically in a best-effort fashion, the mediated schema and mapping need to be refined as they are used. Refining the mediated schema and mapping is the focus of this chapter.

This chapter presents UFeed [58], a pay-as-you-go data integration system that refines the mediated schema and mappings based solely on feedback that a user provides on the answers to her queries that use the mediated schema and mappings. This is in contrast to most prior work, in which refinement is decoupled from the querying process. Instead of fixing mistakes in the schema and mappings by inferring the correct action from feedback on query answers, most prior work presents the user with these mistakes and asks her to fix them, or requires the user to provide further input in addition to her feedback on query answers. One important feature of UFeed is that it is, to the best of my knowledge, the first system that fixes both the mediated schema and the mappings based on user feedback over query answers. UFeed accepts as input a mediated schema and mappings between each source schema and this mediated schema. UFeed does not make any assumptions about the techniques used to create the initial mediated schema and mappings in the setup phase.

In enabling the user to issue queries against the mediated schema, UFeed follows a philosophy that it is better to isolate the user from the details of the mediated schema and mappings, and not to assume that the user has background knowledge about all available data sources. Therefore, UFeed does not require the user to issue her queries on the mediated schema. Instead, UFeed allows the user to issue her query on an individual data source of her choice whose schema she is familiar with, similar to prior work [42]. UFeed maps the query on this data source to a query on the mediated schema and on other data sources. When query answers are shown to the user, she can mark any answer tuple as "correct" or "incorrect". The query itself and this feedback trigger refinement operations in UFeed.

The main contribution of UFeed is to propose a set of well-defined refinement operations that are triggered by the user's interactions with the system. These refinement operations modify the mediated schema and mappings with the goal of improving the quality of query answers. Modifying the mediated schema and mappings presents several challenges: Which source attributes should be part of the mediated schema? What causes an answer to be incorrect (error in the mediated schema and mappings or error in the data)? What if a feedback instance provided by the user is incorrect? UFeed addresses these challenges in its definition of the refinement operations and how they are triggered. Next, an example of the problems that UFeed must tackle in its refinement is presented. This is used as a running example throughout the chapter.

Example 1 *Consider the following source schemas:*
S_1*(county, state, country, region)*
S_2*(country name, gdp per capita, region)*
S_3*(country, region)*
S_4*(name, gdp per capita ppp, region)*

Figure 5.1 shows the gold standard G for integrating the four data sources (i.e., the correct mediated schema, which I created manually), and a mediated schema M that can be the output of some automatic data integration approach. The mediated schema M differs from the gold standard in that it includes the attribute "county" in the mediated attribute as

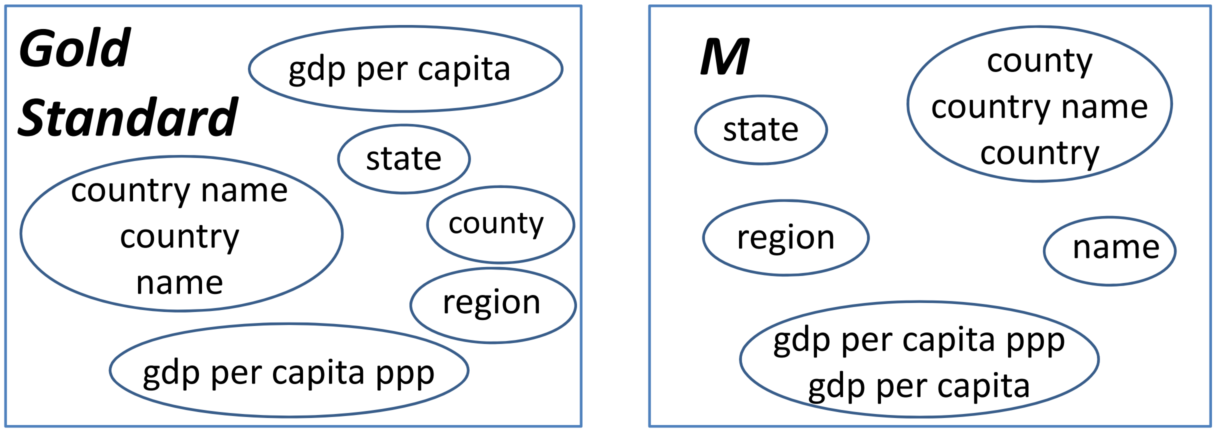


Figure 5.1: The gold standard G for integrating schemas S_1, S_2, S_3, and S_4 in Example 1, and a possible mediated schema M.

{country name, country}, which is supposed to represent the concept "name of a country". The attribute "name" also represents the same concept but it is not part of the same mediated attribute. The mediated schema also combines "gdp per capita" and "gdp per capita ppp". The first represents nominal GDP while the second represents GDP at purchase power parity. These two concepts are related, but distinct, so they should not be part of the same mediated attribute. Figure 5.2 shows the mapping of each source schema to both the mediated schema and the gold standard. When a query is issued against the mediated schema M, the returned answers can have correct answers, missing answers, and/or incorrect answers. For example, when selecting states in a region, all answers are expected to be correct and complete since there are no mistakes in the "state" or "region" attributes in M. However, errors can occur when selecting countries in a region. For example, if a query is issued where the mediated attributes {name} and {region} are chosen, this query will return only a subset of the correct answers because no answers will be returned using the attribute "country" from S_1 and S_3, or the attribute "country name" from S_2. If the query uses the mediated attributes {county, country name, country} and {region}, it will return some correct answers based on the source attributes "country" and "country name", but it will also return incorrect answers based on the source attribute "county". Moreover, the answers from the data source S_4 with the attribute "name" will be missing.

The goal of UFeed is to address problems such as the ones presented in the previous example. UFeed collects user feedback on the answers to queries and refines the mediated schema and mappings until they are correct.

Mapping	Attribute Mappings

Map_1	county → {county, country name, country}
	state → {state}
	region → {region}

Map_1^G	county → {county}
	state → {state}
	country → {country, country name, name}
	region → {region}

Map_2	country name → {county, country name, country} gdp per capita → {gdp per capita ppp, gdp per capita} region → {region}

Map_2^G	country name → {country, country name, name}
	gdp per capita → {gdp per capita}
	region → {region}

Map_3	country → {county, country name, country} region → {region}
Map_3^G	country → {country, country name, name} region → {region}

Map_4	name → {name}
	gdp per capita ppp → {gdp per capita ppp, gdp per capita}
	region → {region}

Map_4^G	name → {country, country name, name} gdp per capita ppp → {gdp per capita ppp} region → {region}

Figure 5.2: The mappings to the mediated schema M and to the gold standard G.

The contributions of this chapter are as follows:

- Refining both the mediated schema and the mappings based solely on feedback given by non-expert users on the answers to their queries.
- Defining a set of complete operations that refine automatically generated mediated schemas and mappings, and describing how these operations are triggered based on feedback on query answers.
- Proving that the UFeed operations can transform any automatically-generated mediated schema and mappings to a gold standard that is the output of manual integration of data sources.
- Evaluating UFeed on real web data sources and showing that it improves the quality of query answers.

5.1 Related Work

5.1.1 Schema Matching and Mapping

Data integration aims at automatically creating a unified mediated schema for a set of data sources and generating mappings between these data sources and the mediated schema. There has been a lot of work on mediated schema creation where the focus was on theoretical aspects of merging schemas [99, 107]. Schema matching and mapping have been extensively studied in the literature [19, 23, 110], and the state of the art is that many matching decisions can be made automatically. Proposed approaches to the problem of schema matching and mapping can be roughly categorized into four categories: 1. *Schema-based* approaches perform matchings using the meta-data associated with the data sources (e.g., Clio [106]). 2. *Instance-based* approaches determine the similarity between schema elements based on the similarity between their instances [83, 92]. 3. *Hybrid* approaches use a combination of the two aforementioned approaches such as LSD [50], Cupid [93], and COMA [49]. 4. *Usage-based* approaches exploit usage information, such as query logs [62] and search click logs [102]. Whenever ambiguity arises in schema matching, involvement of an experienced user (e.g., a data architect) is required. To account for the uncertainty faced by data integration systems due to this ambiguity, probabilistic models of data integration have emerged [42, 43, 52, 53, 94]. The contributions of UFeed start after the mediated schema and mappings are created, since UFeed accepts as input a mediated schema and mappings between each source schema and this mediated schema. UFeed does not make any assumptions about the techniques used to create the initial mediated schema and mappings, and can work with any technique for schema matching and mapping. In particular, it can work with deterministic or probabilistic mediated schemas.

5.1.2 Incorporating Users and User Feedback in Data Integration Systems

Involving users in various tasks related to data integration has been studied in the literature. This spans different data integration problems like entity matching and schema matching and mapping.

Entity Matching

In the Crowd ER system [140], users are asked to confirm or reject entity matches. However, due to the large number of questions that can be asked, the system is more concerned with sorting the questions to choose the questions that need to be answered first (questions that are more challenging to computers). The Corleone [68] system outsources the entire entity matching workflow to the crowd, including blocking and matching.

Schema Matching and Mapping

In [8], user feedback is used in an iterative exploratory process to guide the system towards the best data sources for the user, and the best mediated schema for these sources. In [35], a debugger for understanding and exploring schema mappings is introduced. This debugger computes, and displays the relationships between source and target schemas. Muse [11] refines partially correct mappings generated by an automatic matcher, and asks the user to debug them by examining user-proposed examples.

Some work has been done on verifying automatic decisions during the process of schema matching and mapping. In [81], a large set of candidate matches can be generated using schema matching techniques. These matches need to be confirmed by the user. The candidate matches are sorted based on their importance (i.e., if they are involved in more queries or associated with more data). A user is asked to confirm a match with a "yes" or "no" question. A similar approach has also been proposed in [98, 140, 149]. In [29], the system provides functionality for checking a set of mappings to choose the ones that represent better transformations from a source schema to a target schema. The step of verifying schema mappings is done as part of setting up the data integration system. This results in a significant up-front cost [66].

The work described in the previous paragraph requires the user to provide feedback on the schema. Another research direction focuses on different variations of the idea of utilizing feedback that the user provides on the answers to her queries. In [144], user feedback is obtained over the answers to a keyword search-based data integration query. The feedback is represented as a constraint over the ordering of the returned answers, or as identification of good or bad answers. The feedback is then utilized to improve the ordering of answers to future queries. The system in [33] relies on writing manual rules to perform information extraction or information integration operations. The output view of these operations is then subject to feedback from the user in the form of inserting, editing, or deleting data. This feedback is then reflected on the original data sources and propagated to other views. In the Q system [124, 125, 144], keywords are used to match terms in tuples within the data tables. Foreign keys within the database are used to discover "join paths" through tables, and query results consist of different ways of combining the matched tuples. The queries are ranked according to a cost model and feedback over answers is used to adjust the weights on the edges of the graph to rerank the queries. Other approaches [12, 108] require the user to specify samples of data mapped from a source schema to a target schema in order to generate the mappings between these schemas.

The work closest to UFeed is [18], where the focus is on refining alternative mappings in a pay-as-you-go fashion. The mediated schema is assumed to be correct, and the user issues a query along with constraints over the required precision and recall to limit the number of mappings used to answer the query. The user can give feedback over the returned answers so that the mappings can be annotated with an estimate of their precision and recall. These estimates are used in future queries to refine and select the mappings that would return the level of precision and recall desired by the user. UFeed refines not only the mappings, but also the mediated schema, without overburdening the user with specifying constraints on the quality of query answers. As a comparison to [18], the experimental evaluation in this chapter shows that refining the mappings alone is not sufficient to find high-quality answers to the user's queries.

In contrast to these approaches, UFeed refines the mediated schema and mappings incrementally, relying solely on user queries and feedback on query answers. UFeed infers the parts of the mediated schema and mappings that need to be modified by relying on user queries and feedback on query answers, while shielding the user from the details of the matching and mapping process. UFeed is also agnostic to the degree of confidence the data integration system has about its matches because user feedback is used to directly change the mediated schema and mappings, regardless of whether the data integration system is certain or uncertain about them.

5.2 Preliminaries on Relational Data Integration

This section defines the concepts used by UFeed and provides some required preliminary information on relational data integration.

5.2.1 Schema Matching and Mapping

Source Schema: UFeed focuses on the integration of web tables, which usually have simple schemas that do not adhere to explicit data types or integrity constraints. Thus, a source schema consists of a set of attribute names.

Definition 2 *A source schema S that has n source attributes is defined by: $S = \{a_1, \ldots, a_n\}$.*

For q source schemas, the set of all source attributes in these schemas is $\mathcal{A} = attr(S_1) \cup \cdots \cup attr(S_q)$.

Mediated Attribute: A mediated attribute mA is a grouping of source attributes from different source schemas. Source attributes in a mediated attribute should represent the same real-world concept.

Definition 3 *A mediated attribute is defined by: $mA = \{S_i.a_x, \ldots, S_j.a_y | \forall i, j, i \neq j\}$.*

Mediated Schema: In this chapter, one mediated schema is generated for a number of data sources belonging to the same domain. This approach is known as *holistic schema matching* [121], in contrast to approaches that perform pairwise matching between a pair of schemas. Holistic schema matching is the appropriate approach for web data integration because a large number of data sources needs to be covered by one mediated schema.

Definition 4 *A mediated schema $\mathcal{M}$ is defined by: $\mathcal{M} = \{mA_1, \ldots, mA_m\}$, where m is the number of mediated attributes in the mediated schema.*

Mapping: The mapping from any data source to the mediated schema is represented by a set of correspondences, each between a source attribute and a mediated attribute.

Definition 5 *A mapping $\mathcal{M}ap_i$ between source schema S_i and the mediated schema $\mathcal{M}$ is defined by: $\mathcal{M}ap_i = \{a_j \rightarrow mA_k | j \in [1, |\mathcal{S}_i|], k \in [1, |\mathcal{M}|]\}$.*

The process of generating a mediated schema holistically and generating mappings between each source schema and the mediated schema is referred to in this chapter as *holistic data integration.*

5.2.2 Probabilistic Mediated Schemas and Mappings

UFeed can work with mediated schemas and mappings generated through holistic or probabilistic data integration. The probabilistic model of data integration [42, 52] reflects the uncertainty faced by automatic approaches when integrating heterogeneous data sources. A probabilistic mediated schema can possibly consist of several mediated schemas, each of which is associated with a probability that reflects how likely the mediated schema represents the domain of the data sources.

Definition 6 *A probabilistic mediated schema $\mathcal{PM}$ with p mediated schemas is defined by: $\mathcal{PM} = \{(\mathcal{M}_1, Pr(\mathcal{M}_1)), \ldots, (\mathcal{M}_p, Pr(\mathcal{M}_p)))\}$, where $Pr(\mathcal{M}_i)$ is the probability that mediated schema $\mathcal{M}_i$ is the correct one.*

In [42], the aforementioned uncertainty is captured through the similarity score thresholds. If the similarity between two source attributes is high enough (larger than a threshold), the two attributes are connected with a certain edge. If the similarity is too low (below another threshold), the two attributes are not connected. If the similarity is between the high and low thresholds, the two attributes are connected with an uncertain edge. A mediated schema is created for each subset of uncertain edges.

The mediated schemas should be assigned probabilities that reflect how well they represent the domain of the data sources. In [42], the probabilities are assigned to mediated schemas based on a definition of consistency of the mediated schema. A mediated schema M_i is consistent with a source schema S_j if there is no pair of source attributes in S_j that is in the same mediated attribute in M_i. The probability of any mediated schema is the number of source schemas it is consistent with divided by the total number of consistencies counted.

Similarly, a probabilistic mapping is defined between each source schema $\mathcal{S}_i$ and mediated schemas $\mathcal{M}_j$. The probabilistic mapping can possibly consist of several mappings each of which is associated with a probability.

Definition 7 *A probabilistic mapping $\mathcal{PM}ap_{ij}$ is defined by: $\mathcal{PM}ap_{ij} = \{(\mathcal{M}ap_1, Pr(\mathcal{M}ap_1)), \ldots, (\mathcal{M}ap_l, Pr(\mathcal{M}ap_l))\}$, where $l \geq 1$ is the number of mappings between source schema $\mathcal{S}_i$ and mediated schema $\mathcal{M}_j$.*

In [42], finding the possible mappings and assigning probabilities to them is represented as an optimization problem. The goal of the optimization problem is to find mappings and assign probabilities that are based on the similarity between source attributes and mediated attributes with the goal of maximizing their entropy. Formally, the goal is to *maximize* $\sum_{k=1}^{l} -p_k \log p_k$ subject to:
1. $\forall k \in [1, l], 0 < p_k < 1$. Where l is the number of all possible mappings. The probability of any mapping is between 0 and 1.
2. $\sum_{k=1}^{l} p_k = 1$. The sum of all probabilities of all mappings from a data source to a mediated schema is 1.
3. $\forall i, j, \sum_{k \in [1,l],(i,j) \in m_k} p_k = p_{i,j}$. Where $p_{i,j} = \frac{\sum_{a \in mA_j} sim(a_i, a)}{|mA_j|}$ is the weighted correspondence between source attribute a_i and mediated attribute mA_j.

When queries are issued against the probabilistic mediated schema, answer tuples are computed from each possible mediated schema, and a probability is computed for each answer tuple. An answer tuple from a data source using a specific mapping is assigned the probability of this mapping. It is possible that a tuple is generated using multiple mappings from the same data source that have different probabilities. The probability of a tuple that is generated from one data source using multiple mappings is given by $p = \sum_{i=1}^{k} Pr(t|M_i) * Pr(M_i)$, where $Pr(t|M_i)$ is the probability of the mapping from the source to the mediated schema M_i which is used to find the tuple t, and $Pr(M_i)$ is the probability of the mediated schema. If tuple t can be generated from d data sources and has a probability p_i for data source i, its final probability is $Pr(t) = 1 - \prod_{i=1}^{d}(1 - p_i)$ because it is assumed that the mappings from different data sources are independent of each other. This model is referred to in this chapter as *probabilistic data integration.*

5.2.3 Answering Queries over Relational Mediated Schemas

The focus in this chapter is on *select-project (SP)* queries using a SQL-like syntax. Supporting joins and more complex queries is left as future work. A query has a $SELECT$ clause and a $WHERE$ clause. There is no $FROM$ clause because queries are issued over all data sources. This type of queries conforms with prior work [42].

