4-27-2010

Graph-Based Weakly-Supervised Methods for Information Extraction & Integration

Partha P. Talukdar
University of Pennsylvania, partha@cis.upenn.edu

Graph-Based Weakly-Supervised Methods for Information Extraction & Integration

Abstract

The variety and complexity of potentially-related data resources available for querying --- webpages, databases, data warehouses --- has been growing ever more rapidly. There is a growing need to pose integrative queries across multiple such sources, exploiting foreign keys and other means of interlinking data to merge information from diverse sources. This has traditionally been the focus of research within Information Extraction (IE) and Information Integration (II) communities, with IE focusing on converting unstructured sources into structured sources, and II focusing on providing a unified view of diverse structured data sources. However, most of the current IE and II methods, which can potentially be applied to the pro blem of integration across sources, require large amounts of human supervision, often in the form of annotated data. This need for extensive supervision makes existing methods expensive to deploy and difficult to maintain. In this thesis, we develop techniques that generalize from limited human input, via weakly-supervised methods for IE and II. In particular, we argue that graph-based representation of data and learning over such graphs can result in effective and scalable methods for large-scale Information Extraction and Integration. Within IE, we focus on the problem of assigning semantic classes to entities. First we develop a context pattern induction method to extend small initial entity lists of various semantic classes. We also demonstrate that features derived from such extended entity lists can significantly improve performance of state-of-the-art discriminative taggers.

The output of pattern-based class-instance extractors is often high-precision and low-recall in nature, which is inadequate for many real world applications. We use Adsorption, a graph based label propagation algorithm, to significantly increase recall of an initial high-precision, low-recall pattern-based extractor by combining evidences from unstructured and structured text corpora. Building on Adsorption, we propose a new label propagation algorithm, Modified Adsorption (MAD), and demonstrate its effectiveness on various real-world datasets. Additionally, we also show how class-instance acquisition performance in the graph-based SSL setting can be improved by incorporating additional semantic constraints available in independently developed knowledge bases.

Within Information Integration, we develop a novel system, Q, which draws ideas from machine learning and databases to help a non-expert user construct data-integrating queries based on keywords (across databases) and interactive feedback on answers. We also present an information need-driven strategy for automatically incorporating new sources and their information in Q. We also demonstrate that Q's learning strategy is highly effective in combining the outputs of ``black box'' schema matchers and in re-weighting bad alignments. This removes the need to develop an expensive mediated schema which has been necessary for most previous systems.

Keywords

machine learning, information extraction, data integration, keyword search, semi-supervised learning

Disciplines

Artificial Intelligence and Robotics | Databases and Information Systems

GRAPH-BASED WEAKLY-SUPERVISED METHODS FOR INFORMATION EXTRACTION & INTEGRATION

Partha Pratim Talukdar

A DISSERTATION

in

Computer and Information Science

Presented to the Faculties of the University of Pennsylvania in Partial

Fulfillment of the Requirements for the Degree of Doctor of Philosophy

2010

Prof. Fernando Pereira Supervisor of Dissertation	Prof. Zack Ives Supervisor of Dissertation
Prof. Mark Liberman Supervisor of Dissertation	Prof. Jianbo Shi Graduate Group Chairperson

Dissertation Committee

Prof. William Cohen, Carnegie Mellon University

Prof. Aravind Joshi, University of Pennsylvania

Prof. Ben Taskar, University of Pennsylvania

Prof. Lyle Ungar, University of Pennsylvania

Acknowledgments

First and foremost, I would like to thank my advisors, Prof. Zack Ives, Prof. Mark Liberman, and Prof. Fernando Pereira. This thesis would not have been possible without their guidance and constant support. I consider myself to be immensely fortunate to have had them as my advisors. I started working with Prof. Ives in the latter years of my graduate student life; I wish the collaborations had started earlier. Prof. Ives has always been there whenever I needed him. For that, no amount of gratitude can suffice. I am amazed at the breadth of knowledge and interests that Prof. Liberman possesses. I have thoroughly enjoyed my associations with him, and thank him for always considering my ideas carefully, however crazy they were. Prof. Pereira's enthusiasm, both in and out of science, has been an inspiration for me. His deep insights and scientific taste have contributed significantly to shaping my thought process and scientific pursuits.

I sincerely thank my thesis committee, Prof. William Cohen, Prof. Aravind Joshi, Prof. Ben Taskar, and Prof. Lyle Ungar, for its valuable comments and suggestions which improved this thesis significantly. I thank Prof. Cohen for readily agreeing to serve as the external on the committee, for diligently reading drafts of this thesis, and for providing very helpful critical feedback. I have always found my interactions with Prof. Joshi, Prof. Taskar, and Prof. Ungar to be stimulating and thought-provoking, and have learned a lot in the process. And it is with heartfelt gratitude that I thank Prof. Joshi and Prof. Liberman for encouraging me to come to Penn at the first place.

I feel deeply privileged to have worked with and learned from Prof. Koby Crammer during his tenure at Penn. I am sure that our collaborations will continue in the years to come. Special thanks to Prof. Michael Kearns and Prof. Sudipto Guha for many interesting discussions. My sincere thanks to Google, Thorsten Brants and Marius Paşca for supporting me over three summer internships, which resulted in two chapters of this thesis. Thanks to Louis Monier, Peter Norvig, D. Sivakumar, Wojciech Skut, and Vrishali Wagle for their valuable suggestions. I have enjoyed my collaborations with Rahul Bhagat, Marie Jacob, Salman Mehmood, Deepak Ravichandran, and Joseph Reisinger. I thank the wonderful and efficient staff at CIS and IRCS: Mike Felker, Laurel Sweeny and Cheryl Hickey in particular, for helping me navigate through administrative matters with much ease. I am grateful to the National Science Foundation (NSF) and DARPA for supporting my research.

Sincere thanks to Sarah Cohen-Boulakia for supplying the BioGuide schema, rankings, and expert queries. Special thanks to Elisabetta Manduchi and Chris Stoeckert for sharing the GBCO dataset with us, and to the authors of COMA++ for providing us with assistance.

My sincere thanks to João Graça, the only Ninja I know, for making sure this dissertation is submitted in time. I thank other past and present members of the Penn Structured Learning Group (my academic brothers and sisters): Axel Bernal and Qian Liu for making bioinformatics more accessible; John Blitzer for making sure the servers are free of unnecessary jobs (as determined by John); Mark Dredze for his undying enthusiasm; Kuzman Ganchev for making me realize that not everyone from Bulgaria is a wrestler; Alex Kulesza for sharing his deep air-travel insights, among others; Ryan McDonald for many exciting discussions; and Hanna Wallach for adding a unique touch to the office space. Special thanks to the Penn DB Group, and my other numerous friends at Penn and Philadelphia – Nikhil Dinesh, Ryan Gabbard, Jenny Gillenwater, Liang Huang, Annie Louis, Nick Montfort, Emily Pitler, Ted Sandler, Jeff Vaughn, Jenn Wortman Vaughn, Qiuye Zhao, Rangoli members, and the Philadelphia ISKCON community – for providing a stimulating environment and making my graduate student life enjoyable and truly memorable. Special thanks to Sunish Mohanan for all the thought provoking discussions on a wide range of topics. I can't thank Peg Eileen Bourke and Celine Bourke enough for their warm company over the years and for making a home away from home possible.

Finally, I thank my family: Ma, Deuta, Bamoni, Vindeo, Pranshu, and Asu; Shayna; and my canine friends over the years: Kattu, Kitti, Sweety, and Tik-Tiki; for their support and encouragement through thick and thin. All these would not have been without their care and love.

ABSTRACT

Graph-Based Weakly-Supervised Methods for Information Extraction & Integration

Partha Pratim Talukdar

Supervisors: Prof. Fernando Pereira, Prof. Zack Ives, and Prof. Mark Liberman

The variety and complexity of potentially-related data resources available for querying — webpages, databases, data warehouses — has been growing ever more rapidly. There is a growing need to pose integrative queries ***across*** multiple such sources, exploiting foreign keys and other means of interlinking data to merge information from diverse sources. This has traditionally been the focus of research within Information Extraction (IE) and Information Integration (II) communities, with IE focusing on converting unstructured sources into structured sources, and II focusing on providing a unified view of diverse structured data sources. However, most of the current IE and II methods, which can potentially be applied to the problem of integration *across* sources, require large amounts of human supervision, often in the form of annotated data. This need for extensive supervision makes existing methods expensive to deploy and difficult to maintain. In this thesis, we develop techniques that generalize from limited human input, via weakly-supervised methods for IE and II. In particular, we argue that *graph-based representation of data and learning over such graphs can result in effective and scalable methods for large-scale Information Extraction and Integration*.

Within IE, we focus on the problem of assigning semantic classes to entities. First we develop a context pattern induction method to extend small initial entity lists of various semantic classes. We also demonstrate that features derived from such extended entity lists can significantly improve performance of state-of-the-art discriminative taggers.

The output of pattern-based class-instance extractors is often high-precision and low-recall in nature, which is inadequate for many real world applications. We use Adsorption, a graph based label propagation algorithm, to significantly increase recall of an initial high-precision, low-recall pattern-based extractor by combining evidences from unstructured and structured text corpora. Building on Adsorption, we propose a new label propagation algorithm, Modified Adsorption (MAD), and demonstrate its effectiveness on various real-world datasets. Additionally, we also show how class-instance acquisition performance in the graph-based SSL setting can be improved by incorporating additional semantic constraints available in independently developed knowledge bases.

Within Information Integration, we develop a novel system, Q, which draws ideas from machine learning and databases to help a non-expert user construct data-integrating queries based on keywords (across databases) and interactive feedback on answers. We also present an *information need-driven* strategy for automatically incorporating new sources and their information in Q. We also demonstrate that Q's learning strategy is highly effective in combining the outputs of "black box" schema matchers and in re-weighting bad alignments. This removes the need to develop an expensive mediated schema which has been necessary for most previous systems.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The spectacular growth of the Internet and the World Wide Web (WWW) has made an enormous amount of heterogeneous data – unstructured, semi-structured and structured – available to the common user[1]. The variety and complexity of potentially-related data resources available for querying — webpages, databases, data warehouses, virtual integrated schemas — are growing ever more rapidly. Web search engines have become the virtual gate-keepers of these data by directing users to the data source which *may* contain the information the user is looking for. A user usually represents her information need in a few keywords which are input to a search engine. The search engine then retrieves a list of web pages ranked according to relevance of the webpages to the set of input keywords.

Web search engines have been very effective in such keyword based information retrieval (IR), which in part has fueled the growth of the web. However, they have certain limitations. For example, the current ranking granularity of search engines is at the webpage (single source) level. Hence, they can be effective only when a single source is enough to answer user's query.

Unfortunately, this is not sufficient in many real use cases. This is possibly most marked in the life sciences, where hundreds of large, complex, interlinked, and overlapping data resources have become available on fields like proteomics, genomics, disease studies, and

[1]In this thesis, we shall primarily focus on textual data sources.

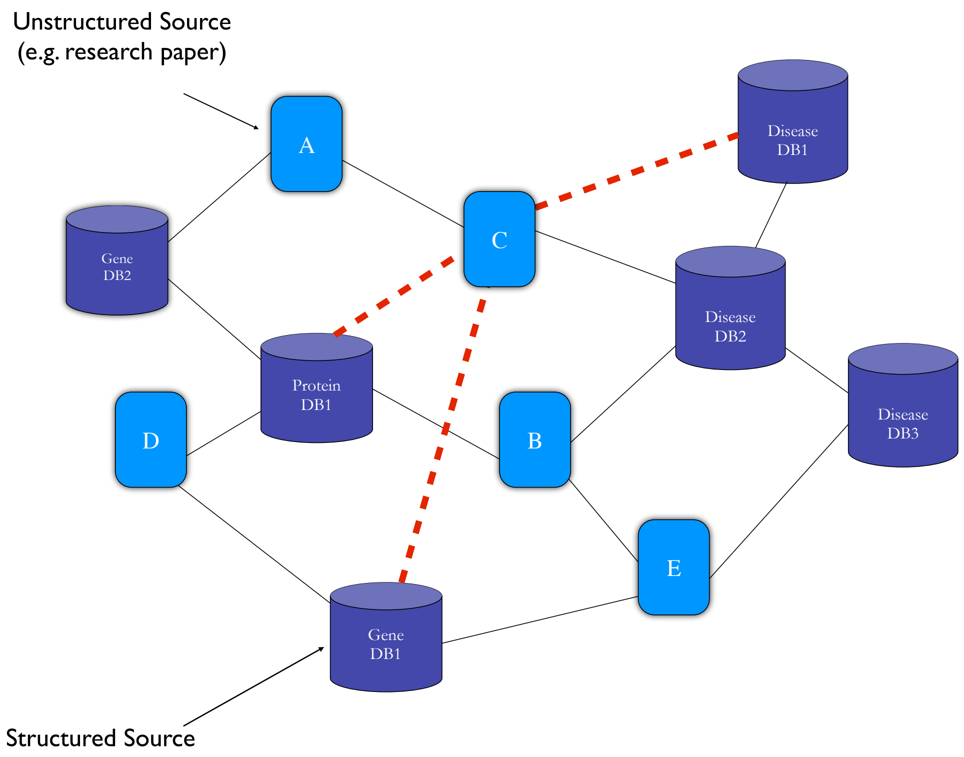


Figure 1.1: Heterogeneous sources along with links (e.g., hyperlinks, foreign-keys, citations, etc.) among them. Cylinders represent structured sources (e.g., relational databases), while rectangles represent unstructured sources (e.g., research papers). The red dotted lines connect sources which are needed together to answer a single information need, e.g., *"find all genes and proteins associated with the disease malaria"*, keywords are underlined. Traditional IR techniques assume the answer to be self-contained in a single source, and hence they are inadequate to answer such queries where there is a need to integrate multiple sources.

pharmacology. *Within* any of these databases, schemas are often massive: for example, the Genomics Unified Schema [Kissinger et al., 2002], used by a variety of databases on parasites, such as CryptoDB and PlasmoDB, has 358 relations, many of which have dozens of attributes each. Biomedical researchers need to pose integrative queries ***across*** multiple sources, exploiting foreign keys, similar terms, and other means of interlinking data: using a plethora of sources can reveal previously undiscovered relationships among biological entities [Boulakia et al., 2007, Mork et al., 2002], and some of these may lead to scientific breakthroughs. One such scenario is shown in Figure 1.1, where a variety of heterogeneous sources are linked with one another and the user is interested in finding out all the *genes* and *proteins* associated with the *disease malaria*. If there existed a single source which was created keeping exactly this information need in mind, then existing IR technology would have been enough to retrieve that source and answer the query. However, as we see in Figure 1.1, this is not the case as information regarding genes, proteins and diseases are spread across multiple sources. Moreover, anticipating future queries and constructing (or populating) data sources accordingly is neither practical nor efficient. Hence, there is a growing need to be able to integrate data across sources to answer user's queries.

The need for ability to query across sources is not limited to the life-sciences. For example, if one were interested in generating a list of the alma maters of all current U.S. city mayors, then there exist no easy way within current IR methods to achieve this goal, without issuing multiple queries and intermediate bookkeeping. In this case, one will have to first obtain the list of U.S. mayors (e.g., from `http://usmayors.org/climateprotection/list.asp`) and then lookup wikipedia pages of individual mayors (e.g., `http://en.wikipedia.org/wiki/Michael_Nutter`) which may contain their alma mater information. It is easy to see that this is a very labor intensive and inefficient process. Even though the necessary information is present and is spread across multiple sources, there exists no effective, automatic way to integrate them. Traditionally, research within Information Integration (II) has focused on this problem. In Section 1.2, we shall look into the current approaches to this problem, their limitations and our proposed solutions to overcome them.

While some of the currently available data sources for querying are structured, there are even larger number of unstructured sources (e.g., web pages, blogs, etc.). Currently, the dominant way to represent such unstructured sources is to treat them as bags-of-words. While this may be effective in standard keyword-based single source IR, this poses additional challenges when there is a need to integrate data across sources, some of which are unstructured. For example, database schemas provide the necessary semantics to integrate data across sources; unfortunately, such schemas are not readily available in case of unstructured sources. For such sources, the information in the data needs to be automatically organized in a structured form, resulting in a schema which in turn can be used for subsequent integration across sources. Hence, before any form of information integration can be attempted, there is a need to organize the information contained within unstructured documents. This has traditionally been the focus of research within Information Extraction (IE). In Section 1.1, we shall briefly look into the current approaches to IE, their limitations and our proposed methods to overcome them.

1.1 Information Extraction (IE)

The goal of Information Extraction (IE) is to automatically extract structured information from unstructured text sources (i.e., convert the rectangles into cylinder in Figure 1.1). For example, given the sentence,

John is a graduate student at the University of Pennsylvania .

an IE system is expected to return the following pieces of information: *"John"* is a `person`, *"University of Pennsylvania"* is a `university` and there is a `student-of` relationship between *"John"* and *"University of Pennsylvania"*; where `person`, `university`, and `student-of` are tags which are pre-determined by the system designer. To achieve this objective, an IE system has to solve two subtasks: (1) Entity Extraction: identifying the strings *"John"* and *"University of Pennsylvania"* as entities and assigning semantic classes to them (`person` and `university`, respectively); (2) Relation Extraction: identifying the binary `student-of` relation and its two arguments. The task of assigning semantic classes to instances (e.g., assigning class `student` to *"John"*) can be thought of as extraction of another special binary relation: the `IS-A` relation. In this thesis, we shall assume that the entities have already been segmented, and instead primarily focus on assigning classes to *pre-segmented* entities. The ability to do this on a large scale can result in a repository of class-instance[2] pairs, which can then be used to bootstrap other IE tasks (e.g., relation extraction), and can also be used in non-IE tasks (e.g., machine translation).

Over the last two decades, several methods for IE have been developed, some of which are based on hand-coded rules, while others are statistical learning based. A recent survey of these methods is presented in [Sarawagi, 2007]. Supervised machine learning techniques achieve state-of-the-art performance on these IE tasks [Florian et al., 2003], [Sang and De Meulder, 1837], [Bunescu and Mooney, 2005], [Culotta and Sorensen, 2004]. Such supervised methods assume availability of high-quality labeled training data, often annotated by human annotators. These annotations need to be very specific. For example, to train an `university` extractor, one has to identify documents containing universities and label specific mentions of universities in the documents. This labeling process is often quite expensive and time consuming. Instances and relations of interest are potentially infinite, therefore labeling data for each such entity and relation type is practically impossible. This requirement for labeled data has turned out to be a critical bottleneck towards wide development and deployment of IE systems.

To overcome the problems listed above, *seed-based* bootstrapped extraction of instances and relations has received considerable interest within the IE community [Hearst, 1992], [Riloff and Jones, 1999], [Brin, 1999], [Agichtein and Gravano, 2000], [Thelen and Riloff, 2002], [Etzioni et al., 2005]. These methods start with a few instances of the class (or relation) of interest and grow that initial set by extracting more instances of the same class from unlabeled text, often using contextual patterns as the extractor. Following this promising line of work, we present a novel context pattern induction method for entity extraction in Chapter 2. Another contribution we make is that on top of extending seed entity lists at high precision, we successfully include membership in these automatically generated lexicons as features in a high quality named entity tagger improving its performance.

[2]We use *entity* and *instance* interchangeably.

The output of pattern-based bootstrapped extraction systems tend to be high-precision, and low-recall in nature. However, for many real-world applications (e.g., Web search) recall is equally important, and hence applicability of pattern-based methods in such applications is limited. To address these problems, in Chapter 3, we present a graph-based method to acquire labeled instances by combining evidence from unstructured and structured text sources in a single framework. A graph-based label propagation algorithm, Adsorption [Baluja et al., 2008, Talukdar et al., 2008c], is used to assign semantic classes to instances. Adsorption is a semi-supervised learning (SSL) algorithm [Zhu, 2005, Chapelle et al., 2006] which reduces dependence on labeled data by exploiting widely available unlabeled data. Using the method presented in Chapter 3, we were able to significantly increase coverage of an initial high-precision low-recall pattern based extractor [Van Durme and Paşca, 2008]. WebTables [Cafarella et al., 2008], a collection of 154 million HTML tables extracted from the web, was used as the (structured) source of instances. In contrast to most previous approaches involving extractions from a *single* source, our proposed graph-based method demonstrated a way to combine information and extractions originating from *multiple* sources and methods, resulting in better and expanded class-instance acquisition overall.

Building on the Adsorption algorithm, we propose a new graph-based label propagation algorithm, Modified Adsorption (MAD) (Chapter 4), which shares all the desirable properties of Adsorption, while remaining more effective in practice. Like Adsorption, MAD can be parallelized which is suitable for our target application setting where there is a pressing need to efficiently process large volumes of data.

Recently, the problem of (instance, attribute) (e.g., (*China, population*)) extraction has started to receive attention [Probst et al., 2007, Pasca and Durme, 2007]. In Chapter 4, we explore whether additional semantic constraints in the form of (instance, attribute) pairs, which are obtained from independently developed knowledge bases, can be used to improve class-instance acquisition performance. In particular, we find preferences which favor two instances sharing attributes to belong to the same class to be very effective. We note that such preferences can be easily incorporated into the graph-based method we propose.

As mentioned previously, in this thesis we have concentrated on developing weakly-supervised, scalable methods for acquiring class-instance pairs, which can also be thought of as extraction of instances of a single relation: the IS-A relation. Developing similar methods for other types of relations is left as future work. In the next section, we review the issues associated with integrating information across sources, assuming all unstructured sources have already been converted into structured form.

1.2 Information Integration (II)

Information Integration remains one of the most difficult challenges in information technology, largely due to the ambiguities involved in trying to semantically merge different data sources. In an ideal world, the data needs of science, medicine, and policy would be met by discovering new data sets and databases the moment they are published, and automatically conveying their contents to users with related information needs, in the form relevant to those users. Instead, we live in a world where both discovery and semantic conversion are for the most part time-consuming, manual processes, causing a great deal of relevant information to be simply ignored. To address these difficulties, some research communities have attempted to define a consensus global schema (*mediated schema*) for their field so that individual sources can be mapped into a common representation. Researchers in machine learning, databases, and the semantic web have made significant progress in the last few years on partially automating these mapping and alignment tasks for given schemas and ontologies [Rahm and Bernstein, 2001].

However, the global-schema approach is poorly suited to automating the process of source discovery and integration in a dynamic scientific community. It is difficult to develop a consensus mediated schema that captures the diverse needs of a large user base and keeps up with new concepts, methods, and types of experimental result.

Going back to our examples from the life sciences, currently biologists have been restricted to queries embodied in Web forms, typically limited to a **single** database, that were created by full-time database maintainers; or they have to become experts in SQL, Web services, the different schemas, and scripting languages to create their own queries that span multiple schemas.

A potentially promising approach to authoring queries without using SQL — primarily used within a *single* database — has been to start with *keyword queries* and to match terms in tuples within the data instance's tables. If multiple keywords are provided and match on different 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 [Bhalotia et al., 2002, Hristidis and Papakonstantinou, 2002]. Path lengths are used as costs for the joined results. In general, results will be highly heterogeneous and specific to the keyword matches in the tuples. It is also possible to support approximate, similarity-based joins [Cohen, 1998] or approximate keyword matches.

Unfortunately, the above work may be insufficient for scientific users, because such work assumes query-insensitive costs for path length, attribute similarity, and other matching steps, when scientists may need costs *specific to the context of the query* (i.e., the setting under which the query is being posed). Preferences for sources may depend on whether users are posing "what-if" types of exploratory queries or refining previous answers; they may depend on the specific query domain; or they may depend on the (perceived or actual) quality of the individual data sources. Recent bioinformatics work [Boulakia et al., 2007, Mork et al., 2002] shows that there are often *many* combinations of data sources that can reveal relationships between biological entities, and biologists need ways of discovering, understanding, and assessing the quality of each of the possible ways of linking data.

Rather than simply answering queries by matching keywords to tuples and finding associations, in Chapter 5 we propose a system, Q, for defining and *interactively refining* families of queries, based on keywords. The user poses keyword queries that are matched against source relations and their attributes; the system uses sequences of associations (e.g., foreign keys, links, schema mappings, synonyms, and taxonomies) to create multiple ranked queries linking the matches to keywords; the set of queries is attached to a Web query form. Now the user and his or her associates may pose specific queries by filling in parameters in the form. Importantly, the answers to this query are ranked and annotated with data provenance, and the user provides *feedback* on the utility of the answers, from which the system ultimately learns to assign costs to sources and associations according to the user's specific information need, as a result changing the ranking of the queries used to generate results.

In contrast to previous approaches to keyword search over databases [Bhalotia et al., 2002], [Botev and Shanmugasundaram, 2005], [Hristidis and Papakonstantinou, 2002], [Kacholia et al., 2005], we seek to *learn* how to score results based on user preferences, since associations between scientific data sources do not necessarily reflect authority, and in any case perceived authority may vary with user characteristics and needs. This naturally complements a number of existing bioinformatics query authoring systems that rely on expert-provided scores [Boulakia et al., 2007, Mork et al., 2002].

Currently, few mechanisms exist for *discovering* relevant sources as they are first published, and for having their data *automatically put into use*. Even though schema alignment tools may be used to align a new source to existing sources, such tools rarely *scale* to large numbers of schemas and relations, and it can be difficult to determine when they have produced the *right mappings*. In order to address these challenges, in Chapter 6, we present an information need-driven strategy for automatically incorporating new sources and their information in Q. We also demonstrate that our learning strategy is highly effective in combining the outputs of "black box" schema matchers and in re-weighting bad alignments.

1.3 Thesis Contributions & Organization

In this document, we support the following thesis:

Graph-based representation of data and learning over such graphs result in effective and scalable methods for large-scale information extraction and integration.

We have found that graph-based methods allow us to have the following: *high performance*, *weak-supervision*, and *scalability* – three crucial ingredients for overcoming the challenges in large-scale Information Extraction and Integration, as mentioned in previous sections.

In this thesis, we make the following contributions:

- In Chapter 2, we propose a novel context pattern induction method for entity extraction. We demonstrate effectiveness of the proposed method by extending seed entity lists of various types at fairly high precision. We also show how performance of a state-of-the-art discriminative tagger can be improved by adding features derived from such extended entity lists.

- In Chapter 3, we use a graph-based semi-supervised label propagation algorithm, Adsorption, for acquiring open-domain labeled classes and their instances from a combination of unstructured and structured text sources. This allows extractions from diverse sources and different methods to be put together in a single framework and perform joint learning and inference. This acquisition method significantly improves coverage compared to a previous set of labeled classes and instances derived from free text, while achieving comparable precision.

- Building on Adsorption, in Chapter 4, we present a new label propagation algorithm, Modified Adsorption (MAD). We compare many label propagation methods on a variety of real-world learning tasks, including class-instance acquisition, and find MAD to be the most effective. We also show how class-instance acquisition performance in the graph-based SSL setting can be improved by including additional semantic constraints available in independently constructed knowledge bases.

- In Chapter 5, we focus on Information Integration and present a novel system, Q, which draws ideas from machine learning and databases to help a non-expert user construct data-integrating queries based on keywords (across databases) and interactive feedback on answers. We evaluate the effectiveness of Q against gold standard costs from domain experts and demonstrate the method's scalability.

- In Chapter 6, we present an information need-driven strategy for automatically incorporating new sources and their information in Q. This is particularly important in today's environment where new data sources are constantly showing up and there is a pressing need to make new source's data available to the user at the earliest. We have also demonstrated that our learning strategy is highly effective in combining the outputs of "black box" schema matchers and in re-weighting bad alignments.

Finally, in Chapter 7, we conclude the thesis by outlining again its main contributions and listing avenues for future work.

Chapter 2

Entity Extraction from Unstructured Text

Some parts of this chapter are based on [Talukdar et al., 2006].

2.1 Introduction

Partial entity lists and massive amounts of unlabeled data are becoming available with the growth of the Web, as well as the increased availability of specialized corpora and entity lists. For example, the primary public resource for biomedical research, MEDLINE, contains over 13 million entries and is growing at an accelerating rate. Combined with these large corpora, the recent availability of entity lists in those domains has opened up interesting opportunities and challenges. Such lists are never complete and suffer from sampling biases. However, we would like to exploit them, in combination with large unlabeled corpora, to speed up the creation of information extraction systems for different domains and languages. In this chapter, we concentrate on exploring the utility of such resources for named entity extraction.

Currently available entity lists contain a small fraction of named entities: there are orders of magnitude more entities present in the unlabeled data[1]. In this chapter, we test the following hypotheses:

i. Starting with a few seed entities, it is possible to induce high-precision context patterns by exploiting entity context redundancy.

ii. New entity instances of the same category can be extracted from unlabeled data with the induced patterns to create high-precision extensions of the seed lists.

iii. Features derived from token membership in the extended lists improve the accuracy of learned named-entity taggers.

Previous approaches to context pattern induction were described by [Riloff and Jones, 1999], [Agichtein and Gravano, 2000], [Thelen and Riloff, 2002], [Lin et al., 2003], and [Etzioni et al., 2005], among others. The main advance in the present method is the combination of grammatical induction and statistical techniques to create high-precision patterns.

The chapter is organized as follows. Section 2.2 describes our pattern induction algorithm. Section 2.3 shows how to extend seed sets with entities extracted by the patterns from unlabeled data. Section 2.4 gives experimental results.

2.2 Context Pattern Induction

The overall method for inducing entity context patterns and extending entity lists is as follows:

1. Let E = seed set, T = text corpus.
2. Find the contexts C of entities in E in the corpus T (Section 2.2.1).
3. Select *trigger words* from C (Section 2.2.2).

[1]For example, based on approximate matching, out of the 2403 unique organization names mentioned in CoNLL-2003 shared task training data, only 22 are present in the Fortune 500 list.

4. For each trigger word, induce a pattern automaton (Section 2.2.3).
5. Use induced patterns P to extract more entities E' (Section 2.3).
6. Rank P and E' (Section 2.3.1).
7. If needed, add high scoring entities in E' to E and return to step 2. Otherwise, terminate with patterns P and extended entity list $E \cup E'$ as results.

2.2.1 Extracting Context

Starting with the seed list, we first find occurrences of seed entities in the unlabeled data. For each such occurrence, we extract a fixed number W (context window size) of tokens immediately preceding and immediately following the matched entity. As we are only interested in modeling the context here, we replace all entity tokens by the single token `-ENT-`. This token now represents a *slot* in which an entity can occur. Examples of extracted entity contexts are shown in Table 2.1. In the work presented in this chapter, seeds are entity instances (e.g., *Google* is a seed for organization category).

increased expression of -ENT- in vad mice
the expression of -ENT- mrna was greater
expression of the -ENT- gene in mouse

Table 2.1: Extracted contexts of known genes with $W = 3$.

The set of extracted contexts is denoted by C. The next step is to automatically induce high-precision patterns containing the token `-ENT-` from such extracted contexts.

2.2.2 Trigger Word Selection

To induce patterns, we need to determine their starting positions. It is reasonable to assume that some tokens are more specific to particular entity classes than others. For example, in the examples shown above, *expression* can be one such word for gene names. Whenever one comes across such a token in text, the probability of finding an entity (of the corresponding entity class) in its vicinity is high. We call such starting tokens *trigger words*. Trigger words mark the beginning of a pattern. It is important to note that simply selecting the first token of extracted contexts may not be a good way to select trigger words. In such a scheme, we would have to vary W to search for useful pattern starts. Instead of that brute-force technique, we propose an automatic way of selecting trigger words. A good set of trigger words is very important for the quality of induced patterns. Ideally, we want a trigger word to satisfy the following:

- It is frequent in the set C of extracted contexts.
- It is specific to entities of interest and thereby to extracted contexts.

We use a term-weighting method to rank candidate trigger words from entity contexts. IDF (Inverse Document Frequency) was used in our experiments but any other suitable term-weighting scheme may work comparably. The IDF weight f_w for a word w occurring in a corpus is given by:

$$f_w = \log\left(\frac{N}{n_w}\right)$$

where N is the total number of documents in the corpus and n_w is the total number of documents containing w. Now, for each context segment $c \in C$, we select a *dominating word* d_c given by

$$d_c = \arg\max_{w \in c} f_w$$

There is one dominating word for each $c \in C$. All dominating words for contexts in C form multiset M. Let m_w be the multiplicity of the dominating word w in M. We sort M by decreasing m_w and select the top n tokens from this list as potential trigger words.

Selection criteria based on dominating word frequency work better than criteria based on simple term weight because high term weight words may be rare in the extracted contexts, but would still be misleadingly selected for pattern induction. This can be avoided by using instead the frequency of dominating words within contexts, as we did here.

2.2.3 Automata Induction

Rather that using individual contexts directly, we summarize them into automata that contain the most significant regularities of the contexts sharing a given trigger word. This construction allows us to determine the relative importance of different context features using a variant of the forward-backward algorithm from HMMs.

2.2.3.1 Initial Induction

For each trigger word, we extract the contexts starting with the word. For example, with *"expression"* as the trigger word, the contexts in Table 2.1 are reduced to those in Table 2.2. Since *"expression"* is a left-context trigger word, only one token to the right of `-ENT-` is retained. Here, the predictive context lies to the left of the slot `-ENT-` and a single token is retained on the right to mark the slot's right boundary. To model predictive right contexts, the token string can be reversed and the same techniques as here applied on the reversed string.[2]

expression of `-ENT-` *in*
expression of `-ENT-` *mrna*
expression of the `-ENT-` *gene*

Table 2.2: Context segments corresponding to trigger word *"expression"*.

Similar contexts are prepared for each trigger word. The context set for each trigger word is then summarized by a pattern automaton with transitions that match the trigger word and also the wildcard `-ENT-`. We expect such automata to model the position in

[2]Experiments reported in this chapter use predictive left context only.

the context of the entity slot and help us extract more entities of the same class with high precision.

We use a simple form of grammar induction to learn the pattern automata. Grammar induction techniques have been previously explored for information extraction (IE) and related tasks. For instance, [Freitag, 1997] used grammatical inference to improve precision in IE tasks.

Context segments are short and typically do not involve recursive structures. Therefore, we chose to use 1-reversible automata to represent sets of contexts. An automaton A is *k-reversible* iff (1) A is deterministic and (2) A^r is deterministic with k tokens of lookahead, where A^r is the automaton obtained by reversing the transitions of A. Wrapper induction using *k-reversible* grammar is discussed by [Chidlovskii, 2000].

In the 1-reversible automaton induced for each trigger word, all transitions labeled by a given token go to the same state, which is identified with that token. Figure 2.1 shows a fragment of a 1-reversible automaton. [Solan et al., 2005] describe a similar automaton construction, but they allow multiple transitions between states to distinguish among sentences.