As mentioned earlier, the philosophy in UFeed is to isolate the user as much as possible from the details of the mediated schema and mappings, since the focus is on users who are not experts and who may not have detailed knowledge about the semantics of all queried data sources. Thus, UFeed adopts the following two-step model for querying the data sources: First, the user writes a query over one source schema S_i. The query over the source schema is rewritten to a query over the mediated schema. This is done by replacing each source attribute with the mediated attribute it maps to according to the schema mapping of this data source. If it is not possible to replace all source attributes in the query with mediated attributes, the query is issued only over the data source it is composed over. Formally, the user issues a query over a source schema S_i using the source attributes $\overline{S_i.a} \subseteq S_i$, where using an overline denotes a set. This query $Q(\overline{S_i.a})$ is rewritten to be issued over the mediated schema $\mathcal{M}$ using the following rule: $Q(\overline{S_i.a}) \rightarrow Q(\mathcal{M}')|\forall S_i.a_j \in \overline{S_i.a} \exists (S_i.a_j \rightarrow \mathcal{M}'.mA_x) \in \mathcal{M}ap_i$, where $\mathcal{M}' \subseteq \mathcal{M}$, $|\mathcal{M}'| = |\overline{S_i.a}|$, and $\mathcal{M}'.mA_x$ is a mediated attribute in $\mathcal{M}'$.

Second, once a query over the mediated schema is obtained, the query is rewritten using the appropriate mappings so that it can be issued over all relevant data sources. For each source schema, if there is a source attribute that maps to a mediated attribute in the query, the query is rewritten so that the source attribute replaces the mediated attribute in the $SELECT$ or $WHERE$ clause. The query is rewritten for all data sources that are represented in the mediated attributes in the query. Formally, the query over the mediated schema that was obtained from the previous step $Q(\mathcal{M}')$ is rewritten to be issued over any source schema S_y according to the following rule: $Q(\mathcal{M}') \rightarrow Q(\overline{S_y.a})|\forall \mathcal{M}'.mA_x \in \mathcal{M}' \exists (S_y.a_l \rightarrow \mathcal{M}'.mA_x) \in \mathcal{M}ap_y$, where $|\mathcal{M}'| = |\overline{S_y.a}|$. The answers to the rewritten queries over all source schemas that satify this rule are combined using a union operation.

5.3 Refinement in UFeed

This section presents how UFeed accepts and stores user feedback and the operations triggered by this feedback. In using feedback to refine the mediated schema and mappings, UFeed has to address the following challenges:

1. Which source attributes should be in the mediated schema? Typically, data integration systems do not include all attributes from all data sources in the mediated schema. Doing so would make the mediated schema too large and semantically incoherent, and would make the mappings too complex and difficult to use. Choosing source attributes based on their frequency in the data sources has been used in prior work [42]. Whether this or some other method is used, the choice of attributes will not be perfect. Desired attributes may be excluded, and undesired ones may exist in

the mediated schema. Even if a suitable frequency threshold is found for a specific domain, this threshold may be different for other domains.

2. What happens if UFeed receives conflicting feedback or performs incorrect refinement? One way to ensure correct feedback is to use feedback that is aggregated from multiple users over a period of time [51, 98]. However, even with this type of feedback aggregation, some of the feedback used by UFeed may be incorrect and result in incorrect refinements. UFeed needs to correct its mistakes based on future instances of correct feedback.

3. How should UFeed respond when the user marks a tuple in a query answer as incorrect? Is the answer incorrect because of an incorrect grouping of source attributes in a mediated attribute, an incorrect mapping from a source attribute to a mediated attribute, or because the data in the data source is incorrect? UFeed should pinpoint the origin of an error using only feedback over query answers.

4. How to adapt mappings to changes in the mediated schema? As the mediated schema is refined, some of the mappings are invalidated and some new ones need to be generated. UFeed should solve this problem without being dependent on the specific algorithm used to generate the mappings.

5.3.1 Attribute Correspondence and Answer Association

When a user issues a query and receives answer tuples, she can mark any of the answer tuples as "correct" (positive feedback) or "incorrect" (negative feedback). This is referred to as a *feedback instance*. Note that the user is not required to provide feedback on all answer tuples. She can choose as many or as few answer tuples as she wants to mark as "correct" or "incorrect".

Each feedback instance updates two in-memory data structures used by UFeed: the *attribute correspondence set* and the *answer association set*. An *attribute correspondence* links a source attributed used in the original query to a source attribute from another data source used in the rewritten query. This link means that the two source attributes represent the same concept. For example, a query over source schema S_3 from Example 1 is *SELECT country WHERE region = North America*. The query uses the attribute *country* which is rewritten to *country name* when issuing the query over S_2 using the mappings in Figure 5.2. If feedback is received over an answer tuple based on this rewritten query, the attribute correspondence entry (*country*, *country name*) is created and associated with the type of feedback received (positive or negative). The attribute correspondence set stores all the attribute correspondences inferred from feedback that is received by UFeed.

An *answer association* links a value in an answer tuple to the source attribute in the rewritten query that this value comes from. For example, (*country name*, *"USA"*) is an answer association. The answer association set stores all the answer associations derived from user feedback.

The attribute correspondence set and answer association set capture all the feedback received by UFeed in a way that allows the system to refine the mediated schema and mappings based on this feedback.

5.3.2 UFeed Operations

UFeed defines a set of abstract and independent operations that target several kinds of flaws in the mediated schema and mappings. The operations address adding/removing source attributes to/from the mediated schema, modifying mappings, and merging/splitting mediated attributes. This section describes these operations and how they are triggered.

Inject

The *Inject* operation overcomes the problem of missing source attributes in the mediated schema by adding source attributes required by the user to the mediated schema. As discussed in Section 5.2.3, queries are formulated over one of the source schemas. Querying an attribute that exists in a source schema but not in the mediated schema is a sufficient indication that this attribute is important to the user and needs to be injected in the mediated schema. This triggers the *Inject* operation. The main question for *Inject* is which mediated attribute the new source attribute should join. UFeed uses a minimum distance classifier to answer this question. The minimum distance classifier chooses the mediated attribute that has the source attribute that is most similar to the newly added source attribute (i.e., nearest neighbor [39]). Other types of classifiers can also be used [54]. A threshold α is introduced so that a source attribute is not forced to join a mediated attribute to which it has a relatively low similarity. A value $\alpha = 0.8$ is used in this chapter. If the new source attribute cannot join any existing mediated attribute, it is placed in a new mediated attribute that contains only this source attribute. Thus, *Inject* can be defined as follows:

Definition 8 *If the current set of source attributes in the mediated schema is* $\mathcal{A}' = attr'(S_1) \cup attr'(S_2) \cup \ldots \cup attr'(S_q)$, *where* $attr'(S_i)$ *is the set of source attributes of* *data source S_i that contribute to the mediated schema. Inject($S_i.a$) performs two steps: 1. $\mathcal{A}' \leftarrow \mathcal{A}' \cup \{S_i.a\}$, and 2. $mA_i \leftarrow mA_i \cup \{S_i.a\}$ for mA_i with the highest similarity to $S_{i.a}$ greater than α OR $mA_{|\mathcal{M}|+1} \leftarrow \{S_i.a\}$ if no mA_i has similarity to $S_i.a$ greater than α.*

Confirm

The *Confirm* operation is triggered when the user marks an answer tuple "correct". Since the answer tuple is correct, this means that the data, the mediated attributes, and mappings used to generate this tuple are correct. The correctness of the data is recorded in the answer association set, and the correctness of the mappings is recorded in the attribute correspondence set. In UFeed, there are two kinds of confirmations: *definite confirmations* and *tentative confirmations*. A definite confirmation is applied to the attribute correspondences and answer associations that are directly touched by the user feedback instance. A tentative confirmation is applied to the answer associations that are indirectly touched by this feedback instance. For example, consider the query $SELECT$ *country* $WHERE$ *region* = *North America* over S_3. This query is rewritten based on Map_1 in Figure 5.2 to be issued over S_1. The rewritten query is $SELECT$ *county* $WHERE$ *region* = *North America*. The same query is also rewritten based on Map_2 to be issued over S_2. The rewritten query is $SELECT$ *country name* $WHERE$ *region* = *North America*. The answers are shown in Figure 5.3. Giving positive feedback over the answer *"Canada"* leads to the creation of the attribute correspondences (S_3.*country*, S_2.*country name*) and (S_3.*region*, S_2.*region*), and the answer associations (S_2.*country name*, *"Canada"*) and (S_2.*region*, *"North America"*). There are also answer associations for the source attributes from S_3, but they are omitted for the clarity of the example. The aforementioned confirmations are all definite, since they are based directly on the tuple "Canada" over which the user provided positive feedback. Definite confirmations are represented in the figure by a solid green line. Other answer tuples that are generated by the same rewritten query are given tentative confirmations, represented by a dotted green line in the figure. Assigning tentative confirmations is based on the reasoning that the positive feedback provided by the user indicates that the value of the source attribute on which this feedback was given is correct, and this source attribute is indeed an instance of the mediated attribute. Other values of the source attribute are likely to be correct, so they should be confirmed. However, there may be errors in the data resulting in some of these values being incorrect. Therefore, the confirmation remains a tentative confirmation, and the confirmation of a value becomes definite only if the user explicitly provides positive feedback on this value.

The *Confirm* operation aims at protecting source attributes in the mediated attributes from being affected by other operations that alter the mediated schema, in particular, the

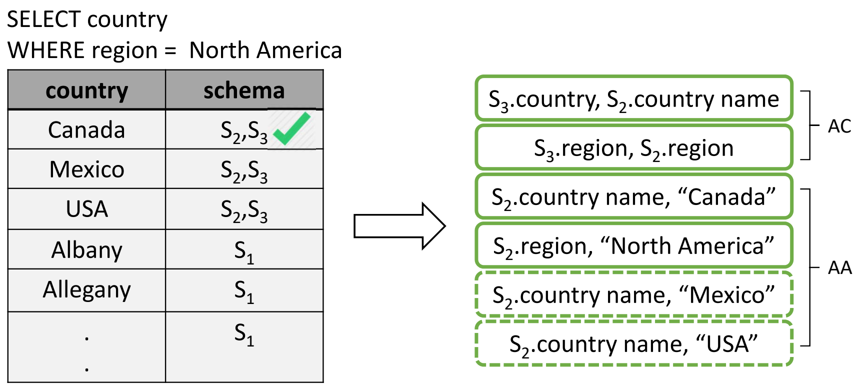

SELECT country
WHERE region = North America
country
schema
Canada
S_2,S_3
Mexico
S_2,S_3
USA
S_2,S_3
Albany
S_1
Allegany
S_1
S_1
S_3.country, S_2.country name
S_3.region, S_2.region
S_2.country name, "Canada"
S_2.region, "North America"
S_2.country name, "Mexico"
S_2.country name, "USA"
AC
AA

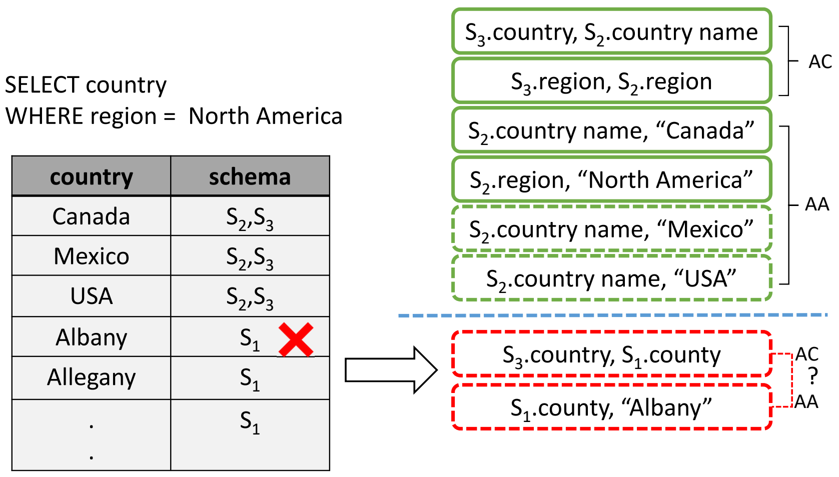
Figure 5.3: Positive feedback over an answer tuple and the resulting attribute correspondences (AC) and answer associations (AA).

Split and *Blacklist* operations that will be discussed next.

Split and Blacklist

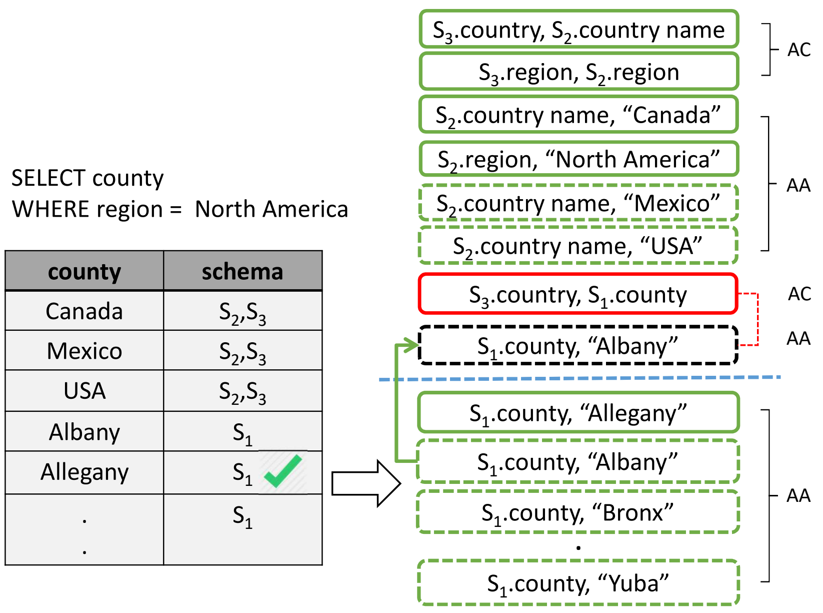
When negative feedback is received over an answer tuple, this means that either the data is incorrect or the mediated schema and mappings are incorrect. In particular, one or more attribute values in the source tuple may be incorrect, or the source attribute does not represent the same concept as the mediated attribute it is part of. Since there is no simple way to distinguish between these two causes of negative feedback, UFeed faces uncertainty about the action to take in response to such feedback. This uncertainty can be resolved when UFeed receives further feedback. UFeed records the attribute correspondences and answer associations created for this feedback instance. The attribute correspondences and answer associations for this feedback instance are *linked* together for future investigation based on future feedback. Figure 5.4 shows the uncertainty faced by UFeed when negative feedback is received over the answer *"Albany"*. UFeed does not know if the answer is incorrect because *country* and *county* should not be in the same mediated attribute, or because *"Albany"* is not a *county*. The attribute correspondence and answer association are linked together as shown in the figure (represented by a dotted red line).

Now, consider the query: *SELECT county WHERE region = North America* and its answers in Figure 5.5. Assume that positive feedback is received over *"Allegany"*. As explained earlier, a definite confirmation is applied to (S_1*.county, "Allegany"*). Note that no attribute correspondence is added because this answer tuple comes from the source schema over which the query is issued. Tentative confirmations are applied to the remaining answer


SELECT country
WHERE region = North America
S_3.country, S_2.country name
S_3.region, S_2.region
AC
S_2.country name, “Canada”
S_2.region, “North America”
S_2.country name, “Mexico”
S_2.country name, “USA”
AA
S_3.country, S_1.county
S_1.county, “Albany”
AC
?
AA

country	schema
Canada	S_2,S_3
Mexico	S_2,S_3
USA	S_2,S_3
Albany	S_1 ✗
Allegany	S_1
. .	S_1

Figure 5.4: Negative feedback over an answer tuple and the resulting linking of the attribute correspondence (AC) and answer association (AA) to which the feedback applies. Attribute correspondences and answer associations from the previous query are shown above the blue dotted line.


SELECT county
WHERE region = North America
S_3.country, S_2.country name
S_3.region, S_2.region
AC
S_2.country name, “Canada”
S_2.region, “North America”
S_2.country name, “Mexico”
S_2.country name, “USA”
AA
S_3.country, S_1.county
AC
S_1.county, “Albany”
AA
S_1.county, “Allegany”
S_1.county, “Albany”
S_1.county, “Bronx”
S_1.county, “Yuba”
AA

county	schema
Canada	S_2,S_3
Mexico	S_2,S_3
USA	S_2,S_3
Albany	S_1
Allegany	S_1 ✓
. .	S_1

Figure 5.5: Positive feedback over an answer tuple to a query asking for counties in North America.

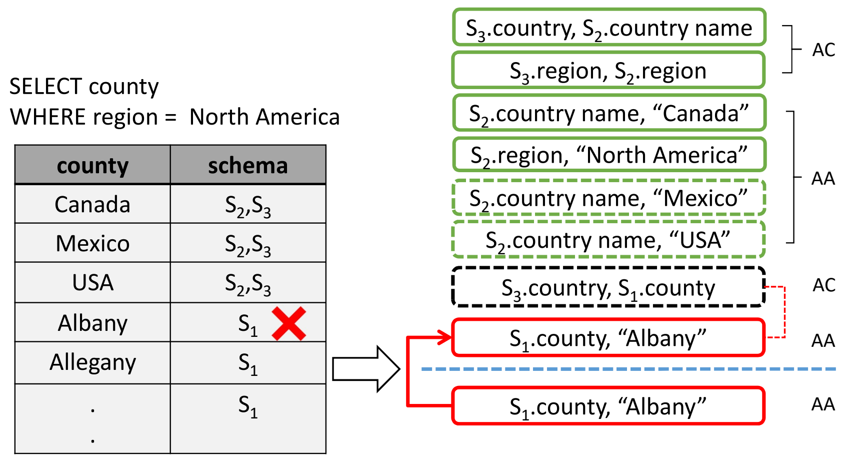
associations as explained earlier. However, the answer association (S_1.*county, "Albany"*) has been previously linked to a negative feedback instance. Conflicting feedback, as explained later in this section, results in updating the status of the entry to "unknown" (represented by a black dotted line), that is, neither correct nor incorrect. With this update, UFeed concludes that the reason for the negative feedback received in Figure 5.4 is that *county* should not be in the same mediated attribute as *country* (because *county* is the source attribute used to generate the answer "Albany"). This triggers the *Split* operation, which splits the source attribute used in the rewritten query from the mediated attribute it is part of, and forms a new mediated attribute that only contains this one source attribute. In this example, *county* is removed from the mediated attribute {*county, country name, country*} and the new mediated attribute {*county*} is added to the mediated schema. If the split source attribute is the only member of a mediated attribute, the mediated attribute is also removed. If the split source attribute does not exist in the mediated attribute, the correspondence between the source attribute and the mediated attribute in the mapping used to answer the query is removed. Thus, the *Split* operation is defined as:

Definition 9 *If $S_i.a$ is a source attribute, where $S_i.a \in \mathcal{A}'$, $Split(S_i.a, mA_x)$ performs three possible actions:*

$$\begin{cases} Remove(mA_x), \mathcal{A}' \leftarrow \mathcal{A}' - \{S_i.a\} \\ \qquad if\ S_i.a \in mA_x\ AND\ |mA_x| = 1 \\ OR\ 1.\ mA_x \leftarrow mA_x - S_i.a \quad 2.\ mA_{|\mathcal{M}|+1} \leftarrow \{S_i.a\} \\ \qquad if\ S_i.a \in mA_x\ AND\ |mA_x| > 1 \\ OR\ Remove(S_i.a \rightarrow mA_x) \\ \qquad if\ S_i.a \notin mA_x \end{cases}$$

UFeed uses the following heuristic: When the user provides negative feedback indicating that an answer tuple is incorrect, UFeed assumes that there is only one mistake that caused this answer tuple to be incorrect. This can be a mistake in the mediated schema or mappings, or it can be erroneous data. If it happens that multiple mistakes cause an answer tuple to be incorrect, UFeed will fix the mistakes one by one based on multiple instances of negative feedback.

To illustrate another way UFeed identifies the cause of negative feedback, assume that instead of giving positive feedback over *"Allegany"*, the user provides negative feedback over *"Albany"*, as shown in Figure 5.6. This feedback is incorrect since *"Albany"* is in fact a county, but it serves the example. In this case, UFeed does not face uncertainty about the reason for this negative feedback because this answer tuple is generated from one source


SELECT county
WHERE region = North America

county	schema
Canada	S_2,S_3
Mexico	S_2,S_3
USA	S_2,S_3
Albany	S_1 ✗
Allegany	S_1
. .	S_1

S_3.country, S_2.country name
S_3.region, S_2.region
AC
S_2.country name, “Canada”
S_2.region, “North America”
S_2.country name, “Mexico”
S_2.country name, “USA”
AA
S_3.country, S_1.county
AC
S_1.county, “Albany”
AA
S_1.county, “Albany”
AA

Figure 5.6: Negative feedback over an answer tuple to a query asking for counties in North America.

(S_1), without using any mappings. UFeed knows now that *“Albany”* is erroneous data in the data source. This triggers the *Blacklist* operation. This operation maintains a blacklist that keeps track of incorrect answer associations in the answer association set. The blacklist is used in the query answering process to remove erroneous data from query answers. In the example, the blacklist removes *“Albany”* from future answers. This negative feedback also updates the status of the attribute correspondence ($S_3.country$, $S_1.county$) to “unknown” until future feedback indicates it is incorrect.

Merge

The *Merge* operation is triggered when two or more answer associations share the data value while having different source attributes. For example, consider the query in Figure 5.7, which finds countries and their “gdp per capita purchase power parity” values in North America. Assume this query is issued after the query in Figure 5.3. Receiving positive feedback over the answer tuple (*“Canada”, “45553”*) results in definite confirmation of the answer associations (S_4*.name, “Canada”*) and (S_4*.gdp per capita ppp, “45553”*). However, the answer association (S_2*.country name, “Canada”*) exists and was confirmed by the user. This triggers the *Merge* operation, which merges the two mediated attributes that contain the two source attributes in the answer associations are in. The *Merge* operation is triggered when two answer associations that share the data value are confirmed with definite or tentative confirmation.

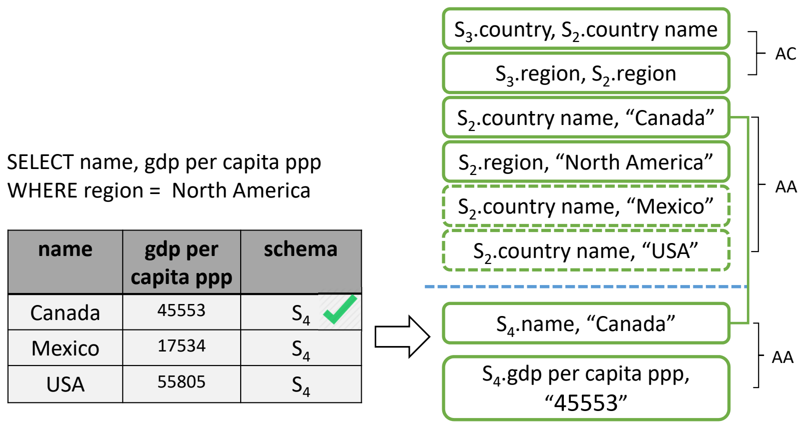

SELECT name, gdp per capita ppp
WHERE region = North America
name
gdp per capita ppp
schema
Canada
45553
S4
Mexico
17534
S4
USA
55805
S4
S3.country, S2.country name
S3.region, S2.region
AC
S2.country name, "Canada"
S2.region, "North America"
S2.country name, "Mexico"
S2.country name, "USA"
AA
S4.name, "Canada"
S4.gdp per capita ppp, "45553"
AA

Figure 5.7: Positive feedback that results in triggering the *Merge* operation.

Definition 10 *If $S_i.a$ and $S_j.b$ are two source attributes, where $S_i.a, S_j.b \in \mathcal{A}'$, $Merge(S_i.a, S_j.b)$ performs two steps: 1. $mA_i \leftarrow mA_i \cup mA_j | S_i.a \in mA_i, S_j.b \in mA_j$. 2. $Remove(mA_j)$.*

Adapt

The *Adapt* operation updates the mappings whenever the mediated schema is changed by other UFeed operations. It is an essential operation for maintaining semantic coherence as the mediated schema changes. Formally, when a mediated schema $\mathcal{M}$ is changed into a new mediated schema $\mathcal{M}'$ via any of the aforementioned UFeed operations, the mapping $\mathcal{M}ap_i$ for any source schema S_i that is affected by the changes should evolve into $\mathcal{M}ap'_i$ that maps S_i to $\mathcal{M}'$.

The UFeed operations that trigger the *Adapt* operation are: *Inject*, *Split*, and *Merge* because these are the operations that make changes to the set of attributes in the mediated schema $\mathcal{A}$ and their groupings into mediated attributes.

Mapping adaptation after *Inject*: The *Inject* operation introduces a new source attribute to the set of source attributes in the mediated schema. This change in the mediated schema will require updating the mapping of the source schema that the injected source attribute is part of. Mapping adaptation in this case is straight forward. Following Definition 8, if the injected source attribute is $S_i.a_x$ the new mapping becomes $\mathcal{M}ap'_i = \mathcal{M}ap_i \cup (S_i.a_x \rightarrow mA_j)$, where $S_i.a_x \in mA_j \wedge (!\exists S_i.a_y \in mA_j | x \neq y)$. This means that the injected source attribute should map to the mediated attribute it is injected in, unless another source attribute from the same source schema is in this mediated attribute. In the latter case, no mapping is generated because it is assumed that no two source attributes in a table should represent the same real-world concept.

Mapping adaptation after *Split*: Definition 9 has three different cases for the *Split* operation. Different mapping adaptations apply in each case.

$$\begin{cases} Remove(S_i.a \rightarrow mA_x) \\ OR\ S_i.a \rightarrow mA_{|M|+1} \\ OR \text{ No Action} \end{cases}$$

For the first case, where the source attribute and the mediated attribute are removed, the correspondence to the mediated attribute is also removed. For the second case, where one source attribute is split from the mediated attribute, the split source attribute maps to the new mediated attribute that only contains the split source attribute. The third case removes the mapping, so no adaptation is needed.

Mapping adaptation after *Merge*: When two mediated attributes are merged according to Definition 10, the mappings from any source attribute to any of the merged mediated attributes should be updated. Following Definition 10, mappings to the mediated attribute mA_i do not need to be updated. Only mappings to the mediated attribute mA_j should be changed to map to mA_i.

5.3.3 Applying UFeed Operations to Probabilistic Mediated Schemas and Mappings

As noted earlier, UFeed operations can be applied on probabilistic mediated schemas and mappings. The challenge in this case is that there are several mediated schemas, each with an associated probability. UFeed uses a simple solution to address this challenge: it deals with each mediated schema independently, as if it were the output of a holistic schema matching system. Whenever two mediated schemas are equivalent (have the exact same grouping of source attributes), one is removed. The ultimate goal of schema refinement in this case is for the probabilistic mediated schema to converge to a single (non-probabilistic) mediated schema.

Calculating probabilities for mediated schemas and mappings in UFeed is dependent on the details of the probabilistic data integration approach. UFeed should incorporate the probability calculation method in order to update the probabilities as the probabilistic mediated schema and mappings are refined. This chapter uses the probability calculation method of [42], which was discussed in Section 5.2.2.

5.3.4 Handling Incorrect Feedback

UFeed expects incorrect feedback to be rare, but it can cancel the effects of such feedback through future correct feedback. UFeed does not know that a feedback instance is incorrect. However, it makes changes to the mediated schema and mappings based on one feedback instance at a time, and the changes that it makes are independent of each other. Therefore, more correct feedback instances will ultimately undo the negative effects caused by incorrect feedback.

Inject: An incorrect *Inject* can be fixed using the *Split* operation. If the injected source attribute is incorrectly placed in an existing mediated attribute, negative feedback over answers that are generated using the injected source attribute will split it from the mediated attribute. If the source attribute forms a mediated attribute on its own, negative feedback removes this source attribute, and consequently the mediated attribute, from the mediated schema.

Confirm: When an attribute correspondence is mistakenly given a definite confirmation, the source attribute cannot be split from the mediated attribute. This confirmation cannot be removed unless a negative feedback is received and it is known that the error comes from the attribute correspondence and not an answer association. In this case, the confirmation is removed and the status of the attribute correspondence is updated to "unknown".

Split: If a source attribute is mistakenly split due to incorrect feedback, this can be simply undone by receiving correct feedback that triggers the *Merge* operation, which merges the source attribute that was split with the mediated attribute it was split from. If the *Split* results in removing the source attribute from the mediated schema, this will require the user to query the removed source attribute to trigger an *Inject* operation to add it to the mediated schema.

Blacklist: To avoid blacklisting a value that may be correct due to incorrect feedback, UFeed uses a *second chance* technique for this specific operation. The second chance technique allows a value to be shown to the user after it is identified by UFeed as erroneous. The vaule is allowed to be shown to the user until another negative feedback is received over it. At that point, the value is blacklisted for future queries.

Merge: An incorrect merge operation that merges two mediated attributes can be undone by receiving negative feedback over any of the merged source attributes. This will trigger a *Split* operation that will split the mistakenly merged source attribute from the new mediated attribute.

5.4 UFeed Completeness

The gold standard is defined as the mediated schema and mappings designed by a human expert through manual integration of the data sources. This section proves that UFeed can transform any automatically generated mediated schema to the gold standard. I decompose the gold standard of n data sources into the following:

1. Set of Source Attributes $\mathcal{A}_G$: This is the set of attributes that is part of the gold standard mediated schema, not to be confused with $\mathcal{A}$ defined in Section 5.2, which is the union of source attributes in all data sources. If all source attributes are of relevance and included in the mediated schema, then $\mathcal{A}_G \equiv \mathcal{A}$. However, in practice, $\mathcal{A}_G \subseteq \mathcal{A}$.

2. Mediated Schema $\mathcal{M}_G$: The mediated schema consists of several mediated attributes, each of which is a cluster of one or more source attribute from $\mathcal{A}_G$. Formally, $\mathcal{M}_G = \{mA_1{}^G, \cdots, mA_k{}^G\}$ where k is the number of mediated attributes in the gold standard, $mA_i{}^G \subseteq \mathcal{A}_G$, $i \in [1, k]$.

Assume the data integration system generates the corresponding components $\mathcal{A}'$ and $\mathcal{M}'$, which are then used by UFeed as the starting point for refinement. This section proves that the UFeed operations can transform $\mathcal{A}'$ and $\mathcal{M}'$ into $\mathcal{A}_G$ and $\mathcal{M}_G$, respectively.

Lemma 5.4.1 *Let $\mathcal{A}_G$ be the set of source attributes in the gold standard mediated schema $\mathcal{M}_G$, and $\mathcal{A}'$ be the set of source attributes of any mediated schema $\mathcal{M}'$. Then $\mathcal{A}'$ can be transformed into $\mathcal{A}_G$ using* Inject *(Definition 8) and* Split *(Definition 9).*

Proof $\mathcal{A}_G$ can differ from $\mathcal{A}'$ as follows:

1. $\exists a \in \mathcal{A}_G, a \notin \mathcal{A}'$. That is, a source attribute a exists in the set of source attributes of the gold standard $\mathcal{A}_G$, and a does not exist in the current set of source attributes $\mathcal{A}'$. According to Definition 8, *Inject(a)* would transform $\mathcal{A}'$ into $\mathcal{A}' \cup a$.

2. $\exists b \in \mathcal{A}', b \notin \mathcal{A}_G$. That is, a source attribute b exists in the current set of source attributes $\mathcal{A}'$, and it does not exist in the set of source attributes of the gold standard $\mathcal{A}_G$. There are two scenarios:

   (a) $b \in mA_x, |mA_x| > 1$. That is, b exists in a mediated attribute mA_x that contains other source attributes. According to Definition 9, *Split(b, mA_x)* $\Rightarrow mA_{|\mathcal{M}'|+1} \leftarrow \{b\}$. This will split b from mA_x and create the new mediated attribute $mA_{|\mathcal{M}'|+1} = \{b\}$. Then, also according to Definition 9, another *Split(b,* $mA_{|\mathcal{M}'|+1}$) will remove $mA_{|\mathcal{M}'|+1}$ which includes b.

(b) $b \in mA_x, |mA_x| = 1$. That is, b exists in a mediated attribute mA_x that contains only b. According to Definition 9, *Split(b, mA_x)* will remove mA_x which includes b.

Therefore, $\mathcal{A}'$ can be transformed into $\mathcal{A}_G$.

Lemma 5.4.2 *Let $\mathcal{M}_G$ be the gold standard mediated schema of a set of source attributes $\mathcal{A}_G$, and $\mathcal{M}'$ be any mediated schema of a set of source attributes $\mathcal{A}'$. If $\mathcal{A}'$ is transformed into $\mathcal{A}_G$ (Lemma 5.4.1), then $\mathcal{M}'$ can be transformed into $\mathcal{M}_G$ using* Merge *(Definition 10) and* Split *(Definition 9).*

Proof The lemma can be proven by construction, by describing a sequence of *Split* and *Merge* operations that result in the desired transformation of $\mathcal{M}''$ to $\mathcal{M}_G$. From Lemma 5.4.1, $\mathcal{A}'$ can be transformed into $\mathcal{A}_G$, and $\mathcal{A}_G = \{a_1, \ldots, a_l\}$, where l is the number of source attributes in $\mathcal{M}_G$. Whenever a Split or Inject is applied to $\mathcal{A}'$, the operation, by definition, also changes the mediated schema $\mathcal{M}'$. Denote by $\mathcal{M}''$ the schema $\mathcal{M}'$ after this transformation, and let $\mathcal{M}'' = \{mA''_1, \ldots, mA''_q\}$, where q is the number of mediated attributes in $\mathcal{M}''$. Note that since $\mathcal{A}'$ has been transformed to $\mathcal{A}_G$, $mA''_j \subseteq \mathcal{A}_G$, for all $j \in [1, q]$. Let all source attributes in $\mathcal{M}''$ be split from the mediated attribute they are in. That is, formally, $\forall a''_i \in mA''_j$, where $|mA''_j| > 1$, *Split(a''_i, mA''_j)*. This will transform $\mathcal{M}''$ into a set of single-source-attribute mediated attributes. Formally, $\mathcal{M}'' = \{\{a_1\}, \ldots, \{a_l\}\}$. For each mediated attribute $mA_i{}^G \in \mathcal{M}_G$, merge mediated attributes in $\mathcal{M}''$ to form $mA_i{}^G$. Formally, in $\mathcal{M}''$, $\forall a_x, a_y \in mA_i{}^G \in \mathcal{M}_G$, *Merge*$(a_x, a_y)$. This leads to $\mathcal{M}''$ becoming equivalent to $\mathcal{M}_G$, which proves the lemma.