Each transition $e = (v, w)$ in a 1-reversible automaton A corresponds to a bigram vw in the contexts used to create A. We thus assign each transition the probability

$$P(w|v) = \frac{C(v,w)}{\Sigma_{w'} C(v,w')}$$

where $C(v,w)$ is the number of occurrences of the bigram vw in contexts for W. With this construction, we ensure words will be credited in proportion to their frequency in contexts. The automaton may overgenerate, but that potentially helps generalization.

2.2.3.2 Pruning

The initially induced automata need to be pruned to remove transitions with weak evidence so as to increase match precision.

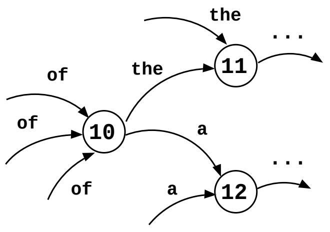


Figure 2.1: Fragment of a 1-reversible automaton

The simplest pruning method is to set a count threshold c below which transitions are removed. However, this is a poor method. Consider state 10 in the automaton of Figure 2.2, with $c = 20$. Transitions $(10, 11)$ and $(10, 12)$ will be pruned. $C(10, 12) \ll c$ but $C(10, 11)$ just falls short of c. However, from the transition counts, it looks like the sequence *"the -ENT-"* is very common. In such a case, it is not desirable to prune $(10, 11)$. Using a local threshold may lead to overpruning.

We would like instead to keep transitions that are used in relatively many probable paths through the automaton. The probability of path p is $P(p) = \prod_{(v,w)\in p} P(w|v)$. Then the posterior probability of edge (v, w) is

$$P(v, w) = \frac{\sum_{(v,w)\in p} P(p)}{\sum_p P(p)} \quad ,$$

which can be efficiently computed by the forward-backward algorithm [Rabiner, 1989]. We can now remove transitions leaving state v whose posterior probability is lower than $p_v = k(\max_w P(v, w))$, where $0 < k \leq 1$ controls the degree of pruning, with higher k forcing more pruning. All induced and pruned automata are trimmed to remove unreachable states.

2.3 Automata as Extractor

Each automaton induced using the method described in Section 2.2.3 represents high-precision patterns that start with a given trigger word. By scanning unlabeled data using these patterns, we can extract text segments which can be substituted for the slot token -ENT-. For example, assume that the induced pattern is *"analyst at -ENT- and"* and that

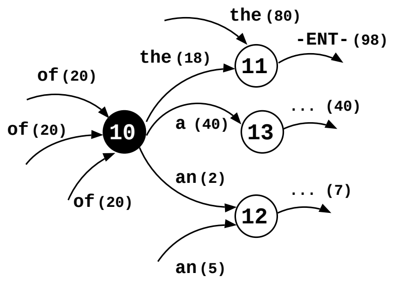


Figure 2.2: Automaton to be pruned at state 10. Transition counts are shown in parenthesis.

the scanned text is *"He is an analyst at the University of California and ..."*. By scanning this text using the pattern mentioned above, we can figure out that the text *"the University of California"* can substitute for "-ENT-". This extracted segment is a candidate extracted entity. We now need to decide whether we should retain all tokens inside a candidate extraction or purge some tokens, such as *"the"* in the example.

One way to handle this problem is to build a language model of content tokens and retain only the maximum likelihood token sequence. However, in the current work, the following heuristic which worked well in practice is used. Each token in the extracted text segment is labeled either *keep* (K) or *droppable* (D). By default, a token is labeled K. A token is labeled D if it satisfies one of the droppable criteria: whether the token is present in a common-word list, whether it is non-capitalized, or whether it is a number.

Once tokens in a candidate extraction are labeled using the above heuristic, the longest token sequence corresponding to the regular expression $\mathtt{K}[\mathtt{D}\ \mathtt{K}]*\mathtt{K}$ is retained and is considered a final extraction. If there is only one K token, that token is retained as the final extraction. In the example above, the tokens are labeled *"the/D University/K of/D California/K"*, and the extracted entity will be *"University of California"*.

To handle run-away extractions, we can set a domain-dependent hard limit on the number of tokens which can be matched with "-ENT-". This stems from the intuition that useful extractions are not very long. For example, it is rare that a person name is longer than five tokens.

2.3.1 Ranking Patterns and Entities

Using the method described above, patterns and the entities extracted by them from unlabeled data are paired. But both patterns and extractions vary in quality, so we need a method for ranking both. Hence, we need to rank both patterns and entities. This is difficult given that there we have no negative labeled data. Seed entities are the only positive instances that are available.

Related previous work tried to address this problem. [Agichtein and Gravano, 2000] seek to extract relations, so their pattern evaluation strategy considers one of the attributes of an extracted tuple as a key. They judge the tuple as a positive or a negative match for the pattern depending on whether there are other extracted values associated with the same key. Unfortunately, this method is not applicable to entity extraction.

The pattern evaluation mechanism used here is similar in spirit to those of [Etzioni et al., 2005] and [Lin et al., 2003]. With seeds for multiple classes available, we consider seed instances of one class as negative instances for the other classes. A pattern is penalized if it extracts entities which belong to the seed lists of the other classes. Let $\mathrm{pos}(p)$ and $\mathrm{neg}(p)$ be respectively the number of distinct positive and negative seeds extracted by pattern p. In contrast to previous work mentioned above, we do not combine $\mathrm{pos}(p)$ and $\mathrm{neg}(p)$ to calculate a single accuracy value. Instead, we discard all patterns p with positive $\mathrm{neg}(p)$ value, as well as patterns whose total positive seed (non distinct) extraction count is less than certain threshold η_{pattern}. This scoring is very conservative. There are several motivations for such a conservative scoring. First, we are more interested in precision than recall. We believe that with massive corpora, large number of entity instances can be extracted anyway. High accuracy extractions allow us to reliably (without any human evaluation) use extracted entities in subsequent tasks successfully (see Section 2.4.3). Second, in the absence of sophisticated pattern evaluation schemes (which we are investigating — Section 2.6), we feel it is best to heavily penalize any pattern that extracts even a single negative instance.

Let G be the set of patterns which are retained by the filtering scheme described above. Also, let $I(e, p)$ be an indicator function which takes value 1 when entity *e* is extracted by pattern *p* and 0 otherwise. The score of e, $S(e)$, is given by

$$S(e) = \Sigma_{p \in G} I(e, p)$$

This whole process can be iterated by including extracted entities whose score is greater than or equal to a certain threshold η_{entity} to the seed list.

2.4 Experimental Results

For the experiments described below, we used 18 billion tokens (31 million documents) of news data as the source of unlabeled data. We experimented with 500 and 1000 trigger words. The results presented were obtained after a single iteration of the Context Pattern Induction algorithm (Section 2.2).

2.4.1 English LOC, ORG and PER

For this experiment, we used as seed sets subsets of the entity lists provided with CoNLL-2003 shared task data.[3] Only multi-token entries were included in the seed lists of respective categories (location (LOC), person (PER) & organization (ORG) in this case). This was done to partially avoid incorrect context extraction. For example, if the seed entity is *"California"*, then the same string present in *"University of California"* can be incorrectly considered as an instance of LOC. A stoplist was used for dropping tokens from candidate extractions, as described in Section 2.3. Examples of top ranking induced patterns and extracted entities are shown in Table 2.9. Seed list sizes and experimental results are shown in Table 2.3. The precision numbers shown in Table 2.3 were obtained by manually evaluating 100 randomly selected instances from each of the extended lists.

[3]A few locally available entities in each category were also added. These seeds are available upon request from the authors.

Category	Seed Size	Patterns Used	Extended Size	Precision
LOC	379	29	3001	70%
ORG	1597	276	33369	85%
PER	3616	265	86265	88%

Table 2.3: Results of LOC, ORG & PER entity list extension experiment with η_{pattern} = 10 set manually.

The overlap[4] between the induced ORG list and the Fortune 500 list has 357 organization names, which is significantly higher than the seed list overlap of 22 (see Section 4.1). This shows that we have been able to improve coverage considerably.

2.4.2 Watch Brand Name

A total of 17 watch brand names were used as seeds. In addition to the pattern scoring scheme of Section 2.3.1, only patterns containing sequence *"watch"* were finally retained. Entities extracted with $\eta_{\text{entity}} = 2$ are shown in Table 2.5. Extraction precision is 85.7%.

This experiment is interesting for several reasons. First, it shows that the method presented in this chapter is effective even with small number of seed instances. From this we conclude that the unambiguous nature of seed instances is much more important than the size of the seed list. Second, no negative information was used during pattern ranking in this experiment. This suggests that for relatively unambiguous categories, it is possible to successfully rank patterns using positive instances only.

2.4.3 Extended Lists as Features in a CRF Tagger

Supervised models normally outperform unsupervised models in extraction tasks. The downside of supervised learning is expensive training data. On the other hand, massive amounts of unlabeled data are readily available. The goal of semi-supervised learning to combine the best of both worlds. Recent research have shown that improve-

[4]Using same matching criteria as in Section 4.1.

Corum, Longines, Lorus, Movado, Accutron, Audemars Piguet, Cartier, Chopard, Franck Muller, IWC, Jaeger-LeCoultre, A. Lange & Sohne, Patek Philippe, Rolex, Ulysse, Nardin, Vacheron Constantin

Table 2.4: Watch brand name seeds.

Rolex	Fossil	Swatch
Cartier	Tag Heuer	Super Bowl
Swiss	Chanel	SPOT
Movado	Tiffany	Sekonda
Seiko	TechnoMarine	Rolexes
Gucci	Franck Muller	Harry Winston
Patek Philippe	Versace	Hampton Spirit
Piaget	Raymond Weil	Girard Perregaux
Omega	Guess	Frank Mueller
Citizen	Croton	David Yurman
Armani	Audemars Piguet	Chopard
DVD	DVDs	Chinese
Breitling	Montres Rolex	Armitron
Tourneau	CD	NFL

Table 2.5: Extended list of watch brand names after single iteration of pattern induction algorithm.

ments in supervised taggers are possible by including features derived from unlabeled data [Miller et al., 2004, Liang, 2005, Ando and Zhang, 2005]. Similarly, automatically generated entity lists can be used as additional features in a supervised tagger.

For this experiment, we started with a conditional random field (CRF) [Lafferty et al., 2001] tagger with a competitive baseline (Table 2.6). The baseline tagger was trained[5] on the full CoNLL-2003 shared task data. We experimented with the LOC, ORG and PER lists that were automatically generated in Section 2.4.1. In Table 2.7, we show the accuracy of the tagger for the entity types for which we had

[5]Standard orthographic information, such as character n-grams, capitalization, tokens in immediate context, chunk tags, and POS were used as features.

System	F1 (Precision, Recall)
[Florian et al., 2003], best single, no list	89.94 (91.37, 88.56)
[Zhang and Johnson, 2003], no list	90.26 (91.00, 89.53)
CRF baseline, no list	89.52 (90.39, 88.66)

Table 2.6: Baseline comparison on 4 categories (LOC, ORG, PER, MISC) on Test-a dataset.

Training Data (Tokens)	Test-a			Test-b		
	No List	Seed List	Unsup. List	No List	Seed List	Unsup. List
9268	68.16	70.91	**72.82**	60.30	63.83	**65.56**
23385	78.36	79.21	**81.36**	71.44	72.16	**75.32**
46816	82.08	80.79	**83.84**	76.44	75.36	**79.64**
92921	85.34	83.03	**87.18**	81.32	78.56	**83.05**
203621	89.71	84.50	**91.01**	84.03	78.07	**85.70**

Table 2.7: CRF tagger F-measure on LOC, ORG, PER extraction.

induced lists. The test conditions are just baseline features with no list membership, baseline plus seed list membership features, and baseline plus induced list membership features. For completeness, we also show in Table 2.8 accuracy on the full CoNLL task (four entity types) without lists, with seed list only, and with the three induced lists. The seed lists (Section 2.4.1) were prepared from training data itself and hence with increasing training data size, the model overfitted as it became completely reliant on these seed lists. From Tables 2.7 & 2.8 we see that incorporation of token membership in the extended lists as additional membership features led to improvements across categories and at all sizes of training data. This also shows that the extended lists are of good quality, since the tagger is able to extract useful evidence from them.

Relatively small sizes of training data pose interesting learning situation and is the case with practical applications. It is encouraging to observe that the list features lead to significant improvements in such cases. Also, as can be seen from Table 2.7 & 2.8, these

Training Data	Test-a			Test-b		
(Tokens)	No List	Seed List	Unsup. List	No List	Seed List	Unsup. List
9229	68.27	70.93	**72.26**	61.03	64.52	**65.60**
204657	89.52	84.30	**90.48**	83.17	77.20	**84.52**

Table 2.8: CRF tagger F-measure on LOC, ORG, PER and MISC extraction.

lists are effective even with mature taggers trained on large amounts of labeled data.

2.5 Related Work

The method presented in this chapter is similar in many respects to some of the previous work on context pattern induction [Riloff and Jones, 1999, Agichtein and Gravano, 2000, Lin et al., 2003, Etzioni et al., 2005], but there are important differences. [Agichtein and Gravano, 2000] focus on relation extraction while we are interested in entity extraction. Moreover, [Agichtein and Gravano, 2000] depend on an entity tagger to initially tag unlabeled data whereas we do not have such requirement. The pattern learning methods of [Riloff and Jones, 1999] and the generic extraction patterns of [Etzioni et al., 2005] use language-specific information (for example, chunks). In contrast, the method presented here is language independent. For instance, the English pattern induction system presented here was applied on German data without any change. Also, in the current method, induced automata compactly represent all induced patterns. The patterns induced by [Riloff and Jones, 1999] extract NPs and that determines the number of tokens to include in a single extraction. We avoid using such language dependent chunk information as the patterns in our case include right[6] boundary tokens thus explicitly specifying the slot in which an entity can occur. Bayesian Sets (BS) [Ghahramani and Heller, 2006] is a recently proposed method for *set expansion* whose goals is similar to the method presented in this chapter. BS requires the candidate

[6]In case of predictive left context.

Induced LOC Patterns
troops in -ENT-to
Cup qualifier against -ENT-in
southern -ENT-town
war - torn -ENT-.
countries including -ENT-.
Bangladesh and -ENT-,
England in -ENT-in
west of -ENT-and
plane crashed in -ENT-.
Cup qualifier against -ENT-,

Extracted LOC Entities
US
United States
Japan
South Africa
China
Pakistan
France
Mexico
Israel
Pacific

Induced PER Patterns
compatriot -ENT-.
compatriot -ENT-in
Rep. -ENT-,
Actor -ENT-is
Sir -ENT-,
Actor -ENT-,
Tiger Woods , -ENT-and
movie starring -ENT-.
compatriot -ENT-and
movie starring -ENT-and

Extracted PER Entities
Tiger Woods
Andre Agassi
Lleyton Hewitt
Ernie Els
Serena Williams
Andy Roddick
Retief Goosen
Vijay Singh
Jennifer Capriati
Roger Federer

Induced ORG Patterns
analyst at -ENT-.
companies such as -ENT-.
analyst with -ENT-in
series against the -ENT-tonight
Today 's Schaeffer 's Option Activity Watch features -ENT-(
Cardinals and -ENT-,
sweep of the -ENT-with
joint venture with -ENT-(
rivals -ENT-Inc.
Friday night 's game against -ENT-.

Extracted ORG Entities
Boston Red Sox
St. Louis Cardinals
Chicago Cubs
Florida Marlins
Montreal Expos
San Francisco Giants
Red Sox
Cleveland Indians
Chicago White Sox
Atlanta Braves

Table 2.9: Top ranking LOC, PER, ORG induced pattern and extracted entity examples.

entities to be given as input, while the method in this chapter *extracts* (and subsequently ranks) such candidate entities from unstructured text. SEAL [Wang and Cohen, 2007] is another promising method recently proposed for set expansion. While SEAL exploits semi-structured text, the method in this chapter processes unstructured text.

Another contribution we make is the fact that on top of extending seed lists at high precision, we successfully include membership in these automatically generated lexicons as features in a high quality named entity tagger improving its performance.

2.6 Summary of Chapter

In this chapter, we have presented a novel language-independent context pattern induction method. Starting with a few seed examples, the method induces in an unsupervised way context patterns and extends the seed list by extracting more instances of the same category at fairly high precision from unlabeled data. We were able to improve a CRF-based high quality named entity tagger by using membership in these automatically generated lists as additional features.

In the next chapter, we shall see how the coverage of an initial high precision extractor, such as the one presented in this chapter, can be significantly increased by graph-based semi-supervised learning methods.

Chapter 3

Graph-based Acquisition of Labeled Entities

Some parts of this chapter are based on [Talukdar et al., 2008c].

3.1 Introduction

3.1.1 Motivation

Users of large document collections can readily acquire information about the instances, classes, and relationships described in the documents. Such relations play an important role in both natural language understanding and Web search, as illustrated by their prominence in both Web documents and among the search queries submitted most frequently by Web users [Jansen et al., 2000]. These observations motivate our work on algorithms to extract instance-class information from Web documents.

While work on named-entity recognition traditionally focuses on the acquisition and identification of instances within a small set of coarse-grained classes, the distribution of instances within query logs indicates that Web search users are interested in a wider range of more fine-grained classes. Depending on prior knowledge, personal interests and immediate needs, users submit for example medical queries about the symptoms of *leptospirosis* or the treatment of *monkeypox*, both of which are instances of *zoonotic diseases*, or the risks and benefits of *surgical procedures* such as *PRK* and *angioplasty*. Other users may be more interested in *African countries* such as *Uganda* and *Angola*, or *active volcanoes* like *Etna* and *Kilauea*. Note that *zoonotic diseases*, *surgical procedures*, *African countries* and *active volcanoes* serve as useful class labels that capture the semantics of the associated sets of class instances. Such interest in a wide variety of specific domains highlights the utility of constructing large collections of fine-grained classes.

Comprehensive and accurate class-instance information is useful not only in search but also in a variety of other text processing tasks including co-reference resolution [McCarthy and Lehnert, 1995].

Pattern-based bootstrapped extractions (Chapter 2) present an interesting approach to address these large-scale class-instance acquisition challenges. However, there are a few limitations.

- The output of seed-based bootstrapped extraction systems tend to be high-precision, and low-recall in nature. This is primarily because of the fact that pattern based extraction methods tend to estimate extraction confidence using redundancy based counts. This, along with a high retention threshold (in the face of uncertainty), leaves very few extractions finally, resulting in lower recall. However, as mentioned above, for many real-world applications recall is equally important, thereby limiting applicability of pattern-based methods in such applications.

- The optimization carried out by bootstrapped extractors is not very well understood. For example, it is not clear what (if any) objective function is being optimized by the methods presented in [Hearst, 1992], [Riloff and Jones, 1999], [Brin, 1999], [Agichtein and Gravano, 2000], [Thelen and Riloff, 2002], [Etzioni et al., 2005], [Pantel and Pennacchiotti, 2006] or in Chapter 2.

- Context pattern based extractors are likely to be effective only when such patterns can be reliably learned from unstructured text. For entities or relations occurring in a variety of contexts, learning such context patterns may not be effective. Some evidence of this is presented in [Bellare et al., 2007].

3.1.2 Contributions

We study the acquisition of open-domain, labeled classes and their instances from both structured and unstructured textual data sources by combining and ranking individual extractions in a principled way with the Adsorption label-propagation algorithm [Baluja et al., 2008], reviewed in Section 4.2 below. To the best of our knowledge, this is one of the first attempts of this kind in the area of minimally-supervised extraction algorithms.

A collection of labeled classes acquired from text [Van Durme and Paşca, 2008] is extended in two ways:

1. Class label coverage is increased by identifying additional class labels (such as *public agencies* and *governmental agencies*) for existing instances such as *Office of War Information*).

2. The overall instance coverage is increased by extracting additional instances (such as *Addison Wesley* and *Zebra Books*) for existing class labels (*book publishers*).

The WebTables database constructed by [Cafarella et al., 2008] is used as the source of additional instances. Evaluations on gold-standard labeled classes and instances from existing linguistic resources [Fellbaum, 1998] indicate coverage improvements relative to that of [Van Durme and Paşca, 2008], while retaining similar precision levels.

3.2 First Phase Extractors

To show Adsorption's ability to uniformly combine extractions from multiple sources and methods, we apply it to: 1) high-precision open-domain extractions from free Web

Class	Size	Examples of Instances
Book Publishers	70	crown publishing, kluwer academic, prentice hall, puffin
Federal Agencies	161	catsa, dhs, dod, ex-im bank, fsis, iema, mema, nipc, nmfs, tdh, usdot
Mammals	956	armadillo, elephant shrews, long-tailed weasel, river otter, weddell seals, wild goat
NFL Players	180	aikman, deion sanders, fred taylor, jamal lewis, raghib ismail, troy vincent
Scientific Journals	265	biometrika, european economic review, nature genetics, neuroscience
Social Issues	210	gender inequality, lack of education, substandard housing, welfare dependency
Writers	5089	bronte sisters, hemingway, kipling, proust, torquato tasso, ungaretti, yeats

Table 3.1: A sample of the open-domain classes and associated instances from [Van Durme and Paşca, 2008].

text [Van Durme and Paşca, 2008], and 2) high-recall extractions from WebTables, a large database of HTML tables mined from the Web [Cafarella et al., 2008]. These two methods were chosen to be representative of two broad classes of extraction sources: free text and structured Web documents.

3.2.1 Extraction from Free Text

Van Durme and Paşca [Van Durme and Paşca, 2008] produce an open-domain set of instance clusters $C \in \mathcal{C}$ that partitions a given set of instances $\mathcal{I}$ using distributional similarity [Lin and Pantel, 2002], and labels using *is-a* patterns [Hearst, 1992]. By filtering the class labels using distributional similarity, a large number of high-precision labeled clusters are extracted. The algorithm proceeds iteratively: at each step, all clusters are tested for label *coherence* and all coherent labels are tested for high cluster *specificity*. Label L is coherent if it is shared by at least $J\%$ of the instances in cluster C, and it is specific if the total number of other clusters $C' \in \mathcal{C}, C' \neq C$ containing instances with label L is less than K. When a cluster is found to match these criteria, it is removed from $\mathcal{C}$ and added to an output set. The procedure terminates when no new clusters can be removed from $\mathcal{C}$.

Table 3.1 shows a few randomly chosen classes and representative instances obtained by this procedure.

3.2.2 Extraction from Structured Text

To expand the instance sets extracted from free text, we use a *table-based extraction* method that mines structured Web data in the form of HTML tables. A significant fraction of the HTML tables in Web pages are assumed to contain coherent lists of instances suitable for extraction. Identifying such tables from scratch is hard, but seed instance lists can be used to identify potentially coherent table columns. In this chapter, we use the WebTables database of around 154 million tables as our structured data source [Cafarella et al., 2008].

We employ a simple ranking scheme for candidate instances in the WebTables corpus $\mathcal{T}$. Each table $\mathbf{T} \in \mathcal{T}$ consists of one or more columns. Each column $g \in \mathbf{T}$ consists of a set of candidate instances $i \in g$ corresponding to row elements. We define the set of unique *seed matches* in g relative to semantic class $C \in \mathcal{C}$ as

$$M_C(g) \overset{\text{def}}{=} \{i \in I(C) : i \in g\}$$

where $I(C)$ denotes the set of instances in seed class C. For each column g, we define its α*-unique class coverage*, that is, the set of classes that have at least α unique seeds in g,

$$Q(g;\alpha) \overset{\text{def}}{=} \{C \in \mathcal{C} : |M_C(g)| \geq \alpha\}.$$

Using M and Q we define a method for scoring columns relative to each class. Intuitively, such a score should take into account not only the number of matches from class C, but also the total number of classes that contribute to Q and their relative overlap. Towards this end, we introduce the scoring function

$$score(C, g; \alpha) \overset{\text{def}}{=} \underbrace{|M_C(g)|}_{\text{seed matches}} \cdot \overbrace{\frac{|M_C(g)|}{|\bigcup_{C' \in Q(g;\alpha)} I(C')|}}^{\text{class coherence}}$$

which is the simplest scoring function combining the number of seed matches with the coherence of the table column. Coherence is a critical notion in WebTables extraction, as some tables contain instances across many diverse seed classes, contributing to extraction noise. The class coherence introduced here also takes into account class overlap; that is, a column containing many semantically similar classes is penalized less than one containing diverse classes.[1] Finally, an extracted instance i is assigned a score relative to class C equal to the sum of all its column scores,

$$score(i, C; \alpha) \overset{\text{def}}{=} \frac{1}{Z_C} \sum_{g \in \mathbf{T}, \mathbf{T} \in \mathcal{T}} score(C, g; \alpha)$$

where Z_C is a normalizing constant set to the maximum score of any instance in class C. This scoring function assigns high rank to instances that occur frequently in columns with many seed matches and high class specificity. The ranked list of extracted instances is post-filtered by removing all instances that occur in less than d unique Internet domains.

In this section, we presented a simple (and heuristic) way to assign classes to instances present in tables. Previously proposed methods such as Bayesian Sets [Ghahramani and Heller, 2006] may be used as an alternative for this task.

3.3 Graph-Based Extraction using Adsorption

To combine the extractions from both free and structured text, we need a representation capable of encoding efficiently all the available information. We chose a graph representation for the following reasons:

- Graphs can represent complicated relationships between classes and instances. For example, an ambiguous instance such as *Michael Jordan* could belong to the class of both *Professors* and *NBA players*. Similarly, an instance may belong to multiple nodes in the hierarchy of classes. For example, *Blue Whales* could belong to both classes *Vertebrates* and *Mammals*, because *Mammals* are a subset of *Vertebrates*.

[1]Note that this scoring function does not take into account class containment: if all seeds are both *wind Instruments* and *instruments*, then the column should assign higher score to the more specific class.

- Extractions from multiple sources, such as Web queries, Web tables, and text patterns can be represented in a single graph.
- Graphs make explicit the potential paths of information propagation that are implicit in the more common local heuristics used for weakly-supervised information extraction. For example, if we know that the instance *Bill Clinton* belongs to both classes *President* and *Politician* then this should be treated as evidence that the class of *President* and *Politician* are related.

Each instance-class pair (i, C) extracted in the first phase (Section 3.2) is represented as a weighted edge in a graph $G = (V, E, W)$, where V is the set of nodes, E is the set of edges and $W : E \to \mathbb{R}^+$ is the weight function which assigns positive weight to each edge. In particular, for each (i, C, w) triple from the set of base extractions, i and C are added to V and (i, C) is added to E, [2] with $W(i, C) = w$. The weight w represents the total score of all extractions with that instance and class. Figure 3.1 illustrates a portion of a sample graph. This simple graph representation could be refined with additional types of nodes and edges, as we discuss in Section 5.7.

In what follows, all nodes are treated in the same way, regardless of whether they represent instances or classes. In particular, all nodes can be assigned class labels. For an instance node, that means that the instance is hypothesized to belong to the class; for a class node, that means that the node's class is hypothesized to be semantically similar to the label's class (Section 3.5).

We now formulate the task of assigning labels to nodes as graph label propagation. We are given a set of instances $\mathcal{I}$ and a set of classes $\mathcal{C}$ represented as nodes in the graph, with connecting edges as described above. We annotate a few instance nodes with labels drawn from $\mathcal{C}$. That is, classes are used both as nodes in the graph and as labels for nodes. There is no necessary alignment between a class node and any of the (class) labels, as the final labels will be assigned by the Adsorption algorithm.

[2]In practice, we use two directed edges, from i to C and from C to i, both with weight w.

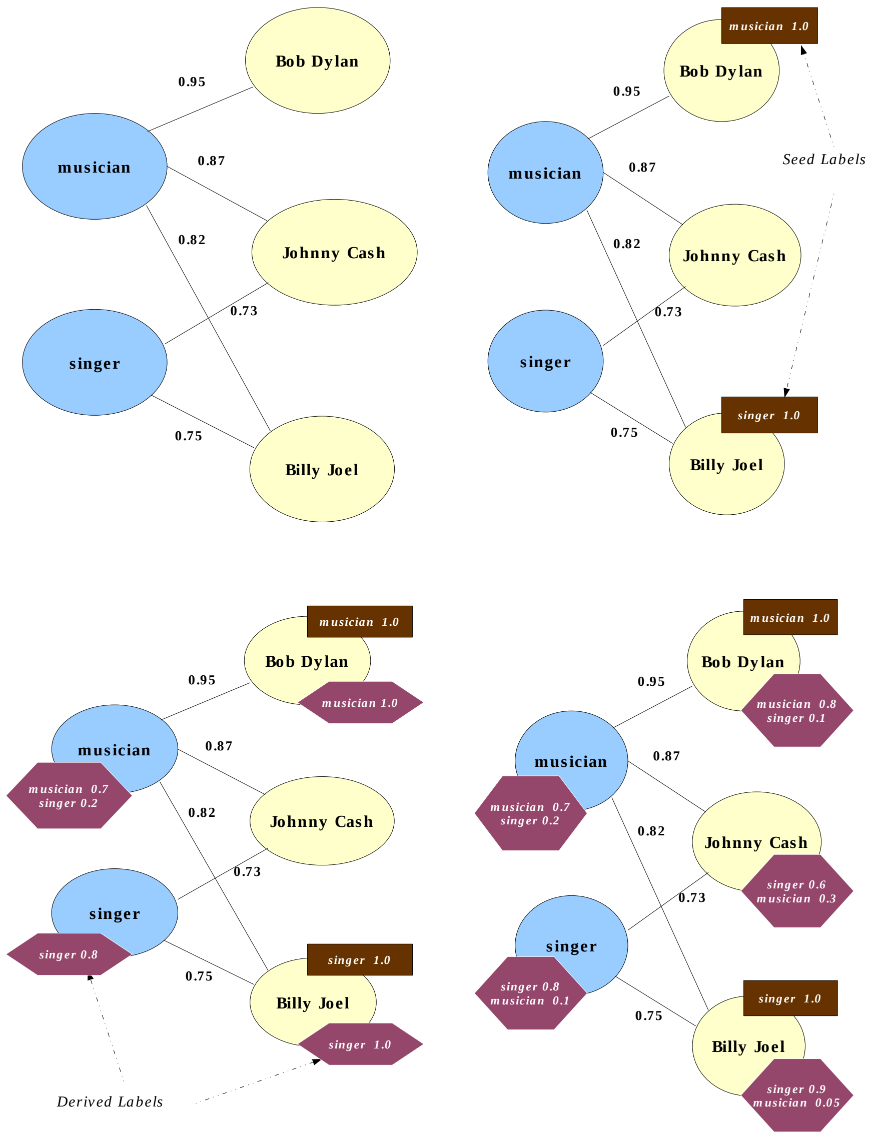


Figure 3.1: Section of a graph during various stages of the Adsorption label propagation algorithm. The initial graph without any labels on nodes is shown in the upper-left corner, while the final graph with labels estimated for all nodes using Adsorption is shown in lower-right corner. For better readability, instance nodes are shaded in yellow while class nodes are shaded in blue.

In order to assign labels to currently unlabeled nodes, the Adsorption label propagation algorithm [Baluja et al., 2008] is now applied to the given graph. Adsorption is a general framework for label propagation, consisting of a few nodes annotated with labels and a rich graph structure containing the universe of all labeled and unlabeled nodes. Adsorption proceeds to label all nodes based on the graph structure, ultimately producing a probability distribution over labels for each node.

More specifically, Adsorption works on a graph $G = (V, E, W)$ and computes for each node v a *label distribution* that represents which labels are more or less appropriate for that node. Several interpretations of Adsorption-type algorithms have appeared in various fields [Azran, 2007, Zhu et al., 2003, Szummer and Jaakkola, 2002, Indyk and Matousek, 2004]. Analysis of the Adsorption algorithm along with its various interpretations are presented in Chapter 4.

Since Adsorption is memoryless, it can scale to large graphs (especially, with limited number of classes) and can be easily parallelized, as described by [Baluja et al., 2008]. We used a MapReduce [Dean and Ghemawat, 2008] based parallel implementation for the experiments in this chapter, using the heuristics from [Baluja et al., 2008] to set the random walk probabilities (see Chapter 4 for details). Adsorption's time complexity is linear in the number of classes propagated, which can be a scalability concern when large number of classes are involved. In practice, this can be speeded up by retaining only top scoring η classes (if any) per node after each class update on a node. We shall get back to such per-node class sparsity constraints in Section 4.6.5. For the experiments in this chapter, we set $\eta = 100$, i.e., a node was allowed to attain a maximum of 100 classes.

3.4 Experiments

3.4.1 Data

As mentioned in Section 4.2, one of the benefits of using Adsorption is that we can combine extractions by different methods from diverse sources into a single framework. To demonstrate this capability, we combine extractions from free-text patterns and from Web

Seed Class	Seed Instances
Book Publishers	millbrook press, academic press, springer verlag, chronicle books, shambhala publications
Federal Agencies	dod, nsf, office of war information, tsa, fema
Mammals	african wild dog, hyaena, hippopotamus, sperm whale, tiger
NFL Players	ike hilliard, isaac bruce, torry holt, jon kitna, jamal lewis
Scientific Journals	american journal of roentgenology, pnas, journal of bacteriology, american economic review, ibm systems journal

Table 3.2: Classes and seeds used to initialize Adsorption.

tables.

Open-domain (instance, class) pairs were extracted by applying the method described by [Van Durme and Paşca, 2008] on a corpus of over 100M English web documents. A total of 924K (instance, class) pairs were extracted, containing 263K unique instances in 9081 classes. We refer to this dataset as A8.

Using A8, an additional 74M unique (instance,class) pairs are extracted from a random 10% of the WebTables data, using the method outlined in Section 3.2.2. For maximum coverage we set $\alpha = 2$ and $d = 2$, resulting in a large, but somewhat noisy collection. We refer to this data set as WT.

3.4.2 Graph Creation

We applied the graph construction scheme described in Section 4.2 on the A8 and WT data combined, resulting in a graph with 1.4M nodes and 75M edges. Since extractions in A8 are not scored, weight of all edges originating from A8 were set at 1[3]. This graph is used in all subsequent experiments.

[3] A8 extractions are assumed to be high-precision and hence we assign them the highest possible weight.

3.5 Evaluation

We evaluated the Adsorption algorithm under two experimental settings. First, we evaluate Adsorption's extraction precision on (instance, class) pairs obtained by Adsorption but not present in A8 (Section 3.5.1). This measures whether Adsorption can add to the A8 extractions at fairly high precision. Second, we measured Adsorption's ability to assign labels to a fixed set of gold instances drawn from various classes (Section 3.5.2).