Theorem 5.4.3 *Let $\mathcal{A}$ be the set of source attributes that is the input to any data integration system, and $\mathcal{A}'$ be the set of source attributes that is part of the mediated schema $\mathcal{M}'$. Let $\mathcal{A}_G$ be the gold standard set of source attributes and $\mathcal{M}_G$ be the gold standard mediated schema. $\mathcal{A}'$ and $\mathcal{M}'$ can be transformed into $\mathcal{A}_G$ and $\mathcal{M}_G$, respectively, using the UFeed operations: Inject, Merge, and Split.*

Proof Source attributes can be added and removed from $\mathcal{A}'$ so it can be transformed into $\mathcal{A}_G$ using Inject and Split (Lemma 5.4.1). Then, the set of mediated attributes in $\mathcal{M}'$ can be altered to form the mediated attributes in $\mathcal{M}_G$. Therefore, $\mathcal{M}'$ can be transformed into $\mathcal{M}_G$ using Merge and Split (Lemma 5.4.2).

It is important to note that the transformation of $\mathcal{A}'$ into $\mathcal{A}_G$ goes hand in hand with altering the mediated attributes in $\mathcal{M}'$ (i.e., Lemma 5.4.1 is applied together with Lemma 5.4.2). Also, it is not necessary to do all the splits before merging the source attributes into mediated attributes. For example, if the gold standard mediated attribute is *{country name, country, name}*, and the current related mediated attributes in $\mathcal{M}'$ are *{country, county}* and *{country name}*, it is not necessary to inject *name* before removing *county* and merging *country* and *country name*. This means that UFeed can apply its operations based on the order of feedback it receives from the user.

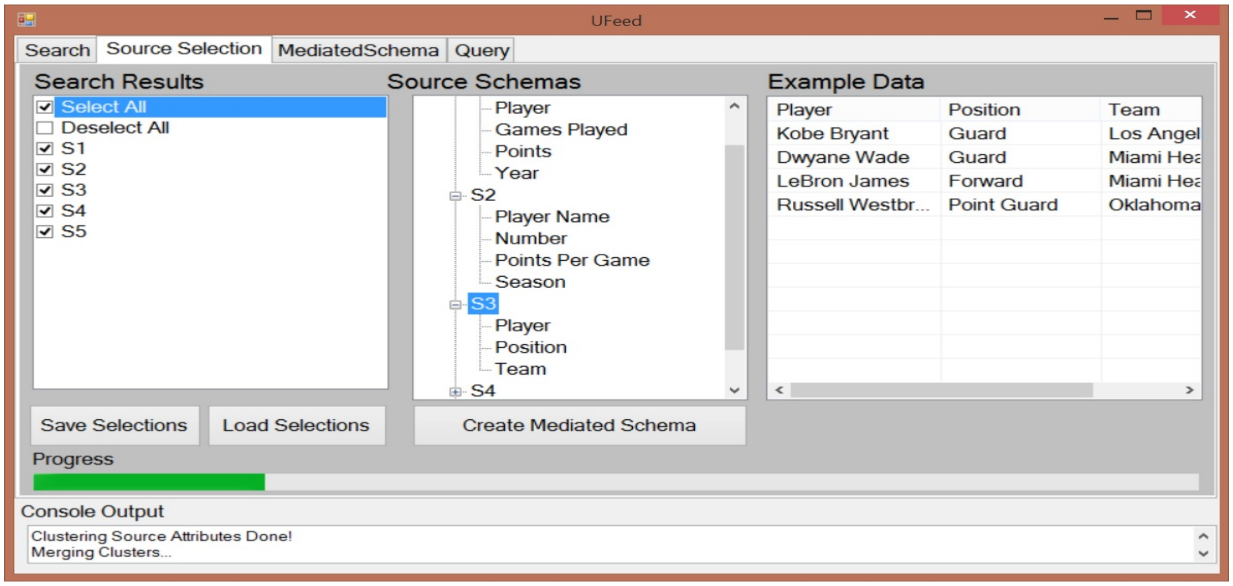
5.5 User Interface of UFeed

UFeed has a simple user interface that demonstrates the functionality of the system by applying it on Google Fusion Tables [1]. The user interface uses the probabilistic schema matching and mapping approach of [42], and the probability computation in Section 5.2.2. This section overviews the different capabilities provided by this user interface.

5.5.1 Generating Mediated Schemas and Mappings

UFeed gives users the ability to select data sources by searching for tables that belong to a domain of interest by issuing a keyword search over Google Fusion Tables. A sample of the results of the search using the keyword "NBA Players" is shown in Figure 5.8. The user can browse the source schemas and click on any of them to view example tuples from the table (the figure shows a screenshot after the user has clicked on schema $S3$). This browsing step enables the user to decide which schemas should be included when creating the mediated schema and mappings. The user can also save her selections for future use.

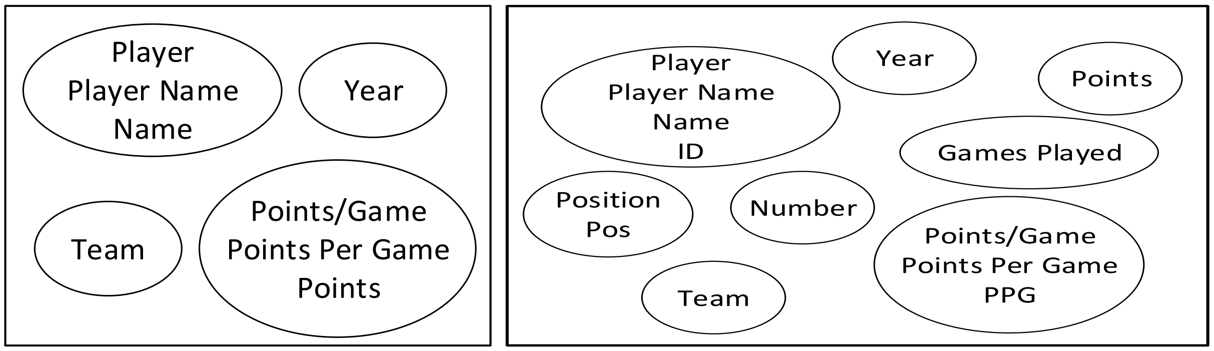
After the user selects the data sources she wishes to integrate, UFeed creates the mediated schema and mappings for these sources using any automatic schema matching and mapping approach. A real deployment of UFeed would not expose web users who are not necessarily data experts to the mediated schema that is generated by this technique. However, this user interface allows the user to see the mediated schemas generated by probabilistic schema mediation, along with their associated probabilities. The user can also browse the source schemas and click on any source attribute to highlight the mediated attribute that the selected source attribute maps to (if a mapping exists). An initial mediated schema and the gold standard for some schemas related to the domain of NBA basketball are shown in Figure 5.9.



Player	Position	Team
Kobe Bryant	Guard	Los Angel
Dwyane Wade	Guard	Miami Hea
LeBron James	Forward	Miami Hea
Russell Westbr...	Point Guard	Oklahoma

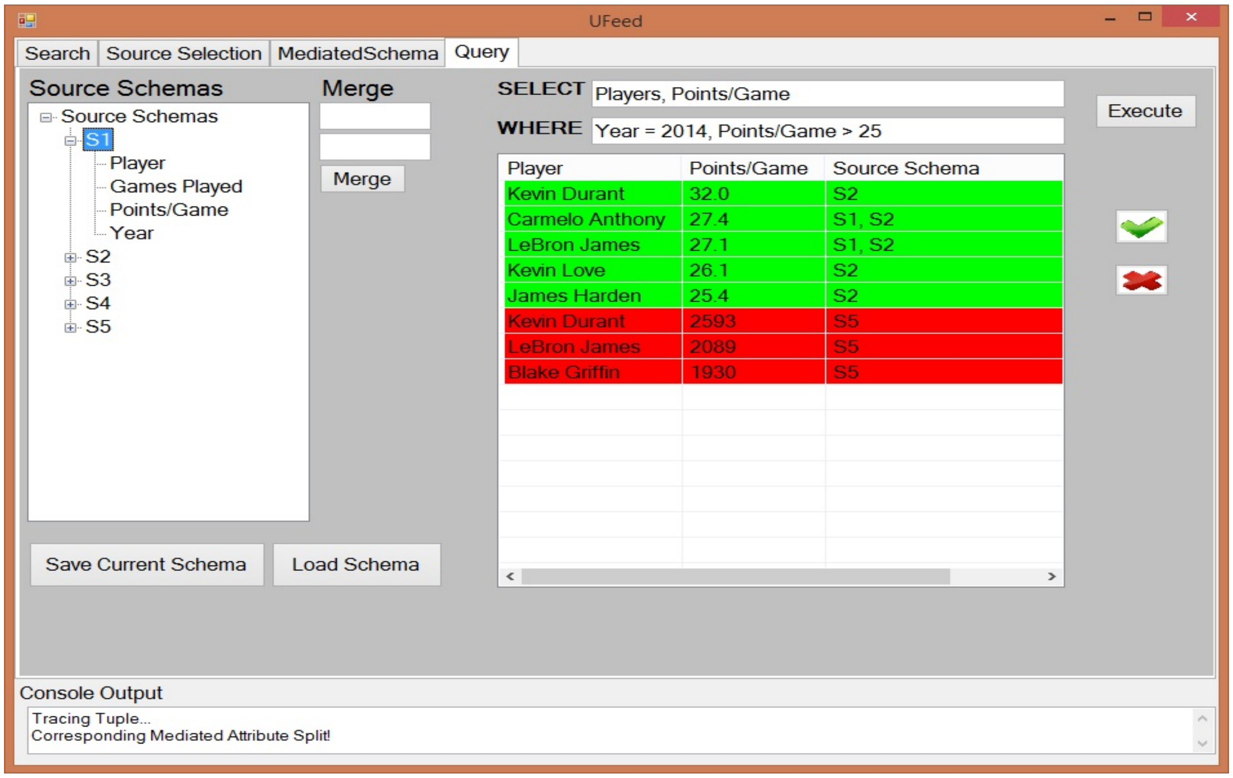
```
Clustering Source Attributes Done!
Merging Clusters...
```

Figure 5.8: Selecting data sources in UFeed.



(a) Initial Mediated Schema (b) Gold Standard

Figure 5.9: The initial mediated schema vs. the manually created gold standard for the domain of NBA basketball.



SELECT: Players, Points/Game

WHERE: Year = 2014, Points/Game > 25

Player	Points/Game	Source Schema
Kevin Durant	32.0	S2
Carmelo Anthony	27.4	S1, S2
LeBron James	27.1	S1, S2
Kevin Love	26.1	S2
James Harden	25.4	S2
Kevin Durant	2593	S5
LeBron James	2089	S5
Blake Griffin	1930	S5

```
Tracing Tuple...
Corresponding Mediated Attribute Split!
```

Figure 5.10: Answers to the query "*SELECT Player, Points/Game WHERE Year = 2014, Points/Game > 25*" (Correct answers are in green and incorrect answers are in red).

5.5.2 Query Processing

UFeed allows the user to issue queries over any of the selected data sources. UFeed tries to find answers to the query in all available data sources using the available mediated schemas and mappings. Figure 5.10 shows an example query over source schema $S1$, where the user is interested in finding the names of NBA players and their points per game score in the 2014 season for players whose score is over 25. To retrieve this information, the user issues the query: *SELECT Player, Points/Game WHERE Year = 2014, Points/Game > 25* (UFeed uses a comma in the WHERE clause in lieu of the AND keyword). The figure shows that answers are generated from 3 different data sources: 1. $S1$ using the mappings *Player* $\rightarrow$ *{Player, Player Name, Name}* and *Points/Game* $\rightarrow$ *{Points/Game, Points Per Game, Points}*. 2. $S2$ using the mappings *Player Name* $\rightarrow$ *{Player, Player Name, Name}* and *Points per game* $\rightarrow$ *{Points/Game, Points Per Game, Points}*. 3. $S5$ using the mappings

Name $\rightarrow$ *{Player, Player Name, Name}* and *Points* $\rightarrow$ *{Points/Game, Points Per Game, Points}*. When the answers are first shown to the user, the tuples are not highlighted red or green. The tuples are highlighted only when feedback is given on them. The answers from different data sources are combined and ranked according to their probabilities. The answers are shown to the user along with the source schema(s) that they come from.

5.5.3 Feedback and Schema Refinement

The user can select any tuple and mark it as correct or incorrect. As discussed in Section 5.3, feedback is represented by the set of attribute correspondences and the set of answer associations. In the example in Figure 5.10, marking the tuple *{James Harden,* 25.4*}* as "correct" will result in giving the tuples values generated using the mapping from $S2$ with tentative confirmations (except from the values of the tuple *{James Harden,* 25.4*}* which are given difinite confirmations because the feedback is received directly over this tuple). Also, the attribute correspondences *(S_1.Player, S_2.Player Name)* and (S_1*.Points/Game, S_2.Points Per Game)* are confirmed. These attribute correspondence confirmations mean that they should not be split from their mediated attributes when future feedback triggers a *Split* operation targeting these mediated attributes.

In the same query answer in Figure 5.10, it is clear that a player cannot score 2000 or more points in a game, so the user can mark any of the last three tuples as "incorrect". UFeed faces the uncertainty of identifying why the answer is incorrect. Assume the negative feedback is received over the tuple *{LeBron James, 2089}*. The candidate causes are: (a) the attribute correspondence *(S_1.Player, S_5.Name)*, (b) the attribute correspondence (*S_1.Points/Game, S_5.Points)*, (c) the answer association *(S_5.Name, LeBron James)*, or (d) the answer association *(S_5.Points, 2089)*. Assume an earlier query was issued to find all NBA players (any positive feedback will confirm (a) and tentatively confirm (c)), and another query for points (positive feedback will either result in difinite or tentative confirm of (d)). Therefore, UFeed will determine that the reason of the tuple *{LeBron James, 2089}* being incorrect is (b). This will result in splitting *Points* from the mediated attribute *{Points/Game, Points Per Game, Points}*.

Keywords	Tables	Attributes/Table	Total Attributes	Tuples/Table	Total Tuples
Internet Usage	38	2–230	886	7–261	3826
Movies	20	1–16	117	10–3201	7422
World GDP	12	2–230	839	12–262	4830

Table 5.1: Data sets used in the experiments.

5.6 Experimental Evaluation

5.6.1 Experimental Setup

The experimental evaluation of UFeed was run on a machine with Intel Core i7 CPU at 2.6 GHz and 8 GB of memory. To generate the starting mediated schema and mappings for UFeed, the techniques in [117] were used for the holistic data integration approach and the techniques in [42] were used for the probabilistic data integration approach. These two techniques are the state-of-the-art in holistic and probabilistic data integration.

Data Sources: The data sources for the experiments were extracted from Google Fusion Tables. To find a set of data sources representing a given domain, a keyword search query is issued using keywords representing the domain on Google Fusion Tables. The top 150 tables returned are considered and manually refined to eliminate redundant or irrelevant ones. The refined set of tables was input to the integration algorithms (holistic and probabilistic) to create the mediated schema(s) and mappings. The mediated schema(s) and mappings are used to answer queries formulated over any data source by returning answers from other data sources that can contribute to the answer set (Section 5.2). Table 5.1 presents the data used in the experiments. Each row represents one domain, and the name of the domain is also the keyword used for searching in Google Fusion Tables.

The holistic data integration approach generates one mediated schema and one set of mappings for each domain. The probabilistic data integration approach generates one mediated schema for the "Movies" domain, and two mediated schemas for each of the "World GDP" and "Internet Usage" domains. The number of probabilistic mappings for each source in the three domains ranged from 0 mappings, where the source does not map to the mediated schema to 8 different possible mappings from one source to the mediated schema of the domain.

Gold Standard: For each domain, I manually created a gold standard mediated schema and the corresponding mapping for each data source. When used, the gold standard returns correct and complete answers to all queries.

Queries: For each domain, I manually constructed a set of queries that trigger the UFeed operations. These queries focus on the parts of the mediated schema that differ from the gold standard. When the queries are issued before UFeed refines the schema, they can return incomplete, incorrect, or empty answers. As the UFeed operations are applied, the quality of the query answers improves. The experiments use these queries because queries that target parts of the mediated schema that are already the same as the gold standard will always return complete and correct answers, and thus offer limited opportunities for testing UFeed. The number of queries in each setting is shown in Table 5.2.

5.6.2 Quality of Query Answers

This section evaluates whether UFeed improves the quality of query answers. To measure the quality of the current mediated schema (or schemas in the case of probabilistic data integration) and mappings, the queries are run over both the gold standard and the current mediated schema and mappings. Assuming the answer set from the gold standard is G and the answer set from the current mediated schema and mappings used by UFeed is A, the quality of query answers is measured using: *Precision* $P = \frac{|A \cap G|}{|A|}$, *Recall* $R = \frac{|A \cap G|}{|G|}$, and *F-measure* $F = \frac{2PR}{P+R}$. The average precision, recall, and F-measure of the answers to all queries in the set of evaluation queries are computed for each domain. It was noted that there is no large gap between precision and recall curve. Therefore, only the F-measure is reported in this section.

In all experiments, feedback is provided to UFeed after each query, and UFeed immediately refines the mediated schema and mappings based on this feedback so that subsequent queries get higher quality answers. Feedback is generated by comparing sample answers from the two answer sets A and G (defined above). However, feedback is not generated for each tuple in A and G. One tuple is randomly chosen from A and is looked for it in G. If it exists, a positive feedback instance is generated. If it does not, a negative feedback instance is generated. After every UFeed operation, all queries in the evaluation set that were affected by this operation are rerun and the F-measure is recomputed.