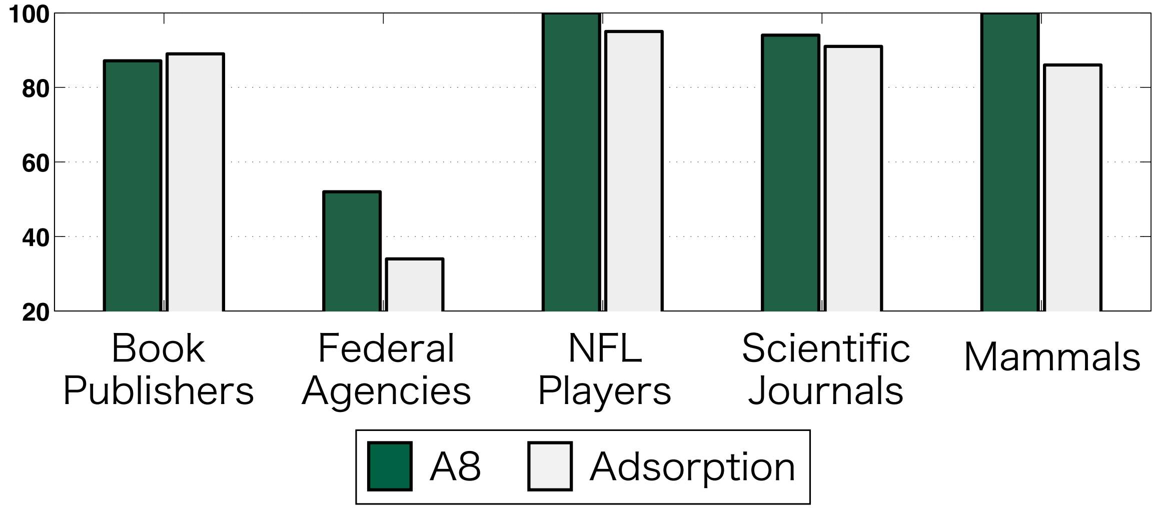


Figure 3.2: Precision at 100 comparisons for five classes in A8 and Adsorption.

3.5.1 Instance Precision

First we manually evaluated precision across five randomly selected classes from A8: Book Publishers, Federal Agencies, NFL Players, Scientific Journals and Mammals. For each class, 5 seed instances were chosen manually to initialize Adsorption. These classes and seeds are shown in Table 3.2. Adsorption was run for each class separately and the resulting ranked extractions were manually evaluated.

Since the extracted instances in A8 are not ranked, we chose 100 random instances (per class) from the A8 extractions to compare to the top 100 instances produced by Ad-

Seed Class	Non-Seed Class Labels Discovered by Adsorption
Book Publishers	small presses, journal publishers, educational publishers, academic publishers, commercial publishers
Federal Agencies	public agencies, governmental agencies, modulation schemes, private sources, technical societies
NFL Players	sports figures, football greats, football players, backs, quarterbacks
Scientific Journals	prestigious journals, peer-reviewed journals, refereed journals, scholarly journals, academic journals
Mammal Species	marine mammal species, whale species, larger mammals, common animals, sea mammals

Table 3.3: Top class labels ranked by their similarity to a given seed class in Adsorption.

sorption. Each of the resulting 500 instance-class pairs (i, C) was presented to two human evaluators, who were asked to evaluate whether the relation "i is a C" was correct or incorrect. The user was also presented with Web search link to verify the results against actual documents. Results from these experiments are presented in Figure 3.2 and Table 3.4. The results in Figure 3.2 show that the A8 extractions have higher precision than Adsorption's extractions. This is not surprising since the A8 system is tuned for high precision. When considering individual evaluation classes, changes in precision scores between A8 and Adsorption vary from a small increase from 87% to 89% for the class Book Publishers, to a significant decrease from 52% to 34% for the class Federal Agencies, with a decrease of 10% as an average over the 5 evaluation classes.

Class	Precision at 100 (non-A8 extractions)
Book Publishers	87.36
Federal Agencies	29.89
NFL Players	94.95
Scientific Journals	90.82
Mammal Species	84.27

Table 3.4: Precision of top 100 Adsorption extractions (for five classes) which were **n**ot present in A8.

Seed Class	Sample of Top Ranked Instances Discovered by Adsorption
Book Publishers	small night shade books, house of anansi press, highwater books, distributed art publishers, copper canyon press
NFL Players	tony gonzales, thabiti davis, taylor stubblefield, ron dixon, rodney hannah
Scientific Journals	journal of physics, nature structural and molecular biology, sciences sociales et santé, kidney and blood pressure research, american journal of physiology–cell physiology

Table 3.5: Random examples of top ranked extractions (for three classes) found by Adsorption which were not present in A8.

Table 3.4 shows the precision of the instances extracted by Adsorption alone, i.e., these instances were not present in A8 extractions. Such an evaluation is important as one of the main motivations of the current work is to increase coverage (recall) of existing high-precision extractors without significantly affecting precision. Results in Table 3.4 show that Adsorption is indeed able to extraction with high precision (in 4 out of 5 cases) new instance-class pairs which were not extracted by the original high-precision extraction set (in this case A8). Examples of a few such pairs are shown in Table 3.5. This is promising as almost all state-of-the-art extraction methods are high-precision and low-recall. The proposed method shows a way to overcome that limitation.

As noted in Section 4.2, Adsorption ignores node type and hence the final ranked extraction may also contain classes along with instances. Thus, in addition to finding new instances for classes, it also finds additional class labels similar to the seed class labels with which Adsorption was run, at no extra cost. Some of the top ranked class labels extracted by Adsorption for the corresponding seed class labels are shown in Table 3.3. To the best of our knowledge, there are no other systems which perform both tasks simultaneously.

3.5.2 Class Label Recall

Next we evaluated each extraction method on its relative ability to assign labels to class instances. For each test instance, the five most probable class labels are collected using each method and the Mean Reciprocal Rank (MRR) is computed relative to a gold standard

Method	MRR (full)	MRR (found only)	# found
A8	0.16	0.47	2718
WT	0.15	0.21	**5747**
AD 1	0.26	0.45	4687
AD 5	0.29	0.48	4687
AD 10	0.30	0.51	4687
AD 25	**0.32**	**0.55**	4687

Table 3.6: Mean-Reciprocal Rank scores of instance class labels over 38 Wordnet classes (WN-gold). MRR (full) refers to evaluation across the entire gold instance set. MRR (found only) computes MRR only on recalled instances.

target set. This target set, WN-gold, consists of the 38 classes in Wordnet containing 100 or more instances.

In order to extract meaningful output from Adsorption, it is provided with a number of labeled seed instances (1, 5, 10 or 25) from each of the 38 test classes. Regardless of the actual number of seeds used as input, all 25 seed instances from each class are removed from the output set from all methods, in order to ensure fair comparison.

The results from this evaluation are summarized in Table 3.6; AD x refers to the adsorption run with x seed instances. Overall, Adsorption exhibits higher MRR than either of the baseline methods, with MRR increasing as the amount of supervision is increased. Due to its high coverage, WT assigns labels to a larger number of the instance in WN-gold than any other method. However, the average rank of the correct class assignment is lower, resulting is lower MRR scores compared to Adsorption. This result highlights Adsorption's ability to effectively combine high-precision, low-recall (A8) extractions with low-precision, high-recall extractions (WT) in a manner that improves *both* precision and coverage.

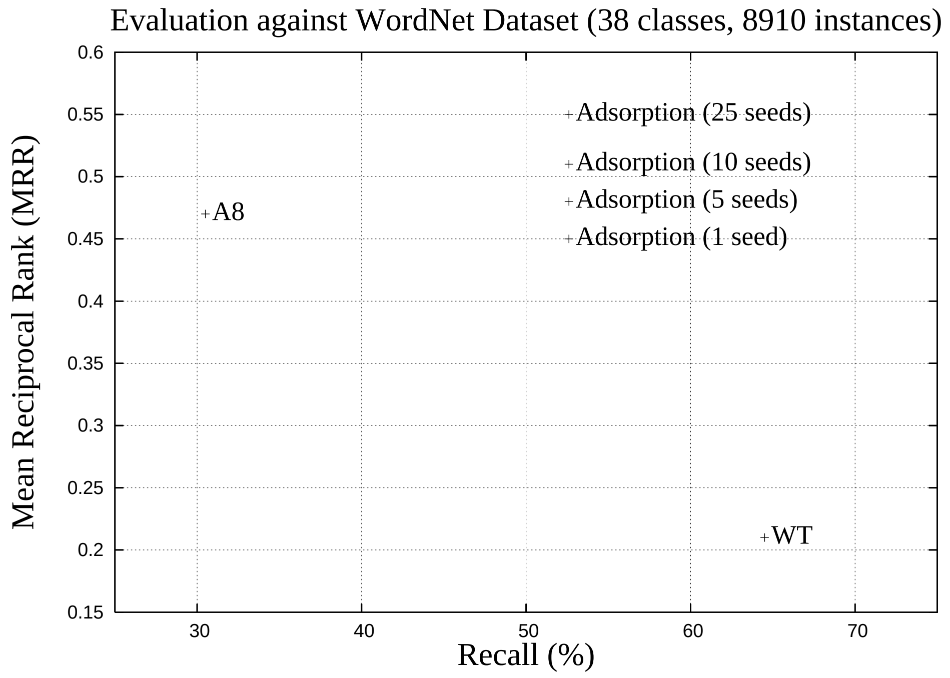


Figure 3.3: Plot of MRR (found only) vs Recall of the data shown in Table 3.6.

3.6 Related Work

Graph-based algorithms for minimally supervised information extraction methods have recently been proposed. For example, SEAL [Wang and Cohen, 2007] is a random walk based method applied over a graph built from entities and relations extracted from semi-structured text. The method presented in this chapter is focused on combining extractions from multiple methods and sources within one framework, with subsequent joint inference. Hence, this proposed method should be interpreted more as a re-ranker and aggregator, rather than an extractor in itself. To this effect, SEAL's output can be one additional input, amongst others, for the proposed method. Also, while SEAL expands one class at a time, the proposed method in this chapter focuses on working with large number of classes at once. Adsorption is known to have a random walk interpretation [Baluja et al., 2008] (also see Chapter 4). From this, we would like to point out that both Adsorption and SEAL have similar inspiration: both are trying to rank nodes in a graph based on similarity of the nodes to a given given set of initial nodes. The re-ranking algorithm of [Bellare et al., 2007] also constructs a graph whose nodes are instances and attributes, as opposed to instances and classes in this chapter. Adsorption can be seen as a generalization of the method proposed in that paper.

3.7 Summary of Chapter

The field of open-domain information extraction has been driven by the growth of Web-accessible data. We have staggering amounts of data from various structured and unstructured sources such as general Web text, online encyclopedias, query logs, web tables, or link anchor texts. Any proposed algorithm to extract information needs to harness several data sources and do it in a robust and scalable manner. The method in this chapter represents a first step towards that goal. In doing so, we achieved the following:

1. Improved coverage relative to a high accuracy instance-class extraction system while maintaining adequate precision.
2. Combined information from two different sources: free text and web tables.
3. Demonstrated a graph-based label propagation algorithm that given as little as five seeds per class achieved good results on a graph with more than a million nodes and 70 million edges.

In the next chapter, we shall take a closer look at the details of the Adsorption algorithm, and also its various extensions.

Chapter 4

Adsorption Algorithm and its Extensions

Some parts of this chapter are based on [Talukdar and Crammer, 2009] and [Talukdar and Pereira, 2010].

4.1 Introduction

In Chapter 3, we used the Adsorption algorithm [Baluja et al., 2008] for large-scale class-instance acquisition from a combination of unstructured as well as structured sources. Adsorption is a recently proposed graph based semi-supervised algorithm which has been successfully used for different tasks, such as recommending YouTube videos to users [Baluja et al., 2008]. Adsorption has many desirable properties: it can perform multi-label classification, it can be parallelized and hence can be scaled to handle large data sets which is of particular importance for semi-supervised algorithms. Even though Adsorption works well in practice, to the best of our knowledge it has never been analyzed before; for example, it is currently unknown whether Adsorption is solving any well defined optimization. Hoping to fill this gap, we make the following contributions in this chapter:

- We analyze the Adsorption algorithm [Baluja et al., 2008] and show that there does not exist an objective function with continuous second partial derivative whose local optimization would be the output of the Adsorption algorithm.
- Motivated by this negative result, we propose a new graph based semi-supervised algorithm (Modified Adsorption, MAD), which is similar to Adsorption and shares its desirable properties, but has important differences.
- We state the learning problem as an optimization problem and develop efficient (iterative) methods to solve it. We also list the conditions under which the optimization algorithm – MAD – is guaranteed to converge.
- The transition to an optimization based learning algorithm provides a flexible and general framework that enables us to specify a variety of requirements. We demonstrate this framework using data with dependent labels, resulting in the Modified Adsorption with Dependent Labels (MADDL) algorithm.
- We report systematic comparison of various graph-based SSL algorithms on different real-world learning tasks, including class-instance acquisition, where we find MAD to be the most effective. We also demonstrate how class-instance acquisition performance in the graph-based SSL setting can be improved by incorporating additional semantic constraints available in independently developed knowledge bases.

4.2 Adsorption Algorithm

Adsorption [Baluja et al., 2008] is a general algorithmic framework for transductive learning where the learner is often given a small set of labeled examples and a very large set of unlabeled examples. The goal is to label all the unlabeled examples, and possibly, under the assumption of label-noise, also to relabel the labeled examples.

As many other related algorithms [Zhu et al., 2003], [Szummer and Jaakkola, 2002], [Indyk and Matousek, 2004], the Adsorption algorithm assumes that the learning problem is given in a *graph* form, where examples or instances are represented as nodes or vertices and edges code *similarity* between examples. Some of the nodes are associated with a pre-specified label, which is correct in the noise-free case, or can be subject to label-noise. Additional information can be given in the form of *weights* over the labels. Adsorption propagates label-information from the labeled examples to the entire set of vertices via the edges. The labeling is represented using a non-negative score values for each label, high value for some label indicates high-association of a vertex (or its corresponding instance) with that label. If the scores are additively normalized they can be thought of as a conditional distribution over the labels given the node (or example) identity.

More formally, Adsorption is fed with an undirected graph $G = (V, E, W)$, where a node $v \in V$ corresponds to examples, an edge $e = (a, b) \in V \times V$ indicates that the label of the two vertices $a, b \in V$ should be similar and the weight $W_{ab} \in \mathbb{R}_+$ measures the strength of this similarity.

We denote the total number of examples or vertices by $n = |V|$, by n_l the number of examples for which we have prior knowledge of their label and by n_u the number of unlabeled examples to be labeled. Clearly $n_l + n_u = n$. Let $\mathcal{L}$ be the set of possible labels, their number is denoted by $m = |\mathcal{L}|$ and with out loss of generality we assume that the possible labels are $\mathcal{L} = \{1 \ldots m\}$. Each instance $v \in V$ is associated with two column-vectors $\mathbf{Y}_v, \hat{\mathbf{Y}}_v \in \mathbb{R}_+^m$. The $1th$ element of the vector $\mathbf{Y}_v$ encodes the prior knowledge for vertex v. The higher the value of $\mathbf{Y}_{vl}$ the stronger we a-priori believe that the label of v should be $l \in \mathcal{L}$ and a value of zero $\mathbf{Y}_{vl} = 0$ indicates no prior about the label l for vertex v. If all the elements of $\mathbf{Y}_v$ are zero, that is $\mathbf{Y}_{vl} = 0$ for $l = 1 \ldots m$ then the vertex v is labeled. The second vector $\hat{\mathbf{Y}}_v \in \mathbb{R}_+^m$ is the output of the algorithm, using similar semantics as $\mathbf{Y}_v$. For example, a high value of $\hat{\mathbf{Y}}_{v,l}$ indicates that the algorithm believes that the vertex v should have the label l. Note the algorithm does not necessarily enforce the equality $\hat{\mathbf{Y}}_v = \mathbf{Y}_v$. We denote by $\mathbf{Y}, \hat{\mathbf{Y}} \in \mathbb{R}_+^{n \times m}$ the matrices whose rows are $\mathbf{Y}_v$ and $\hat{\mathbf{Y}}_v$ respectively. Finally, we denote by $\mathbf{0}_d$ the all-zeros row vector of dimension d.

4.2.1 Random-Walk View

The Adsorption algorithm can be viewed as a controlled random what over the graph G. The control is formalized via three possible actions denoted by *inj, cont, abnd* with pre-defined probabilities $p_v^{inj}, p_v^{cont}, p_v^{abnd} \geq 0$ per vertex $v \in V$. Clearly their sum is unit $p_v^{inj} + p_v^{cont} + p_v^{abnd} = 1$. To label a some vertex $v \in V$, either unlabeled or labeled, we make a random walk start at v facing three options: with probability p_v^{inj} the random-walk stops and returns (i.e. *inject*) the pre-defined vector information $\mathbf{Y}_v$. We constrain $p_v^{inj} = 0$ for unlabeled vertices v. Second, with probability p_v^{abnd} the random-walk *abandons* the labeling process and returns the all-zeros vector $\mathbf{0}_m$. Third, with probability p_v^{cont} the random *continues* to one of v's neighbors v' with probability proportional to $W_{vv'} \geq 0$. Note that by definition $W_{v'v} = 0$ if $(v, v') \notin V$.

We can summarize the above process with the following set of equations. The transition probabilities are,

$$\Pr\left[v'|v\right] = \begin{cases} \frac{W_{v'v}}{\displaystyle\sum_{u\,:\,(u,v)\in E} W_{uv}} & (v', v) \in E \\ 0 & \text{otherwise} \end{cases} . \tag{4.1}$$

The (expected) score $\hat{\mathbf{Y}}_v$ for note $v \in V$ is given by,

$$\hat{\mathbf{Y}}_v = \left(p_v^{inj} \times \mathbf{Y}_v\right) + \left(p_v^{cont} \times \sum_{v'\,:\,(v',v)\in E} \Pr\left[v'|v\right] \hat{\mathbf{Y}}_{v'}\right) + p_v^{abnd} \times \mathbf{0}_m . \tag{4.2}$$

4.2.2 Averaging View

For this view we add a designated symbol called the dummy label and denoted by $\nu \notin V$. This additional label explicitly encodes ignorance about the correct label and it means that a dummy label can be used instead. Explicitly, we add an additional row to all the vectors defined above, and have now that $\mathbf{Y}_v, \hat{\mathbf{Y}}_v \in \mathbb{R}_+^{m+1}$ and $\mathbf{Y}, \hat{\mathbf{Y}} \in \mathbb{R}_+^{n\times(m+1)}$. We set $\mathbf{Y}_{v\nu} = 0$, that is, a-priori no vertex is associated with the dummy label, and replace the zero vector $\mathbf{0}_m$ with the vector $\mathbf{r} \in \mathbb{R}_+^{m+1}$ where $\mathbf{r}_l = 0$ for $l \neq \nu$ and $\mathbf{r}_\nu = 1$. In words, if the random-walk

Algorithm 1: Adsorption Algorithm

```
Input:
  - Graph:            $G = (V, E, W)$
  - Prior labeling:   $\mathbf{Y}_v \in \mathbb{R}^{m+1}$ for $v \in V$
  - Probabilities:    $p_v^{inj}, p_v^{cont}, p_v^{abnd}$ for $v \in V$
Output:
  - Label Scores:     $\hat{\mathbf{Y}}_v$ for $v \in V$
1: $\hat{\mathbf{Y}}_v \leftarrow \mathbf{Y}_v$ for $v \in V$ {Initialization}
2:
3: repeat
4:     $D_v \leftarrow \frac{\sum_u W_{uv} \hat{\mathbf{Y}}_u}{\sum_u W_{uv}}$ for $v \in V$
5:     for all $v \in V$ do
6:         $\hat{\mathbf{Y}}_v \leftarrow p_v^{inj} \times \mathbf{Y}_v + p_v^{cont} \times D_v + p_v^{abnd} \times \mathbf{r}$
7:     end for
8: until convergence
```

is abandoned, then the corresponding labeling vector is zero for all true labels in $\mathcal{L}$, and an arbitrary value of unit for the dummy label ν. This way, there is always a non-negative score for at least one label, either the real one or the dummy label.

The averaging view then defines a set of *fixed-point* equations to update the predicted labels. A summary of the equations appears in Algorithm 6. The algorithm is run until convergence. Alternatively, it can be run until the label distribution on each node ceases to change within some tolerance value; or it can be run for a fixed number of iterations, which is what we used in practice. Also, since Adsorption is memoryless, it scales to tens of millions of nodes with dense edges and can be easily parallelized [Baluja et al., 2008].

Baluja et. al. [Baluja et al., 2008] show that up to the additional dummy label, these two views are equivalent. It remains to specify the values of p_v^{inj}, p_v^{cont} and p_v^{abnd}. For the experiments reported in Section 6.5 we set their value using the following heuristics (adapted from Baluja et. al. [Baluja et al., 2008]) which depends on a parameter $\beta = 2$.

For each node v we define two quantities: c_v and d_v and will define

$$p_v^{cont} \propto c_v \quad ; \quad p_v^{inj} \propto d_v \ .$$

The first quantity $c_v \in [0, 1]$ is monotonically decreasing with the number of neighbors for node v in the graph G. Intuitively, the higher the value of c_v, the lower the number of neighbors of vertex v, and hence higher the information they contain about the labeling of v. The other quantity $d_v \geq 0$ is monotonically increasing with the entropy (for labeled vertices), and in this case we prefer to use the prior-information rather than the computed quantities from the neighbors.

Specifically we first compute the entropy of the transition probabilities for each node,

$$H[v] = -\sum_u \mathrm{P}r\left[u|v\right] \log \mathrm{P}r\left[u|v\right] \ ,$$

and then pass it through the following monotonically decreasing function,

$$f(x) = \frac{\log \beta}{\log(\beta \ + \ e^x))} \ .$$

Note that $f(0) = log(\beta)/\log(\beta + 1)$ and that $f(x)$ goes to zero, as x goes to infinity. We define,

$$c_v = f\left(H[v]\right) \ .$$

Next we define,

$$d_v = \begin{cases} (1 - c_v) \times \sqrt{H[v]} & \text{the vertex v is labeled} \\ 0 & \text{the vertex v is unlabled} \end{cases}$$

Finally, to ensure proper normalization of p_v^{cont}, p_v^{inj} and p_v^{abnd}, we define,

$$z_v = \max(c_v + d_v, 1) \ ,$$

and

$$p_v^{cont} = \frac{c_v}{z_v} \quad ; \quad p_v^{inj} = \frac{d_v}{z_v} \quad ; \quad p_v^{abnd} = 1 - p_v^{cont} - p_v^{abnd} \ .$$

Thus, abandonment occurs only when the continuation and injection probabilities are low enough.

We note that neighborhood entropy based estimation of the three random walk probabilities above – p_v^{inj}, p_v^{cont}, and p_v^{abnd} – is not necessarily the only possible way out. Depending on the domain and the classification task at hand, other appropriate estimation choices may be used. However, what is interesting is the fact that, through these three per-node probabilities, Adsorption allows us to control the amount of information flowing through each node, and thereby facilitating greater control during label propagation.

4.3 Analysis of the Adsorption Algorithm

Our next goal is to find an objective function that the Adsorption algorithm minimizes. Our starting point is line 6 of Algorithm 6. We note that when the algorithm converges, both sides of the assignments operator equal each other before the assignment takes place. Thus when the algorithm terminates we have for all $v \in V$:

$$\hat{\mathbf{Y}}_v = p_v^{inj} \times \mathbf{Y}_v + p_v^{cont} \times \frac{1}{N_v} \sum_u W_{uv} \hat{\mathbf{Y}}_u + p_v^{abnd} \times \mathbf{r} \ ,$$

where

$$N_v = \sum_{v'} W_{v'v} \ .$$

The last set of equalities is equivalent to,

$$G_v = 0 \text{ for } v \in V \ , \tag{4.3}$$

where we define,

$$G_v \left(\{\hat{\mathbf{Y}}_u\}_{u \in V} \right) = \left[p_v^{inj} \times \mathbf{Y}_v + p_v^{cont} \times \frac{1}{N_v} \sum_u W_{uv} \hat{\mathbf{Y}}_u + p_v^{abnd} \times \mathbf{r} \right] - \hat{\mathbf{Y}}_v \ .$$

Now, if the Adsorption algorithm was minimizing some objective function (denoted by $\mathcal{Q}\left(\{\hat{\mathbf{Y}}_u\}_{u \in V} \right)$) the termination condition of Eq. (4.3) was in fact a condition on the vector of its partial derivatives where we would identify

$$G_v = \frac{\partial}{\partial \hat{\mathbf{Y}}_v} \mathcal{Q} \; . \tag{4.4}$$

Since the functions G_v are linear (and thus have continuous derivatives), necessary and sufficient conditions for the existence of a function $\mathcal{Q}$ such that (4.4) holds is that the derivatives of G_v are symmetric [Katz, 1979], that is,

$$\frac{\partial}{\partial \hat{\mathbf{Y}}_v} G_u = \frac{\partial}{\partial \hat{\mathbf{Y}}_u} G_v, \;\; \forall \, u, v$$

Computing and comparing the derivatives we get,

$$\frac{\partial}{\partial \hat{\mathbf{Y}}_u} G_v = p_v^{cont} \left(\frac{W_{uv}}{N_v} - \delta_{u,v} \right) \neq p_u^{cont} \left(\frac{W_{vu}}{N_u} - \delta_{u,v} \right) = \frac{\partial}{\partial \hat{\mathbf{Y}}_v} G_u \; , \tag{4.5}$$

where $\delta_{u,v}$ is an indicator variable with $\delta_{u,v} = 1$ when $u = v$, and 0 otherwise. Please note that equation (4.5) is true since in general $N_u \neq N_v$ and $p_v^{cont} \neq p_u^{cont}$. To conclude we proved the following proposition:

Proposition 1 *There does not exist a function $\mathcal{Q}$ with continuous second partial derivatives such that the Adsorption algorithm converges when gradient of $\mathcal{Q}$ is equal to zero.*

In other words, we searched for a (well-behaved) function $\mathcal{Q}$ such that its local optimal would be the output of the Adsorption algorithm, and showed that this search will always fail. Please note that Proposition 1 is applicable only to the set of functions with continuous second partial derivative, and that there *may* exist some function outside this set, whose local optimization is the output of the Adsorption algorithm. Search for such a function is an interesting avenue for future work.

We use the negative result in Proposition 1 to define a new algorithm, which builds on the Adsorption and is optimizing a function of the unknowns $\hat{\mathbf{Y}}_v$ for $v \in V$.

4.4 New Algorithm: Modified Adsorption (MAD)

Our starting point is Sec. 4.2.2 where we assume to have been given a weighted-graph $G = (V, E, W)$ and a matrix $\mathbf{Y} \in \mathbb{R}_+^{n \times (m+1)}$ and are seeking for a labeling-matrix $\hat{\mathbf{Y}} \in$

$\mathbb{R}_+^{n\times(m+1)}$. In this section it is more convenient to decompose the matrices $\mathbf{Y}$ and $\hat{\mathbf{Y}}$ into their columns, rather than rows. Specifically, we denote by $\mathbf{Y}_l \in \mathbb{R}_+^n$ the lth column of $\mathbf{Y}$ and similarly by $\hat{\mathbf{Y}}_l \in \mathbb{R}_+^n$ the lth column of $\hat{\mathbf{Y}}$. (We distinguish the rows and columns of the matrices $\mathbf{Y}$ and $\hat{\mathbf{Y}}$ using their index, the columns are indexed with the label index l, while the rows are indexed are with a vertex index v, u).

We build on previous research in the area of graph-based semi-supervised learning (SSL) [Zhu et al., 2003], [Subramanya and Bilmes, 2008] and construct an objective that reflects three requirements as follows. First, for the labeled vertices, we would like the output of the algorithm to be close to the a-priori given labels, that is $\mathbf{Y}_v \approx \hat{\mathbf{Y}}_v$. Second, for pair of vertices that are close according to the input graph, we would like their labeling to be close, that is $\hat{\mathbf{Y}}_u \approx \hat{\mathbf{Y}}_v$ if W_{uv} is large. Third, we want the output to be as uninformative as possible, this serves as additional regularization, that is $\hat{\mathbf{Y}}_v \approx \mathbf{r}$. We now further develop the objective in light of the three requirements.

We use the Euclidean distance to measure discrepancy between two quantities, and start with the first requirement above,

$$\begin{aligned}\sum_v p_v^{inj} \sum_l \left(\mathbf{Y}_{vl} - \hat{\mathbf{Y}}_{vl}\right)^2 = & \quad \sum_l \sum_v p_v^{inj} \left(\mathbf{Y}_{vl} - \hat{\mathbf{Y}}_{vl}\right)^2 \\ = & \quad \sum_l \left(\mathbf{Y}_l - \hat{\mathbf{Y}}_l\right)^\top \mathbf{S} \left(\mathbf{Y}_l - \hat{\mathbf{Y}}_l\right) \ ,\end{aligned}$$

where we defined the diagonal matrix $\mathbf{S} \in \mathbb{R}^{n\times n}$ and $\mathbf{S}_{vv} = p_v^{inj}$ if vertex v is labeled and $\mathbf{S}_{vv} = 0$ otherwise. The matrix $\mathbf{S}$ captures the intuition that for different vertices we enforce the labeling of the algorithm to match the a-priori labeling with different extent.

Next, we modify the similarity weight between vertices to take into account the difference in degree of various vertices. In particular we define $\mathbf{W}'_{vu} = p_v^{cont} \times \mathbf{W}_{vu}$. Thus, a vertex u will *not* be similar to a vertex v if either the input weights $\mathbf{W}_{vu}$ are low or the vertex v has a large-degree (p_v^{cont} is low). We write the second requirement as,

$$\sum_{v,u} \mathbf{W}'_{vu} \left\| \hat{\mathbf{Y}}_v - \hat{\mathbf{Y}}_u \right\|^2 = \sum_{v,u} \mathbf{W}'_{vu} \sum_l \left(\hat{\mathbf{Y}}_{vl} - \hat{\mathbf{Y}}_{ul} \right)^2 = \sum_l \sum_{v,u} \mathbf{W}'_{vu} \left(\hat{\mathbf{Y}}_{vl} - \hat{\mathbf{Y}}_{ul} \right)^2$$

$$= \sum_l \sum_v \left(\sum_u \mathbf{W}'_{vu} \right) \|\hat{\mathbf{Y}}_{vl}\|^2 + \sum_l \sum_u \left(\sum_v \mathbf{W}'_{vu} \right) \|\hat{\mathbf{Y}}_{ul}\|^2 - 2 \sum_l \sum_{u,v} \mathbf{W}'_{vu} \hat{\mathbf{Y}}_{ul} \hat{\mathbf{Y}}_{vl}$$

$$= \sum_l \hat{\mathbf{Y}}_l^\top \mathbf{L} \hat{\mathbf{Y}}_l \ ,$$

where,

$$\mathbf{L} = \mathbf{D} + \bar{\mathbf{D}} - \mathbf{W}' - \mathbf{W}'^\top,$$

and $\mathbf{D}$ and $\bar{\mathbf{D}}$ are $n \times n$ diagonal matrices with

$$\mathbf{D}_{uu} = \sum_v \mathbf{W}'_{vu} \quad , \quad \bar{\mathbf{D}}_{vv} = \sum_u \mathbf{W}'_{vu} \ .$$

Finally we define the matrix $\mathbf{R} \in \mathbb{R}_+^{n \times (m+1)}$ where the vth row of $\mathbf{R}$ equals $p_v^{abnd} \times \mathbf{r}$ defined above. In other words the first m columns of $\mathbf{R}$ equal zero, and the last m+1th column equal the elements of p_v^{abnd}. The third requirement above is thus written as,

$$\sum_{vl} \left(\hat{\mathbf{Y}}_{vl} - \mathbf{R}_{vl} \right)^2 = \sum_l \left\| \hat{\mathbf{Y}}_l - \mathbf{R}_l \right\|^2 \ .$$

We now combine the three terms above into a single objective, giving to each term a different importance using the weights $\mu_1, \mu_2, \mu_3 > 0$

$$C(\hat{\mathbf{Y}}) \ = \ \sum_l \left[\mu_1 \left(\mathbf{Y}_l - \hat{\mathbf{Y}}_l \right)^\top \mathbf{S} \left(\mathbf{Y}_l - \hat{\mathbf{Y}}_l \right) \ + \ \mu_2 \hat{\mathbf{Y}}_l^T \, \mathbf{L} \, \hat{\mathbf{Y}}_l \ + \ \mu_3 \left\| \hat{\mathbf{Y}}_l - \mathbf{R}_l \right\|^2 \right] \quad (4.6)$$

We remind the reader that $\hat{\mathbf{Y}}_l, \mathbf{Y}_l, \mathbf{R}_l$ are the lth columns (each of size $1 \times n$) of the matrices $\hat{\mathbf{Y}}, \mathbf{Y}$ and $\mathbf{R}$ (resp.). Note that each term in the sum can be optimized independently.

4.4.1 Solving the Optimization Problem

We now develop an algorithm to optimize (4.6) similar to the quadratic cost criteria [Bengio et al., 2007]. Differentiating Equation 4.6 w.r.t. $\hat{\mathbf{Y}}_l$ we get,

$$\begin{aligned} \frac{1}{2} \frac{\delta C(\hat{\mathbf{Y}}_)}{\delta \hat{\mathbf{Y}}_l} &= \mu_1 \mathbf{S}(\hat{\mathbf{Y}}_l - \mathbf{Y}_l) + \mu_2 \mathbf{L} \hat{\mathbf{Y}}_l + \mu_3 (\hat{\mathbf{Y}}_l - \mathbf{R}_l) \\ &= (\mu_1 \mathbf{S} + \mu_2 \mathbf{L} + \mu_3 \mathbf{I}) \hat{\mathbf{Y}}_l - (\mu_1 \mathbf{S} \mathbf{Y}_l + \mu_3 \mathbf{R}_l) \ . \end{aligned} \quad (4.7)$$

Differentiating once more we get,

$$\frac{1}{2}\frac{\delta C(\hat{\mathbf{Y}})}{\delta \hat{\mathbf{Y}}_l \delta \hat{\mathbf{Y}}_l} = \mu_1 \mathbf{S} + \mu_2 \mathbf{L} + \mu_3 \mathbf{I} \ ,$$

and since both $\mathbf{S}$ and $\mathbf{L}$ are symmetric and positive semidefinite matrices (PSD), we get that the Hessian (Equation 4.8) is PSD as well, since positive semi-definiteness is preserved under non-negative scalar multiplication and addition. Hence, we get the optimal minima is obtained by setting the first derivative (i.e. Equation (4.7)) to 0, and we get,

$$(\mu_1 \mathbf{S} + \mu_2 \mathbf{L} + \mu_3 \mathbf{I})\ \hat{\mathbf{Y}}_l = (\mu_1 \mathbf{S}\mathbf{Y}_l + \mu_3 \mathbf{R}_l) \ .$$

Hence, the new labels ($\hat{\mathbf{Y}}$) can be obtained by a matrix inversion followed by matrix multiplication. However, this can be quite expensive when large matrices are involved. A more efficient way to obtain the new label scores is to solve a set of linear equations using Jacobi iteration [Saad, 2003], which we describe in Section 4.4.2 (also see [Bengio et al., 2007]).

4.4.2 Jacobi Method

Given the following linear system (in x)

$$\mathbf{M}x = b$$

the Jacobi iterative algorithm defines the approximate solution at the (t+1)th iteration given the solution at time tth iteration as follows,

$$x_i^{(t+1)} = \frac{1}{\mathbf{M}_{ii}} \left(b - \sum_{j \neq i} \mathbf{M}_{ij} x_j^{(t)} \right) \ . \tag{4.8}$$

We apply the iterative algorithm to our problem by substituting $x = \hat{\mathbf{Y}}_l$, $\mathbf{M} = \mu_1 \mathbf{S} + \mu_2 \mathbf{L} + \mu_3 \mathbf{I}$ and $b = \mu_1 \mathbf{S}\mathbf{Y}_l + \mu_3 \mathbf{R}_l$ in (4.8),

$$\hat{\mathbf{Y}}_{vl}^{(t+1)} = \frac{1}{\mathbf{M}_{vv}} \left(\mu_1 (\mathbf{S}\mathbf{Y}_l)_v + \mu_3 \mathbf{R}_{vl} - \sum_{u \neq v} \mathbf{M}_{vu} \hat{\mathbf{Y}}_{ul}^{(t)} \right) \tag{4.9}$$

Let us compute the values of $(\mathbf{S}\mathbf{Y}_l)_i$, $\mathbf{M}_{ij(j\neq i)}$ and $\mathbf{M}_{ii}$. First,

$$\mathbf{M}_{vu(v\neq u)} = \mu_1 \mathbf{S}_{vu} + \mu_2 \mathbf{L}_{vu} + \mu_3 \mathbf{I}_{vu} \ .$$

Note that since $\mathbf{S}$ and $\mathbf{I}$ are diagonal, we have that $\mathbf{S}_{vu} = 0$ and $\mathbf{I}_{vu} = 0$ for $u \neq v$. Substituting the value of $\mathbf{L}$ we get,

$$\mathbf{M}_{vu(v\neq u)} = \mu_2 \mathbf{L}_{vu} = \mu_2 \left(\mathbf{D}_{vu} + \bar{\mathbf{D}}_{vu} - \mathbf{W}'_{vu} - \mathbf{W}'_{uv} \right) \ ,$$

and as before the matrices $\mathbf{D}$ and $\bar{\mathbf{D}}$ are diagonal and thus $\mathbf{D}_{vu} + \bar{\mathbf{D}}_{vu} = 0$. Finally, substituting the values of $\mathbf{W}'_{vu}$ and $\mathbf{W}'_{uv}$ we get,

$$\mathbf{M}_{vu(v\neq u)} = -\mu_2 \times \left(p_v^{cont} \times \mathbf{W}_{vu} + p_u^{cont} \times \mathbf{W}_{uv}\right) \ . \quad (4.10)$$

We now compute the second quantity,

$$(\mathbf{S}\mathbf{Y}_l)_{vu} = \mathbf{S}_{vv}\mathbf{Y}_{vv} + \sum_{t\neq v} \mathbf{S}_{vt}\mathbf{Y}_{tu} = p_v^{inj} \times \mathbf{Y}_{vv} \ ,$$

where the second term in the first line equals zero since $\mathbf{S}$ is diagonal. Finally, the third term,

$$\begin{aligned} \mathbf{M}_{vv} &= \mu_1 \mathbf{S}_{vv} + \mu_2 \mathbf{L}_{vv} + \mu_3 \mathbf{I}_{vv} \\ &= \mu_1 \times p_v^{inj} + \mu_2 (\mathbf{D}_{vv} - \mathbf{W}'_{vv}) + \mu_3 \\ &= \mu_1 \times p_v^{inj} + \mu_2 \times p_v^{cont} \times \sum_{u\neq v} \mathbf{W}_{vu} + \mu_3 \ . \end{aligned}$$

Plugging the above equations into (4.9) and using the fact that the diagonal elements of $\mathbf{W}$ are zero, we get,

$$\hat{\mathbf{Y}}_v^{(t+1)} = \frac{1}{\mathbf{M}_{vv}} \left(\mu_1 p_v^{inj}\, \mathbf{Y}_v + \mu_2 \sum_u \left(p_v^{cont}\mathbf{W}_{vu} + p_u^{cont}\mathbf{W}_{uv}\right) \hat{\mathbf{Y}}_u^{(t)} + \mu_3 p_v^{abnd}\, \mathbf{R}_v \right) \quad (4.11)$$

We call the new algorithm MAD (for Modified-Adsorption) and it is summarized in Algorithm 2. Please note that for a graph G which is invariant to permutations of the vertices, along with $\mu_1 = 2\mu_2 = \mu_3 = 1$, Adsorption and MAD coincide. Also, both algorithms have the same time and space complexities.

Algorithm 2: MAD Algorithm

```
Input:
  - Graph:            $G = (V, E, W)$
  - Prior labeling:   $\mathbf{Y}_v \in \mathbb{R}^{m+1}$ for $v \in V$
  - Probabilities:    $p_v^{inj}, p_v^{cont}, p_v^{abnd}$ for $v \in V$
Output:
  - Label Scores:     $\hat{\mathbf{Y}}_v$ for $v \in V$
1: $\hat{\mathbf{Y}}_v \leftarrow \mathbf{Y}_v$ for $v \in V$ {Initialization}
2: $\mathbf{M}_{vv} \leftarrow \mu_1 \times p_v^{inj} + \mu_2 \times p_v^{cont} \times \sum_u \mathbf{W}_{vu} + \mu_3$
3: repeat
4:     $D_v \leftarrow \sum_u \left(p_v^{cont}\mathbf{W}_{vu} + p_u^{cont}\mathbf{W}_{uv}\right)\hat{\mathbf{Y}}_u$
5:     for all $v \in V$ do
6:         $\hat{\mathbf{Y}}_v \leftarrow \frac{1}{\mathbf{M}_{vv}}\left(\mu_1 \times p_v^{inj} \times \mathbf{Y}_v + \mu_2 \times D_v + \mu_3 \times p_v^{abnd} \times \mathbf{R}_v\right)$
7:     end for
8: until convergence
```

4.4.3 Convergence

A sufficient condition for the iterative process of Equation (4.8) to converge is that $\mathbf{M}$ is strictly diagonally dominant [Saad, 2003], that is if,

$$|\mathbf{M}_{vv}| > \sum_{u \neq v} |\mathbf{M}_{vu}| \quad \text{for all values of } v$$

We have,

$$\begin{aligned}
|\mathbf{M}_{vv}| - \sum_{u \neq v} |\mathbf{M}_{vu}| &= \mu_1 \times p_v^{inj} + \mu_2 \times \sum_{u \neq v}\left(p_v^{cont} \times \mathbf{W}_{vu} + p_u^{cont} \times \mathbf{W}_{uv}\right) + \mu_3 - \\
&\qquad \mu_2 \times \sum_{u \neq v}\left(p_v^{cont} \times \mathbf{W}_{vu} + p_u^{cont} \times \mathbf{W}_{uv}\right) \\
&= \mu_1 \times p_v^{inj} + \mu_3 \qquad (4.12)
\end{aligned}$$

Note that $p_v^{inj} \geq 0$ for all v and that μ_3 is a free parameter in (4.12). Thus we can guarantee a strict diagonal dominance (and hence convergence) by setting $\mu_3 > 0$.

4.5 Modified Adsorption with Dependent Labels (MADDL)

In many machine learning tasks, labels are not mutually exclusive. For example, in hierarchical classification, labels are organized in a tree. In open-domain Information Extraction, labels are extracted from text itself without any subsequent filtering. This often leads to dependent labels (e.g. synonyms). In this section, we extend the MAD algorithm to handle dependence among labels. Specifically, we shall additional terms to the objective for each pair of dependent labels. Let C be a $m \times m$ matrix where m is the number of labels as before. Each entry, $C_{ll'}$, of this matrix C represents the dependence or similarity among the labels l and l'. By encoding dependence in this pairwise fashion, we can encode dependencies among labels represented as arbitrary graphs. The extended objective is shown in Equation 4.13.

$$C(\hat{\mathbf{Y}}) \;=\; \sum_l \Bigg[\mu_1 \left(\mathbf{Y}_l - \hat{\mathbf{Y}}_l\right)^\top \mathbf{S} \left(\mathbf{Y}_l - \hat{\mathbf{Y}}_l\right) \;+\; \mu_2 \hat{\mathbf{Y}}_l^T \, \mathbf{L} \, \hat{\mathbf{Y}}_l \;+\; \mu_3 \left\|\hat{\mathbf{Y}}_l - \mathbf{R}_l\right\| \\ + \mu_4 \sum_i \sum_{l,l'} C_{ll'} (\hat{\mathbf{Y}}_{il} - \hat{\mathbf{Y}}_{il'})^2 \Bigg] \quad (4.13)$$

The last term in Equation 4.13 penalizes the algorithm if similar labels (as determined by the matrix C) are assigned different scores, with severity of the penalty controlled by μ_4.

Label Graph: This additional regularization term in Equation 4.13 can be thought of as a Laplacian smoothness penalty over a *label graph*, as opposed to the *instance graph* we have concentrated on so far, where each node is a label (total $m+1$), with edges connecting similar labels and the edge weight representing the degree of similarity between the pair of labels. In other words, C is the edge weight matrix of this *label graph*. The labels estimated by MADDL over this *label graph* is given by $\hat{\mathbf{Y}}^\top$, where $\hat{\mathbf{Y}}$ is the estimated label matrix in the *instance graph*. Since $\hat{\mathbf{Y}}$ is of size $n \times (m+1)$, $\hat{\mathbf{Y}}^\top$ is of size $(m+1) \times n$. Hence, for each node in the original *instance graph*, there is a corresponding label which is propagated over the *label graph.* We note that there is no direct relationship between the edge weight matrices of the *instance graph* (W) and the *label graph* (C).

Now, analyzing the objective in Equation 4.13 in the manner outlined in Section 4.4, we arrive at the update rule shown in Equation 4.14.

$$\begin{aligned}\hat{\mathbf{Y}}_{vl}^{(t+1)} &= \frac{1}{\mathbf{M}_{vv}^{l}} \Bigg(\mu_1 p_v^{inj}\, \mathbf{Y}_{vl} + \mu_2 \sum_u \left(p_v^{cont} \mathbf{W}_{vu} + p_u^{cont} \mathbf{W}_{uv} \right) \hat{\mathbf{Y}}_{ul}^{(t)} + \\ &\qquad \mu_3 p_v^{abnd}\, \mathbf{R}_{vl} + \mu_4 \sum_{l'} C_{ll'} \hat{\mathbf{Y}}_{il'} \Bigg) \end{aligned} \tag{4.14}$$

where,

$$\mathbf{M}_{vv}^{l} = \mu_1 \times p_i^{inj} + \mu_2 \times p_i^{cont} \times \sum_{j \neq i} \mathbf{W}_{ij} + \mu_3 + \mu_4 \sum_{l'} C_{ll'}$$

Replacing Line 6 in MAD (Algorithm 2) with Equation 4.14, we end up with a new algorithm: Modified Adsorption with Dependent Labels (MADDL).

4.6 Class-Instance Acquisition Experiments

In this section, we compare performances of the three graph-based SSL algorithms – LP-ZGL [Zhu et al., 2003], Adsorption, and MAD – on the class-instance acquisition task over a variety of graphs constructed from different sources. For all experiments, we use Mean Reciprocal Rank (MRR) as the evaluation metric as in [Talukdar et al., 2008c]. We used iterative implementations of the graph-based SSL algorithms, and the number of iterations was treated as a hyperparameter which was tuned, along with other hyperparameters, on a separate held out set, details regarding which are presented in Table 4.1. Statistics of various graph constructed and used in the experiments of Section 4.6 are presented in Table 4.2.

Experiment Setting	Algorithm	Iterations	μ_1	μ_2	μ_3
Section 4.6.1, 2 seeds per class	LP-ZGL	3	-	-	-
	Adsorption	3	-	-	-
	MAD	4	1	1	1e2
Section 4.6.1, 10 seeds per class	LP-ZGL	2	-	-	-
	Adsorption	2	-	-	-
	MAD	4	1	1e2	1e3
Section 4.6.2, 2 seeds per class	LP-ZGL	3	-	-	-
	Adsorption	2	-	-	-
	MAD	7	1	1	1e-4
Section 4.6.2, 10 seeds per class	LP-ZGL	3	-	-	-
	Adsorption	2	-	-	-
	MAD	7	1	1e2	1e2
Section 4.6.3, 2 seeds per class	LP-ZGL	4	-	-	-
	Adsorption	6	-	-	-
	MAD	17	1	1	1e-4
Section 4.6.3, 10 seeds per class	LP-ZGL	4	-	-	-
	Adsorption	4	-	-	-
	MAD	15	1	1	1e-4

Table 4.1: Tuned hyperparameter values used in the experiments of Sections 4.6.1, 4.6.2, and 4.6.3. During tuning, range of values tried for different hyperparameters were: maximum 20 iterations; $\mu_1 \in \{1\}$; $\mu_2, \mu_3 \in \{$1e-4, 1e-2, 1, 1e2, 1e3, 1e4$\}$.

4.6.1 Freebase Graph with Pantel Classes

Freebase [Metaweb Technologies, 2009][1] is a large collaborative knowledge base. The knowledge base harvests information from many open data sets (e.g. Wikipedia, MusicBrainz), which is augmented by contributions from users. It contains structured information about many diverse topics (equivalent to a Wikipedia article). For example, *Bob Dylan*, *Neuroscience* are examples of topics. A topic can have several *properties* and their corresponding values, which we shall refer to as *cell-values*. Properties are grouped together into *types*. A topic is assigned one or more types. A type can be thought of as a relational table. Types in turn are grouped into *domains*. In the example in Figure 4.2, *Bob Dylan* is a topic whose properties are present in the types *people-person* and *film-*

[1] `http://www.freebase.com/`

Graph	Vertices	Edges	Avg. Deg.	Min. Deg.	Max. Deg.
Freebase-1 (Section 4.6.1)	32970	957076	29.03	1	13222
Freebase-2 (Section 4.6.2)	301638	2310002	7.66	1	137553
TextRunner (Section 4.6.3)	175818	529557	3.01	1	2738
YAGO (Section 4.6.6)	142704	777906	5.45	0	74389
TextRunner + YAGO (Section 4.6.6)	237967	1307463	5.49	1	74389

Table 4.2: Statistics of various graphs used in experiments in Section 4.6. Some of the test instances (added for fair comparison with the TextRunner graph, Section 4.6.6) in the YAGO graph had no attributes in YAGO KB, and hence these instance nodes had degree 0 in the YAGO graph.

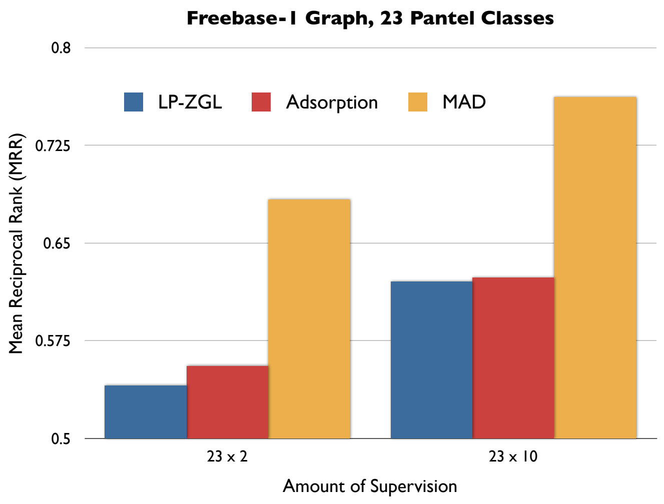


Figure 4.1: Comparison of three graph transduction methods on a graph constructed from the Freebase dataset (see Section 4.6.1), with 23 classes. All results are averaged over 4 random trials.

music_contributor. A type can be uniquely identified by prefixing its domain name. In type *people-person*, *Gender* is a property and *Male* is the corresponding cell-value.

For our current purposes, we can think of the Freebase dataset as a collection of rela-

Table id: people-person

Name	Place of Birth	Gender
...	...	...
Isaac Newton	Lincolnshire	Male
Bob Dylan	Duluth	Male
Johnny Cash	Kingsland	Male
...	...	...

Table id: film-music_contributor

Name	Film_Music_Credits
...	...
Bob Dylan	No Direction Home
...	...

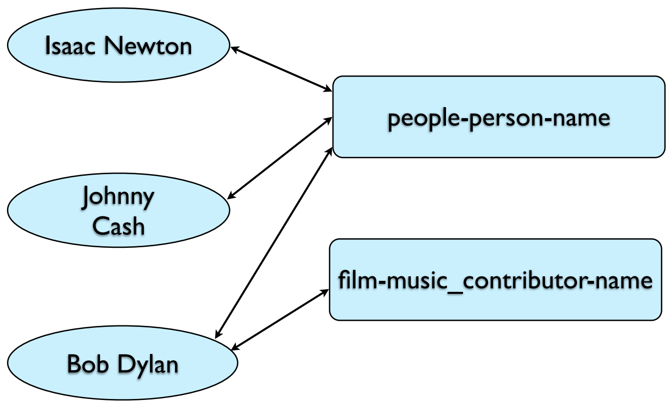
Figure 4.2: Examples of two tables (*types*) from Freebase, one table is from the *people* domain while the other is from the *film* domain.

tional tables, where each table is assigned a unique ID. We use the following process to convert the Freebase data tables into a graph:

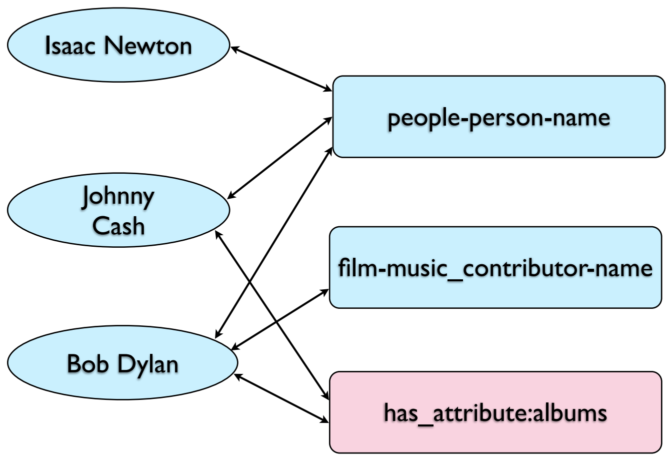
- Create a node for each unique cell-value (those retained after filtering)
- Create a node for each unique property name, where unique property name is obtained by prefixing the unique type name to the property name. For example, in the example in Figure 4.2, *people-person-gender* is a unique property name.
- Add an edge of weight 1.0 from cell-value node v and unique property node p, iff value v is present in the column corresponding to property p in one of the types. Similarly, add an edge in the reverse direction.

By applying this graph construction process on the first column of the tables (types) in Figure 4.2, we end up with the graph shown in Figure 4.3 (a).

We applied the same graph construction process on a subset of the Freebase dataset consisting of topics from 18 randomly selected domains: *astronomy, automotive, biology, book, business, chemistry, comic_books, computer, film, food, geography, location, people,*



(a)



(b)

Figure 4.3: (a) Example of a section of the graph constructed from the two tables in Figure 4.2. Nodes corresponding to unique property name is rectangular in shape, while node corresponding to an entity (or cell-value) is oval shaped. (b) The graph in part (a) augmented with an attribute node, *has_attribue:albums*, along with the edges incident on it. This results is additional constraints for the nodes *Johnny Cash* and *Bob Dylan* to have similar labels (see Section 4.6.6).

religion, spaceflight, tennis, travel, wine. The topics in this subset were further filtered and only cell-values with frequency 10 or more were retained. The resulting graph, Freebase-1 (Table 4.2), consisted of 32k nodes and 957k edges.

The authors of [Pantel et al., 2009] have made available a set of gold class-instance pairs derived from Wikipedia, which is also downloadable from http://ow.ly/13B57. From this set, we selected all classes which had more than 10 instance overlap with the Freebase graph constructed above. This resulted in 23 classes, which along with their overlapping instances were used as the gold standard set for the experiments in this section.

Experimental results with 2 and 10 seeds per class are shown in Figure 4.1. From Figure 4.1, we observe that LP-ZGL and Adsorption performed comparably on this dataset, with MAD significantly outperforming both methods.

4.6.2 Freebase Graph with Wordnet Classes

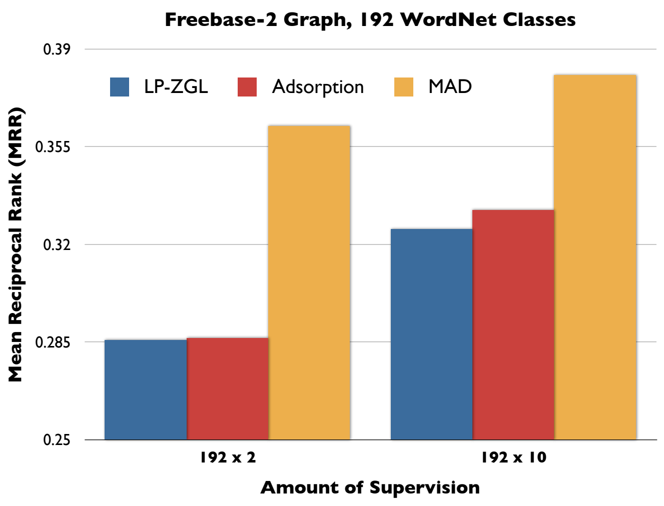


Figure 4.4: Comparison of graph transduction methods on a graph constructed from the Freebase dataset (see Section 4.6.2). All results are averaged over 10 random trials

In order to scale up the evaluation setting in terms of graph size, we constructed a larger graph, Freebase-2 (Table 4.2), from the same 18 domains as in Section 4.6.1, and using the same graph construction process. The resulting graph consisted of 301k nodes and 2.3m edges. In order to scale up the number of classes, we selected all Wordnet (WN) classes, available in the YAGO KB [Suchanek et al., 2007], which had more than 100 instance overlap with the larger Freebase graph constructed above. This resulted in 192 WN classes which were used for the experiments in this section. The reason behind imposing such frequency constraint during class selection is to make sure that each class is left with sufficient number of instances during testing.

Experimental results comparing LP-ZGL, Adsorption, and MAD with 2 and 10 seeds per class are shown in Figure 4.4. In order to emphasizes the large scale nature of these evaluations, we point out that a total of 292k test nodes were used during the evaluation in the setting with 10 seeds per class. Once again, we observe MAD outperforming both LP-ZGL and Adsorption. It is interesting to note that MAD with 2 seeds per class outperforms LP-ZGL and Adsorption even with 10 seeds per class.

4.6.3 TextRunner Graph with Wordnet Classes

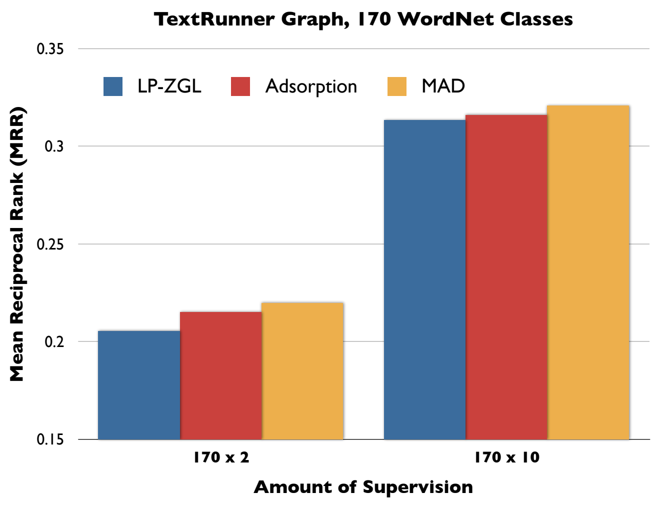


Figure 4.5: Comparison of graph transduction methods on a graph constructed from the hypernym tuples extracted by the TextRunner system [Banko et al., 2007] (see Section 4.6.3). All results are averaged over 10 random trials.

In contrast to graph construction from structured tables as in Sections 4.6.1, 4.6.2, in this section we use hypernym tuples extracted by TextRunner [Banko et al., 2007], an open domain IE system, to construct the graph[2]. An example of a hypernym tuple extracted by TextRunner is *(http, protocol, 0.92)*, where 0.92 is the extraction confidence. In order to convert such tuples into a graph, we construct a node corresponding to the instance (*http*), a node for the class (*protocol*), and connect the nodes with two directed edges in opposite direction, with extraction confidence (*0.92*) set as edge weights. Following this process, we constructed a graph from the TextRunner output with 175k nodes and 529k edges. We call this graph the TextRunner Graph (Table 4.2). As in Section 4.6.2, we use WN class-instance pairs as the gold set. In this case, we considered all WN classes, once again from YAGO KB [Suchanek et al., 2007], which had more than 50 instances overlapping with the constructed graph. This resulted in 170 WN classes being used in the experiments in this section.

Experimental results with 2 and 10 seeds per class are shown in Figure 4.5. Performances of the three methods are comparable in this setting, with MAD achieving the highest overall MRR.

4.6.4 Discussion

If we correlate the graph statistics in Table 4.2 with the results of sections 4.6.1, 4.6.2, and 4.6.3, we see that MAD is most effective for graphs with high average degree, that is, graphs where nodes tend to connect to many other nodes. For instance, the Freebase-1 graph has a high average degree of 29.03, with a corresponding large advantage for MAD over the other methods. Even though this might seem mysterious at first, it becomes clearer if we look at the objectives minimized by different algorithms. We find that the objective minimized by LP-ZGL (Equation **??**) is *under-regularized*, i.e., its model parameters ($\hat{\mathbf{Y}}$) are not constrained enough, compared to MAD (Equation **??**, specifically the third term), resulting in overfitting in case of highly connected graphs. In contrast, MAD is able to avoid such overfitting because of its minimization of a well regularized objective (Equation **??**). Based on this, we suggest that average degree, an easily computable structural property

[2]We thank the authors of [Banko et al., 2007] for kindly sharing their system's output

of the graph, may be a useful indicator in choosing which graph-based SSL algorithm should be applied on a given graph.

Unlike MAD, Adsorption does not optimize any well defined objective [Talukdar and Crammer, 2009], and hence any analysis along the lines described above is not possible. The heuristic choices made in Adsorption may have lead to its sub-optimal performance compared to MAD; we leave it as a topic for future investigation.

4.6.5 Effect of Per-Node Class Sparsity

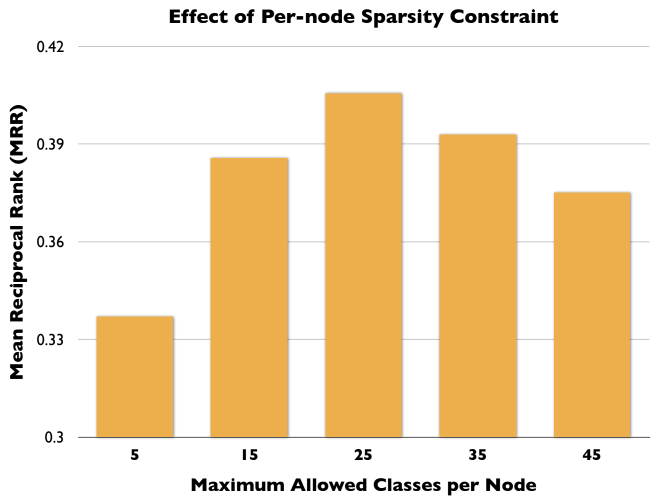


Figure 4.6: Effect of per node class sparsity (maximum number of classes allowed per node) during MAD inference in the experimental setting of Figure 4.4 (one random split).

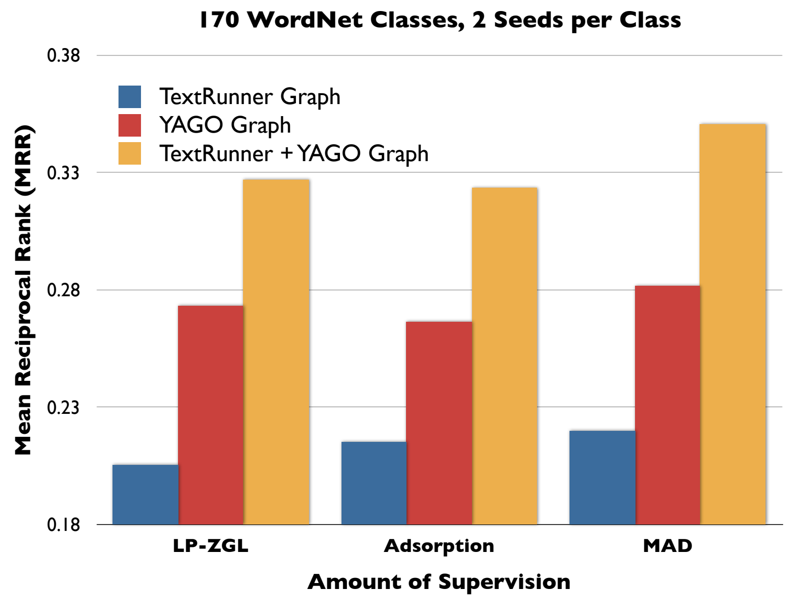
For all the experiments in Sections 4.6.1, 4.6.2, and 4.6.6, a node was allowed to attain a maximum of 15 classes during inference. After each update on a node, all classes except for the top scoring 15 classes were discarded. Without such sparsity constraints, a node in a connected graph will end up acquiring all the labels injected into the graph. This is undesirable for two reasons: (1) for experiments involving large number of classes (as in previous section and true in general in case of open domain IE), this increases the space requirement and also slows down inference; (2) a particular node is unlikely to belong a large number of classes. In order to estimate the effect of such sparsity constraints, we varied the number of classes allowed per node from 5 to 45 on the graph and setup of Figure 4.4. The results for MAD inference over the development split are shown in Figure 4.6. We observe that performance can vary significantly as the maximum number of classes allowed per node is changed, with the performance peaking at 25. This suggests that sparsity constraints during graph based SSL may have a crucial role to play, a topic which needs further investigation.

4.6.6 TextRunner Graph with Semantic Constraints from YAGO

Recently, the problem of instance-attribute extraction has started to receive attention [Probst et al., 2007, Bellare et al., 2007, Pasca and Durme, 2007]. Example of an instance-attribute pair is (*Bob Dylan, albums*). Given a set of seed instance-attribute pairs, these methods attempt to extract more instance-attribute pairs automatically from different sources, e.g. unstructured text, query logs etc. In this section, we explore whether class-instance assignment in the graph-based SSL setting can be improved by incorporating new semantic constraints derived from (instance, attribute) pairs. In particular, we experiment with the following type of constraint: two instances with a common attribute are likely to belong to the same class. For example, in Figure 4.3 (b), instances *Johnny Cash* and *Bob Dylan* are more likely to belong to the same class as they have a common attribute, *albums*. Because of the *smooth* labeling bias of graph-based SSL methods, such constraints are naturally captured by the graph-based SSL methods. All that is necessary is the introduction of bidirectional (instance, attribute) edges to the graph, as shown in Figure 4.3 (b).

In Figure 4.7, we compare class-instance acquisition performance of the three graph-based SSL methods on the following three graphs (Table 4.2):

> **TextRunner Graph**: Graph constructed from the hypernym tuples extracted by TextRunner, as in Figure 4.5 (Section 4.6.3).



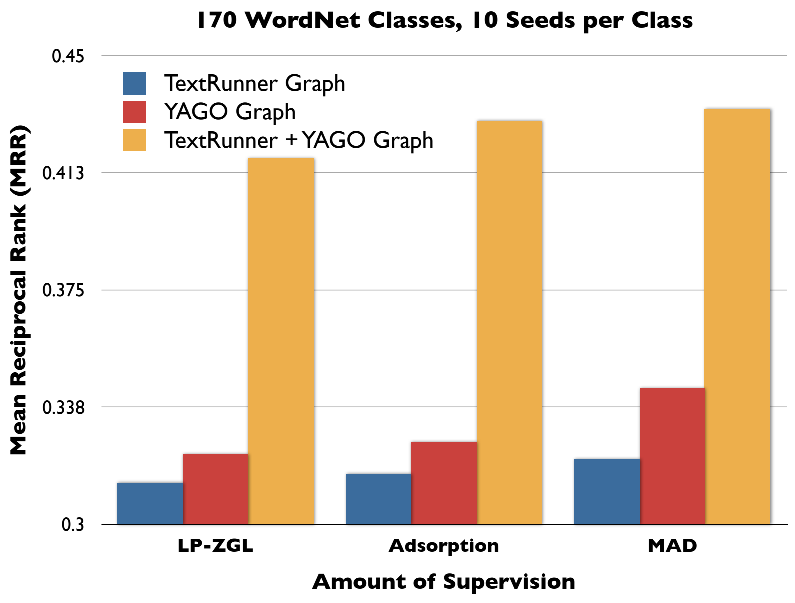


Figure 4.7: Comparison of class-instance acquisition performance on the three different graphs described in Section 4.6.6. All results are averaged over 10 random trials, with the hyperparameters tuned on a separate held out set. Addition of YAGO attributes to the TextRunner graph significantly improves performance.

YAGO Graph: Graph constructed from the (instance, attribute) pairs obtained from the YAGO KB [Suchanek et al., 2007], with 142k nodes and 777k edges.

YAGO Attribute	**Top-3 WordNet Classes Assigned by MAD** (example instances for each class are shown in brackets)
has_currency	**wordnet_country_108544813 (Burma, Afghanistan)** wordnet_region_108630039 (Aosta Valley, Southern Flinders Ranges) wordnet_state_108654360 (Agusan del Norte, Bali)
works_at	**wordnet_scientist_110560637 (Aage Niels Bohr, Adi Shamir)** wordnet_person_100007846 (Catherine Cornelius, Jamie White) wordnet_chancellor_109906986 (Godon Brown, Bill Bryson)
has_capital	**wordnet_state_108654360 (Agusan del Norte, Bali)** wordnet_region_108630039 (Aosta Valley, Southern Flinders Ranges) wordnet_country_108544813 (Burma, Afghanistan)
born_in	**wordnet_boxer_109870208 (George Chuvalo, Fernando Montiel)** wordnet_chancellor_109906986 (Godon Brown, Bill Bryson) wordnet_celebrity_109903153 (Donald Trump, Iain Lee)
has_isbn	**wordnet_book_106410904 (Past Imperfect, Berlin Diary)** wordnet_magazine_106595351 (Railway Age, Investors Chronicle) wordnet_episode_106396330 (Breaking Bread, Apocalypse Cow)

Table 4.3: Top-3 (out of 170) WordNet classes assigned by MAD on 5 randomly chosen YAGO attribute nodes (out of 80) in the TextRunner + YAGO graph used in Figure 4.7 (see Section 4.6.6), with 10 seeds per class used. A few example instances of each WordNet class is shown within brackets. Top ranked class for each attribute is shown in bold.