The order of the queries run, and consequently the UFeed operations, has some effect on the quality of the query answers. To take into account the effect of query ordering, each experiment is rerun 10 times with a different random ordering of the queries and the average F-measure for the 10 runs is reported along with its 95% confidence interval. The random orderings are created under some simple ordering constraints. For example, a source attribute cannot be merged with another source attribute before it is injected in the mediated schema.

	Internet Usage	Movies	World GDP
Holistic	49	32	40
Probabilistic	85	65	74

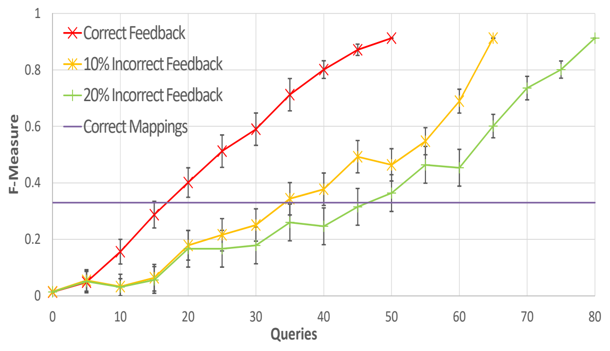
Table 5.2: Number of queries in the different settings.

UFeed recovers from incorrect feedback by relying on future correct feedback. To evaluate the effectiveness of this approach, UFeed is tested with 10% and 20% incorrect feedback instances chosen at random and it is shown that UFeed can overcome their effect through additional correct feedback.

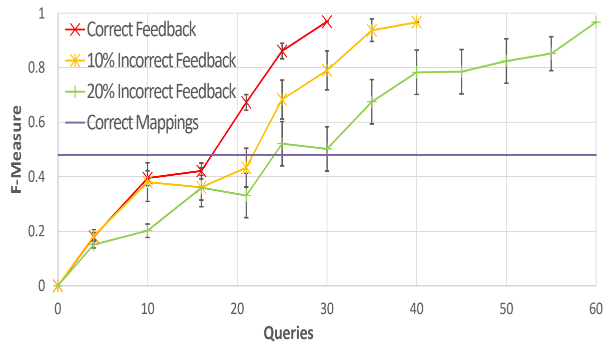
As mentioned earlier, to the best of my knowledge, the only other technique that uses feedback over query answer tuples is [18]. In that paper, query feedback is used to refine the mappings but not the mediated schema. To demonstrate that this is not sufficient, fully correct mappings are generated from the data sources to the automatically generated mediated schema by manually modifying the results of the automatic mapping. It is shown that the F-measure obtained using these correct mappings, which is the best that a technique that only refines the mappings such as [18] can obtain. It is important to emphasize that UFeed and [18] target different domains. While UFeed targets web tables that are usually large in numbers and have simple schemas, and therefore require generating a mediated schema as an interface to access them, as well as mappings (one-to-one) between each table and the mediated schema, the approach in [18] targets a small number of tables in a one source schema to one target schema fashion of mapping modification. The approach of [18] is used here as a comparison point to emphasize the importance of targeting *both* mediated schema and mappings in the refinement step of the pay-as-you-go approach to web data integration. UFeed will not perform well in the domain targeted by [18] due to the lack of support for complex mappings (1-to-n, n-to-1, or n-to-m).

Figure 5.11 shows the quality of the query answers (F-measure) for the three domains shown in Table 5.1 for holistic and probabilistic data integration. Since all the queries in the set of evaluation queries touch parts of the initial mediated schema that differ from the gold standard, the F-measure starts at a very low value. The figure shows that UFeed consistently reaches a high value of F-measure (greater than 0.9). As expected, UFeed is slower to converge when there is incorrect feedback, but the important point to note is that UFeed converges to the same (high) value of the F-measure even if up to 20% of the feedback is incorrect. The confidence intervals are narrow, showing that while the order of queries has some effect on the quality of the results, this effect is small.

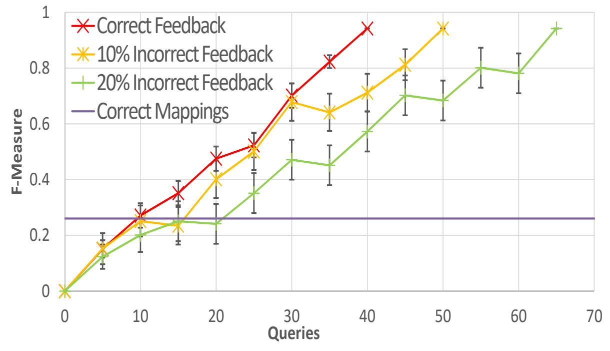
The figure shows that having correct mappings (the best for a technique such as [18]) is not sufficient to guarantee high quality answers (the horizontal line in the plots). The



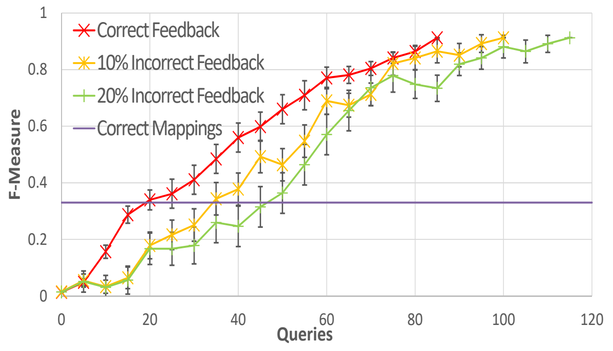
(a) Internet - Holistic



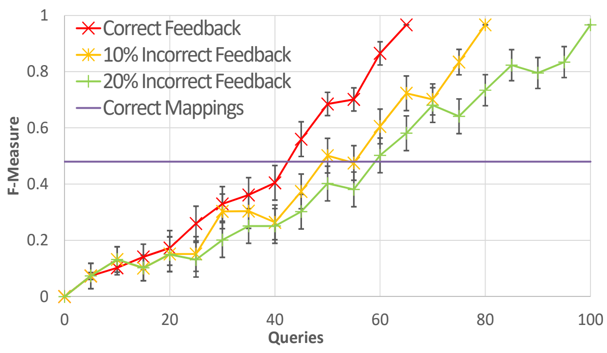
(b) Movies - Holistic



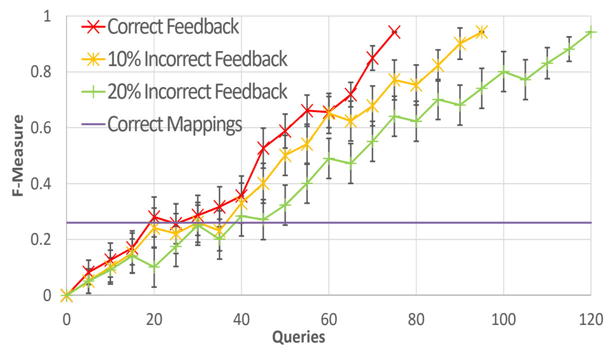
(c) World GDP - Holistic



(d) Internet - Probabilistic



(e) Movies - Probabilistic



(f) World GDP - Probabilistic

Figure 5.11: Quality of query answers.

automatically generated mappings can be refined to map to the correct mediated attribute, if it exists. However, there are typically mediated attributes that are either a group of source attributes representing different concepts and incorrectly grouped together (affecting precision), or an incomplete group that is missing relevant source attributes (affecting recall).

It is important to note that the user does not need to wait until the mediated schema is the same as the gold standard to receive benefit from UFeed. Whenever a UFeed operation is triggered, the answers to the query that triggered this operation are improved. Thus, there is immediate improvement to the answers that the user is currently interested in. When there is incorrect feedback, UFeed requires more operations (i.e., more queries) to converge, but still reaches a high value of F-measure.

5.6.3 Distance to the Gold Standard

Another question to ask is how close the current mediated schema is to the gold standard mediated schema. Since mediated schema generation can be viewed as a clustering problem (clustering source attributes into mediated attributes), the F-measure of clustering output [71, 120] is used to measure the distance between any mediated schema and the gold standard. This measure is computed as follows: Each mediated attribute represents a *cluster* of source attributes. A *contingency matrix* is constructed for all the clusters in the gold standard and the current mediated schema. Each cell (c_G, c_M) in the matrix counts the number of source attributes that exist in the two clusters, c_G in the gold standard and c_M in the current mediated schema. The distance between the current mediated schema and the gold standard is computed as follows:

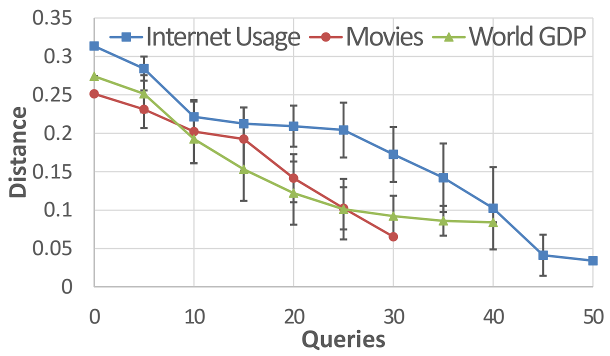
1. Precision, recall, and F-measure are computed for each cell (c_G, c_M), where $Precision$ $(c_G, c_M) = \frac{|(c_G, c_M)|}{|c_M|}$, $Recall(c_G, c_M) = \frac{|(c_G, c_M)|}{|c_G|}$, and $F(c_G, c_M) = \frac{2PR}{P+R}$.

2. The F-measure of each gold standard cluster is computed as the maximum value of all the F-measures for this cluster: $F(c_G) = \underset{|c_M|}{max} F(c_G, c_M)$.

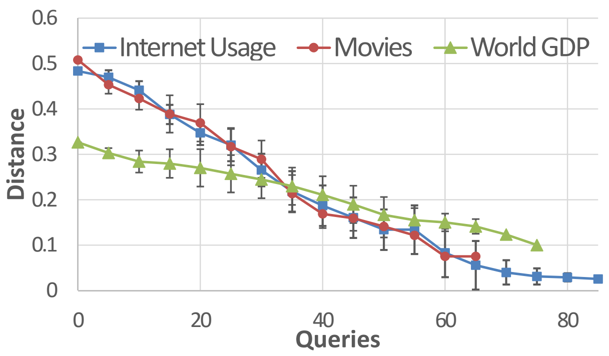
3. The F-measure of the clustering output is computed as the weighted average of the F-measures of all clusters of the gold standard: $F = \frac{\sum\limits_{c_G \in C} |c_G| F(c_G)}{\sum\limits_{c_G \in C} |c_G|}$.

4. Finally, the distance between the two mediated schemas is computed as $Distance = 1 - F$.

Figure 5.12 shows the distance between the current mediated schema and the gold standard after each query. The distance is shown for the three domains in the case where



(a) Holistic



(b) Probabilistic

Figure 5.12: Distance to the gold standard.

only correct feedback is used. For the probabilistic mediated schema, the mediated schema with the minimum distance to the gold standard is used. The figure shows that the distance steadily decreases as queries are issued and the UFeed operations are applied, which demonstrates the effectiveness and efficiency of UFeed.

5.7 Conclusion and Future Work

This chapter presented UFeed, a system that closes the loop of pay-as-you go relational data integration on the web by providing mechanisms to improve the quality of the mediated schema and mappings based on feedback received from the user. A key feature of UFeed is that it refines both the mediated schema and the mappings from the data sources to this schema. Another key feature is that the refinement relies solely on the queries issued by the user and feedback on the answers to these queries, without the need to directly manipulate the mediated schema or mappings. UFeed refinement is based on a set of well-defined operations that are proved to be complete. An experimental evaluation with real data shows that UFeed is effective at improving quality for holistic and probabilistic data integration.

UFeed opens up several directions for future work. One extension to UFeed is to handle more complex mappings (1-to-n, n-to-1, or n-to-m). Another extension is to handle more complex queries, specifically joins, which would require UFeed to rewrite queries over multiple tables. Another direction for future work is to discover integrity constraints in web tables (e.g., prmiary keys and foreign keys) and use them in refining the mediated schema and mappings. A longer term direction for futrue work is to explore the possibility of transforming web relational tables to RDF before integrating them with native RDF data sets. This will transform the problem of generating a mediated schema and mappings to one of inferring the equivalence of properties (originally attributes) and instances (originally tuples) and issuing queries that use entailments to find answers from the RDF graph.

Chapter 6

Conclusion and Future Work

6.1 Thesis Conclusion

Due to the recent advances in the field of information extraction, which have helped in automating the construction and extraction of structured data, the research community is presented with a significantly large number of heterogeneous structured data sets in different formats on the web. To unlock the full potential of these data sets, there needs to be a way of easily accessing them as one unified whole. This goal has been addressed by numerous works in the area of data integration. An important class of work focuses on automatic linking of of equivalent entities in heterogeneous RDF data sets. Another important class of work focuses on automatically creating a mediated schema for web relational tables through schema matching and mapping techniques. A shortcoming of these automatic data integration techniques is that their output can be incomplete and contain errors. Therefore, a subsequent step is needed in which the output of these techniques is refined over time as it is used. This refinement step is based on users interacting with the output of the data integration techniques and providing feedback. This feedback can then be used as the basis for refining the data integration output. For systems targeted towards the web, the users should be expected to be non-expert users.

This thesis focused on the interaction of non-expert users with the output of data integration systems, and refining this output based on feedback gathered during this interaction. The contributions of the thesis can be summarized as follows:

- Chapter 3 discussed incorporating user feedback over query answers on RDF data to improve the quality of *owl:sameAs* links. The chapter presented the ALEX system, analyzed the *owl:sameAs* links between pairs of RDF data sets from the linked open data cloud, discussed my approach of modeling this problem as a reinforcement learning problem, proved the soundness of this approach, discussed optimizations for faster convergence, showed the ALEX user interface, and presented an evaluation of ALEX, showing that it can efficiently learn how to explore the search space to find new correct links.

- Chapter 4 presented the architecture of Sapphire, how Sapphire initializes its cached data for a new endpoint through SPARQL queries, how the predictive user model suggests query completions, and ways to change the query to find better answers, the Sapphire user interface, and an evaluation of Sapphire based on a user study and a qualitative and quantitative comparison.

- Chapter 5 discussed the UFeed operations that are triggered by user feedback over query answers to directly refine the automatically created mediated schema and mappings, the proof of how the UFeed oprations could transform any automatically created mediated schema to a gold standard mediated schema that can be created by a data expert, the UFeed user interface, and an evaluation of UFeed.

To recap, the thesis incorporated non-expert users into different aspects of data integration. One aspect is facilitating the querying process over a large number of interconnected RDF data sets (Sapphire). The thesis also showed that giving feedback over answers to queries involving linked entities from different RDF data sets helps in improving the quality of links between these data sets by removing incorrect links and discovering new correct links (ALEX). Finally, in the relational model, the thesis showed that giving feedback over answers to queries involving relational tables helps in refining the mediated schema and mappings of these tables, and consequently improves the quality of query answers (UFeed).

6.2 Future Work

The contributions of this thesis present some directions for future work. An important direction for future work in ALEX is confirming the effectiveness of ALEX through a user study using real applications on the Linked Open Data cloud. Such a study with real users who may generate incorrect feedback should enable validating the robustness of ALEX beyond the experiments in this thesis.

As future work for UFeed, it is possible to support more complex schemas and queries. UFeed currently supports selection queries on single-table schemas, and this covers a large fraction of the use cases on the web. Nevertheless, it would be useful to extend UFeed to support data sources and mediated schemas that consist of multiple tables. This requires extending UFeed to support queries with joins, issuing these queries to the data sources and learning from feedback on their answers.

Looking beyond this thesis at the broad area of data integration, I see that there is an ever increasing need for integrating large numbers of data sets in different formats (e.g., RDF, relational tables, JSON, text, etc.). This thesis has focused on data integration on the web, but large scale data integration is also seen in the enterprise world in the context of *data lakes*, which are repositories of all kinds of data collected from the operation of the enterprise stored in its raw format. The philosophy of data lakes is to collect as much data as possible without burdening oneself with imposing structure on the data, and then to impose structure later, when the data is used (referred to as *schema on read*).

Large scale data integration on the web and in data lakes raises several challenges related to the topic of this thesis. One challenge is storing such a massive amount of heterogeneous data. Another is preprocessing the data for indexing and automatic data integration. For example, the automatic linking that is done prior to invoking ALEX and the indexing done by Sapphire may need to be completely different at this larger scale. A third challenge is identifying the data sets to query; we cannot rely on a federated RDF query processor as in ALEX and Sapphire since the data is not all RDF. Also, since the data is not RDF, composing queries needs to be revisited. Finally, user interaction, which is the major focus of this thesis, may need to be revisited to accommodate data that is much larger scale and much more heterogeneous. All of these challenges represent rich areas for future work.

References

[1] Google Fusion Tables. http://research.google.com/tables.

[2] RDF 1.1 concepts and abstract syntax. http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/.

[3] RDF schema 1.1, http://www.w3.org/TR/rdf-schema/.

[4] Schema-agnostic queries over large-schema databases. http://sites.google.com/site/eswcsaq2015/.

[5] SPARQL 1.1 query language. http://www.w3.org/TR/sparql11-query/.

[6] Turtle - terse RDF triple language. https://www.w3.org/TeamSubmission/turtle/.

[7] Andrejs Abele, John P. McCrae, Paul Buitelaar, Anja Jentzsch, and Richard Cyganiak. Linking open data cloud diagram. http://lod-cloud.net/. 2017.

[8] Ashraf Aboulnaga and Kareem El Gebaly. μbe: User guided source selection and schema mediation for internet scale data integration. In *Proceedings of the International Conference on Data Engineering (ICDE)*, 2007.