TextRunner + YAGO Graph: Union of the two graphs above, with 237k nodes and 1.3m total edges.

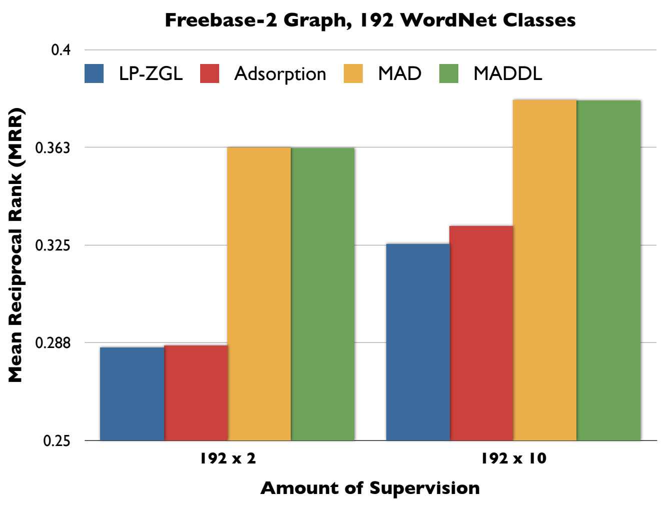
In all experimental settings with 2 and 10 seeds per class in Figure 4.7, we observe that the three methods consistently achieved best performance on the TextRunner + YAGO graph. This suggests that addition of attribute based semantic constraints from YAGO to the TextRunner graph results in a better connected graph which in turn results in better inference by the graph-based SSL algorithms, compared to using either of the sources, i.e., TextRunner output or YAGO attributes, in isolation. This further illustrates the advantage of aggregating information across sources [Talukdar et al., 2008c, Pennacchiotti and Pantel, 2009]. However, we are the first, to the best of our knowledge, to demonstrate the effectiveness of attributes in class-instance acquisition. We note that this work is similar in spirit to the recent work in [Carlson et al., 2010] which also demonstrate the benefits of additional constraints in SSL.

Because of the label propagation behavior, graph-based SSL algorithms assign classes to all nodes reachable (over the graph) from at least one of the labeled instance nodes. This allows us to check the classes assigned to nodes corresponding to YAGO attributes in the TextRunner + YAGO graph, as shown in Table 4.3. Even though the experimental setting was designed for class-instance acquisition, it is encouraging to see that the graph-based SSL algorithm (MAD in Table 4.3) is able to learn class-attribute relationships, an important by-product which has been the focus of recent studies [Reisinger and Pasca, 2009]. For example, the algorithm is able to learn that *works_at* is an attribute of the WordNet class *wordnet_scientist_110560637*, and thereby its instances (e.g. *Aage Niels Bohr, Adi Shamir*).

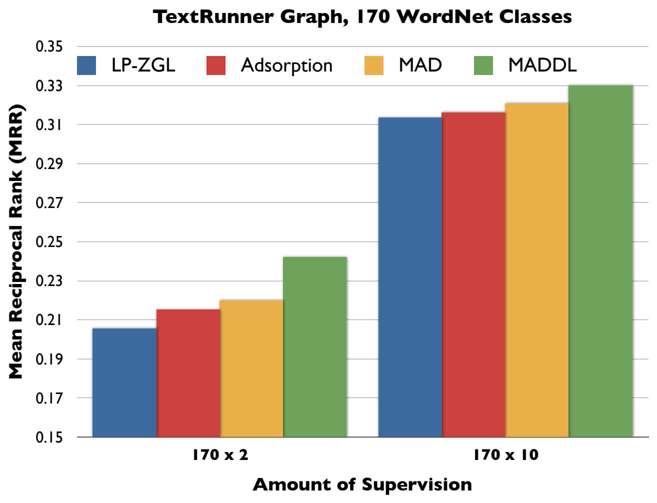
4.6.7 Effect of Class Similarity Constraints

In this section, we explore whether addition of class similarity constraints (e.g. synonym labels) in MADDL can improve overall class-instance acquisition performance. Class similarities for MADDL were obtained using Jiang and Conrath similarity [Jiang and Conrath, 1997] over the Wordnet graph. The similarities returned by this measure were manually browsed through and a few of the top scoring pairs were retained: 37 similarity pairs in case of Figure 4.8 (a), and 76 in case of Figure 4.8 (b). During inference, each node in the graph was allowed to have a maximum of 15 classes, as in previous experiments. Experimental results comparing MADDL with three other methods on two real-world graphs are presented in Figure 4.8.

From the results in Figure 4.8, we observe that MADDL is comparable or better than the three other methods, and it is most effective in the TextRunner graph. During the experiments in Figure 4.8, we observed that the MAD algorithm, which has no explicit knowledge of class similarity, is able to assign similar classes comparable scores on the nodes. Quality of the graph, where instance nodes from semantically similar classes have high



(a)



(b)

Figure 4.8: Comparison of MADDL with other methods on the class-instance acquisition task over two graphs: (a) Freebase-2 graph; (b) TextRunner graph (see Table 4.2). All results are averaged over 10 random trials. In (a), 37 class similarity pairs were used in MADDL; while in (b), 76 class similarity pairs were used.

connectivity, may be one reason for this. In this respect, the TextRunner graph is *noisier* compared to the Freebase-2 graph, as the former is constructed from automatic extractions. It is in such noisier graphs that we expect MADDL, and class similarity constraints, to be effective, as observed in Figure 4.8 (b).

Overall, we feel that the gains due to MADDL in these experiments are somewhat limited, and that there is significant room for future exploration in this direction, some of which we list here:

- Firstly, there is a need to evaluate effect of different similarity measures, as so far we have experimented only with one [Jiang and Conrath, 1997].
- Secondly, the number of similarity constraints used in these experiments are quite sparse, e.g. only 37 similarity constraints are used in Figure 4.8 (a), out of a total of 36672 possibilities. It will be interesting to see whether performance can be improved by constraining the inference problem further by including more label constraints in MADDL.

4.7 Experimental Results on Non-IE Tasks

In this section, we compare MAD with various state-of-the-art learning algorithms on some non-IE tasks: classification tasks (e.g. text classification in Sec. 4.7.1) and sentiment analysis in Sec. 4.7.2). In Sec. 4.7.3, we also provide experimental evidence showing that MAD is quite insensitive to wide variation of values in its hyper-parameters. In Sec. 4.7.4, we present evidence showing how MADDL can be used to obtain smooth ranking for sentiment prediction, a particular instantiation of classification with dependent labels.

4.7.1 Text Classification

World Wide Knowledge Base (WebKB) is a text classification dataset widely used for evaluating transductive learning algorithms. Most recently, the dataset was used by Subramanya and Bilmes [Subramanya and Bilmes, 2008], who kindly shared their preprocessed complete WebKB graph with us. There are a total of $4,204$ vertices in the

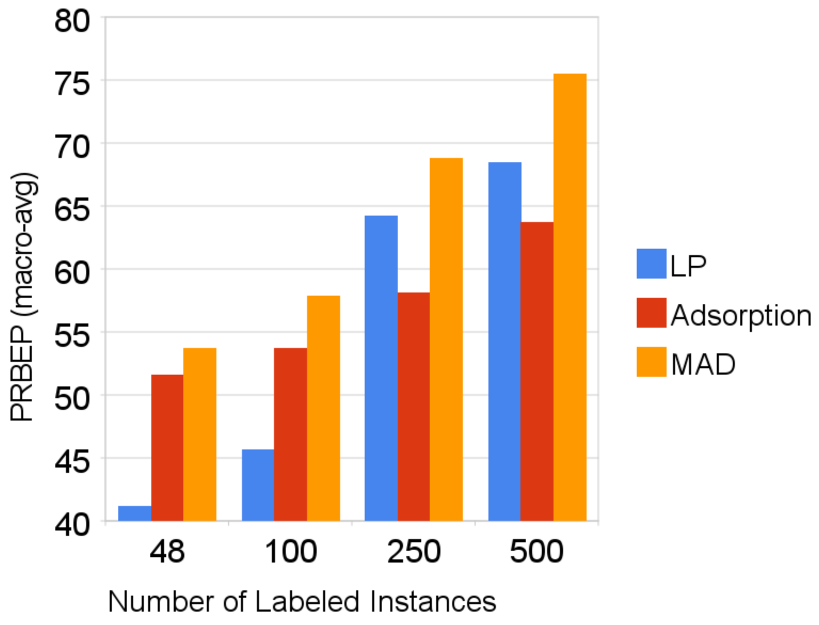


Figure 4.9: PRBEP (macro-averaged) for the WebKB dataset with 3148 testing instances. All results are averages over 20 randomly generated transduction sets.

Class	SVM	TSVM	SGT	LP	AM	Adsorption	MAD
course	46.5	43.9	29.9	45.0	**67.6**	61.1	62.8
faculty	14.5	31.2	42.9	40.3	42.5	52.8	**52.9**
project	15.8	17.2	17.5	27.8	42.3	**52.6**	**52.6**
student	15.0	24.5	**56.6**	51.8	55.0	39.8	48.6
average	23.0	29.2	36.8	41.2	51.9	51.6	**54.2**

Table 4.4: PRBEP for the WebKB data set with $n_l = 48$ training and 3148 testing instances. All results are averages over 20 randomly generated transduction sets. The last row is the macro-average over all the classes. MAD is the proposed approach. Results for SVM, TSVM, SGT, LP and AM are reproduced from Table 2 of [Subramanya and Bilmes, 2008].

graph, with the nodes labeled with one of four categories: *course, faculty, project, student*. A K-NN graph is created from this complete graph by retaining only top K neighbors of each node. K is treated as a hyper-parameter. We follow previous experimental protocol [Subramanya and Bilmes, 2008] and use Precision-Recall Break Even Point (PRBEP) [Raghavan et al., 1989] as the the evaluation metric. Performance comparison of

MAD and Adsorption for increasing n_l are shown in Figure 4.9, where MAD outperforms Adsorption for all values of n_l.

The dataset was randomly partitioned into four sets. A transduction set was generated by first selecting one of the four splits at random and then sampling n_l training documents from it, the remaining three sets working as the test set. This process was repeated 21 times to generate as many transduction sets. The first transduction set was used to tune the hyper-parameters, with search over the following ranges (following [Subramanya and Bilmes, 2008]): $K \in \{10, 50, 100, 500, 1000, 2000, 4204\}, \mu_2 \in \{1e\text{–}8, 1e\text{–}4, 1e\text{–}2, 1, 10, 1e2, 1e3\}, \mu_3 \in \{1e\text{–}8, 1e\text{–}4, 1e\text{–}2, 1, 10, 1e2, 1e3\}$. μ_1 was not tuned and was set to 1 in all the experiments. MAD's parameter sensitivity is discussed in Section 4.7.3.

As in previous work [Subramanya and Bilmes, 2008], Precision-Recall Break Even Point (PRBEP) [Raghavan et al., 1989] is the the evaluation metric. Same evaluation measure, dataset and the same experimental protocol makes the results reported here directly comparable to those reported in [Subramanya and Bilmes, 2008]. For easier readability, the results from Table 2 [Subramanya and Bilmes, 2008] are reproduced in Table 4.4 of this chapter, comparing performance of Adsorption based methods (Adsorption and MAD) to many previously proposed approaches: SVM [Joachims, 1999], Transductive-SVM [Joachims, 1999], Spectral Graph Transduction (SGT) [Joachims, 2003] and Label Propagation (LP) [Zhu and Ghahramani, 2002]. For MAD (the proposed approach) with $n_l = 48$, the optimal parameters were found to be: $K = 2000$, $\mu_1 = \mu_2 = 1$, $\mu_3 = 10$. The first four rows in Table 4.4 shows PRBEP for individual categories, with the last line showing the macro-averaged PRBEP across all categories. The MAD algorithm described above outperforms all other methods for two of the four categories and achieves the best performance overall (for n_l = 48).

Performance comparison of MAD and Adsorption for increasing n_l are shown in Figure 4.9. Comparing these results against Fig. 2 in [Subramanya and Bilmes, 2008], it seems that MAD outperforms all other methods compared (except AM

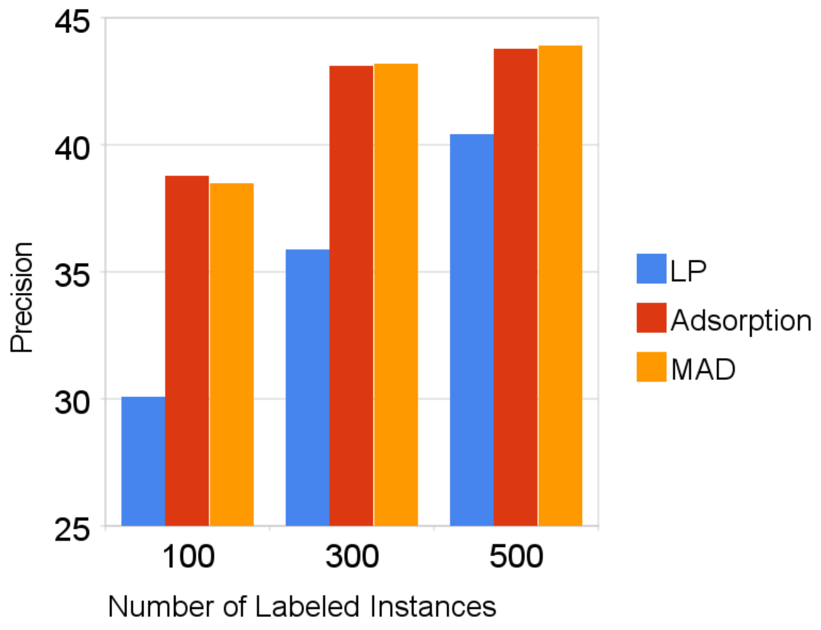


Figure 4.10: Precision for the Sentiment Analysis dataset with 3568 testing instances. All r esults are averages over 4 randomly generated transduction sets.

[Subramanya and Bilmes, 2008]) for all values of n_l. MAD performs better than AM for n_l = 48, but achieves second best solution for the other three values of n_l.

4.7.2 Sentiment Analysis

The goal of sentiment analysis is to automatically assign polarity scores to text collections, with a high score reflecting positive sentiment (user likes) and a low score reflecting negative sentiment (user dislikes). In this section, we report results on sentiment classification in the transductive setting. From Section 4.7.1 and [Subramanya and Bilmes, 2008], we observe that Label Propagation (LP) [Zhu and Ghahramani, 2002] is one of the best performing L2-norm based transductive learning algorithm. Hence, we compare the performance of MAD against Adsorption and LP.

For the experiments in this section, we use a set of $4,768$ user reviews from the electronics domain [Blitzer et al., 2007]. Each review is assigned one of the four scores: 1

(worst), 2, 3, 4 (best). We create a K-NN graph from these reviews by using cosine similarity as the measure of similarity between reviews. As before, K is treated as a hyper-parameter. We create 5 transduction sets from this data using the process described in Section 4.7.1. One transduction set is used to tune hyper-parameters while the rest are used for evaluation, with hyper-parameter search over the ranges: $K \in \{10, 100, 500\}, \mu_2 \in \{1e\text{–}4, 1, 10\}, \mu_3 \in \{1e\text{–}8, 1, 1e2, 1e3\}$. As before, μ_1 was not tuned and was set to 1 in all experiments. PRBEP results for different algorithms are shown in Figure 4.10. From this, we note that MAD outperforms LP, while Adsorption is quite competitive.

4.7.3 Parameter Sensitivity

We evaluated the sensitivity of MAD to variations of its μ_2 and μ_3 hyper-parameters. We used a 2000-NN graph constructed from the WebKB dataset and a 500-NN graph constructed from the Sentiment dataset. In both cases we tried three values for μ_2 and four values for μ_3 both ranging in at least 5 order of magnitude. For the WebKB the PRBEP remains almost fixed ranging between $45.1 - 46.8$ and for the sentiment data the PRBEP varies in the range $35.7 - 38.6$. We note that in both cases the algorithm is less sensitive to the value of μ_3 than the value of μ_2.

4.7.4 Smooth Ranking for Sentiment Analysis

We revisit the sentiment prediction problem in Section 4.7.2, but with the additional requirement that ranking of the labels (1, 2, 3, 4) generated by the algorithm should be smooth i.e. we prefer the ranking $1 > 2 > 3 > 4$ over the ranking $1 > 4 > 3 > 2$, where $3 > 2$ means that the algorithm ranks label 3 higher than label 2. Even though heuristic based post-processing of the ranking is one way to approach the problem, the goal here is to develop a principled way. We use the framework of stating requirements as an objective to be optimized. We use the MADDL algorithm of Sec. 4.5 initializing the matrix C as

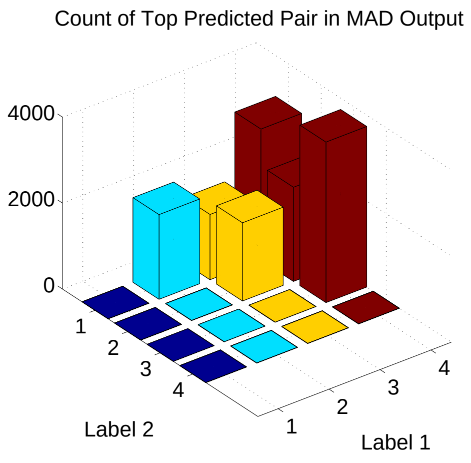


Figure 4.11: Plot of counts of top predicted labels pairs (order ignored) in MAD's predictions with $\mu_1 = \mu_2 = 1$, $\mu_3 = 100$.

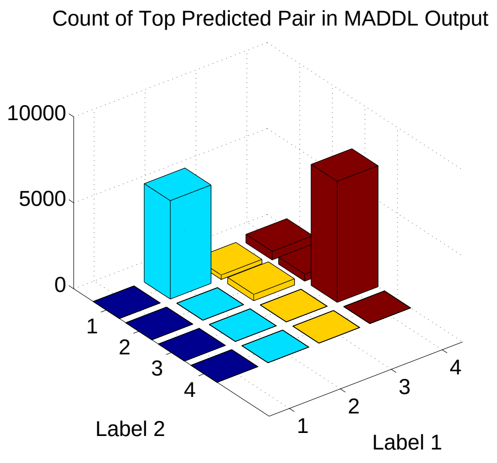


Figure 4.12: Plot of counts of top predicted labels pairs (order ignored) in MADDL's predictions (Section 4.5), with $\mu_1 = \mu_2 = 1$, $\mu_3 = 100$ and $\mu_4 = 1e3$.

	μ_4				
	0	1	10	100	1000
Prediction Loss (L1) at rank 1	0.78	0.78	0.78	0.78	0.77
Prediction Loss (L1) at rank 2	1.02	1.00	0.96	0.96	0.95

Table 4.5: Average prediction loss at ranks 1 & 2 (for various values of μ_4) for sentiment prediction. All results are averaged over 4 runs. See Section 4.7.4 for details.

follows (assuming that labels 1 and 2 are related, while labels 3 and 4 are related):

$$C_{12} = C_{21} = 1 \quad , \quad C_{34} = C_{43} = 1$$

with all other entries in matrix C set to 0. Such constraints (along with appropriate μ_4 in Equation (4.13)) will force the algorithm to assign similar scores to dependent labels, thereby assigning them adjacent ranks in the final output. MAD and MADDL were then used to predict ranked labels for vertices on a 1000-NN graph constructed from the sentiment data used in Sec. 4.7.2, with 100 randomly selected nodes labeled. For this experiment we set $\mu_1 = \mu_2 = 1$, $\mu_3 = 10$. Results for increasing values of μ_4 are shown in

Table 4.5. L1 loss computed between the gold label and labels predicted at ranks $r = 1, 2$ are computed. MADDL with $\mu_4 = 0$ corresponds to MAD. From Table 4.5 we observe that with increasing μ_4, MADDL is able to put a label at $r = 2$ which is related (as per C) to the label at $r = 1$, but at the same time maintain the quality of prediction at $r = 1$ (represented by the almost constant first row in Table 4.5), thereby ensuring a smoother ranking.

Another view of the same phenomenon is shown in Fig. 4.11 and Fig. 4.12. In these figures, we plot the counts of top predicted label pair (order of prediction is ignored for better readability) generated by the MAD and MADDL algorithms. By comparing these two figures we observe that label pairs (e.g. (2,1) and (4,3)) favored by C (above) are more frequent in MADDL's predictions than in MAD's. At the same time, non-smooth predictions (e.g. (4, 1)) are virtually absent in MADDL's predictions while they are quite frequent in MAD's. These clearly demonstrate MADDL's ability to generate smooth predictions in a principled way, and more generally the ability handle data with non-mutually exclusive or dependent labels.

4.8 Summary of Chapter

In this chapter we have analyzed the Adsorption algorithm [Baluja et al., 2008] and proposed a new graph based semi-supervised learning algorithm, MAD. We have developed efficient (iterative) solution to solve our convex optimization based learning problem, and also listed the conditions under which the algorithm is guaranteed to converge. Transition to an optimization based learning algorithm allows us to easily extend the MAD algorithm to handle data with dependent labels, resulting in the MADDL algorithm. We reported systematic comparison of various graph-based SSL algorithms on different real-world learning tasks, including class-instance acquisition, where we found MAD to be the most effective. We also demonstrated how class-instance acquisition performance in the graph-based SSL setting can be improved by incorporating additional semantic constraints available in independently developed knowledge bases.

In the next chapter, we shall transition into the Information Integration (II) portion of this thesis, and explore how data-integrating structured queries can be constructed from keyword queries and user feedback over answers.

Chapter 5

Learning Data-Integrating Queries

Some parts of this chapter are based on [Talukdar et al., 2008a].

5.1 Introduction

In this chapter, we present Q, a system with which a non-expert user can author new query templates and Web forms, to be reused by anyone with related information needs. The user poses keyword queries that are matched against source relations and their attributes; the system uses sequences of associations (e.g., foreign keys, links, schema mappings, synonyms, and taxonomies) to create multiple ranked queries linking the matches to keywords; the set of queries is attached to a Web query form. Now the user and his or her associates may pose specific queries by filling in parameters in the form. Importantly, the answers to this query are ranked and annotated with data provenance, and the user provides *feedback* on the utility of the answers, from which the system ultimately learns to assign costs to sources and associations according to the user's specific information need, as a result changing the ranking of the queries used to generate results. We evaluate the effectiveness of our method against "gold standard" costs from domain experts and demonstrate the method's scalability.

Challenges. The mode of interaction in Q creates a number of fundamental challenges.

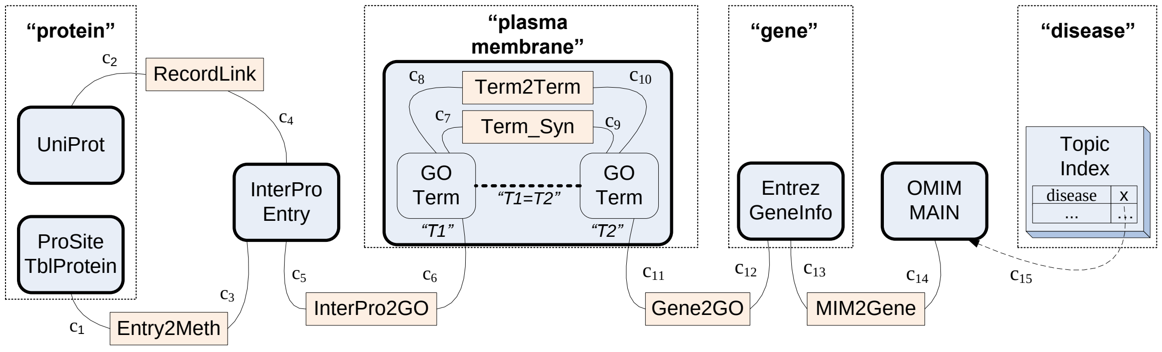


Figure 5.1: Schema elements (nodes) matching a query about proteins, diseases, and genes related to "plasma membranes." The relations come from different bioinformatics sources, plus site-provided correspondence tables (e.g., InterPro2GO), the results of a record linking tool (RecordLink), a Term that may be used directly or combined with its superclasses or synonyms through ontologies (GO Term2Term, Term_Syn), and instance-level keyword matching (topic index). Rounded rectangles represent conceptual entities and squared rectangles represent tables relating these entities; Q considers there to be a weighted *association* edge based on the foreign key or link dereference.

CQ2: $q(prot, gene, typ, dis)$:- $TblProtein(id, prot, \ldots)$, $Entry2Meth(ent, id, \ldots)$,
$InterPro2Go(ent, gid1)$, $\mathbf{Term\_Syn}(gid1, gid2)$, $Term(gid2, typ)$,
$Gene2GO(gid2, gild)$, $GeneInfo(gild, gene, \ldots)$, $MIM2Gene(gild, mId)$,
$MAIN(mId, dis, \ldots)$, $typ =$ 'plasma membrane'

CQ3: $q(prot, gene, typ, dis)$:- $TblProtein(id, prot, \ldots)$, $Entry2Meth(ent, id, \ldots)$,
$InterPro2Go(ent, gid1)$, $\mathbf{Term2Term}(\_,$ 'part_of', $gid1, gid2)$, $Term(gid2, typ)$,
$Gene2GO(gid2, gild)$, $GeneInfo(gild, gene, \ldots)$, $MIM2Gene(gild, mId)$,
$MAIN(mId, dis, \ldots)$, $typ =$ 'plasma membrane'

Table 5.1: Excerpts of some potential queries from the graph of Figure 5.1, with important differences highlighted in boldface.

First, we must have a unified, end-to-end model that supports computation of ranked queries, which produce correspondingly ranked results, and it must be possible to *learn new query rankings from feedback* over the results, ultimately converging to rankings consistent with user expectations. In support of this, we must be able to find the top-k queries, begin producing answers, and learn from feedback at interactive-level speeds. We must always be able to determine results' provenance, as a means of connecting feedback to the originating query or queries. Additionally, it is essential that we be able to *generalize* feedback to results other than the one to which the user directly reacted. In part this is due to the "curse of dimensionality": we must generalize if we are to learn from small numbers of examples.

Contributions. In our approach, edge weights encode shared learned knowledge about the usefulness of particular schema elements, across multiple queries and users with similar preferences and goals. Multiple users in the same lab or the same subfield may wish to share the same view and continuously refine it. They may also wish to pose other related queries, and have feedback affect the entire set of queries together. On the other hand, groups of users with very different goals (say, highly speculative exploration of associations, versus refinement of results to find the highest-reliability links) can have their own set of views with a different set of weight assignments. In essence, each sub-community is defining its own integrated schema for viewing the data in the system — which includes not only a set of schema mappings (associations) but also a set of weights on the mappings and source relations. This represents a *bottom-up*, *community-driven* scheme for integrating data. Our chapter makes the following contributions:

- We bring together ideas from data integration, query-by-example, data provenance, and machine learning into a novel unified framework for authoring *through feedback* families of queries corresponding to a bioinformatics Web form, but potentially spanning many databases. We exploit the output from record linking and schema mapping tools.

- We develop efficient search strategies for exploring and ranking associations among schema elements, links, synonyms, hypernyms, and mapping tables — producing top-ranked queries at interactive speeds. This depends on a new approximation scheme for the K-best Steiner tree problem (explained in Section 5.5), which scales to much larger graphs than previous methods.

- We develop efficient techniques for integrating data provenance and online learning techniques — learning at interactive speeds.

- We experimentally validate the efficacy of our solutions, using real schemas and associations.

Roadmap. Section 5.3 presents our basic architecture and operation. Section 5.4 describes how queries are answered in our system, and Section 5.5 describes how feedback is given and processed. We present experimental results showing scalability and rapid learning in Section 5.6, and we conclude and discuss future work in Section 5.7.

5.2 Related Work

The problem of providing ranked, keyword-based answers to queries has been the subject of many studies. Most focus on providing *answers* based on the keywords, rather than on constructing persistent *views* that can be used for multiple keywords. We briefly review this work and explain why our problem setting is different.

Information retrieval [Baeza-Yates and Ribeiro-Neto, 1999] focuses on providing ranked documents as query answers, given keyword queries. It does not generate or learn structured queries that combine data from multiple databases. Similarly, while natural-language query interfaces have been studied that create structured queries over databases, e.g., [Popescu et al., 2003], our goal is to take keyword queries, not natural language, and a large and diverse set of data sources and associations, and to learn the appropriate structural queries.

At a high level, our work seems most similar to keyword search over databases [Bhalotia et al., 2002, Botev and Shanmugasundaram, 2005, Hristidis and Papakonstantinou, 2002, Kacholia et al., 2005]. There, keyword matches involve both data items and metadata. Results are typically scored based on a combination of match scores between keywords and data values, length of the join path between the matched data items, and possibly node authority scores [Balmin et al., 2004, Guo et al., 2003]. The NAGA system [Kasneci et al., 2008] additionally considers an ontology-like knowledge graph and a generative model for ranking query answers. In many of these systems, ranking is based on costs that are computed in additive fashion (much like the cost model we adopt). In contrast to all of these systems, we seek to *learn* how to score results based on user preferences, since associations between scientific data sources do not necessarily reflect authority, and in any case perceived authority may vary with user characteristics and needs. This naturally complements a number of existing bioinformatics query authoring systems that rely on expert-provided scores [Boulakia et al., 2007, Mork et al., 2002].

Existing "top-k query answering" [Cohen, 1998, Gravano et al., 2003, Li et al., 2005, Marian et al., 2004] provides the highest-scoring answers in answering ranked queries. Our goal is to identify possible queries to provide answers by connecting terms, to separately rank each combination, to output the results using this rank, and finally to enable feedback on the answers. The step in which results are output could be performed using top-k query processing algorithms.

Our work uses machine learning in a way that is complementary to other learning-based tools and techniques for data integration. Schema matching tools [Rahm and Bernstein, 2001] provide correspondences and possibly complete schema mappings between different pre-existing schemas: we use mappings as input, in the form of associations that help our system to create cross-site queries. We can also learn the quality of the mappings based on feedback. Finally, recent work focuses on learning to construct mashups [Tuchinda and Knoblock, 2008], in a way that generalizes the information extraction problem and suggests new columns to be integrated: this is also complementary to our work, which focuses on determining how to decide which queries and answers should be used to populate the results.

A method that learns to rank pairs of nodes based on their graph-walk similarity is presented in [Minkov et al., 2006, Minkov and Cohen, 2007]. A similar method that learns the random walk probabilities in a graph satisfying pairwise node ordering constraints is

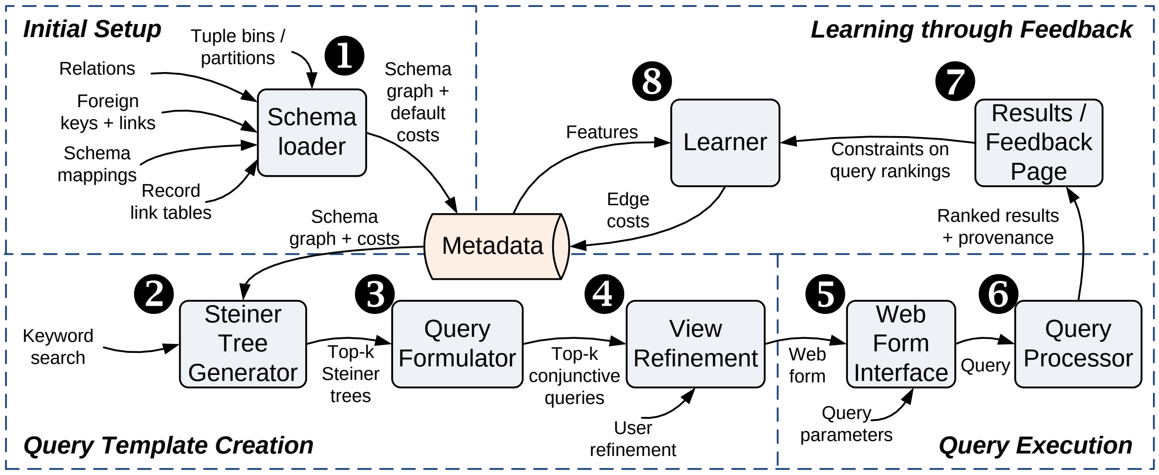


Figure 5.2: Architectural stages of the Q System.

presented in [Agarwal et al., 2006]. In contrast, the learning method used in this chapter learns to rank *trees* derived from the query graph, and not just node pairs. However, all these methods share the common objective of learning the costs on edges of the graph on which they operate.

5.3 Architecture and Operation

We divide system operation into four major phases: initial setup, query template creation, query execution, and learning through feedback. We discuss each of these phases in order, focusing on the modules and dataflow.

5.3.1 Initial Setup

Refer to Figure 5.2 to see the components of the Q System. During initial setup, Q's **Schema Loader** (the box highlighted with the numeral 1 in the figure) is initially given a set of data sources, each with its own schema. Data items in each schema might optionally contain links (via URLs, foreign keys, or coded accession numbers) to other data sources. Additionally, we may be given certain known correspondences or transformations as the result of human input or data integration tools: for instance, we may have schema mappings between certain elements, created for data import, export, or peer-to-peer integration [Halevy et al., 2003]; some data items may be known to reference an externally defined taxonomy or ontology such as GeneOntology (GO); and tools may be able to discover (possibly approximate) associations between schema elements. All such information will be encoded in the *schema graph*, which is output by the Schema Loader and saved in a metadata repository.

Figure 5.1 features two classes of relations as nodes: blue rounded rectangles represent entities, and orange elongated rectangles represent cross-references, links, or correspondence tables. Edges represent associations between nodes (generally indicating potential joins based on equality of attributes). The schema graph in the example illustrates a common feature of many bioinformatics databases, which is that they frequently contain cross-referencing tables: *Entry2Meth*, *InterPro2GO*, etc., represent the database maintainers' current (incomplete, possibly incorrect) information about inter-database references. Additionally, our example includes a correspondence table, *RecordLink*, that was created by a schema mapping/record linking tool, which matches *UniProt* and *InterPro* tuples. As previously described, any of the associations encoded as edges may have a *cost* that captures its likely utility to the user: this may be based on reliability, trustworthiness, etc., and the system will attempt to learn that cost based on user feedback. These costs are normally initialized to the same default value.

5.3.2 Query Template Creation

The user defining a query template poses a **keyword query**

```
protein "plasma membrane" gene disease
```

which is matched against the schema graph by the **Steiner Tree Generator** (box #2 in Figure 2). A pre-processing step consists of matching keywords against graph elements: We can see from the figure that the first term matches against *UniProt* and *TblProtein* (based on substring matching against both relation and attribute names). The term "plasma membrane" does not match against any table names or attributes — but rather against a term in the GO ontology, which includes (as encoded data) standardized terms. Terms in the ontology have both subclasses (*Term2Term*) and synonyms (*Term_Syn*), and hence the system must consider these in the query answers as well. The keyword "gene" matches as a substring against *GeneInfo*, and finally, "disease" matches against an entry in an index from topics to databases. Implicitly, as part of the keyword matching process, the Q System adds a node to the schema graph for each keyword, and an edge to each matching node. (For visual differentiation in the figure, we indicate these edges by drawing a dashed rectangle around each keyword and its matching nodes.)

Now, given keyword matches against nodes, the Steiner Tree Generator can determine the *best* (top-k) queries matching the keywords. Its goal is to find the k trees of minimal cost contained in the schema graph, each of which includes all of the desired (keyword) nodes, *plus* any additional nodes and edges necessary to connect them. This is technically a Steiner tree; the cost of each Steiner tree is the sum of edge costs. (We discuss below how edge costs are obtained.) Note the subtlety that this module does not generate queries to compute the top-k *answers*; rather, it produces the top-k-scoring *queries* according to our schema graph. These may return more or fewer than k answers; but commonly each query will return more than one answer.

The **Query Formulator** (box 3) takes each of the top-k Steiner trees and converts it into a conjunctive query (nodes become relations, edges become joins, and the cost of the query is the sum of the costs of the edges in the Steiner tree).

At the **View Refinement** stage (box 4), the top-scoring queries are combined into a disjoint union (i.e., aligning like columns and padding elsewhere with nulls, as described in Section 5.5), forming what we term a *union view*. Next, the query author may *refine* the query, adding projections, renaming or aligning columns, and so on. At this stage, the view is optionally given a name and made persistent for reuse. An associated Web form is automatically generated, as in Figure 5.1b. (Recall that our "view" actually represents a template for a family of queries with similar information needs, which are to be parameterized by the user to actually pose a query.)

5.3.3 Query Execution

Any user with permissions (not only the author) may access the **Web Form Interface** (box 5), parameterize the fields of the query through the Web form, and execute it. This invokes the **Query Processor** (box 6), which is a distributed relational query engine extended to annotate all tuples with their *provenance* or lineage [Buneman et al., 2001, Cui, 2001, Green et al., 2007b], which is essential for later allowing the system to take feedback on tuples and convert it into feedback on queries. Of course, the query processor must also return these annotated results in increasing order of cost, where they receive the cost of the query that produces them. (If a tuple is returned by more than one query, it is annotated with the provenance of all of its producer queries, and given the cost of the lowest-cost query.)

5.3.4 Learning through Feedback

Once the user has posed a query, he or she may look over the results in the **Results/Feedback Page** (box 7) and provide *feedback* to the system, with respect to the relative ordering and set of answers. The system will generalize this feedback to the queries producing the answers. Then the **Learner** (box 8) will adjust costs on the schema graph, thus potentially changing the set of queries associated with the Web form, and altering the set of answers to the query. The new results are computed and returned at interactive speeds, and the user may provide feedback many times. Our goal is to learn the costs corresponding to the user's mental model of the values of the respective sources.

In the subsequent two sections, we discuss the implementation of the main modules in detail. We omit further discussion of Module 1, the Schema Loader, as it is straightforward to implement. Our discussion begins with the query creation and answering stages (boxes 2-6), and then we move on to discuss the feedback and learning stages (boxes 7 and 8).

5.4 Queries and Query Answers

In this section, we begin by discussing the details of the schema graph (Section 5.4.1) and cost model (Section 5.4.2), which form the basis of all query generation. Section 5.4.3 then considers how keywords are matched against the graph, and Section 5.4.4 addresses the key problem of finding the best queries through Steiner tree generation. Finally, we discuss how Steiner trees are converted into query templates (Section 5.4.5), and how these templates are parameterized and executed (Section 5.4.6).

5.4.1 Foundation: the Schema Graph

As its name connotes, the *schema graph* is primarily at the schema and relationship level: nodes represent source relations and their attributes and edges represent associations between the elements. Our query system additionally supports matches at the tuple level — which is especially useful when searching topic indices and ontologies (as in Figure 5.1) — but our emphasis is primarily on the metadata level, as explained in the previous section.

Nodes. Nodes represent source relations containing data that may be of interest. The query answers should consist of attributes from a set of source nodes.

Edges. Within a given database, the most common associations are **references**: a foreign key pointing to another relation, a hyperlink pointing to content in another database, etc. However, a variety of additional associations may relate nodes, particularly across different sources: **subclass** ("is-a") is very common in ontologies or class hierarchies; **maps-to** occurs when there exists a view, schema mapping, synonym, or correspondence table specifying a relationship between two different tables; **similar-to** describes an association that requires a similarity join. All edges have cost expressions associated with them.

5.4.2 Cost Model

The *costs* associated with edges in the schema graph are simple weighted linear combinations of edge *features*. Features are domain-specific functions on edges that encode the aspects of those edges that are relevant to user-ranking of queries: in essence, they capture distinctions that may be relevant to a user's preference for an edge as part of the query. The identities of edge end-nodes are the simplest and most obvious features to use: the cost will be a function of the nodes being associated by the edge. However, more general features, for instance the type of association (subclass, maps-to, similar-to) used to create an edge, are also potentially useful. Each feature has a corresponding *weight*, representing the relative contribution of that feature to the overall cost of the query: this is set to a default value and then *learned*. Crucially, the use of common features in computing costs allows the Q System to share information about relevance across different queries and edges, and thus learn effectively from a small number of user interactions.

We discuss features and how they are learned in Section 5.5. For purposes of explaining query answering in this section, we note that the cost of a tree is a weighted linear combination of the features of the edges in the tree. This model was carefully chosen: it allows simple and effective learning of costs for the features from user feedback [Crammer et al., 2006a].

Intuitions behind the cost model. An edge cost in our model can be thought of as the logarithm of the odds (in the sense of betting) that using that edge in a query leads to *worse* answers (from the user's point of view) than including the average alternative edge. Conversely, lower costs correspond to better odds that using the edge will lead to better answers. Since the costs are parameterized by a shared weight vector $\mathbf{w}$, feedback from a few queries will typically affect edges involved in many different queries. Selecting query trees according to the updated weights will increase the odds that user-favored answers are shown first.

We observe that our cost model somewhat resembles that of other keyword query systems (e.g. [Kacholia et al., 2005]), which do not use features or learning, but often use an additive model based on edge costs. Our notion of cost and its use in query construction is different from the probabilities in probabilistic databases [Dalvi and Suciu, 2004], which represent uncertainty about whether specific relationships hold. A low-cost answer tuple in our model is not necessarily very probable, but simply one that was derived by a query that involves associations favored by the user. Our costs are also different from edge capacities in authority flow models [Balmin et al., 2004, Varadarajan et al., 2008], which encode the relative strength of edges as carriers of authority between nodes. A low-cost edge in our model is not necessarily one that passes more authority from its source to its target, but simply one that supports a join that has proven useful.

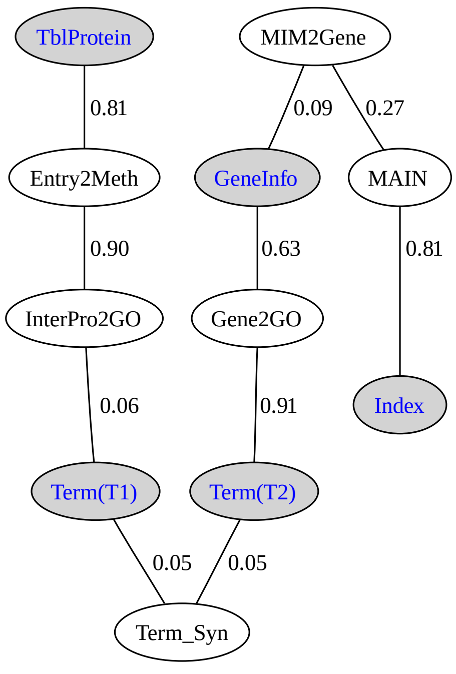
5.4.3 Matching Keyword Queries

Given a user's keyword query, the Q System begins by matching it against nodes in the schema graph. A keyword query consists of a set of terms $Q = \{q_1, \ldots, q_n\}$. Let N_q be the set of nodes in the schema graph that match $q \in Q$, and let $N = \bigcup_{q \in Q} N_q$. A node matches a term if its label (consisting of the relation and attribute names) contains the term as a substring, or, in special cases (e.g., for taxonomies and synonym tables), the *instance* of the relation represented by the node contains the term.

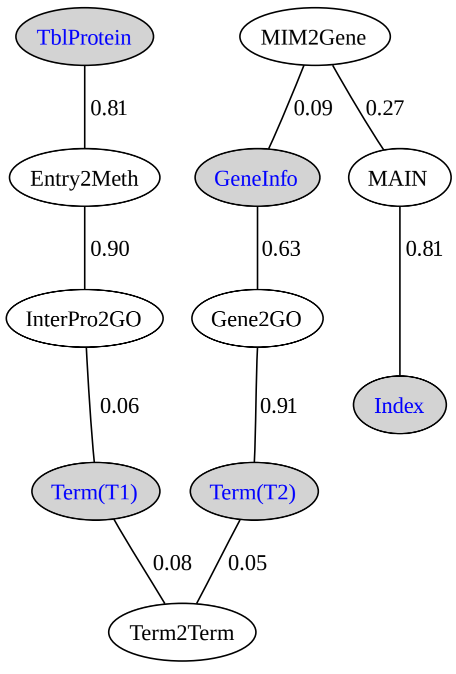
For each $q \in Q$, we add a special *keyword node* q to the graph, and also edges (q, n) for all $n \in N_q$. These new edges can be assigned costs according to an appropriate scoring function, for instance TF/IDF. The system now attempts to find the k lowest-cost *Steiner trees* that contain all of the keyword nodes.

Each such tree T also includes non-keyword nodes that are needed to complete a (connected) tree. As discussed previously, the cost of T is the sum of costs of its edges, and those costs are weighted combinations of edge features. Formally, the feature weights form a *weight vector* $\mathbf{w}$, and the cost $C(T, \mathbf{w})$ of T is the sum of $\mathbf{w}$-dependent edge costs:

$$C(T, \mathbf{w}) = \sum_{e \in E(T)} C(e, \mathbf{w}) \tag{5.1}$$



(a) Cost=4.58



(b) Cost=4.61

Figure 5.3: Steiner trees for queries CQ2 and CQ3 in Table 5.1. Nodes matching query keywords are shaded, with blue text.

where $E(T)$ is the set of edges of T.

We next discuss the process of finding Steiner trees. The goal here is to quickly (at interactive rates) produce an ordered list of subtrees of the schema graph that purport to satisfy the information need specified by a set of keywords. That ordered list is determined by the current feature weight vector $\mathbf{w}$. Later, the learning process will adjust this weight vector so that the order of the returned query trees corresponds better to user preferences about the order of the corresponding answers.

5.4.4 Steiner Tree Generation

The task of our Steiner Tree Generator is not merely to find a single Steiner tree in the graph, as is customary in the literature — but to find the *top k Steiner trees* in order to find the k best queries. Here, we are faced with the question of whether to find the actual

$$\begin{aligned}
&\textsc{Steiner}\ (G,S,C):\\
&\min_{\substack{\mathbf{x},\mathbf{y}\\ r\in V(G)}} \sum_{(i,j)\in E(G)} C(i,j)\times y_{ij}\\
&\quad \text{s.t. } S' = S - \{r\}\\
&\qquad \sum_{h\in V(G)} x^k_{rh} - \sum_{j\in V(G)} x^k_{jr} = 1\ \ \forall k\in S' && \text{(C1)}\\
&\qquad \sum_{h\in V(G)} x^k_{kh} - \sum_{j\in V(G)} x^k_{jk} = -1\ \ \forall k\in S' && \text{(C2)}\\
&\qquad \sum_{h\in V(G)} x^k_{ih} - \sum_{j\in V(G)} x^k_{ji} = 0\ \ \forall i\in V(G)\setminus S && \text{(C3)}\\
&\qquad x^k_{ij} \le y_{ij}\ \forall (i,j)\in E(G), k\in S' && \text{(C4)}\\
&\qquad x^k_{ij} \ge 0\ \forall (i,j)\in E(G), k\in S' && \text{(C5)}\\
&\qquad y_{ij} \in \{0,1\} && \text{(C6)}
\end{aligned}$$

Figure 5.4: Mixed integer program for min-cost Steiner trees.

k lowest-cost Steiner trees, or to settle for an approximation. For small graphs we use an exact algorithm for finding the k lowest-cost Steiner trees, and for larger graphs we develop a heuristic. This allows us to find the optimal solution for small schema graphs, and yet to scale gracefully to larger schemas.

5.4.4.1 Steiner Trees via Integer Programming

We first formalize the Steiner tree problem. Let G be a directed graph with nodes and edges given by $V(G)$ and $E(G)$, respectively. Each edge $e = (i, j) \in E(G)$ has a cost $C(e)$. We also have a set of nodes $S \subseteq V(G)$. A directed subtree T in G connecting the nodes in S is known as a Steiner tree for S. The nodes in $V(T) \setminus S$ are called Steiner nodes. The cost of T is $C(T) = \sum_{e\in E(T)} C(e)$. Finding the minimum cost Steiner tree on a directed graph (STDG) is a well-known NP-hard problem [Wong, 1981, Garey and Johnson, 1979].

Finding a minimum-cost Steiner tree on a directed graph [Wong, 1981] can be expressed as a mixed integer program (MIP) [Wolsey, 1998] in a standard way (Figure 5.4) [Wong, 1981]. This encoding requires one of the nodes r in $V(G)$ to be chosen as root of the Steiner tree. Hence, the minimum cost Steiner tree can be obtained by running the appropriate MIP with each node in $V(G)$ taken as root separately and then selecting the lowest-cost tree from at most $|V(G)|$ candidates. This can be time consuming especially for large schema graphs. For the experiments reported in this chapter, we convert every schema graph edge (which, despite describing a foreign key, is really a bidirectional association) to a pair of directed edges. With such bi-directional edges, one can find the minimum cost Steiner tree by solving STEINER (G, S, C) with any of the nodes in S fixed as root, avoiding the need to iterate over all the vertices in the graph. Unless otherwise stated, we assume the graph to be bi-directional in what follows. In STEINER (G, S, C), an edge $(i, j) \in E(G)$ is included in the solution iff $y_{ij} = 1$. The MIP STEINER (G, S, C) that finds the lowest-cost (according to cost function C) Steiner tree in G and containing nodes S can be viewed as a network flow problem where x_{ij}^k specifies the amount of flow of commodity k flowing on edge (i, j). Flow on an edge (i, j) is allowed only if that edge is included in the solution by setting $y_{ij} = 1$. This is enforced by constraint C4. All flows are nonnegative (constraint C5). Flow of commodity k originates at the root r (constraint C1) and terminates at node k (constraint C2). Conservation of flow at Steiner nodes is enforced by constraint C3.

However, we need more than just the minimum-cost tree: we need the k lowest-cost trees. To achieve this, we modify the MIP of Figure 5.4 so that it can be called multiple times with constraints on the sets of edges that the solution can include. The modified program is shown on Figure 5.5.

The MIP STEINERIE(G, S, I, X, C) finds the lowest cost (according to cost function C) Steiner subtree of G rooted at r that contains the nodes in S, which must contain the edges in I and cannot contain any edge in X. C9 guarantees that there is flow of at least one commodity on all edges in I. T1 with C7-C9 enforce the inclusion constraints, while the exclusion constraints are enforced by C10. We must also ensure the result will be a tree by requiring flow to pass through the source nodes of the edges in I. Step T1 expands S by including source nodes of the edges in I. This ensures there is a directed path from

$$
\begin{aligned}
&\textsc{SteinerIE}(G, S, I, X, C): \\
&\quad \min_{\substack{\mathbf{x},\mathbf{y} \\ r \in S}} \sum_{(i,j) \in E(G)} c(i,j) \times y_{ij} \\
&\quad S^+ = S \cup \{i : (i,j) \in I\} && \text{(T1)} \\
&\quad \text{s.t.} \\
&\qquad \text{Constraints C1-C6 from } \textsc{Steiner}(G, S^+, C) \\
&\qquad \sum_{h \in V(G)} y_{hr} = 0 && \text{(C7)} \\
&\qquad \sum_{h \in V(G)} y_{hi} \leq 1 \;\; \forall i \in V(G) \setminus \{r\} && \text{(C8)} \\
&\qquad \sum_{k \in S'} x_{ij}^k \geq 1 \;\; \forall (i,j) \in I && \text{(C9)} \\
&\qquad y_{ij} = 0 \;\; \forall (i,j) \in X && \text{(C10)}
\end{aligned}
$$

Figure 5.5: MIP for Steiner tree with inclusions and exclusions.

root r to the source nodes of the edges that must be included. C7 ensures that there is no incoming active edge into the root. C8 ensures that all nodes have at most one incoming active edge.

5.4.4.2 K-Best Steiner Trees

To obtain the k lowest-cost Steiner trees, where k is a predetermined constant, we use KBESTSTEINER (Algorithm 3), which uses the MIP STEINERIE as a subroutine. KBESTSTEINER is a simple variant of a previous top k answers algorithm [Kimelfeld and Sagiv, 2006, algorithm DQFSearch], which in turn generalizes a previous k-best answers algorithm for discrete optimization problems [Lawler, 1972].

We are not the first to use lowest-cost Steiner trees to rank keyword query results, but we are the first to use the resulting rankings for learning. In addition, in the previous work [Kimelfeld and Sagiv, 2006], the graph represents actual data items and their associations, and the Steiner trees are possible answers containing given keywords. Since data graphs can be very large, the method is primarily of theoretical rather than practical inter-

Algorithm 3: KBESTSTEINER(G, S, C, k). **I**nput: Schema graph G, keyword nodes S, edge cost function C, number of returned trees k. **O**utput: List of at most k trees sorted by increasing cost.

```
1:  $Q \leftarrow$ empty priority queue
    $\{Q$ contains triples $(I, X, T)$ sorted by $T$'s cost.$\}$
2:  $T = $ STEINERIE$(G, S, \emptyset, \emptyset, C)$
3:  **if** $T \neq null$ **then**
4:    $Q$.INSERT$((\emptyset, \emptyset, T))$
5:  **end if**
6:  $A \leftarrow$ empty list
7:  **while** $Q \neq \emptyset \wedge k > 0$ **do**
8:    $k = k - 1$
9:    $(I, X, T) \leftarrow Q$.DEQUEUE$()$
10:   $A$.APPEND$(T)$
11:   Let $\{e_1, \ldots, e_m\} = E(T) \setminus I$
12:   **for** $i = 1$ to $m$ **do**
13:     $I_i \leftarrow I \cup \{e_1, \ldots, e_{i-1}\}$
14:     $X_i \leftarrow X \cup \{e_i\}$
15:     $T_i \leftarrow$ STEINERIE$(G, S, I_i, X_i, C)$
16:     **if** $T_i$ is a valid tree **then**
17:       $Q$.INSERT$((I_i, X_i, T_i))$
18:     **end if**
19:   **end for**
20: **end while**
21: return $A$
```

est. In our application, however, we work on much smaller schema graphs, and each tree corresponds to a whole query that may yield many answers, not a single answer.

5.4.4.3 K-Best Steiner Tree Approximation

As we show in Section 5.6, our Steiner formulation works for medium-scale schema graphs (around 100 nodes). To scale k-best inference to much larger schema graphs, we developed the following novel pruning heuristic.

Shortest Paths Complete Subgraph Heuristic (SPCSH) We explore using reductions [Duin and Volgenant, 1989, Winter and Smith, 1992] to prune the schema graph to scale up KBESTSTEINER to larger schema graphs. SPCSH keeps only the subgraph induced by the m shortest paths between each pair of nodes in S. The intuition for this is that there should be significant edge overlap between the k-best Steiner trees and the subgraph induced by the m-shortest paths, thereby providing good approximation to the original problem while reducing problem size significantly. SPCSH then computes the k-best Steiner trees by invoking KBESTSTEINER on the reduced subgraph.

Algorithm 4: SPCSH(G, S, C, k, m). **I**nput: Schema graph G, keyword nodes S, edge cost function C, number of returned trees k, number of shortest paths to be used m. **O**utput: List of at most k trees sorted by increasing cost.

```
1: $L \leftarrow$ empty list
2: for all $(u, v) \in S \times S$ do
3:     $P \leftarrow G$.SHORTESTPATHS$(u, v, C, m)$
4:     $L$.APPEND$(P)$
5: end for
6: $G_{(S,C,m)} \leftarrow G$.SUBGRAPH$(L)$
7: return KBESTSTEINER$(G_{(S,C,m)}, S, C, k)$
```

In SPCSH, G.SHORTESTPATHS(u, v, C, m) returns at most m shortest (least costly) paths between nodes u and v of G using C as the cost function. Efficient algorithms to solve this problem are known [Yen, 1971]. SPCSH is similar to the distance network heuristic (DNH) for Steiner tree problems on undirected graphs [Winter, 1987, Winter and Smith, 1992], but there are crucial differences. First, DNH works on the set S-induced complete distance network in G while SPCSH uses a subgraph of G directly. Second, DNH uses a minimum spanning tree (MST) approximation while we use exact inference, implemented by KBESTSTEINER, on the reduced subgraph. Third, DNH considers only the shortest path for each vertex pair in $S \times S$, while SPCSH considers m shortest paths for each such vertex pair.

5.4.5 From Trees to Query Templates

The next task is to go from top k Steiner trees to a set of conjunctive queries, all outputting results in a common schema and returning *only* attributes in which the query author is interested. This is accomplished by first converting the top Steiner trees into conjunctive queries; then combining the set of conjunctive queries into a union view that produces a unified output relation; next, supporting user refinement of the view, e.g., to add projections; finally, naming and saving the view persistently with a Web form.

Converting Steiner trees to conjunctive queries. The task of generating conjunctive queries from Steiner trees is fairly straightforward. Each node in the Steiner tree typically represents a relation; traversing an edge requires a join. (In a few cases, a keyword may match on a tuple in, e.g., a topic index or ontology, and now the match represents a selection predicate.) In our implementation, the edges in our schema graph are annotated with the appropriate dereferencing information, typically foreign keys and keys. Here the query is formed by adding relations plus predicates relating keys with foreign keys. We require a query engine that supports queries over remote sources (such as the ORCHESTRA engine we use, described in Section 5.4.6), and we assume the existence of "wrappers" to abstract non-relational data into relational views.

Union view. The union of the queries derived from the top k Steiner trees form a single *union view*. Since each tree query may consist of source data from relations with different schemas, an important question is how to represent the schema for the union view. To create an initial version of the union view, we adopt a variation of the *outer union* (disjoint union) operator commonly used in relational-XML query processing [Carey et al., 2000]. Essentially, we "align" keys and attributes that have the same name, and pad each result with nulls where it does not have attributes.

View refinement. Next, allow the user to refine the view definition by adding projections, or aligning additional attributes from different source relations. This is done through an AJAX-based Web interface, which provides rapid feedback on how user selections affect the output. Projection and attribute alignment are achieved as follows. In a scrollable pane, we create a column for each keyword k_i. Then, for each conjunctive query in the view, we output a row in this pane, in which we populate each column i with the schema of the relation r_i that matches k_i. Each attribute in the schema is associated with a check box — unchecking the check box will project the attribute out from the view. Additionally, there is a text field through which the attribute can be renamed as it is output in the view. If two source attributes are renamed to the same name, then their output will be automatically aligned in the same output column.

Web form. The result of refinement is an intuitive Web-based form created from (and backed by) the view, as previously shown in Figure 5.1b. To reiterate, this form represents not one query but a *family* of queries, as it may be parameterized the the user. The query author will name and save the view and Web form, making it available for parameterization and execution.

5.4.6 Executing a Query

The user of a Web form (who may or may not be its creator) may retrieve the form via a bookmark, or by looking it up by its name and/or description. Each field in the Web form has a check box, which can be deselected to further project out information. The biologist may add selection predicates by filling in values in text boxes, or, for attributes with only a few values, by selecting from a drop-down list. Finally, alongside each item, there is a description of one or more sources from which the attribute is obtained — depending on space constraints — to help the biologist understand what the attribute actually means.

Query execution with provenance. Once the query is parameterized, the user will request its execution. Based on the query or queries that produced it, each tuple output by the query processor receives a *score*, which is the cost of the query that generated it. If a tuple is derived from multiple queries, it receives the lowest (*minimum-cost*) score. Rather than build our own query engine specifically for the Q System, we adopt the query processor used in the ORCHESTRA system [Green et al., 2007a].

When computing query results, ORCHESTRA also records their provenance in the form of a derivation graph, which can be traversed and retrieved. The same tuple may be derived from more than one query: hence in queries produced by the Q System, the provenance of a tuple is a tree-structured representation specifying which queries were applied to which source tuples, in order to derive the result tuple.

The existing ORCHESTRA system encodes provenance as a graph represented in relations, since it must support *recursive* queries whose provenance may be cyclic. Since all queries from the Q System are *tree-structured* and thus acyclic, we modified the query answering system to compute the provenance *in-line* with the query results: each tuple is annotated with a string-typed attribute containing the provenance tree expression, including the keys and names of the specific source tuples, and any special predicates applied (e.g., tests for similarity). This annotation adds only the overhead of casting attributes to strings and concatenating them to query processing — rather than materializing extra relations.

We note that, for situations in which all of the top k queries' cost expressions are independent of tuple data, we can simplify even further, and simply tag each tuple with the ID of the query. However, for regularity across all answers, we use the previous scheme that encodes full details about the source tuples.

In our experience and that of our collaborators, the majority of bioinformatics queries have selective conditions, so we work under the assumption that any given query typically returns few answers. This has an important benefit in our context: it means that we can compute the entire set of answers satisfying the top queries — and as a result, compute the complete provenance for each tuple in terms of the queries. We need this complete information in order to provide proper feedback to the learning stages of the system, which we describe next.

5.5 Learning from Feedback

Interaction with the Q System does not stop once query answers have been returned. Instead, the user is expected to provide feedback that helps the system learn which answers — thus, which queries and ultimately which *features* in the schema graph — are of greater relevance to the user.

The user provides feedback through the Results/Feedback Page, which shows query results in a table. When the user "mouses over" a tuple, Q provides a pop-up balloon showing the provenance of the tuple, in terms of the Steiner tree(s) that produced it; in many situations this is useful in helping the user understand how the tuple was created. The user may click on a button to tell our Q System that a given tuple should be *removed* from the answer set, another button instructing Q to move the tuple to the *top* of the results, or may input a number to indicate a new position this tuple should have in the output. In the cases we consider here, the cost (and thus rank) of a tuple is dependent solely on the query, and therefore the feedback applies to all tuples from the same query.

5.5.1 Basis of Edge Costs: Features

As we discussed previously, edge costs are based on features that allow the Q System to share throughout the graph what it learned from user feedback on a small number of queries. Such features may include the identity of nodes or edge end-nodes, or the overall quality of the match for an edge representing an approximate join. We now define features and their role in costs more precisely. Let the set of predefined features be $F = \{f_1, \ldots, f_M\}$. A feature maps edges to scalar values. In this chapter, all feature values are binary, but in general they could be real numbers measuring some property of the edge. For each edge (i, j), we denote by $\mathbf{f}(i, j)$ the *feature vector* that specifies the values of all the features of the edge. Each feature f_m has a corresponding weight w_m. Informally, lower feature weights indicate stronger preference for the edges that have those features.

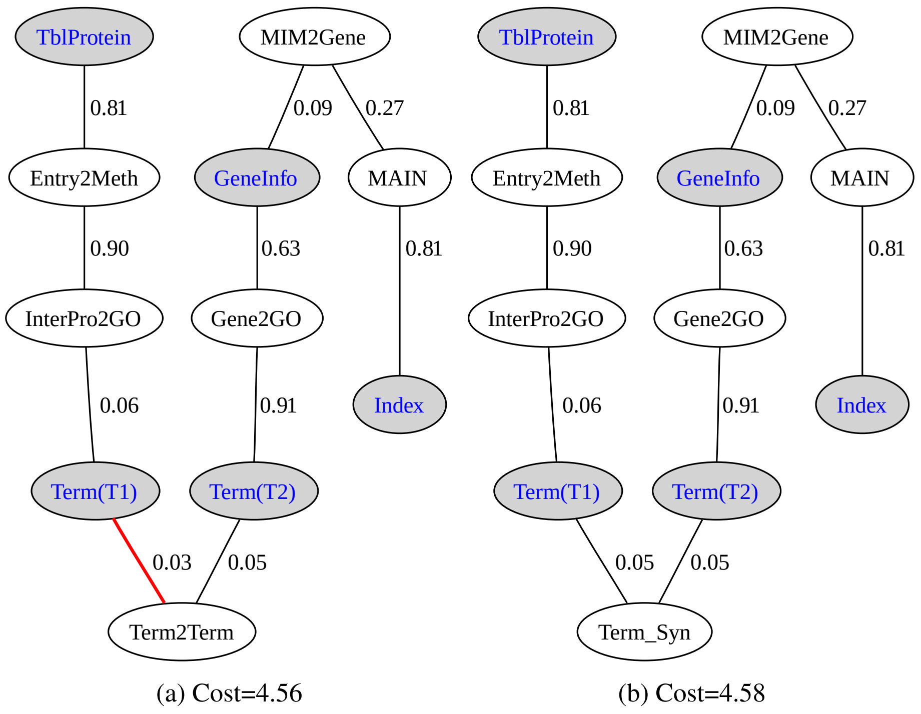


Figure 5.6: Re-ranked Steiner trees with costs updated as discussed in the text. The updated edge is thicker and red.

Edge costs are then defined as follows:

$$C((i,j),\mathbf{w}) = \sum_m w_m \times f_m(i,j) = \mathbf{w}\cdot\mathbf{f}(i,j) \tag{5.2}$$

where m ranges over the feature indices.

To understand features, weights, and the learning process, consider an example with the two Steiner trees in Figure 5.3, which correspond to queries CQ2 and CQ3 in Table 5.1. Their costs are derived from features such as the following, which test the identity of edge end-nodes:

$$f_8(i,j) = \begin{cases} 1 & \text{if } i = \textit{Term(T1)} \ \& \ j = \textit{Term2Term} \\ 0 & \text{otherwise} \end{cases}$$

$$f_{25}(i,j) = \begin{cases} 1 & \text{if } i = \textit{Term(T1)} \\ 0 & \text{otherwise} \end{cases}$$

Suppose that $w_8 = 0.06, w_{25} = 0.02$. Then the score of the edge $(i = \textit{Term(T1)}, j = \textit{Term2Term})$ in Figure 5.3(b)[1] would be $C(i,j) = w_8 \times f_8(i,j) + w_{25} \times f_{25}(i,j) = 0.08$.

Now suppose that, as mentioned in the previous section, the user is presented with tuples generated by the tree queries of Figures 5.3(a) and (b), annotated with provenance information. Since CQ2's tree has a lower cost than CQ3's tree, tuples generated by executing CQ2 are ranked higher. The difference between CQ2 and CQ3 is that while CQ2 uses the synonym relation (*Term_Syn*), CQ3 uses the ontology relation (*Term2Term*). Suppose that the user prefers tuples produced by CQ3 to those produced by CQ2. To make that happen, the learning algorithm would update weights to make the cost of the second tree lower than the cost of the first tree so that in a subsequent execution, tuples from the second tree are ranked higher. Setting $w_8 = 0.01, w_{25} = 0.02$ would achieve this, causing the two tree costs be as shown in Figure 5.6. Of course, the key questions are *which* weights to update, and by *how much*. We now discuss the actual learning algorithm.

5.5.2 Learning Algorithm

We use an *online* learning algorithm, that is, an algorithm that updates its weights after receiving each training example. Algorithm 9 is based on the Margin Infused Relaxed Algorithm (MIRA) [Crammer et al., 2006a]. MIRA has been successfully applied to a number of learning problems on sequences, trees, and graphs, including dependency parsing in natural-language processing [McDonald and Pereira, 2006] and gene prediction in bioinformatics [Bernal et al., 2007].

The weights are all zero as Algorithm 9 starts. After receiving feedback from the user on the r^{th} query S_r about a top answer, the algorithm computes the list B of the k lowest-cost Steiner trees using the current weights. The user feedback for interaction r is represented by the keyword nodes S_r and the *target tree* T_r that yielded the query

[1]For the sake of simplicity, we consider only simple paths here. However, the Q System is capable of handling arbitrary tree structures. This is an improvement over previous systems [Boulakia et al., 2007] that can handle path queries only.

Algorithm 5: ONLINELEARNER(G, U, k). **I**nput: Schema graph G, user feedback stream U, required number of query trees k. **O**utput: Updated costs of edges in G.

```
1: $\mathbf{w}^{(0)} \leftarrow \mathbf{0}$
2: $r = 0$
3: while $U$ is not exhausted do
4:     $r = r + 1$
5:     $(S_r, T_r) = U.\text{NEXT}()$
6:     $C_{r-1}(i, j) = \mathbf{w}^{(r-1)} \cdot \mathbf{f}_{ij} \;\; \forall (i, j) \in E(G)$
7:     $B = \text{KBESTSTEINER}(G, S_r, C_{r-1}, K)$
8:     $\mathbf{w}^{(r)} = \arg\min_{\mathbf{w}} \left\| \mathbf{w} - \mathbf{w}^{(r-1)} \right\|$
9:     s.t. $C(T, \mathbf{w}) - C(T_r, \mathbf{w}) \geq L(T_r, T) \; \forall T \in B$
10:         $\mathbf{w} \cdot \mathbf{f}_{ij} > 0 \; \forall (i, j) \in E(G)$
11: end while
12: Let $C(i, j) = \mathbf{w}^{(r)} \cdot \mathbf{f}_{ij} \; \forall (i, j) \in E(G)$
13: Return $C$
```

answers most favored by the user. The algorithm then updates the weights so that the cost of each tree $T \in B$ is worse than the target tree T_r by a margin equal to the mismatch or *loss* $L(T_r, T)$ between the trees. If $T_r \in B$, because $L(T_r, T_r) = 0$, the corresponding constraint in the weight update is trivially satisfied. The update also requires that the cost of each edge be positive, since non-positive edge costs are not allowed in the Steiner MIP. An example loss function, which is used in the experiments reported in this chapter, is the *symmetric loss*:

$$L(T, T') = |E(T) \setminus E(T')| + |E(T') \setminus E(T)| \tag{5.3}$$

The learning process proceeds in response to continued user feedback, and finally returns the resulting edge cost function.