[9] Maribel Acosta, Amrapali Zaveri, Elena Simperl, Dimitris Kontokostas, Sören Auer, and Jens Lehmann. Crowdsourcing linked data quality assessment. In *Proceedings of the International Semantic Web Conference (ISWC)*. 2013.

[10] Sanjay Agrawal, Surajit Chaudhuri, and Gautam Das. DBXplorer: Enabling keyword search over relational databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2002.

[11] Bogdan Alexe, Laura Chiticariu, Renée J. Miller, and Wang-Chiew Tan. Muse: Mapping understanding and design by example. In *Proceedings of the International Conference on Data Engineering (ICDE)*, 2008.

[12] Bogdan Alexe, Balder ten Cate, Phokion G. Kolaitis, and Wang-Chiew Tan. Designing and refining schema mappings via data examples. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2011.

[13] Arvind Arasu, Christopher Ré, and Dan Suciu. Large-scale deduplication with constraints using dedupalog. In *Proceedings of the International Conference on Data Engineering (ICDE)*, 2009.

[14] Marcelo Arenas, Gonzalo I. Diaz, and Egor V. Kostylev. Reverse engineering SPARQL queries. In *Proceedings of the International World Wide Web Conference (WWW)*, 2016.

[15] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. DBpedia: A nucleus for a web of open data. In *Proceedings of the International Semantic Web Conference (ISWC)*. 2007.

[16] David Aumueller, Hong-Hai Do, Sabine Massmann, and Erhard Rahm. Schema and ontology matching with COMA++. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2005.

[17] Romain Beaumont, Brigitte Grau, and Anne-Laure Ligozat. SemGraphQA@ QALD5: LIMSI participation at QALD5@ CLEF. In *CLEF Working Notes Papers*, 2015.

[18] Khalid Belhajjame, Norman W. Paton, Suzanne M. Embury, Alvaro A. A. Fernandes, and Cornelia Hedeler. Incrementally improving dataspaces based on user feedback. *Information Systems*, 38(5), 2013.

[19] Zohra Bellahsene, Angela Bonifati, and Erhard Rahm. *Schema matching and mapping*. Springer, 2011.

[20] Sonia Bergamaschi, Elton Domnori, Francesco Guerra, Raquel Trillo Lado, and Yannis Velegrakis. Keyword search over relational databases: a metadata approach. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2011.

[21] Tim Berners-Lee. Linked data-design issues. http://www.w3.org/DesignIssues/LinkedData.html, 2006.

[22] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific American*, 2001.

[23] Philip A. Bernstein, Jayant Madhavan, and Erhard Rahm. Generic schema matching, ten years later. *Proceedings of the VLDB Endowment (PVLDB)*, 4(11), 2011.

[24] Gaurav Bhalotia, Arvind Hulgeri, Charuta Nakhe, Soumen Chakrabarti, and Shashank Sudarshan. Keyword searching and browsing in databases using BANKS. In *Proceedings of the International Conference on Data Engineering (ICDE)*, 2002.

[25] Indrajit Bhattacharya and Lise Getoor. Collective entity resolution in relational data. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 2007.

[26] Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked data-the story so far. *International Journal on Semantic Web and Information Systems*, 5(3), 2009.

[27] Christian Bizer, Tom Heath, Kingsley Idehen, and Tim Berners-Lee. Linked data on the web. In *Proceedings of the International World Wide Web Conference (WWW)*, 2008.

[28] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2008.

[29] Angela Bonifati, Giansalvatore Mecca, Alessandro Pappalardo, Salvatore Raunich, and Gianvito Summa. Schema mapping verification: The spicy way. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*, 2008.

[30] Elena Cabrio, Julien Cojan, Alessio Palmero Aprosio, Bernardo Magnini, Alberto Lavelli, and Fabien Gandon. QAKiS: an open domain QA system based on relational patterns. In *Proceedings of the International Semantic Web Conference (ISWC)*, 2012.

[31] Michael J. Cafarella, Alon Halevy, and Nodira Khoussainova. Data integration for the relational web. *Proceedings of the VLDB Endowment (PVLDB)*, 2(1), 2009.

[32] Michael J. Cafarella, Alon Halevy, Daisy Zhe Wang, Eugene Wu, and Yang Zhang. WebTables: exploring the power of tables on the web. *Proceedings of the VLDB Endowment (PVLDB)*, 1(1), 2008.

[33] Xiaoyong Chai, Ba-Quy Vuong, AnHai Doan, and Jeffrey F. Naughton. Efficiently incorporating user feedback into information extraction and integration programs. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2009.

[34] Gong Cheng, Danyun Xu, and Yuzhong Qu. Summarizing entity descriptions for effective and efficient human-centered entity linking. In *Proceedings of the International Conference on World Wide Web (WWW)*, 2015.

[35] Laura Chiticariu and Wang-Chiew Tan. Debugging schema mappings with routes. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, 2006.

[36] Eric Chu, Akanksha Baid, Xiaoyong Chai, AnHai Doan, and Jeffrey Naughton. Combining keyword search and forms for ad hoc querying of databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2009.

[37] Philipp Cimiano, Christina Unger, and John McCrae. Ontology-based interpretation of natural language. *Synthesis Lectures on Human Language Technologies*, 7(2), 2014.

[38] William Cohen, Pradeep Ravikumar, and Stephen Fienberg. A comparison of string metrics for matching names and records. In *KDD workshop on data cleaning and object consolidation*, volume 3, 2003.

[39] Thomas Cover and Peter Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1), 1967.

[40] Isabel F. Cruz, Flavio Palandri Antonelli, and Cosmin Stroe. Agreementmaker: efficient matching for large real-world schemas and ontologies. *Proceedings of the VLDB Endowment (PVLDB)*, 2(2), 2009.

[41] Wanyun Cui, Yanghua Xiao, Haixun Wang, Yangqiu Song, Seung-won Hwang, and Wei Wang. KBQA: Learning question answering over QA corpora and knowledge bases. *Proceedings of the VLDB Endowment (PVLDB)*, 10(5), 2017.

[42] Anish Das Sarma, Xin Dong, and Alon Halevy. Bootstrapping pay-as-you-go data integration systems. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2008.

[43] Anish Das Sarma, Xin Luna Dong, and Alon Halevy. Uncertainty in data integration. *Managing and Mining Uncertain Data*, 2009.

[44] Renaud Delbru, Nickolai Toupikov, Michele Catasta, Giovanni Tummarello, and Stefan Decker. Hierarchical link analysis for ranking web data. In *Proceedings of the European Semantic Web Conference (ESWC)*, 2010.

[45] Gianluca Demartini, Djellel Eddine Difallah, and Philippe Cudré-Mauroux. ZenCrowd: leveraging probabilistic reasoning and crowdsourcing techniques for large-scale entity linking. In *Proceedings of the International World Wide Web Conference (WWW)*, 2012.

[46] Elena Demidova, Xuan Zhou, and Wolfgang Nejdl. A probabilistic scheme for keyword-based incremental query construction. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 24(3), 2012.

[47] Gonzalo Diaz, Marcelo Arenas, and Michael Benedikt. SPARQLByE: Querying RDF data by example (demo). *Proceedings of the VLDB Endowment (PVLDB)*, 9(13), 2016.

[48] Kyriaki Dimitriadou, Olga Papaemmanouil, and Yanlei Diao. Explore-by-example: An automatic query steering framework for interactive data exploration. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2014.

[49] Hong-Hai Do and Erhard Rahm. COMA: a system for flexible combination of schema matching approaches. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, 2002.

[50] AnHai Doan, Pedro Domingos, and Alon Y. Halevy. Reconciling schemas of disparate data sources: A machine-learning approach. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2001.

[51] AnHai Doan and Robert McCann. Building data integration systems: A mass collaboration approach. In *Proceedings of the Workshop on Information Integration on the Web*, 2003.

[52] Xin Dong and Alon Halevy. Indexing dataspaces. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2007.

[53] Xin Luna Dong, Alon Halevy, and Cong Yu. Data integration with uncertainty. *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, 2007.

[54] Richard O. Duda, David G. Stork, and Peter E. Hart. *Pattern classification and scene analysis*. Wiley, 2nd edition, 2000.

[55] Julian Eberius, Maik Thiele, Katrin Braunschweig, and Wolfgang Lehner. Top-k entity augmentation using consistent set covering. In *Proceedings of the International Conference on Scientific and Statistical Database Management (SSDBM)*, 2015.

[56] Ahmed El-Roby and Ashraf Aboulnaga. ALEX: Automatic link exploration in linked data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2015.

[57] Ahmed El-Roby and Ashraf Aboulnaga. ALEX: Automatic link exploration in linked data (demo). In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*, 2016.

[58] Ahmed El-Roby and Ashraf Aboulnaga. UFeed: Refining web data integration based on user feedback. In *Proceedings of the ACM on Conference on Information and Knowledge Management (CIKM)*, 2017.

[59] Ahmed El-Roby, Khaled Ammar, Ashraf Aboulnaga, and Jimmy Lin. Sapphire: Querying RDF data made simple (demo). *Proceedings of the VLDB Endowment (PVLDB)*, 2016.

[60] Ahmed K. Elmagarmid, Panagiotis G. Ipeirotis, and Vassilios S Verykios. Duplicate record detection: A survey. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 2007.

[61] Hazem Elmeleegy, Jayant Madhavan, and Alon Halevy. Harvesting relational tables from lists on the web. *Proceedings of the VLDB Endowment (PVLDB)*, 2(1), 2009.

[62] Hazem Elmeleegy, Mourad Ouzzani, and Ahmed K. Elmagarmid. Usage-based schema matching. In *Proceedings of the International Conference on Data Engineering (ICDE)*, 2008.

[63] Oren Etzioni, Michael Cafarella, Doug Downey, Stanley Kok, Ana-Maria Popescu, Tal Shaked, Stephen Soderland, Daniel S. Weld, and Alexander Yates. Web-scale information extraction in KnowItAll (preliminary results). In *Proceedings of the International World Wide Web Conference (WWW)*, 2004.

[64] Anthony Fader, Luke Zettlemoyer, and Oren Etzioni. Open question answering over curated and extracted knowledge bases. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2014.

[65] Alfio Ferrara, Davide Lorusso, and Stefano Montanelli. Automatic identity recognition in the semantic web. In *Proceedings of the International Workshop on Identity and Reference on the Semantic Web (IRSW)*, 2008.

[66] Michael Franklin, Alon Halevy, and David Maier. From databases to dataspaces: a new abstraction for information management. *ACM SIGMOD Record*, 2005.

[67] George W. Furnas, Thomas K. Landauer, Louis M. Gomez, and Susan T. Dumais. The vocabulary problem in human-system communication. *Communications of the ACM*, 30(11), 1987.

[68] Chaitanya Gokhale, Sanjib Das, AnHai Doan, Jeffrey F. Naughton, Narasimhan Rampalli, Jude Shavlik, and Xiaojin Zhu. Corleone: Hands-off crowdsourcing for entity matching. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2014.

[69] Andrew V. Goldberg and Renato Fonseca F. Werneck. Computing point-to-point shortest paths from external memory. In *The Algorithm Engineering and Experiments (ALENEX) and Analytic Algorithmics and Combinatorics (ANALCO)*, 2005.

[70] Jorge Gracia, Mathieu d'Aquin, and Eduardo Mena. Large scale integration of senses for the semantic web. In *Proceedings of the International World Wide Web Conference (WWW)*, 2009.

[71] Maria Halkidi, Yannis Batistakis, and Michalis Vazirgiannis. Cluster validity methods: part I. *ACM Sigmod Record*, 31:40–45, 2002.

[72] Bin He and Kevin Chen-Chuan Chang. Statistical schema matching across web query interfaces. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2003.

[73] Bin He, Tao Tao, and Kevin Chen-Chuan Chang. Organizing structured web sources by query schemas: a clustering approach. In *Proceedings of the ACM International Conference on Information and Knowledge Management (CIKM)*, 2004.

[74] Hao He, Haixun Wang, Jun Yang, and Philip S. Yu. Blinks: ranked keyword searches on graphs. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2007.

[75] Vagelis Hristidis and Yannis Papakonstantinou. Discover: Keyword search in relational databases. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, 2002.

[76] Wei Hu, Jianfeng Chen, and Yuzhong Qu. A self-training approach for resolving object coreference on the semantic web. In *Proceedings of the International World Wide Web Conference (WWW)*, 2011.

[77] Frank K. Hwang, Dana S. Richards, and Pawel Winter. *The Steiner tree problem*, volume 53. Elsevier, 1992.

[78] Robert Isele and Christian Bizer. Active learning of expressive linkage rules using genetic programming. *Web Semantics: Science, Services and Agents on the World Wide Web*, 23, 2013.

[79] H. V. Jagadish, Adriane Chapman, Aaron Elkiss, Magesh Jayapandian, Yunyao Li, Arnab Nandi, and Cong Yu. Making database systems usable. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2007.

[80] Yves R. Jean-Mary, E. Patrick Shironoshita, and Mansur R. Kabuka. Ontology matching with semantic verification. *Web Semantics: Science, Services and Agents on the World Wide Web*, 7(3), 2009.

[81] Shawn R. Jeffery, Michael J. Franklin, and Alon Y. Halevy. Pay-as-you-go user feedback for dataspace systems. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2008.

[82] Ernesto Jiménez-Ruiz and Bernardo Cuenca Grau. LogMap: Logic-based and scalable ontology matching. In *Proceedings of the International Semantic Web Conference (ISWC)*, 2011.

[83] Jaewoo Kang and Jeffrey F. Naughton. On schema matching with opaque column names and data values. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2003.

[84] Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*. 1972.

[85] Esther Kaufmann and Abraham Bernstein. How useful are natural language interfaces to the semantic web for casual end-users? In *Proceedings of the International Semantic Web Conference (ISWC)*, 2007.

[86] Esther Kaufmann, Abraham Bernstein, and Renato Zumstein. Querix: A natural language interface to query ontologies based on clarification dialogs. In *Proceedings of the International Semantic Web Conference (ISWC)*, 2006.

[87] Christoph Kiefer, Abraham Bernstein, and Markus Stocker. The fundamentals of iSPARQL: A virtual triple approach for similarity-based semantic web tasks. In *Proceedings of the International Semantic Web Conference (ISWC)*, 2007.

[88] Krys J. Kochut and Maciej Janik. SPARQLeR: Extended SPARQL for semantic association discovery. In *Proceedings of the European Semantic Web Conference (ESWC)*, 2007.

[89] Jens Lehmann and Lorenz Bühmann. AutoSPARQL: Let users query your knowledge base. In *Proceedings of the European Semantic Web Conference (ESWC)*, 2011.

[90] Juanzi Li, Jie Tang, Yi Li, and Qiong Luo. RiMOM: A dynamic multistrategy ontology alignment framework. *IEEE Transactions on Knowledge and Data Engineering*, 21(8), 2009.

[91] Vanessa Lopez, Miriam Fernández, Enrico Motta, and Nico Stieler. PowerAqua: supporting users in querying and exploring the semantic web. *Semantic Web*, 3(3), 2011.

[92] Jayant Madhavan, Philip A. Bernstein, AnHai Doan, and Alon Halevy. Corpus-based schema matching. In *Proceedings of the International Conference on Data Engineering (ICDE)*, 2005.

[93] Jayant Madhavan, Philip A. Bernstein, and Erhard Rahm. Generic schema matching with cupid. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, 2001.

[94] Matteo Magnani, Nikos Rizopoulos, Peter Mc. Brien, and Danilo Montesi. Schema integration based on uncertain semantic mappings. In *Conceptual Modeling–ER*. 2005.

[95] Hatem A. Mahmoud and Ashraf Aboulnaga. Schema clustering and retrieval for multi-domain pay-as-you-go data integration systems. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2010.

[96] Essam Mansour, Amin Allam, Spiros Skiadopoulos, and Panos Kalnis. ERA: Efficient serial and parallel suffix tree construction for very long strings. *Proceedings of the VLDB Endowment (PVLDB)*, 5(1), 2011.

[97] Essam Mansour, Ahmed El-Roby, Panos Kalnis, Aron Ahmadia, and Ashraf Aboulnaga. RACE: A scalable and elastic parallel system for discovering repeats in very long sequences. *Proceedings of the VLDB Endowment (PVLDB)*, 2013.

[98] Robert McCann, Warren Shen, and AnHai Doan. Matching schemas in online communities: A web 2.0 approach. In *Proceedings of the International Conference on Data Engineering (ICDE)*, 2008.

[99] Renée J. Miller, Yannis E. Ioannidis, and Raghu Ramakrishnan. The use of information capacity in schema integration and translation. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, 1993.

[100] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of Machine Learning*. The MIT Press, 2012.

[101] Davide Mottin, Matteo Lissandrini, Yannis Velegrakis, and Themis Palpanas. Exemplar queries: Give me an example of what you need. *Proceedings of the VLDB Endowment (PVLDB)*, 7(5), 2014.

[102] Arnab Nandi and Philip A. Bernstein. HAMSTER: using search clicklogs for schema and taxonomy matching. *Proceedings of the VLDB Endowment (PVLDB)*, 2009.

[103] Axel-Cyrille Ngonga Ngomo and Sören Auer. Limes-a time-efficient approach for large-scale link discovery on the web of data. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2011.

[104] Khai Nguyen, Ryutaro Ichise, and Bac Le. Interlinking linked data sources using a domain-independent system. In *The Joint International Semantic Technology Conference*, 2012.

[105] Jan Noessner, Mathias Niepert, Christian Meilicke, and Heiner Stuckenschmidt. Leveraging terminological structure for object reconciliation. In *The Semantic Web: Research and Applications*. 2010.

[106] Lucian Popa, Yannis Velegrakis, Mauricio A. Hernández, Renée J. Miller, and Ronald Fagin. Translating web data. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, 2002.