The edge features used in the experiments of the next section are simply the identities of the source and target nodes, plus a single *default* feature that is on for all edges. The default feature weight serves as a cost offset that is automatically adjusted by Algorithm 9 to ensure that all edge costs are positive.

5.6 Experimental Results

Our Q prototype consists of four primary components. The k-best Steiner tree algorithm uses the MOSEK 5.0 integer linear program solver, run on a dual-core Linux machine (2.6.18.8 kernel) with 12GB RAM. Query refinement is provided by a Java servlet running on Apache Tomcat 6.0.14. Query answering with data provenance is performed by the ORCHESTRA system [Green et al., 2007a], implemented in Java 6 and supported by IBM DB2 9.1 on a Windows 2003, Xeon 5150 server. Finally, the machine learning component is also implemented in Java 6.

5.6.1 Experimental Roadmap

In this chapter, we answer the following questions experimentally:

- Can the system start with default costs on all edges, and based on limited feedback over query answers, generalize the feedback to learn new rankings that enable it to produce "gold standard" (i.e., correct and complete according to expert opinion) queries? How many feedback iterations are necessary?
- How long is the response time (1) in processing feedback and generating new top-k queries, and (2) simply in generating top-k queries from the schema graph?
- How does our performance scale to large, real cross-database schema graphs?

We note that our evaluation focuses purely on the tasks of learning and generating queries. Our interest is not in measuring the response times of the query engine, which is orthogonal to this work. The Q System returns the top k queries in pipelined fashion, and most modern data integration query processors begin pipelining query answers as soon as they receive a query [Chandrasekaran et al., 2003, Ives et al., 1999]. We also do not duplicate the work of [Boulakia et al., 2007, Mork et al., 2002] by performing user studies comparing with existing keyword search systems: that previous work already demonstrated that

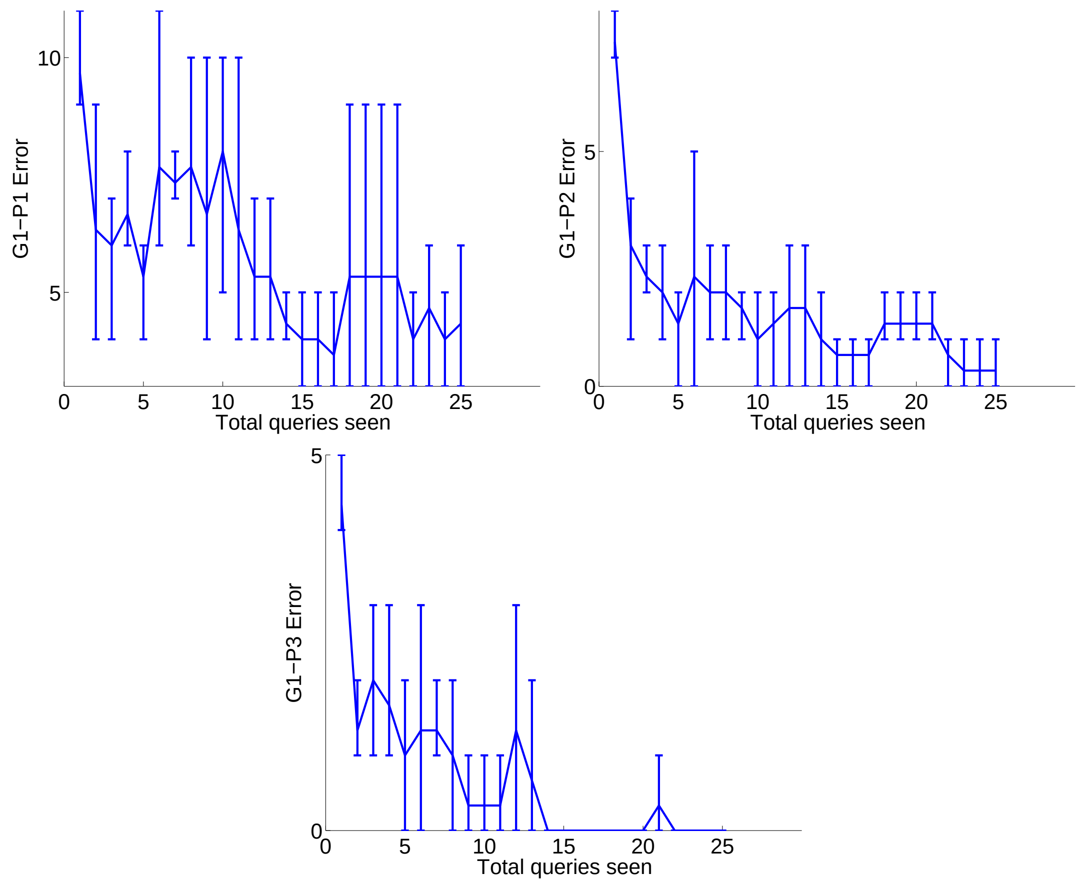


Figure 5.7: Learning curves of top k trees, $k = 1, 2, 3$ against gold standard as feedback is provided, with error bars showing best/worst performance, based on different feedback orders. There are 25 expert queries and the results are averaged over 3 random permutations of the queries.

query answers need to be ranked by source authority/quality and not according to keyword search metrics like path length or term similarity.

Data Sets and Methodology

We conducted our experiments (except for the final experiment focusing on very large schema graphs) using data from a previous biomedical information integration system, BioGuide (`www.bioguide-project.net`) [Boulakia et al., 2007]. BioGuide is, to a significant extent, a baseline for comparison, as it provides ranked answers over a schema graph, given weights set by biological domain experts. The data that the BioGuide developers kindly supplied includes schema graphs with record linking tables between bioinformatics data sources, edge costs determined by a panel of experts based on reliability and completeness judgments, and expert queries (http://bioguide-project.net/project/BioGuideQueryExamples.htm). An example of an expert query is, "What are the related proteins and genes associated with the disease narcolepsy?" From such queries, a set of keywords on concepts can be easily extracted. These form our query workload.

Since BioGuide does not support the kind of learning from feedback we describe here, we used the BioGuide schema graph and set the expert-determined edge costs to create a "gold standard" against which to compare automatic learning. For a given query, the lowest-cost Steiner tree according to expert costs is taken to be what the simulated user prefers, and is used both as feedback in the learning process and as the gold standard for evaluation. Our goal in the near future is to work with our bioinformatics collaborators to deploy the Q System in real applications, and to conduct user studies in this context to confirm our preliminary results.

5.6.2 Learning against Expert Costs

We first investigate how quickly (in terms of feedback steps) the Q System can learn edge costs that yield the same query rankings as the gold standard obtained from expert-provided costs. Note that this is stronger than simply learning, based on feedback, which query the user prefers: our goal is to take feedback over a subset of the queries, and generalize that in a way that lets the system correctly predict which future queries are preferred.

We converted each expert query into a *query template* in which each keyword picks out a single table. For instance, for the narcolepsy query mentioned above, the template would be *"What are the related proteins (in [DB1]) and genes (in [DB2]) associated with disease Narcolepsy in [DB3]?"*. Here, *[proteins], [genes] and [disease]* are entities while *[DB1], [DB2]* and *[DB3]* are table names that need to be filled in. Using substring matching on table names, we populated these query templates by filling in the *[DB]* slots; each such instantiated template forms a query. For the experiments reported in this section, we generated 25 such queries and matched them over the BioGuide schema graph.

We created a feedback stream by randomly selecting a sequence in which the queries will be posed. For each such stream, we paired each query with the corresponding lowest-cost Steiner tree over our schema graph according to expert edge costs. We then applied Algorithm 9 to each stream, with the goal of learning the feature weightings that returned top query. At each point in the stream, our goal is to measure how well the top k algorithm's results for *all of the 25 queries* agree with the gold standard for those queries.

Thus, we simulate the interaction between the algorithm and a user who poses successive queries, examines their answers, supplies feedback about which is the best answer to this query, and moves on to the next query. However, to measure the quality of learning at each point, we need more than just the current query. We also need all the queries that could have been posed, both past and future ones, since the algorithm may change its weights in response to a later interaction in a way that hurts performance with previously submitted queries. The system behavior we aim for is that as this process continues, the queries preferred by the system will agree better with the user's preferences.

The results appear in Figure 5.7. For $k = 1, 2$ & 3, we plot the mean and min/max error bars (across the different random query-feedback stream sequences; note these are not confidence intervals) of how many of the 25 queries fail to have the gold standard tree within the top k trees computed with current weights. We conclude that the learning algorithm converges rapidly: that is, it quickly learns to predict the best Steiner tree consistent with the experts' opinion. After 10-15 query/feedback steps, the system is returning the best Steiner tree in one of the top three positions, and often the top position: Q begins to return the correct queries for *all* queries, given feedback on 40-60% of them.

5.6.3 Feedback and Query Response Time

Given that our goal is to provide an interactive query interface to the user, it is vital that our feedback process, as well as the creation of new top queries based on the feedback, be at a rate that is sufficient for interactivity.

To evaluate the feedback and query generation times, we fix the schema graph and measure (1) the time to process a "typical" user feedback given over a set of top answers, and (2) the time to create a set of queries based on that feedback. Assuming a search engine-like behavior pattern, the user will not look at answers that are beyond the first page of results; moreover, the user will only provide feedback on a few items. Hence, we measured the time taken to retrain and regenerate the set of top queries based on feedback. This took an average of 2.52 sec., which is easily an interactive rate.

A separate but related question is how quickly we can generate queries, given an existing set of weight assignments. Such a case occurs when a user retrieves an existing Web form and simply poses a query over the existing schema graph. We term this the *decoding time*, and Table 5.2 shows the total time it takes to generate the top 1, 5, 10, and 20 queries over the BioGuide schema graph (whose parameters are shown). In general, 5-10 queries should be sufficient to return enough answers for a single screenful of data — and these are returned in 2-4 seconds. Particularly since query generation and query processing can be pipelined, we conclude that response rates are sufficient for user interaction.

Test K	Graph (G) Size $(Nodes, Edges)$	Decoding Time (s)
1	(28, 96)	0.11
5	(28, 96)	2.00
10	(28, 96)	4.02
20	(28, 96)	8.32

Table 5.2: Average per-query decoding times (sec.) for requesting top-1 through -20 results over BioGuide schema.

5.6.4 Schema Graph Size Scalability

We have shown that the Q system scales well to increased user demand for answers. A second question is how well the system scales to larger schema graphs — a significant issue in the life sciences. Given that the Steiner tree problem is NP-hard, we will need to use our SPCSH algorithm (Sec. 5.4.4.3), but now the question is how well it performs (both in running time and precision.) To evaluate this, we used a different real-world schema graph based on mappings between the Genomics Unified Schema (www.gusdb.org), BioSQL (www.biosql.org), and relevant portions of Gene Ontology (www.geneontology.org). We call this the GUS-BioSQL-GO schema. The schema graph had 408 relations (nodes) and a total of 1366 edges. The edge weights were set randomly.

K	KBEST-STEINER (s)	SPCSH (s)	Speedup	Approx. Ratio (α)	Symm. Loss
1	1.2	0.1	12.0	1.0	0
2	43.8	3.0	14.6	1.0	0
3	111.8	5.5	20.3	1.0	0
5	1006.9	13.9	72.4	1.0	0

Table 5.3: Decoding times (sec.) of KBESTSTEINER and SPCSH with K ranging from 1 to 5, and m from 1 to 3. Also shown are the speedup factors, the approximation ratio, α, between the cost of SPCSH's and KBESTSTEINER's top predictions (α = 1.0 is optimal) and the symmetric loss (Equation 6.2) between the top predictions of the two methods. Results were averaged over 10 queries, each consisting of 3 keywords.

We use SPCSH to compute top-K Steiner trees on the GUS-BioSQL-GO schema graph. SPCSH is an approximate inference scheme, while KBESTSTEINER performs exact inference. Hence, the top prediction of KBESTSTEINER is always optimal. In Table 5.3, SPCSH's decoding time and inference quality is compared against KBESTSTEINER on the GUS-BioSQL-GO schema. Table 5.3 demonstrates that SPCSH computes the k-best Steiner trees (for various values of k) at a much faster rate than the previous method, while maintaining quality of prediction (α is 1.0). In fact, in our experiments, SPCSH's predictions were always optimal. We believe this demonstrates that our approach scales to large schema graphs without sacrificing result quality. We also reiterate that the time to generate the first query is of primary importance here: other queries can be pipelined to execution as they are produced.

5.7 Summary of Chapter

In this chapter, we have addressed the problem of helping scientific users author queries over interrelated biological and other data sources, *without* having to understand the full complexity of the underlying schemas, source relationships, or query languages. In general, such scientists would like to rely on expert-provided *weightings* for data sources and associations, and use the best of these to guide their queries. However, experts are not always available to assess the data and associations, and moreover, the utility of a given association may depend heavily on the user's context and information need.

Our approach is based on matching keywords to a schema graph with weighted edges, allowing the user to refine the query, then providing answers with data provenance. As the user provides *feedback* about the quality of the answers, we *learn* new weightings for the edges in the graph (associations), which can be used to refine the query and any related future queries.

We have demonstrated that our approach balances the task of finding the top-k queries with the ability to *learn* new top queries based on feedback. The Q System learns weights that return the top answers rapidly — both in terms of number of interactions, as well as in the computation time required to process the interactions. Using real schemas and mappings, we have shown that we can use an exact top-k solution to handle schema graphs with dozens of nodes and hundreds of edges, and we can easily scale to hundreds of nodes and thousands of edges with our SPCSH approximation algorithm, which in all cases returned the *same* top answers as the exact algorithm.

In the next chapter, we shall explore how information in new sources can be quickly made available to the user within the Q system.

Chapter 6

Automatically Incorporating New Sources in Q

Some parts of this chapter are based on [Talukdar et al., 2010].

6.1 Introduction

In this chapter, we propose a *information need-driven* integration strategy for automatically incorporating new sources and their information in the Q system which was presented in Chapter 5. We shall continue to call the new system Q, as it essentially extends capabilities of the original Q system (presented in Chapter 5), while reusing many of its core components. The improved Q optionally starts with a set of databases that may be interlinked through the use of common identifiers or through correspondence tables, but it does not have a full mediated schema. A user specifies an *information need* through a keyword query. Using ideas from keyword search in databases [Bhalotia et al., 2002, Botev and Shanmugasundaram, 2005, Hristidis and Papakonstantinou, 2002, Kacholia et al., 2005] and keyword-based data integration [Talukdar et al., 2008b], our system defines a ranked view consisting of a union of conjunctive queries over different combinations of the sources. This view is made persistent. As users (or a Web crawler) register new databases, each such source's *relevance* to existing views is considered, using information about data-value overlap as well as schema alignment costs from existing schema matchers. If the source is found to be highly relevant to the query, then the query results are refreshed as appropriate. Now the users of the view may provide *feedback* on the contents of the view: certain new results may be valuable, or possibly erroneous. As the system gets feedback about erroneous results, it adjusts the costs it has assigned to specific mappings or alignments so that associations responsible for the errors are avoided.

Our approach is related to recent work on interactive, user-driven integration, where the system makes a best-effort attempt to get the information, and a user or community of users attempts to refine the results, in particular dataspaces [Franklin et al., 2005] and best-effort integration [Shen et al., 2008]. The key distinctions are that we attempt to *automatically* discover semantic links among data sources, and to use a data-driven approach to providing feedback to the system. Any form of automatic schema alignment is likely to make errors, especially at scale; the challenge is to determine when and where there are mistakes. Simply "eyeballing" the output mapping is unlikely to help identify what is correct. However, if a domain expert is looking at data from the perspective of a particular information need, he or she is (1) likely to invest some effort in ensuring the quality of results, (2) likely to recognize when results do not make sense.

Only very recently has learning been applied to feedback over database query answers, and that work was limited to learning cost assignments for *exact match* conditions. Given a set of keywords and a known set of schema element correspondences, the cross-database query system in [Talukdar et al., 2008b] sought to learn from user feedback the most useful "join paths" for combining the relations. In this chapter we go significantly beyond that work, as the improved Q learns not only how to adjust weights for individual alignments, but also how to combine the outputs from different schema matchers. These tasks require careful formulation of the learning problem, in particular with respect to how learner uses *features* of the potential links between data sources.

We make the following contributions:

- We create a novel "pluggable" architecture that uses matching tools to create alternative potential alignments.
- We develop an automatic, *information need-driven* strategy for schema alignments that, for a given top-k keyword query and a new source, only aligns tables against the new source if there is potential to affect the top-k query results.
- We develop a unified representation for data values and attribute labels, using edge costs to measure relatedness; this facilitates both ranked querying and learning.
- We incorporate state-of-the-art alignment components from the database [Do and Rahm, 2007] and machine learning [Talukdar et al., 2008d] literature, and show how to combine their outputs.
- We propose the use of the random-walk-inspired algorithm *Modified Adsorption (MAD)* [Talukdar and Crammer, 2009], presented in Chapter 4, to detect schema alignments, and study its effectiveness instead of, and in combination with, the COMA++ tool [Do and Rahm, 2007].
- We apply a machine learning algorithm called MIRA [Crammer et al., 2006b], to learn not only correct attribute alignments, but also how to combine information from multiple matching tools. Unlike the learning techniques applied in schema matching tools, our techniques are based on *feedback over answers*.

We experimentally evaluate our techniques over bioinformatics schemas and data, demonstrating effectiveness of the proposed methods.

The remainder of this chapter is structured as follows. First, in Section 6.2 we review the basics of keyword search-based data integration, introduce our specific model, and describe our basic problem setup. Section 6.3 then presents our solution to the problem of determining when a *new source* is relevant to an existing view, through the use of focused schema alignment tasks. Section 6.4 describes how we learn to adjust the alignments among attributes, and their weights, from user feedback. We experimentally analyze our system's effectiveness in Section 6.5. We discuss related work in Section 6.6, before concluding and describing future work in Section 6.7.

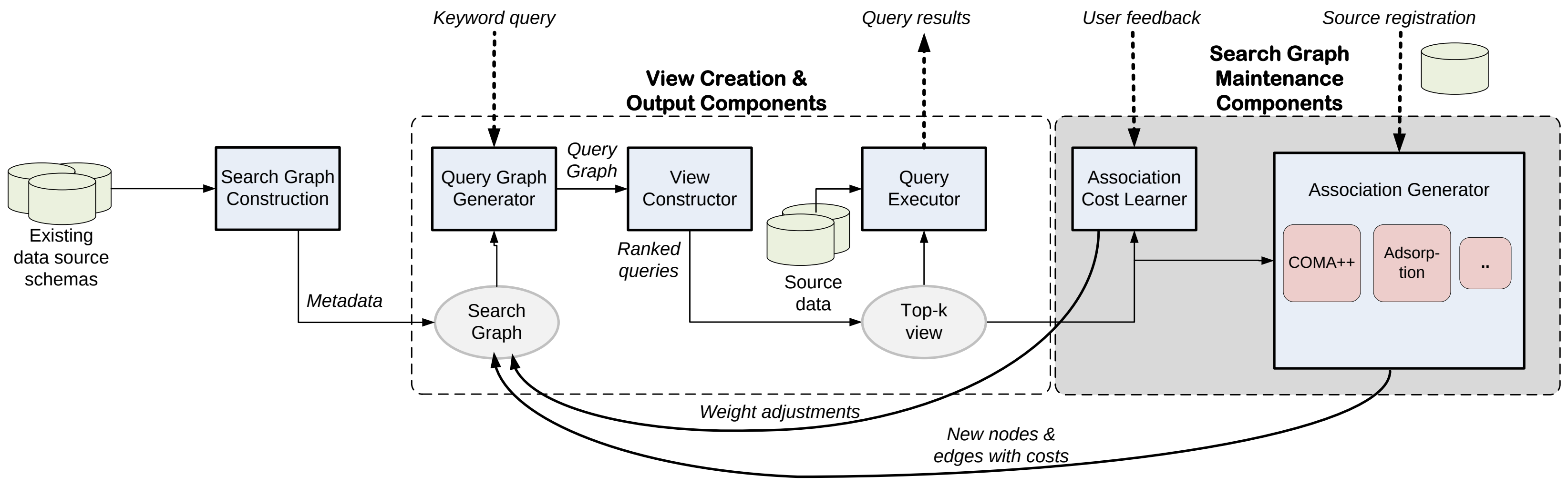


Figure 6.1: Basic architecture of Q. The initial search graph comes from the sources known at startup. At query time this is expanded into a query graph, from which queries and ultimately results are generated. The *search graph maintenance* modules, the focus of this chapter, handle user feedback and accept new source registrations, in order to update the search graph with new alignments — triggering recomputation of the query graph and query results in response.

6.2 Search-Based Integration

This chapter adopts a *keyword search* query model [Bhalotia et al., 2002, Hristidis and Papakonstantinou, 2002, Kacholia et al., 2005, Talukdar et al., 2008b] in which keywords are matched against elements in one or more relations in different data sources. The system attempts to find links between the relations matching the given keywords. Such links are proposed by different kinds of *associations* such as foreign key relationships, value overlaps or global identifiers, similarity predicates, or hyperlinks. In general, there may be multiple relations matching a search keyword, and multiple attribute pairs may align between relations, suggesting many possible ways to join relations in order to answer the query.

Figure 6.1 shows the basic architecture of our Q system. We start with an initial *search graph* generated from existing data source relations and the associations among them. During the *view creation and output* stage, a keyword search is posed against this search graph, and results in a *top-k view* containing answers believed to be relevant to the user. The definition and contents of this view are maintained continuously: both the top-scoring queries and their results may need to be updated in response to changes to the underlying search graph made (1) directly by the user, who may provide feedback that changes the costs of certain queries and thus query answers; (2) by the system, as new data sources are discovered, and their attributes are found to *align* with the existing relations in the search graph, in a way that results in new top-k answers for the user's view. We refer to the process of updating the schema graph's nodes and associations as *search graph maintenance*. In fact, there is interplay between the two graph maintenance mechanisms and the view creation and output stage, as the system may propose an alignment, the view's contents may be updated, the user may provide feedback on these results, and the view output may be updated once again. All of this is focused around alignments that are relevant to the user's ongoing information need.

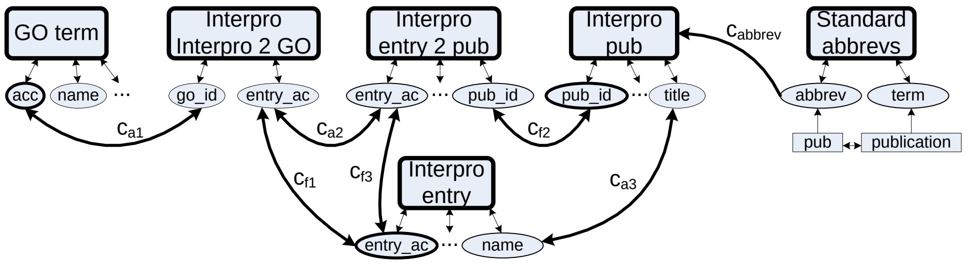


Figure 6.2: Search graph with weighted associations, indicated by bidirectional edges with cost terms c_i. Note the association between the table `pub`, the abbreviation `pub`, and the term `publication`, specified in the `abbrevs` table.

6.2.1 Initial Search Graph Construction

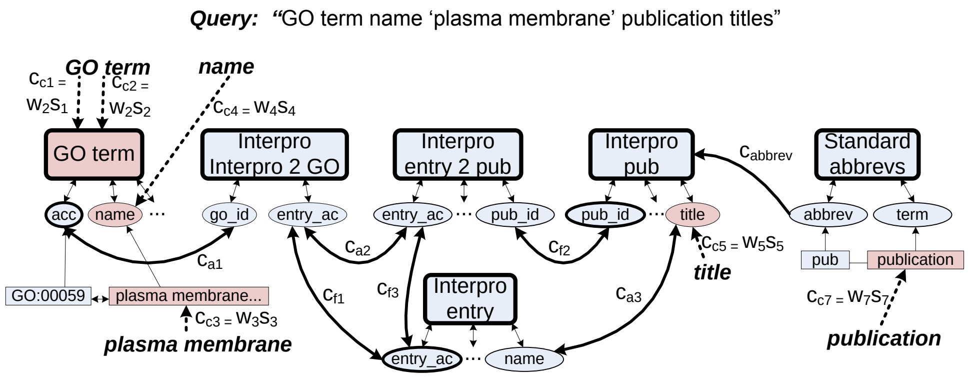


Figure 6.3: Query graph, given a series of keyword search terms. In general, each keyword may match a node with a *similarity score* s_{ci}, for which a *weight coefficient* w_{ci} is to be assigned by the system.

Before any queries are processed, an initial *search graph* is created (leftmost module in Figure 6.1) to represent the relations and potential join links that we already know about. Q first scans the metadata in each data source, determining all attribute and relation names, foreign keys, external links, common identifiers, and other auxiliary information. The basic search graph (see Figure 6.2 for an example) consists of two types of nodes: relations, represented by rounded rectangles, and attributes, represented by ellipses. We add undirected edges between attributes and the relations that contain them (with zero-cost, indicated as thin lines with no annotations), and between tables connected by a key-foreign-key relationship (bold lines with costs $c_{f1}, \ldots, c_{f3}$) initialized to a default foreign key cost c_d.

The graph is extended with bidirectional *association* edges drawn from the results of hand-coded schema alignments (or possibly the results of schema matching tools, such as the ones we consider in this study, which are a *label propagation* algorithm and the COMA++ schema matcher). Such associations may be within the same database (such as those added between `InterPro2GO` and `entry2pub`, or `entry.name` and `pub.title`) or across databases. Each of these associations receives a cost ($c_{a1}, \ldots, c_{a3}$ in Figure 6.2) based on the alignment confidence level.

Each tuple in each of the tables is a virtual node of the search graph, linked by zero-cost edges to its attribute nodes. However, for efficiency reasons we will add tuple nodes as needed for query interpretation. Once the search graph has been fully constructed, Q is ready for querying, and ready to learn adjustments to the costs c_{ci}, c_{aj}, and c_{fk} or to have new association edges added.

6.2.2 Views from Keyword Queries

Given a keyword query $Q = \{K_1, \ldots, K_m\}$, we dynamically expand the search graph into a query graph as follows. For each $K_i \in Q$, we use a keyword similarity metric (by default tf-idf, although other metrics such as edit distance or n-grams could be used) to match the keyword against all schema elements and all pre-indexed data values in the data sources. We add a node representing K_i to the graph (see Figure 6.3, where keyword nodes are represented as boldfaced italicized words). We then add an edge from K_i to each graph node (approximately) matching it. Each such edge is assigned a set of costs, including mismatch cost (e.g., s_2 in the figure) that is lower for closer matches, and costs related to the relevance of the relations connected by the edge. The edge also has an adjustable weight (for instance w_2) that appropriately scales the edge cost to yield an overall edge cost (for instance c_2). Additionally, we "lazily" bring in data values as necessary. For each *database tuple* matching the keyword, we add a node for each value in the tuple, with a similarity edge between the value and the K_i node (e.g., $w_{c3}s_3$ to `plasma membrane`, where s_3 is the mismatch cost and w_{c3} represents the starting weight for that edge). To complete the graph, we add zero-cost edges between tuple value nodes and their corresponding attribute nodes.

From this query graph, each tree with leaf nodes $K_1 \dots K_m$ represents a possible join query (each relation node in the tree, or connected to a node in the tree by a zero-cost edge, represents a query atom, and each non-zero-cost edge represents a join or selection condition). As described in [Talukdar et al., 2008b], Q runs a *top-k Steiner tree algorithm* (using an exact algorithm at small scales, and an approximation algorithm [Talukdar et al., 2008b] at larger scales; STAR [Kasneci et al., 2009] could also be used) to find the k lowest-cost Steiner trees.

From each such tree Q, we generate a conjunctive SQL query that constructs a list of items for the SQL `select`, `from`, and `where` clauses, and an associated cost expression for the particular query. For efficiency reasons, we only incorporate value-based similarity predicates in matching keywords to data or metadata, not in joining one item with another; hence the cost of each query is independent of the tuples being processed. (In ongoing work we are incorporating similarity joins and other operations that vary in cost from one tuple to the next.)

The individual SQL statements must be unioned together in increasing order of associated cost. This actually requires a disjoint or "outer" union: each query may output different attributes, and we want a single unified table for output. However, we would like to place conceptually "compatible" output attributes from different queries into the same column.

We start by defining the query output schema Q_A to match the output schema of the first query's select-list L_A. Then, for each successive query, we iterate over each attribute a in its select-list. Let n_a be the node in the query graph with label a. Suppose there exists some similarity edge $(n_a, n_{a'})$ with cost below a threshold t, and $label(n_{a'})$ appears in Q_A. If the current query is not already outputting an attribute corresponding to $label(n_{a'})$, then we rename attribute a to $label(n_{a'})$ in the output. Otherwise, we simply add a as a new attribute to Q_A. Then we create a multiway disjoint union SQL query, in which each "branch" represents one of the queries produced from a query tree. Each "branch" also outputs a cost (its e term). Finally, we execute the queries and return answers in ranked order, annotated with *provenance* information about their originating queries.

6.2.3 Search Graph Maintenance

The novel aspect of our system is its ability to *maintain* the search graph and adjust the results of existing user queries accordingly, as highlighted on the right side of Figure 6.1. We assume that a user's query has described an ongoing *information need* for that user, and that he or she will make future as well as current use of the query results. Hence we save the results of the query as a view, and we focus on enabling the user to *refine* the view by giving feedback and adjusting the weights given to various associations, and on *incorporating new data sources* if good associations can be found with the existing relations in the search graph, and the contents of these new sources affect the contents of the top-k tuples in the user's view.

The core capabilities for user feedback were addressed in our previous work [Talukdar et al., 2008b], so we concentrate here on discovering new associations (alignments) with relevant sources (Section 6.3), and on using feedback to refine and repair such associations (Section 6.4).

6.3 Adding New Data Sources

Once a keyword search-based view has been defined as in the previous section, Q switches into *search graph maintenance mode*. One crucial maintenance process, discussed in this section, decides if and how to incorporate new sources into the current view as the system is notified of their availability.

Q includes a *registration* service for new tables and data sources: this mechanism can be manually activated by the user (who may give a URL to a remote JDBC source), or could ultimately be triggered directly by a Web crawler that looks for and extracts tables from the Web [Cafarella et al., 2008] or the deep Web [Madhavan et al., 2008, Zhang et al., 2004] .

6.3.1 Basic Approach

When a new source is registered, the first step is to incorporate each of its underlying tables into the search graph. The search graph is in effect the data model queried by Q. It contains both metadata (relation and attribute nodes) and data (tuple values), related by edges that specify possible ways of constructing a query. The lower the cost of an edge, the more likely that the edge will be relevant to answering queries involving one of the nodes it links.

When a new source is encountered, the first step is to determine potential *alignments* between the new source's attributes and those in existing tables: these alignments will suggest (1) potential joins to be used in query answering, and (2) potential alignments of attributes in query output, such that the same column in the query answers contains results from different sources. We note that in both cases, it is desirable that aligned attributes come from the same domains (since, in the case of joins, no results would be produced unless there are shared data values among the attributes).

Of course, this task requires a set of *alignment primitives* (schema matching algorithms) used to match attributes, which we describe in Section 6.3.2. But there are additional architectural challenges that must be faced at the overall system level. As the search graph grows in size, the cost of adding new associations becomes increasingly expensive: regardless of the specific primitives used, the cost of alignment tends to be at least quadratic in the number of compatible attributes. We must find ways of reducing the space of possible alignments considered. Moreover, not all of these proposed alignments may be good ones: most schema matching or alignment algorithms produce false positives.

We exploit the fact that a bad alignment will become apparent when (and *only when*) it affects the top-k results of a user query whose results are closely inspected. We develop an *information need-driven* strategy where we consider only alignments that have the potential

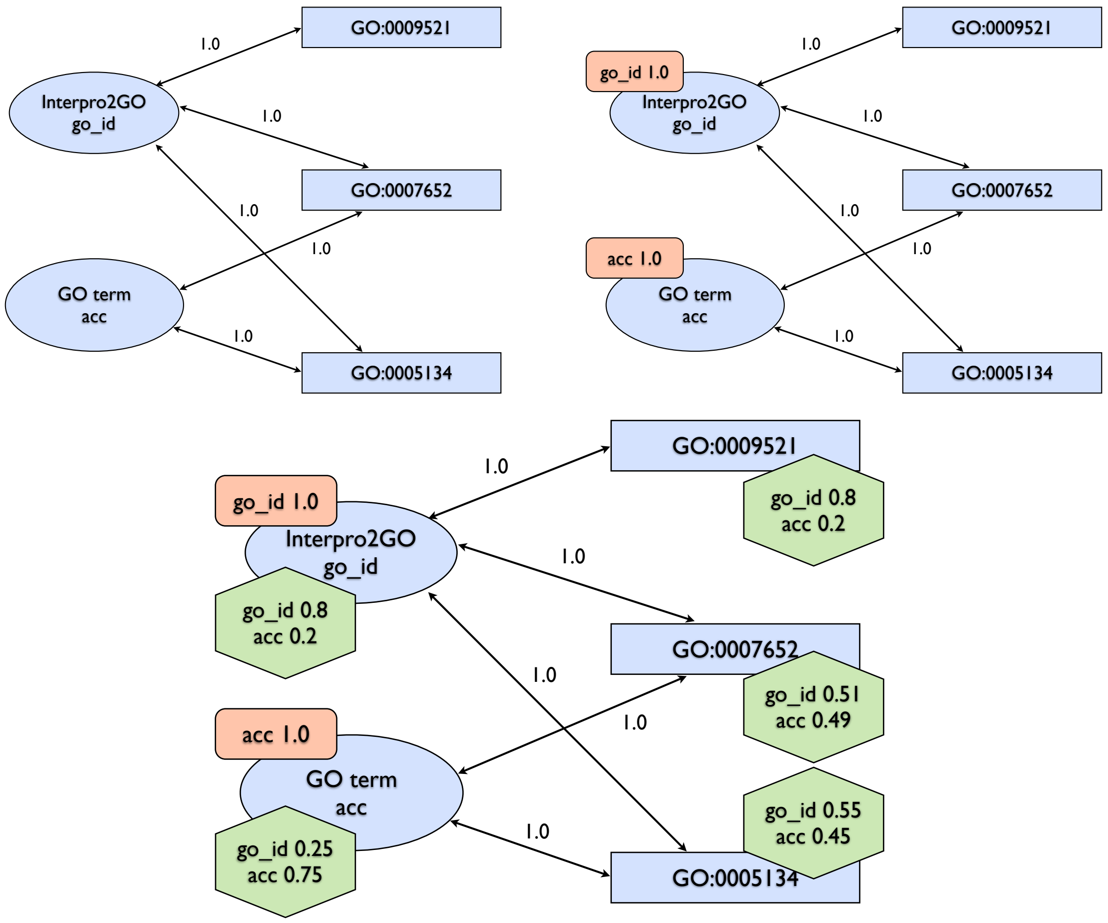


Figure 6.4: Propagation of labels in a column-value graph, using the Modified Adsorption (MAD) algorithm (Section 6.3.2.2). From left to right: the original graph with two column nodes and three value nodes; each column node injected with its own label (labels inside the rectangle); after two iterations of label propagation with estimated labels shown inside hexagons.

to affect existing user queries (Section 6.3.3). As we later show in Section 6.5, this restricts the space of potential alignments to a small subset of the search graph, which grows at a much lower rate than the search graph itself. We then develop techniques for correcting bad alignments through user feedback on the results of their queries (Section 6.4).