[107] Rachel Pottinger and Philip A. Bernstein. Creating a mediated schema based on initial correspondences. *IEEE Data Engineering Bulletin*, 2002.

[108] Li Qian, Michael J. Cafarella, and H. V. Jagadish. Sample-driven schema mapping. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2012.

[109] Bastian Quilitz and Ulf Leser. Querying distributed RDF data sources with SPARQL. In *Proceedings of the European Semantic Web Conference (ESWC)*, 2008.

[110] Erhard Rahm and Philip A. Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal*, 10(4), 2001.

[111] Stefan Ruseti, Alexandru Mirea, Traian Rebedea, and Stefan Trausan-Matu. Qanswer-enhanced entity matching for question answering over linked data. In *CLEF Working Notes Papers*, 2015.

[112] Fatiha Saıs, Nathalie Pernelle, and Marie-Christine Rousset. L2R: a logical method for reference reconciliation. In *Proceedings of the Association for the Advancement of Artificial Intelligence Conference (AAAI)*, 2007.

[113] Fatiha Saıs, Nathalie Pernelle, and Marie-Christine Rousset. Combining a logical and a numerical method for data reconciliation. *Journal on Data Semantics*, 12(12), 2009.

[114] Khalid Saleem, Zohra Bellahsene, and Ela Hunt. Porsche: Performance oriented schema mediation. *Information Systems*, 33(7), 2008.

[115] Mayssam Sayyadian, Hieu LeKhac, AnHai Doan, and Luis Gravano. Efficient keyword search across heterogeneous relational databases. In *Proceedings of the International Conference on Data Engineering (ICDE)*, 2007.

[116] Andreas Schwarte, Peter Haase, Katja Hose, Ralf Schenkel, and Michael Schmidt. FedX: Optimization techniques for federated query processing on linked data. In *Proceedings of the International Semantic Web Conference (ISWC)*. 2011.

[117] Len Seligman, Peter Mork, Alon Halevy, Ken Smith, Michael J. Carey, Kuang Chen, Chris Wolf, Jayant Madhavan, Akshay Kannan, and Doug Burdick. OpenII: an open source information integration toolkit. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2010.

[118] Satinder P. Singh and Richard S. Sutton. Reinforcement learning with replacing eligibility traces. *Machine Learning*, 22(1-3), 1996.

[119] Dezhao Song and Jeff Heflin. Domain-independent entity coreference for linking ontology instances. *Journal of Data and Information Quality (JDIQ)*, 4(2), 2013.

[120] Michael Steinbach, George Karypis, and Vipin Kumar. A comparison of document clustering techniques. In *Proceedings of the KDD Workshop on Text Mining*, 2000.

[121] Weifeng Su, Jiying Wang, and Frederick Lochovsky. Holistic schema matching for web query interfaces. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*, 2006.

[122] Fabian M. Suchanek, Serge Abiteboul, and Pierre Senellart. PARIS: probabilistic alignment of relations, instances, and schema. *Proceedings of the VLDB Endowment (PVLDB)*, 5(3), 2011.

[123] Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning.* MIT Press, 1998.

[124] Partha Pratim Talukdar, Zachary G. Ives, and Fernando Pereira. Automatically incorporating new sources in keyword search-based data integration. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2010.

[125] Partha Pratim Talukdar, Marie Jacob, Muhammad Salman Mehmood, Koby Crammer, Zachary G. Ives, Fernando Pereira, and Sudipto Guha. Learning to create data-integrating queries. *Proceedings of the VLDB Endowment (PVLDB)*, 1(1), 2008.

[126] Giovanni Tummarello, Richard Cyganiak, Michele Catasta, Szymon Danielczyk, Renaud Delbru, and Stefan Decker. Sig.ma: Live views on the web of data. *Proceedings of the International World Wide Web Conference (WWW)*, 2010.

[127] Giovanni Tummarello, Renaud Delbru, and Eyal Oren. Sindice.com: Weaving the open linked data. In *Proceedings of the International Semantic Web Conference and 2nd Asian Semantic Web Conference (ISWC/ASWC)*, 2007.

[128] E. Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14(3), 1995.

[129] Christina Unger, Lorenz Bühmann, Jens Lehmann, Axel-Cyrille Ngonga Ngomo, Daniel Gerber, and Philipp Cimiano. Template-based question answering over RDF data. In *Proceedings of the International World Wide Web Conference (WWW)*, 2012.

[130] Christina Unger, Philipp Cimiano, Vanessa Lopez, and Enrico Motta. Question answering over linked data (QALD-1). In *Workshop on Question Answering Over Linked Data (QALD-1)*, 2011.

[131] Christina Unger, Corina Forascu, Vanessa Lopez, Axel-Cyrille Ngonga Ngomo, Elena Cabrio, Philipp Cimiano, and Sebastian Walter. Question answering over linked data (QALD-4). In *CLEF Working Notes Papers*, 2014.

[132] Christina Unger, Corina Forascu, Vanessa Lopez, Axel-Cyrille Ngonga Ngomo, Elena Cabrio, Philipp Cimiano, and Sebastian Walter. Question answering over linked data (QALD-5). In *CLEF Working Notes Papers*, 2015.

[133] Christina Unger, John McCrae, Sebastian Walter, Sara Winter, and Philipp Cimiano. A lemon lexicon for DBpedia. In *NLP and DBpedia Workshop*, 2013.

[134] Julius Volz, Christian Bizer, Martin Gaedke, and Georgi Kobilarov. Discovering and maintaining links on the web of data. In *Proceedings of the International Semantic Web Conference (ISWC)*. 2009.

[135] Julius Volz, Christian Bizer, Martin Gaedke, and Georgi Kobilarov. Silk-A link discovery framework for the web of data. In *Proceedings of the Workshop on Linked Data on the Web (LDOW)*, 2009.

[136] Haofen Wang, Qiaoling Liu, Thomas Penin, Linyun Fu, Lei Zhang, Thanh Tran, Yong Yu, and Yue Pan. Semplore: A scalable IR approach to search the web of data. *Web Semantics: Science, Services and Agents on the World Wide Web*, 7(3), 2009.

[137] Jiying Wang, Ji-Rong Wen, Fred Lochovsky, and Wei-Ying Ma. Instance-based schema matching for web databases by domain-specific query probing. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, 2004.

[138] Zhichun Wang, Xiao Zhang, Lei Hou, Yue Zhao, Juanzi Li, Yu Qi, and Jie Tang. RiMOM results for OAEI 2010. In *Proceedings of the International Conference on Ontology Matching*, 2010.

[139] Peter Weiner. Linear pattern matching algorithms. In *Annual Symposium on Switching and Automata Theory*, 1973.

[140] Steven Euijong Whang, Peter Lofgren, and Hector Garcia-Molina. Question selection for crowd entity resolution. *Proceedings of the VLDB Endowment (PVLDB)*, 6(6), 2013.

[141] Kun Xu, Sheng Zhang, Yansong Feng, and Dongyan Zhao. Answering natural language questions via phrasal semantic parsing. In *CLEF Working Notes Papers*, 2014.

[142] Mohamed Yahya, Klaus Berberich, Shady Elbassuoni, and Gerhard Weikum. Robust question answering over the web of linked data. In *Proceedings of the ACM International Conference on Information and Knowledge Management (CIKM)*, 2013.

[143] Mohamed Yakout, Kris Ganjam, Kaushik Chakrabarti, and Surajit Chaudhuri. Infogather: entity augmentation and attribute discovery by holistic matching with web tables. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2012.

[144] Zhepeng Yan, Nan Zheng, Zachary G. Ives, Partha Pratim Talukdar, and Cong Yu. Actively soliciting feedback for query answers in keyword search-based data integration. *Proceedings of the VLDB Endowment (PVLDB)*, 6(3), 2013.

[145] Shengqi Yang, Yinghui Wu, Huan Sun, and Xifeng Yan. Schemaless and structureless graph querying. *Proceedings of the VLDB Endowment (PVLDB)*, 7(7), 2014.

[146] Cong Yu and Lucian Popa. Semantic adaptation of schema mappings when schemas evolve. In *VLDBProceedings of the International Conference on Very Large Databases (VLDB)*, 2005.

[147] Jeffrey Xu Yu, Lu Qin, and Lijun Chang. Keyword search in relational databases: A survey. *IEEE Data Engineering Bulletin*, 33(1), 2010.

[148] Gideon Zenz, Xuan Zhou, Enrico Minack, Wolf Siberski, and Wolfgang Nejdl. From keywords to semantic queries-Incremental query construction on the semantic web. *Web Semantics: Science, Services and Agents on the World Wide Web*, 7(3), 2009.

[149] Chen Jason Zhang, Lei Chen, H.V. Jagadish, and Chen Caleb Cao. Reducing uncertainty of schema matching via crowdsourcing. *Proceedings of the VLDB Endowment (PVLDB)*, 6(9), 2013.

[150] Weiguo Zheng, Lei Zou, Wei Peng, Xifeng Yan, Shaoxu Song, and Dongyan Zhao. Semantic SPARQL similarity search over RDF knowledge graphs. *Proceedings of the VLDB Endowment (PVLDB)*, 9(11), 2016.

[151] Qi Zhou, Chong Wang, Miao Xiong, Haofen Wang, and Yong Yu. Spark: Adapting keyword query to semantic search. In *Proceedings of the International Semantic Web Conference (ISWC)*, 2007.

[152] Erkang Zhu, Ken Q. Pu, Fatemeh Nargesian, and Renée J Miller. Interactive navigation of open data linkages. *Proceedings of the VLDB Endowment (PVLDB)*, 10(12), 2017.

APPENDICES

Appendix A

Similarity Function used in ALEX

In ALEX, a generic similarity function that depends on the data types of the compared values is used. The data types supported by the similarity function are strings, dates, and numbers (integers, floats, doubles, etc.). If the data types encountered are not supported, they are treated as strings. The following sections show how the similarity score is calculated depending on the data type.

A.1 Similarity Scores for Strings

To compare strings, the Jaro-Winkler similarity score [38] is used. This similarity function is based on the number of transpositions between two input strings while favoring strings matching from the beginning.

First, the Jaro similarity is explained, then I show how matches from the beginning of the matched strings are favored in Jaro-Winkler similarity. The Jaro similarity sim_{jaro} between two input strings s_1 and s_2 is defined as:

$$sim_{jaro}(s_1, s_2) = \begin{cases} 0 & \text{if} \quad m = 0 \\ \frac{1}{3}\left(\frac{m}{s_1} + \frac{m}{s_2} + \frac{m-t}{m}\right) & \text{otherwise} \end{cases} \tag{A.1}$$

where m is the number of character matches and t is half the number of transpositions. The number of matches is counted only if the distance between the matching characters is within $\left\lfloor \frac{max(|s_1|,|s_2|)}{2} \right\rfloor$. The transpositions are the matches of characters but in different positions between the two strings.

The Jaro-Winkler similarity differs in that it adds a prefix scale p that favors strings matching from the beginning until a specified prefix length l. Formally, the Jaro-Winkler distance is defined by:

$$sim_{jaro-winkler}(s_1, s_2) = sim_{jaro}(s_1, s_2) + lp(1 - sim_{jaro}(s_1, s_2)) \tag{A.2}$$

Where l is the length of common prefix between s_1 and s_2 up to 4 characters, and p is the prefix scale. In this thesis, the value $p = 0.1$ is used.

A.2 Similarity Score for Dates

The date similarity score is based on the number of days between the two compared dates, which is used as an absolute difference value that is normalized later to obtain a value between 0 and 1. The normalization step is based on the minimum and maximum number of days encountered as a difference between two dates in the step of generating the search space of feature sets. Formally, the similarity function is defined as:

$$Similarity(d_1, d_2) = 1 - \frac{DiffDays(d_1, d_2) - MIN_{days}}{MAX_{days} - MIN_{days}} \tag{A.3}$$

Where $DiffDays$ is the function that returns the number of days between two input dates, MIN_{days} is the minimum number of days encountered as a difference between two dates in the feature sets generation step, and MAX_{days} is the maximum number of days encountered. Note that these values of MIN_{days} and MAX_{days} are computed per feature. For example, the MIN_{days} and MAX_{days} values for a feature *(date of birth, birth date)* are different than the minimum and maximum values for another feature *(date published, established on)*.

A.3 Similarity Score for Numbers

In ALEX, numbers are compared in a similar fashion to the comparison of dates. The similarity function is defined as:

$$Similarity(n_1, n_2) = 1 - \frac{ABS(n_1, n_2) - MIN_N}{MAX_N - MIN_N} \tag{A.4}$$

Where ABS is a function that returns the absolute difference between the two numbers, MIN_N is the minimum absolute difference between any two numbers, and MAX_N is the maximum absolute difference. Again, these minimum and maximum values are per feature.

Appendix B

Distinctive Features Found in the Analysis of Links

This appendix lists the distinctive features identified in the analysis of *owl:sameAs* links in Section 3.4. The features are listed for every pair of data sets used in the analysis.

1. DBpedia - Drugbank:

   (a) `http://xmlns.com/foaf/0.1/depiction` –
   `http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugbank/synonym`

   (b) `http://www.w3.org/2000/01/rdf-schema#label` –
   `http://www4.wiwiss.fu-berlin.de/diseasome/resource/diseasome/name`

   (c) `http://dbpedia.org/property/drugbank` –
   `http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugbank/primaryAccessionNo`

   (d) `http://dbpedia.org/ontology/drugbank` –
   `http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugbank/limsDrugId`

   (e) `http://dbpedia.org/property/drugbank` –
   `http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugbank/limsDrugId`

   (f) `http://dbpedia.org/property/inchikey` –
   `http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugbank/inchiKey`

   (g) `http://dbpedia.org/ontology/drugbank` –
   `http://xmlns.com/foaf/0.1/page`

(h) http://dbpedia.org/ontology/drugbank –
http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugbank/primaryAccessionNo

(i) http://xmlns.com/foaf/0.1/depiction –
http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugbank/genericName

(j) http://dbpedia.org/ontology/thumbnail –
http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugbank/synonym

(k) http://dbpedia.org/property/stdinchikey –
http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugbank/inchiKey

(l) http://dbpedia.org/ontology/atcSuffix –
http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugbank/atcCode

(m) http://dbpedia.org/property/drugbank –
http://xmlns.com/foaf/0.1/page

(n) http://dbpedia.org/property/atcSuffix –
http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugbank/atcCode

(o) http://dbpedia.org/property/name –
http://www4.wiwiss.fu-berlin.de/diseasome/resource/diseasome/name

(p) http://xmlns.com/foaf/0.1/depiction –
http://xmlns.com/foaf/0.1/page

(q) http://dbpedia.org/ontology/thumbnail –
http://xmlns.com/foaf/0.1/page

(r) http://xmlns.com/foaf/0.1/name –
http://www.w3.org/2000/01/rdf-schema#label

(s) http://xmlns.com/foaf/0.1/name –
http://www4.wiwiss.fu-berlin.de/diseasome/resource/diseasome/name

(t) http://www.w3.org/2000/01/rdf-schema#label –
http://xmlns.com/foaf/0.1/page

(u) http://dbpedia.org/ontology/atcPrefix –
http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugbank/atcCode

(v) http://www.w3.org/2000/01/rdf-schema#label –
http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugbank/genericName

(w) http://dbpedia.org/property/atcPrefix –
http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugbank/atcCode

   (x) http://dbpedia.org/ontology/thumbnail –
   http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugbank/genericName

   (y) http://dbpedia.org/ontology/iupacName –
   http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugbank/chemicalIupacName

   (z) http://www.w3.org/2000/01/rdf-schema#label –
   http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugbank/synonym

2. DBpedia - Lexvo:

   (a) http://dbpedia.org/property/iso –
   http://lexvo.org/ontology#iso639P3PCode

   (b) http://xmlns.com/foaf/0.1/name –
   http://lexvo.org/ontology#label

   (c) http://www.w3.org/2000/01/rdf-schema#label –
   http://www.w3.org/2008/05/skos#prefLabel

   (d) http://www.w3.org/2000/01/rdf-schema#label –
   http://lexvo.org/ontology#label

   (e) http://dbpedia.org/property/glottorefname –
   http://lexvo.org/ontology#label

   (f) http://dbpedia.org/property/glottorefname –
   http://www.w3.org/2008/05/skos#prefLabel

   (g) http://www.w3.org/2000/01/rdf-schema#label –
   http://www.w3.org/2000/01/rdf-schema#label

   (h) http://xmlns.com/foaf/0.1/name –
   http://www.w3.org/2008/05/skos#prefLabel

   (i) http://dbpedia.org/property/glotto –
   http://lexvo.org/ontology#nearlySameAs

   (j) http://dbpedia.org/property/name –
   http://www.w3.org/2008/05/skos#prefLabel

   (k) http://xmlns.com/foaf/0.1/name –
   http://www.w3.org/2000/01/rdf-schema#label

   (l) http://dbpedia.org/property/name –
   http://lexvo.org/ontology#label

(m) http://dbpedia.org/ontology/iso6393Code –
http://lexvo.org/ontology#iso639P3PCode

(n) http://dbpedia.org/property/glottorefname –
http://www.w3.org/2000/01/rdf-schema#label

(o) http://dbpedia.org/property/ethnicity –
http://www.w3.org/2000/01/rdf-schema#label

(p) http://dbpedia.org/property/name –
http://www.w3.org/2000/01/rdf-schema#label

3. DBpedia - OpenCyc:

(a) http://www.w3.org/2000/01/rdf-schema#label –
http://sw.cyc.com/CycAnnotations_v1#label

(b) http://www.w3.org/2000/01/rdf-schema#label –
http://sw.opencyc.org/concept/Mx4rwLSVCpwpEbGdrcN5Y29ycA

(c) http://dbpedia.org/property/name –
http://sw.cyc.com/CycAnnotations_v1#label

(d) http://xmlns.com/foaf/0.1/name –
http://sw.opencyc.org/concept/Mx4rwLSVCpwpEbGdrcN5Y29ycA

(e) http://xmlns.com/foaf/0.1/name –
http://www.w3.org/2000/01/rdf-schema#label

(f) http://xmlns.com/foaf/0.1/name –
http://sw.opencyc.org/concept/wikipediaArticleURL

(g) http://xmlns.com/foaf/0.1/name –
http://sw.cyc.com/CycAnnotations_v1#label

(h) http://www.w3.org/2000/01/rdf-schema#label –
http://www.w3.org/2000/01/rdf-schema#label

(i) http://dbpedia.org/property/name –
http://sw.opencyc.org/concept/Mx4rwLSVCpwpEbGdrcN5Y29ycA

(j) http://dbpedia.org/property/name –
http://www.w3.org/2000/01/rdf-schema#label

(k) http://www.w3.org/2000/01/rdf-schema#label –
http://sw.opencyc.org/concept/wikipediaArticleURL

(l) `http://dbpedia.org/property/name` – `http://sw.opencyc.org/concept/wikipediaArticleURL`

4. DBpedia - UK Learning:
    (a) `http://dbpedia.org/property/name` – `http://xmlns.com/foaf/0.1/isPrimaryTopicOf`
    (b) `http://www.w3.org/2000/01/rdf-schema#label` – `http://www.w3.org/2000/01/rdf-schema#label`
    (c) `http://xmlns.com/foaf/0.1/name` – `http://www.w3.org/2000/01/rdf-schema#label`
    (d) `http://www.w3.org/2000/01/rdf-schema#label` – `http://xmlns.com/foaf/0.1/isPrimaryTopicOf`
    (e) `http://xmlns.com/foaf/0.1/name` – `http://xmlns.com/foaf/0.1/isPrimaryTopicOf`
    (f) `http://dbpedia.org/property/name` – `http://www.w3.org/2000/01/rdf-schema#label`

5. Lexvo - Geonames:
    (a) `http://www.w3.org/2000/01/rdf-schema#label` – `http://www.geonames.org/ontology#wikipediaArticle`
    (b) `http://www.w3.org/2000/01/rdf-schema#label` – `http://www.geonames.org/ontology#inCountry`
    (c) `http://lexvo.org/ontology#label` – `http://www.geonames.org/ontology#locationMap`
    (d) `http://lexvo.org/ontology#label` – `http://www.geonames.org/ontology#name`
    (e) `http://www.w3.org/2000/01/rdf-schema#label` – `http://www.geonames.org/ontology#alternateName`
    (f) `http://lexvo.org/ontology#label` – `http://www.geonames.org/ontology#wikipediaArticle`
    (g) `http://lexvo.org/ontology#label` – `http://www.geonames.org/ontology#alternateName`
    (h) `http://lexvo.org/ontology#label` – `http://www.geonames.org/ontology#inCountry`

(i) `http://lexvo.org/ontology#label` –
`http://www.w3.org/2000/01/rdf-schema#seeAlso`

(j) `http://www.w3.org/2000/01/rdf-schema#label` –
`http://www.geonames.org/ontology#countryCode`

(k) `http://www.w3.org/2000/01/rdf-schema#label` –
`http://www.w3.org/2000/01/rdf-schema#seeAlso`

(l) `http://www.w3.org/2000/01/rdf-schema#label` –
`http://www.geonames.org/ontology#name`

(m) `http://lexvo.org/ontology#label` –
`http://www.geonames.org/ontology#countryCode`

6. Linkedmdb - DBpedia:

(a) `http://dbpedia.org/property/hasPhotoCollection` –
`http://dbpedia.org/property/name`

(b) `http://dbpedia.org/property/hasPhotoCollection` –
`http://www.w3.org/2000/01/rdf-schema#label`

(c) `http://www.w3.org/2000/01/rdf-schema#label` –
`http://www.w3.org/2000/01/rdf-schema#label`

(d) `http://dbpedia.org/property/hasPhotoCollection` –
`http://xmlns.com/foaf/0.1/name`

(e) `http://dbpedia.org/property/hasPhotoCollection` –
`http://dbpedia.org/ontology/thumbnail`

(f) `http://dbpedia.org/property/hasPhotoCollection` –
`http://xmlns.com/foaf/0.1/depiction`

7. Linkedmdb - Geonames:

(a) `http://data.linkedmdb.org/resource/movie/country_name` –
`http://www.geonames.org/ontology#countryCode`

(b) `http://data.linkedmdb.org/resource/movie/country_name` –
`http://www.geonames.org/ontology#name`

(c) `http://data.linkedmdb.org/resource/movie/country_fips_code` –
`http://www.w3.org/2000/01/rdf-schema#seeAlso`

(d) `http://data.linkedmdb.org/resource/movie/country_name` –
`http://www.geonames.org/ontology#inCountry`

(e) http://www.w3.org/2000/01/rdf-schema#label –
http://www.geonames.org/ontology#alternateName

(f) http://data.linkedmdb.org/resource/movie/country_iso_alpha2 –
http://www.w3.org/2000/01/rdf-schema#seeAlso

(g) http://data.linkedmdb.org/resource/movie/country_fips_code –
http://www.geonames.org/ontology#officialName

(h) http://data.linkedmdb.org/resource/movie/country_name –
http://www.geonames.org/ontology#alternateName

(i) http://data.linkedmdb.org/resource/movie/country_name –
http://www.w3.org/2000/01/rdf-schema#seeAlso

(j) http://data.linkedmdb.org/resource/movie/country_iso_alpha2 –
http://www.geonames.org/ontology#name

(k) http://www.w3.org/2000/01/rdf-schema#label –
http://www.geonames.org/ontology#inCountry

(l) http://data.linkedmdb.org/resource/movie/country_iso_alpha2 –
http://www.geonames.org/ontology#alternateName

(m) http://data.linkedmdb.org/resource/movie/country_iso_alpha3 –
http://www.geonames.org/ontology#name

(n) http://data.linkedmdb.org/resource/movie/country_iso_alpha3 –
http://www.geonames.org/ontology#countryCode

(o) http://data.linkedmdb.org/resource/movie/country_iso_alpha2 –
http://www.geonames.org/ontology#officialName

(p) http://www.w3.org/2000/01/rdf-schema#label –
http://www.geonames.org/ontology#officialName

(q) http://www.w3.org/2000/01/rdf-schema#label –
http://www.geonames.org/ontology#wikipediaArticle

(r) http://data.linkedmdb.org/resource/movie/country_iso_alpha3 –
http://www.geonames.org/ontology#officialName

(s) http://data.linkedmdb.org/resource/movie/country_fips_code –
http://www.geonames.org/ontology#alternateName

(t) http://www.w3.org/2000/01/rdf-schema#label –
http://www.geonames.org/ontology#countryCode

(u) http://data.linkedmdb.org/resource/movie/country_iso_alpha3 – http://www.geonames.org/ontology#alternateName

(v) http://www.w3.org/2000/01/rdf-schema#label – http://www.geonames.org/ontology#name

(w) http://data.linkedmdb.org/resource/movie/country_iso_alpha3 – http://www.w3.org/2000/01/rdf-schema#seeAlso

(x) http://data.linkedmdb.org/resource/movie/country_name – http://www.geonames.org/ontology#officialName

(y) http://data.linkedmdb.org/resource/movie/country_iso_alpha2 – http://www.geonames.org/ontology#countryCode

(z) http://data.linkedmdb.org/resource/movie/country_name – http://www.geonames.org/ontology#wikipediaArticle

8. NYTimes - DBpedia:

   (a) http://www.w3.org/2004/02/skos/core#prefLabel – http://xmlns.com/foaf/0.1/name

   (b) http://www.w3.org/2004/02/skos/core#prefLabel – http://www.w3.org/2000/01/rdf-schema#label

   (c) http://www.w3.org/2004/02/skos/core#prefLabel – http://dbpedia.org/property/name

9. NYTimes - Geonames:

   (a) http://www.w3.org/2003/01/geo/wgs84_pos#lat – http://www.w3.org/2003/01/geo/wgs84_pos#lat

   (b) http://www.w3.org/2004/02/skos/core#prefLabel – http://www.geonames.org/ontology#name

   (c) http://www.w3.org/2004/02/skos/core#prefLabel – http://www.geonames.org/ontology#alternateName

   (d) http://www.w3.org/2003/01/geo/wgs84_pos#long – http://www.w3.org/2003/01/geo/wgs84_pos#long

   (e) http://www.w3.org/2004/02/skos/core#prefLabel – http://www.geonames.org/ontology#officialName

   (f) http://www.w3.org/2004/02/skos/core#prefLabel – http://www.geonames.org/ontology#wikipediaArticle

   (g) http://www.w3.org/2004/02/skos/core#prefLabel –
   http://www.w3.org/2000/01/rdf-schema#seeAlso

   (h) http://www.w3.org/2004/02/skos/core#prefLabel –
   http://www.geonames.org/ontology#locationMap

10. OpenCyc - Umbel:

   (a) http://sw.cyc.com/CycAnnotations_v1#label –
   http://www.w3.org/2004/02/skos/core#prefLabel

   (b) http://sw.cyc.com/CycAnnotations_v1#label –
   http://umbel.org/umbel#isRelatedTo

   (c) http://www.w3.org/2000/01/rdf-schema#label –
   http://umbel.org/umbel#isRelatedTo

   (d) http://www.w3.org/2000/01/rdf-schema#label –
   http://www.w3.org/2004/02/skos/core#altLabel

   (e) http://sw.opencyc.org/concept/Mx4rwLSVCpwpEbGdrcN5Y29ycA –
   http://umbel.org/umbel#isRelatedTo

   (f) http://www.w3.org/2000/01/rdf-schema#label –
   http://www.w3.org/2004/02/skos/core#prefLabel

   (g) http://sw.opencyc.org/concept/Mx4rwLSVCpwpEbGdrcN5Y29ycA –
   http://www.w3.org/2004/02/skos/core#prefLabel

   (h) http://sw.cyc.com/CycAnnotations_v1#label –
   http://www.w3.org/2004/02/skos/core#altLabel

   (i) http://sw.opencyc.org/concept/Mx4rwLSVCpwpEbGdrcN5Y29ycA –
   http://www.w3.org/2004/02/skos/core#altLabel

Appendix C

Initialization of Sapphire

This appendix presents the SPARQL queries used in initializing Sapphire. SPARQL queries are typically provided with limited resources by the remote endpoints. A long-running query that is expected to consume a lot of resources may be rejected by the remote endpoint. If the query is accepted, it will likely time out. Therefore, the initialization queries of Sapphire are broken down into multiple queries that are less resource-intensive and therefore less likely to time out. These queries are as follows.

1. Finding predicates sorted by their frequency (not a resource-intensive query):

```
Q1) SELECT DISTINCT ?p (COUNT(*) AS ?frequency)
WHERE {
?s ?p ?o
}
GROUP BY ?p
ORDER BY DESC(?frequency)
```

2. Finding literals and most significant literals: The queries used to find literals need to be carefully structured to minimize their execution time and the chances of timing out. The key to achieving this goal is increasing the selectivity of the query. The focus in Sapphire is on two common characteristics of RDF data that are relevant to Sapphire: 1. Entities are associated with RDF types or schema classes. 2. Literals of interest in Sapphire are associated with a limited set of predicates.

Some data sets are well-structured and have a hierarchy of RDF schema classes, with each entity in the data set belonging to a class. This is the case for most of the data sets that encountered on the LOD cloud. This characteristic can be exploited by restricting the retrieval of literals to part of the class hierarchy.

The following query finds all classes and their subclasses in a data set:

```
Q2) PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
SELECT DISTINCT ?class ?subclass
WHERE{
?class a owl:Class.
?class rdfs:subClassOf ?subclass
}
```

For data sets that do not have an RDF schema class hierarchy, the RDF type can be exploited. Note that RDF type is the most used predicate in the LOD cloud. The following query is used to find all types in the data set sorted by their frequency:

```
Q3) SELECT DISTINCT ?o (COUNT(?s) AS ?frequency)
WHERE{
?s a ?o.
}
GROUP BY ?o
ORDER BY DESC(?frequency)
```

In both cases, the following query is used to find predicates sorted by the number of associations to literals:

```
Q4) SELECT DISTINCT ?p (COUNT(?o) AS ?frequency)
WHERE{
?s ?p ?o.
Filter (isliteral(?o))
}
GROUP BY ?p
ORDER BY DESC(?frequency)
```

The top k of these predicates are filtered based on whether they satisfy the filtering conditions on the language of the literals they are associated with and the length of these literals. This filtering is done by issuing the following query multiple times, once for each predicate. The placeholder `$PREDICATE$` is replaced with the current predicate being queried:

```
Q5) SELECT DISTINCT ?o
WHERE{
?s $PREDICATE$ ?o.
Filter (isliteral(?o) && lang(?o) = 'en' &&
strlen(str(?o)) < 80)
}
LIMIT 1
```

After issuing these queries to retrieve and filter predicates, if the data set uses RDF schema classes, Sapphire constructs the tree representing the class hierarchy. Starting from the root of this tree, the following query is issued to find literals associated with entities of a certain class (type) `$TYPE$` with a predicate `$PREDICATE$`. This query is issued iteratively, iterating over all classes and predicates:

```
Q6) SELECT DISTINCT ?o
WHERE{
?s a $TYPE$.
?s $PREDICATE$ ?o.
Filter (isliteral(?o) && lang(?o) = 'en' &&
strlen(str(?o)) < 80).
}
```

If a query on the class `$TYPE$` times out, queries over subclasses of this class are issued. If the query succeeds and returns an answer, then issuing the same query over the subclasses is redundant and is therefore not done.

In the case of data sets that do not use an RDF schema class hierarchy, a different way to reduce query result size is needed. For this, I use `LIMIT` and `OFFSET`. Specifically, the following query is issued multiple times, iterating over `$TYPE$` and `$PREDICATE$`, and using `LIMIT` and `OFFSET` to paginate the answers so that the query does not time out:

```
Q7) SELECT DISTINCT ?o
WHERE{
?s a $TYPE$.
?s $PREDICATE$ ?o.
Filter (isliteral(?o) && lang(?o) = 'en' &&
strlen(str(?o)) < 80).
}
```

```
LIMIT $LIMIT$
OFFSET $OFFSET$
```

Finally, the most significant literals need to be found. The following query template is used for this, and it is issued iteratively similar to Q7:

```
Q8) SELECT DISTINCT ?o (COUNT(?subject) AS ?frequency)
WHERE{
?s a $TYPE$.
?subject ?p ?s.
?s $PREDICATE$ ?o.
FILTER(lang(?o) = 'en' && strlen(str(?o)) < 80)
}
GROUP BY ?o
ORDER BY DESC(?frequency)
LIMIT $LIMIT$
OFFSET $OFFSET$
```

Recall that `$PREDICATE$` is associated with literals. Therefore, the literal filter is not added and only the filters on language and length are used.

Much of the complexity of the above queries is to avoid timeouts at the remote endpoints. This is important when using Sapphire in a *federated* architecture. Sapphire can also be used in a *warehousing* architecture, where all the data sets are stored locally on the same server as Sapphire. In the warehousing architecture, no limitations are placed on querying the data set, e.g., no resource constraints and no timeouts. This makes finding literals much simpler since long-running SPARQL queries can be easily issued without worrying about timeouts.

Specifically, the following query can be used to find literals filtered by length and language in the warehousing architecture (`$LIMIT$` can be used to restrict the number of results returned):

```
Q9) SELECT DISTINCT ?o
WHERE{
?s ?p ?o.
FILTER(isliteral(?o) && lang(?o) = 'en' &&
strlen(str(?o)) < 80)
}
```

```
GROUP BY ?o
LIMIT $LIMIT$
```

The following query finds the most significant literals in the warehousing architecture if there are no timeout constraints (again, with `$LIMIT$` and `$OFFSET$` if needed):

```
Q10) SELECT DISTINCT ?o (COUNT(?s1) AS ?frequency)
WHERE{
?s1 ?p ?s2.
?s2 ?p2 ?o.
FILTER(isliteral(?o) && lang(?o) = 'en' &&
strlen(str(?o)) < 80)
}
GROUP BY ?o
ORDER BY DESC(?frequency)
LIMIT $LIMIT$
OFFSET $OFFSET$
```

Appendix D

Evaluation Questions for the Sapphire User Study

D.1 Easy Queries

1. Country in which the Ganges starts
2. John F. Kennedy's vice president
3. Time zone of Salt Lake City
4. Tom Hanks's wife
5. Children of Margaret Thatcher
6. Currency of the Czech Republic
7. Designer of the Brooklyn Bridge
8. Wife of U.S. president Abraham Lincoln
9. Creator of Wikipedia
10. Depth of lake Placid

D.2 Medium Queries

1. Instruments played by Cat Stevens
2. Parents of the wife of Juan Carlos I
3. U.S. state in which Fort Knox is located
4. Person who is called Frank The Tank
5. Birthdays of all actors of the television show Charmed
6. Country in which the Limerick Lake is located
7. Person to which Robert F. Kennedy's daughter is married
8. Number of people living in the capital of Australia

D.3 Difficult Queries

1. Chess players who died in the same place they were born in
2. Books by William Goldman with more than 300 pages
3. Books by Jack Kerouac which were published by Viking Press
4. Films directed by Steven Spielberg with a budget of at least $80 million
5. Most populous city in Australia
6. Films starring Clint Eastwood direct by himself
7. Presidents born in 1945
8. Find each company that works in both the aerospace and medicine industries
9. Number of inhabitants of the most populous city in Canada