6.3.2 Alignment Primitives

Since we focus here on system architecture and learning methods, our goal with Q is to develop an architecture and learning methods that are agnostic as to the specific schema matching or attribute alignment techniques used, such that we can benefit from existing methods in databases and machine learning.

To demonstrate the architecture's ability to accommodate different schema matching algorithms, we incorporate two complementary types of matchers in Q. The first type consists of typical similarity-based schema matchers from the database community that rely on pairwise matches between source and target relations, primarily looking at schema rather than instance-level features, and which we aim to plug into our architecture as "black boxes". The second kind are matchers that globally aggregate the compatibilities between data instances. To that end, we develop a new schema matching technique that looks at "type compatibility" in a way that considers *transitivity*: if attribute A has 50% overlap in values with attribute B, and attribute B has 50% overlap in values with source C, all three attributes likely come from the same domain even if A and C do not share many values. Here we adapt a technique from the machine learning and Web community called *label propagation* that exploits transitivity and data properties, which has not previously been applied to schema matching. We briefly review both kinds of matchers, then describe how we incorporate them into Q.

6.3.2.1 Alignment with Metadata Matcher

Prior work on schema matching has shown that it is useful to consider *multiple* kinds of features, both at the data and metadata level, when determining alignments. Many different schema matchers that incorporate multiple features have been proposed in recent years [Rahm and Bernstein, 2001], with one of the most sophisticated being COMA++ [Do and Rahm, 2007]. The creators of the COMA++ schema matching tool graciously provided a copy of their system, so our specific implementation incorporates COMA++ through its Java API. This system is described in detail elsewhere [Do and Rahm, 2007]. Briefly, we used COMA++'s default structural relationship and substring matchers over metadata to produce proposed alignments[1].

6.3.2.2 Alignment with Label Propagation

Our second matcher focuses on which attributes are type-compatible at the instance level. The notion of *label propagation* has been used in recent machine learning work for finding associated metadata based on weighted transitive relationships across many sources. Informally, this work represents a generalization of some of the ideas in similarity flooding [Melnik et al., 2002] or the Cupid algorithm [Madhavan et al., 2001], but at a larger scale. In label propagation, we are given a graph $G = (V, E, W)$ with nodes V, directed edges E, and a weight function $W : E \rightarrow \mathbb{R}$ that assigns a weight (higher is better) to each edge. Assume some of the nodes $i \in V$ initially are given labels l_i. Labels are propagated from each node along its out-edges to its neighboring nodes with a probability proportional to edge weight, eventually yielding a label probability distribution L_i for each node. Intuitively, this model is similar to PageRank [Brin and Page, 1998], except that it computes how likely a "random surfer" *starting at an initial node with a particular label* will end up at some other node, based on a Markovian (memory- or history-free) behavioral assumption. In this work, we use the Modified Adsorption (MAD) [Talukdar and Crammer, 2009] label propagation algorithm.

MAD is one of a family of related label propagation algorithms used in several areas [Zhu et al., 2003]. While these algorithms can be explained in several ways [Baluja et al., 2008], for simplicity we will rely here on the *random walk* interpretation of MAD.

Let $G_{\mathrm{r}} = (V, E_{\mathrm{r}}, W_{\mathrm{r}})$ be the edge-reversed version of the original graph $G = (V, E, W)$, where $(a, b) \in E_{\mathrm{r}}$ iff $(b, a) \in E$, and $W_{\mathrm{r}}(a, b) = W(b, a)$. Now, choose a

[1]COMA++ also optionally includes instance-level matching capabilities, but despite our best efforts and those of the authors, we were only able to get the metadata matching capabilities of COMA++ to work through its Java API.

node of interest $q \in V$. To estimate L_q for $q \in V$, we perform a random walk on G_{r} starting from q to generate samples for a random label variable L. After reaching a node i during the walk, we have three choices:

1. With probability p_i^{cont}, continue the random walk to a neighbor of i.
2. With probability p_i^{abnd}, abandon the random walk. This abandonment probability makes the random walk stay relatively close to its source when the graph has high-degree nodes. When the random walk passes through such a node, it is likely that further transitions will be into regions of the graph unrelated to the source. The abandonment probability mitigates that effect.
3. With probability p_i^{inj}, stop the random walk and emit either L_i if i is one of the initially labeled nodes.

L_q will converge to the distribution over labels L emitted from random walks initiated from node q. In practice, we use an equivalent iterative fixpoint view of MAD [Talukdar and Crammer, 2009], shown in Algorithm 6. In this algorithm, I_v is the injected label distribution that a node is seeded with; R_v is a label distribution with a single peak corresponding to a separate "none of the above" label $\top$. This dummy label allows the algorithm to give low probability to all labels at a node if the evidence is insufficient. In the next section, we shall see how MAD can be used to discover attribute associations.

6.3.2.3 Combining Matchers in Q

We now describe how we fit each type of matcher into Q, starting with the "black box" interface to COMA++. Later in the chapter, we discuss how we can combine the outputs of multiple matchers, using user feedback to determine how to weigh each one.

COMA++ as a black-box matcher. An off-the-shelf "black box" schema matcher typically does *pairwise* schema matching, meaning that each new source attribute gets aligned with only a *single* attribute in the existing set of data sources (rather than, e.g., an attribute in each of the existing data sources). Moreover, matchers tend to only output their *top* alignment, even when other potential alignments are considered. Our goal in Q is to determine the top-Y (where Y is typically 2 or 3) candidate alignments for each attribute, unless the top alignment has very high confidence: this way we can later use user feedback to "suppress" a bad alignment and see the results of an alternative.

To get alignments between the new source's attributes and all sources, we do a pairwise schema alignment between the new source and each existing source. We thus obtain what COMA++ assumes to be the top attribute alignments between each relation pair.

While we do not do this in our experiments, it is feasible (if expensive) to go beyond this, to force COMA++ to reveal its top-Y overall alignments. Between each pair of schemas, we can first compute the top alignment. Next, for each alignment pair (A, B) that does not have a high confidence level, remove attribute A and re-run the alignment, determining what the "next best" alignment with B would be (if any). Next re-insert A and remove B, and repeat the process. If there are additional schema matching constraints (e.g., no two source attributes may map to the same target attribute), we can again iterate over each alignment pair (A, B). Now remove *all* attributes from A's schema that are "type compatible" with A, except for A itself; and run the alignment. Then replace those attributes, and repeat the process removing attributes type-compatible with B other than B itself.

Ultimately, we will have obtained from the matcher a set of associations (equivalent here to the alignments) and their confidence levels. Depending on the matcher used, the confidence scores may need to be normalized to a value between 0 and 1; in the case of COMA++, its output already falls within this range. These confidence scores will be used in forming a new edge cost (Section 6.3.4).

MAD to discover compatible datatypes. We developed a matcher module (parallelizable for Hadoop MapReduce), which performs MAD across schemas, using techniques described in [Talukdar and Crammer, 2009]. While this matcher implementation is in some sense a part of Q, it is implemented in a way that does not provide any special interfaces, i.e., from Q's perspective it remains a black box. This matcher first creates an internal label propagation graph that incorporates both metadata and data. From the search graph, we take all relation attributes from all sources, and create a node in the label propagation graph for each attribute, labeled with its canonical name. We also take all data values and create a label propagation graph node for each unique value. We add to the graph an edge between a value node and each node representing an attribute in which the value appears. Now we *annotate* or label each attribute node with its name. A sample graph is shown in the left portion of Figure 6.4; for simplicity, all the edges have weight 1.0.

We run the MAD algorithm over this graph, propagating sets of annotations from node to node. The algorithm runs until the label distribution on each node ceases to change beyond some tolerance value. Alternatively, the algorithm can be run for a fixed number of iterations. Each value node ultimately receives a distribution describing how strongly it “belongs” to a given schema attribute, and each attribute node receives a distribution describing how closely it matches other attribute nodes.

In the graph in the second column in the Figure 6.4, we see that the attribute nodes are annotated with labels matching their names, each with probability 1. These labels are propagated to the neighboring nodes and multiple iterations are run until convergence is reached (shown in the rightmost graph). At the end, we see that all data values are annotated with *both* `go_id` and `acc` since there is significant value overlap between the two attributes. Note that MAD does not require direct pairwise comparison of sources. This is very desirable as such pairwise comparisons can be expensive when many sources are involved.

We use the label distributions generated by MAD to generate uncertainty levels from which edge costs will be derived for Q’s search graph. For each node n in the MAD graph, we select the *top-*Y attributes from its label distribution, and we add an edge in the search graph between the attribute node for l and the attribute node for n. The confidence level for each such edge will be $L_n(l)$. Section 6.3.4 describes how this level is combined with other weighted parameters to form an edge cost.

Algorithm 6: Modified Adsorption (MAD) Algorithm

Input: **G**raph: $G = (V, E, W)$, **S**eed labeling: $I_v \in \mathbb{R}^{m+1}$ for $v \in V$, **P**robabilities: $p_v^{inj}, p_v^{cont}, p_v^{abnd}$ for $v \in V$, **L**abel priors: $R_v \in \mathbb{R}^{m+1}$ for $v \in V$, **O**utput: **L**abel Scores: L_v for $v \in V$

```
1: $L_v \leftarrow I_v$ for $v \in V$ {Initialization}
2: $\mathbf{M}_{vv} \leftarrow \mu_1 \times p_v^{inj} + \mu_2 \times p_v^{cont} \times \sum_u \mathbf{W}_{vu} + \mu_3$
3: **repeat**
4:     $D_v \leftarrow \sum_i \left(p_v^{cont} \times \mathbf{W}_{vi} + p_i^{cont} \times \mathbf{W}_{iv}\right) \times I_i$
5:     **for all** $v \in V$ **do**
6:         $L_v \leftarrow \frac{1}{\mathbf{M}_{vv}} \times (\mu_1 \times p_v^{inj} \times I_v + \mu_2 \times D_v +$
7:                                   $\mu_3 \times p_v^{abnd} \times R_v)$
8:     **end for**
9: **until** convergence
```

Algorithm 7: VIEWBASEDALIGNER$(G, G^{'}, K, C, \alpha)$. **I**nput: Search graph G, new source $G^{'}$, keywords (K) associated with current view, cost function C, cost threshold α. **O**utput: Augmented schema graph $G^{''}$, with alignments between G and $G^{'}$.

```
1: $G^{''} \leftarrow$ G $\cup G^{'}$
2: $S \leftarrow \emptyset$
3: **for** $k \in K$ **do**
4:     $S = S \cup$ GETCOSTNEIGHBORHOOD$(G, C, \alpha, k)$
5: **end for**
6: **for** $v \in S$ **do**
7:     A = BASEMATCHER $(G^{'}, v)$
8:     E$(G^{''}) \leftarrow$ E$(G^{''}) \cup A$
9: **end for**
10: Return $G^{''}$
```

6.3.3 Searching for Associations

We just saw how to harness individual schema matchers to find alignments of sources, and hence association edges between existing and new source relations. However, we need to ensure that the alignment algorithms can be applied scalably, as we increase the number of data sources that we have discovered.

Of course, the simplest (though least scalable) approach is to simply perform exhaustive matching: upon the registration of a new data source, we iterate over all existing data

Algorithm 8: PREFERENTIALALIGNER$(G, G^{'}, P)$. **I**nput: Search graph G, new source $G^{'}$, vertex cost function P. **O**utput: Augmented schema graph $G^{''}$, with alignments between G and $G^{'}$.

```
1: $G^{''} \leftarrow$ G $\cup G^{'}$
2: $V_s = $ SORT$(V(G), P)$
3: for $i = 1$ to $\mathbf{V}_s$.length do
4:     r = GETRELATIONNODE ($V_s[i]$)
5:     A = BASEMATCHER ($G^{'}$, r)
6:     E($G^{''}$) $\leftarrow$ E($G^{''}$) $\cup A$
7: end for
8: Return $G^{''}$
```

sources in turn, and run our alignment algorithm(s). We term this approach EXHAUSTIVE, and note that it will scale quadratically in the number of attributes in each source. As we shall see in Section 6.5, even for small numbers of attributes schema alignment takes time, and with large numbers of sources it may be costly to find new associations.

As was previously noted, we can exploit the fact that new associations are only "visible" to users if they appear in any queries returning top-k results. Hence we exploit existing user views, and the existing scores of top-k results, to restrict the search space of alignments. As new queries are materialized within the system, we would incrementally consider further alignments that might affect the results of those queries.

Algorithm 7 shows code for VIEWBASEDALIGNER, which reduces the number of schema alignment comparisons (calls to BASEMATCHER) through a pruning strategy that is guaranteed to provide the same top-k answer set for a query as EXHAUSTIVE. Given an existing schema graph $G = (V, E, C)$ where C is a nonnegative real-valued cost function for each edge (discussed in the next section), a set of keyword nodes K, and the cost α of the kth top-scoring result for the user view, VIEWBASEDALIGNER considers alignments between the new source's schema graph G', and the projection of the graph that is within α of any keyword node. To affect the view output, any new node from G' must be a member of a Steiner tree with cost $\leq \alpha$; given that edge costs are always non-negative, our pruning heuristic guarantees that we have considered all possible alignments that could lead to this

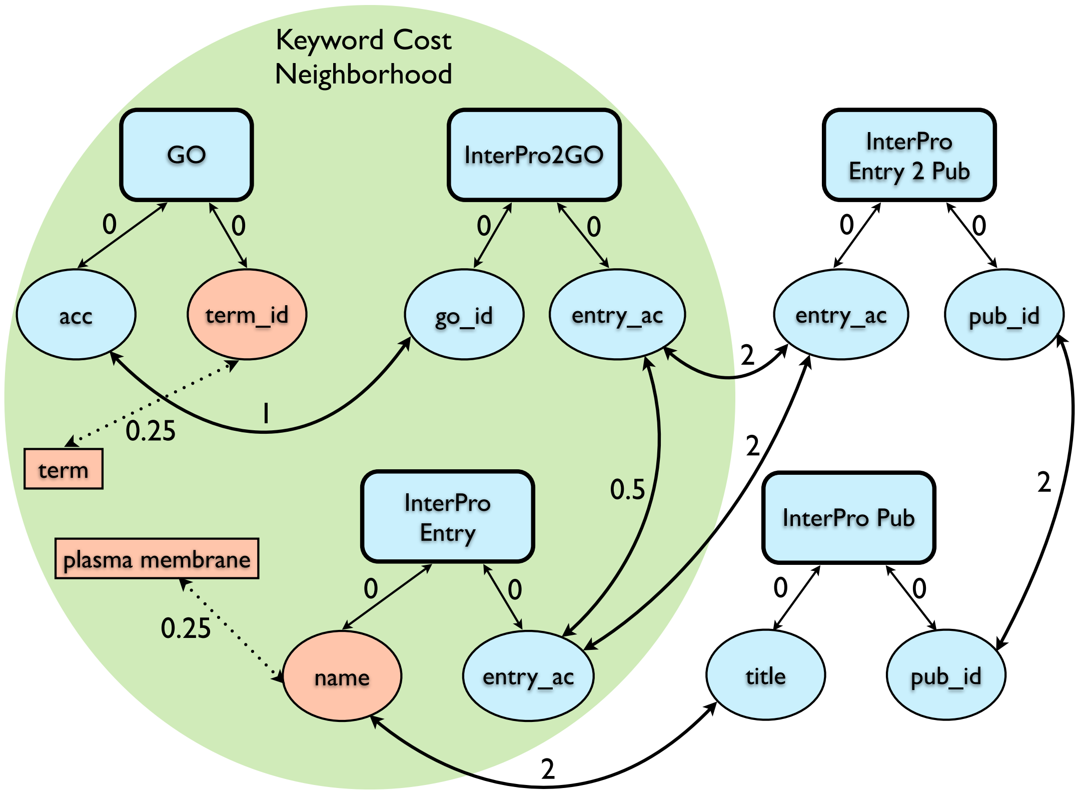


Figure 6.5: A schema graph for the keywords *term* and *plasma membrane*. Edges are annotated with costs. The shaded region is the α-cost neighborhood (α = 2) of the two keywords, i.e. all nodes reachable with cost $\leq$ 2 from a keyword.

condition.

We illustrate this with an example in Figure 6.5, where we assume two keywords have matched and the kth best score $\alpha = 2$. Here, VIEWBASEDALIGNER only considers alignments between a new source and those nodes within the shaded cost neighborhood. This yields savings in comparison with EXHAUSTIVE, which would additionally need to compare the new source against the two sources outside of the region. Of course, in a real search graph many more sources are likely to be *outside* the region than inside it.

If we need even more aggressive pruning, we can adapt ideas from network formation in social networks [Barabasi and Albert, 1999], and assume the existence of an alignment prior (P) over vertices of the existing search graph G, specifying a preference ordering for associations with the existing nodes. This can capture, e.g., that we might want to align with highly authoritative or popular relations. Algorithm 8 shows pseudocode for such a PREFERENTIALALIGNER. A new source, G', is compared against the existing nodes in G in the order of the ranking imposed by the prior P. The prior might itself have been estimated from user feedback over answers of keyword queries, using techniques similar to those of the next section, or it might be computed using alternate methods such as link analysis [Balmin et al., 2004].

6.3.4 Measuring Schema Graph Edge Quality

As we take the output of the aligner and use it to create an association in the search graph, we would like to set the edge cost in a principled way: ideally the value is not simply a hard-coded "default cost," nor just the confidence value of the aligner, but rather it should take into account a number of factors. For instance, the edge cost might take into account costs associated with the relations being joined, derived from their authoritativeness or relevance; and when we are using multiple matchers to create an alignment, we might want to perform a weighted sum of their confidence scores.

We use a cost function for each edge that considers a combination of multiple weighted components, some of which may be shared across edges, and others of which may be exclusive to a specific edge. We formalize this by describing the cost of an edge as a sum of weights times *feature values* (also called *scores*). The weights will be *learned* by Q (Section 6.4), whereas the features are the base cost components whose value does not change. For instance, to incorporate the uncertainty score from a black-box schema matcher, we capture it as a feature, whose associated weight we will learn and maintain. In some cases, we consider features to be Boolean-valued: for instance, if we want to learn a different weight for each edge, then we will create a feature for that edge whose value is 1 for that edge (and 0 elsewhere).

Let the set of predefined features across the search graph be $F = \{f_1, \ldots, f_M\}$. Formally, a feature maps edges to real values. For each edge (i, j), we denote by $\mathbf{f}(i, j)$ the *feature vector* that specifies the values of all the features of the edge. Each feature f_m has a corresponding weight w_m. Informally, lower feature weights indicate stronger preference for the edges that have those features. Edge costs are then defined as follows:

$$C((i,j),\mathbf{w}) = \sum_m w_m \times f_m(i,j) = \mathbf{w} \cdot \mathbf{f}(i,j) \tag{6.1}$$

where m ranges over the feature indices.

When we add a new association edge based on an alignment, we set its cost based on the following weighted features:

- A *default feature* shared with all edges and set to 1, whose weight thus comprises a default cost added to all edges.
- A feature for the confidence value of each schema matcher, whose weight represents how heavily we (dis)favor the schema matcher's confidence scores relative to the other cost components.
- A feature for each relation R connected by the association, whose value is 1 for this relation$R$, and whose weight represents the negated logarithm of the R's *authoritativeness*.
- A feature that uniquely identifies the edge itself, whose value is 1, and whose weight comprises a cost added to the edge.

Together, the weighted features form an edge cost that is initialized based not only on the alignment confidence levels, but also on information shared with other nodes and edges.

6.4 User Feedback & Corrections

When the user sees a set of results, he or she may notice a few results that seem either clearly correct or clearly implausible. In Q the user may provide feedback by optionally annotating each query answer to indicate a valid result, invalid result, or a ranking constraint (tuple t_x should be scored higher than t_y). Q first *generalizes* this feedback by taking each tuple, and, by looking at its provenance, replacing it with the *query tree that* *produced it*, using a scheme similar to [Talukdar et al., 2008b]. Recall that our model is one of tuple and edge *costs* so a lower cost results in higher ranking.

The *association cost learner* converts each tuple annotation into a constraint as follows:

- A query that produces correct results is constrained to have a cost at least as low as the top-ranked query result.
- A query Q_x that should be ranked above some other query Q_y is constrained to have a cost that is lower than Q_y's cost.

These constraints are fed into an algorithm called MIRA [Crammer et al., 2006b], which has previously been shown to be effective in learning edge costs from user feedback on query results [Talukdar et al., 2008b]. We briefly summarize the key ideas of MIRA here, and explain how we are using it in a less restricted way here, learning over *real-valued* features, as opposed to the Boolean features in the previous work [Talukdar et al., 2008b].

Relationship between Edge Costs and Features. Recall from Section 6.3.4 that each edge is initialized with a cost composed of multiple weighted features: the product of the weight and the feature value comprise a default cost given to every edge, a weighted confidence score from each schema alignment algorithm, the authoritativeness of the two relations connected by the edge, and an additional cost for the edge itself. Q's association cost learner takes the constraints from user feedback and determines a weight assignment for each feature — thus assigning a cost to every edge.

Learning Algorithm. The learning algorithm (Algorithm 9) reads training examples sequentially and updates its weights after receiving each of the examples based on how well the example is classified by the current weight vector. The algorithm, which was first used in [Talukdar et al., 2008b], is a variant of the Margin Infused Ranking Algorithm (MIRA) [Crammer et al., 2006b]. We previously showed in [Talukdar et al., 2008b] that MIRA effectively learning top-scoring queries from user feedback; however, in that work only binary features were used, while here we need to include real-valued features from similarity costs. Using real-valued features directly in the algorithm can cause poor learn-

Algorithm 9: ONLINELEARNER(G, U, k). **I**nput: Search graph G, user feedback stream U, required number of query trees k, zero-cost constraint edges A. **O**utput: Updated costs of edges in G.

```
1:  $\mathbf{w}^{(0)} \leftarrow \mathbf{0}$
2:  $r = 0$
3:  while $U$ is not exhausted do
4:      $r = r + 1$
5:      $(S_r, T_r) = U.\text{NEXT}()$
6:      $C_{r-1}(i,j) = \mathbf{w}^{(r-1)} \cdot \mathbf{f}_{ij} \ \ \forall (i,j) \in E(G)$
7:      $B = \text{KBESTSTEINER}(G, S_r, C_{r-1}, K)$
8:      $\mathbf{w}^{(r)} = \arg\min_{\mathbf{w}} \left\| \mathbf{w} - \mathbf{w}^{(r-1)} \right\|$
9:      s.t. $C(T, \mathbf{w}) - C(T_r, \mathbf{w}) \geq L(T_r, T), \ \ \forall T \in B$
10:             $\mathbf{w} \cdot \mathbf{f}_{ij} = 0 \ \forall (i,j) \in A$
11:             $\mathbf{w} \cdot \mathbf{f}_{ij} > 0 \ \forall (i,j) \in E(G) \setminus A$
12: end while
13: Let $C(i,j) = \mathbf{w}^{(r)} \cdot \mathbf{f}_{ij} \ \forall (i,j) \in E(G)$
14: Return $C$
```

ing because of the different ranges of different real-valued and binary features. Therefore, as described above, we *bin* the real-valued features into empirically determined bins; the real-valued features are then replaced by features indicating bin membership.

The weights are all zero as Algorithm 9 starts. After receiving feedback from the user on the r^{th} query S_r about a top answer, the algorithm retrieves the list B of the k lowest-cost Steiner trees using the current weights. The user feedback for interaction r is represented by the keyword nodes S_r and the *target tree* T_r that yielded the query answers most favored by the user. The algorithm then updates the weights so that the cost of each tree $T \in B$ is worse than the target tree T_r by a margin equal to the mismatch or *loss* $L(T_r, T)$ between the trees. If $T_r \in B$, because $L(T_r, T_r) = 0$, the corresponding constraint in the weight update is trivially satisfied. The update also requires that the cost of each edge be positive, since non-positive edge costs will result in non-meaningful Steiner tree computations. To accomplish this, we include the *default feature* listed above, whose weight serves as a uniform cost offset to all edge weights in the graph, which keeps the edge costs positive. Some edges in the query graph are constrained to have a fixed edge cost, irrespective of learning. For example, attribute-relation edges have a cost of zero that should always be maintained. We achieve this by adding such constraints to the MIRA algorithm. Our implementation requires a modification of MIRA (shown in Algorithm 9) that takes as input a set A specifying edges with zero cost constraints.

An example loss function, used in our experiments, is the *symmetric loss* with respect to the edges E present in each tree:

$$L(T, T') = |E(T) \setminus E(T')| + |E(T') \setminus E(T)| \tag{6.2}$$

The learning process proceeds in response to continued user feedback, and finally returns the resulting edge cost function.

6.5 Experimental Analysis

In this section, we use Q as a platform to validate our strategy of performing schema alignment in a query-guided manner (Section 6.5.1), as well as our techniques for using user feedback over data to correct bad alignments (Section 6.5.2). The search graph maintenance modules in Q comprise approximately 4000 lines of Java code, and all experiments were run on a Dell PowerEdge 1950 computer running RedHat Enterprise Linux 5.1 with 8GB RAM. We used the COMA++ 2008 API, and a Java-based implementation of our MAD-based schema matcher.

Our focus in Q is on supporting bioinformatics applications, and hence wherever possible, we use real biological databases and compare with *gold standard* results, i.e., reference results supplied by domain experts. This enables us to perform an experimental study without having to conduct extensive user studies.

For the first set of experiments, we use a dataset for which we have logs of actual SQL queries executed by Web forms, such that we can determine which proposed source associations are actually valid (as witnessed by having real queries use them). This dataset, GBCO[2], consists of 18 relations (which we model as separate sources) with 187 attributes.

[2]http://www.betacell.org/

In the second set of experiments, we used a different dataset, based on the widely used (and linked) Interpro and GO databases, where we could obtain *keyword queries* and find multiple alternative means of answering these queries. This dataset consists of 8 closely interlinked tables with 28 attributes.

6.5.1 Incorporating New Sources

We first look at the cost of adding new data sources to an existing search graph, in a way that keeps the alignment task tractable by limiting it to the "neighborhood" of an existing query. We set up the experiment, using the GBCO dataset described above, as follows.

We first scanned through the GBCO query logs for *pairs* of SQL queries, where one query represented an *expansion* of the other, *base*, query: i.e., the *expanded query* either joined or unioned additional relations with the base query. Intuitively, the expanded query tells us about new sources that would be useful to add to an existing search graph that had been capable of answering the base query. When the expanded query represents the union of the base query with a new query subexpression, then clearly adding the new data source results in new association edges that provide further data for the user's view. When the expanded query represents an additional join of the base query with new data, this also affects the contents of the existing view if the additional join represents a segment of a new top-scoring Steiner tree for the same keyword query.

For each base query, we constructed a corresponding keyword query, whose Steiner trees included the relations in the base query. Next, we initialized the search graph to include all sources *except* the ones unique to the expanded query. We initially set the weights in the search graph to default values, then provided feedback on the keyword query results, such that the SQL base query from our GBCO logs was returned as the top query. For all experiments in this section, the edge costs learned in the process were used as the value of the function C in the VIEWBASEDALIGNER algorithm. The vertex cost function P in PREFERENTIALALIGNER was similarly estimated from the weights of features corresponding to source identities.

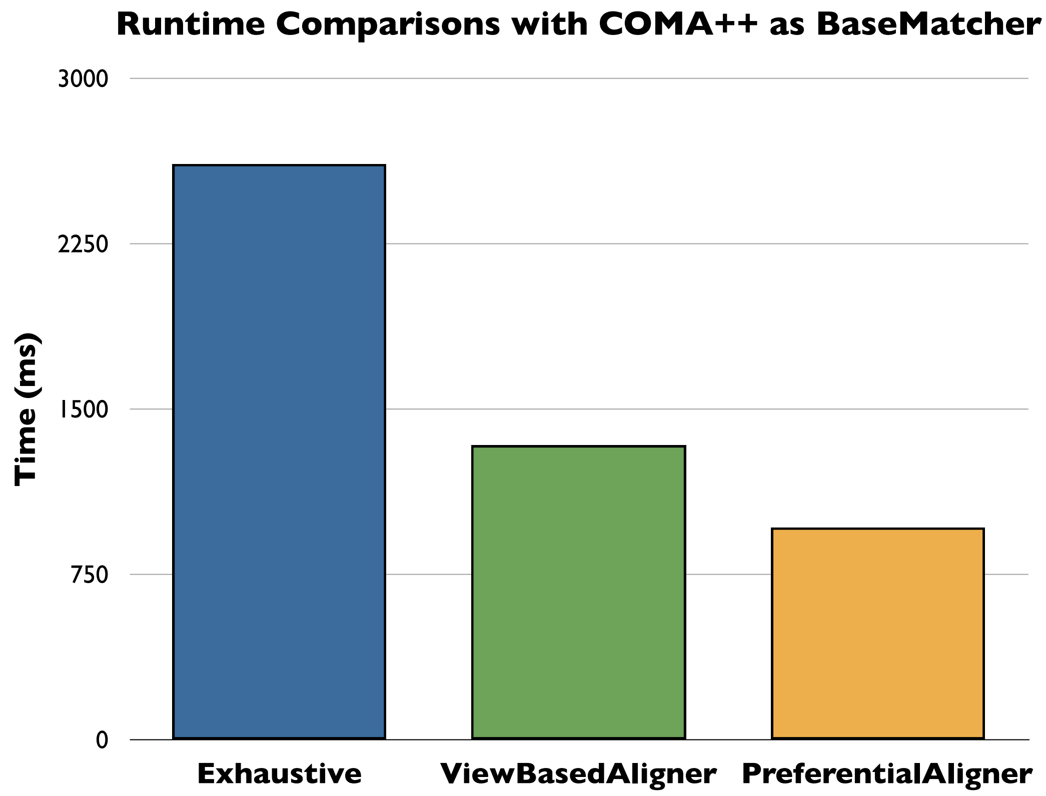


Figure 6.6: Running times (averaged over intro of 40 sources) when aligning a new source to a set of existing sources (COMA++ as base matcher). VIEWBASEDALIGNER and PREFERENTIALALIGNER significantly reduce running times vs. EXHAUSTIVE.

6.5.1.1 Cost of Alignment

Our first experiment measures the cost of performing alignments between the new source and a schema graph containing all of the other sources — using our EXHAUSTIVE, VIEWBASEDALIGNER, and PREFERENTIALALIGNER search strategies, with the COMA++ matcher. Figure 6.6 compares the running times of these strategies. Figure 6.7 shows the number of pairwise attribute comparisons necessary, under two different sets of assumptions. The *Value Overlap Filter* case assumes we have a content index available on the attributes in the existing set of sources and in the new source; we only make compare attributes that have shared values (hence can join). More representative is likely to be the *No Additional Filter* case, which has only metadata to work from.

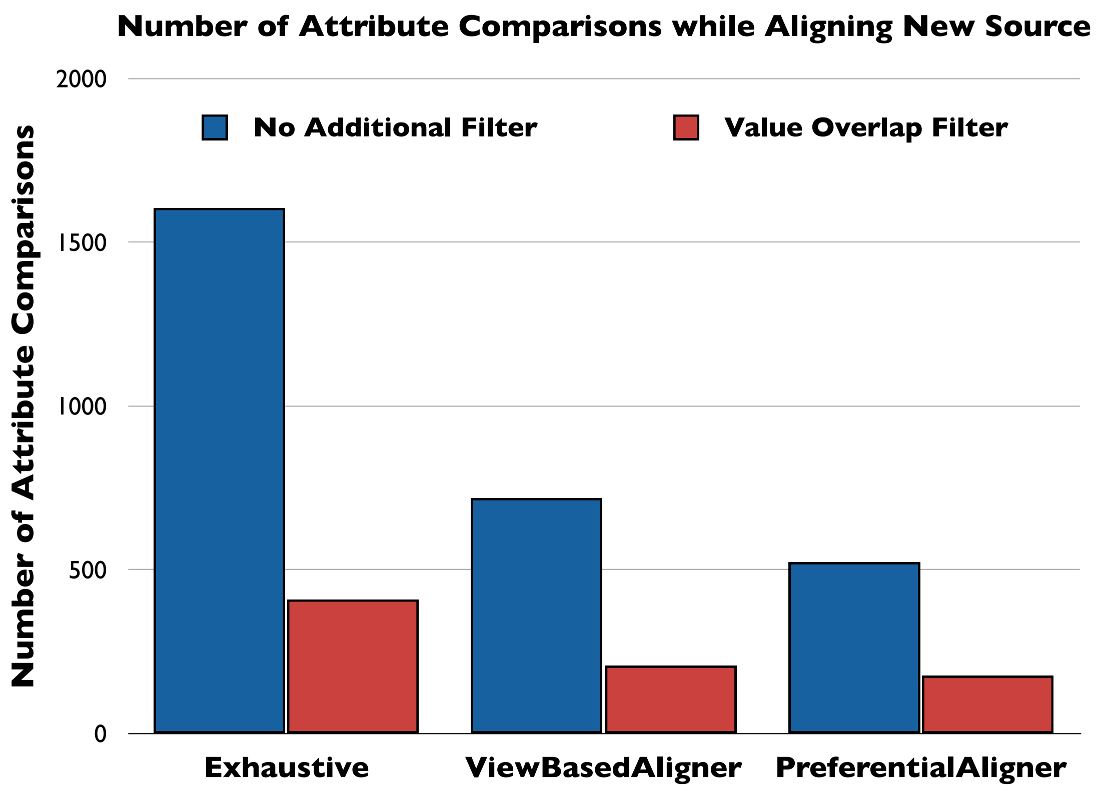


Figure 6.7: Pairwise attribute comparisons performed in aligning new source(s) to existing sources (averaged over intro of 40 sources in 16 trials, where each trial introduces one or more new sources). VIEWBASEDALIGNER and PREFERENTIALALIGNER significantly reduce comparisons vs. EXHAUSTIVE.

We observe that, regardless of whether a value overlap filter is available, limiting the search to the neighborhood of the existing query (i.e., our information need-driven pruning strategy) provides significant speedups (about 60%) versus doing an exhaustive set of comparisons, even on a search graph that is not huge. Recall that VIEWBASEDALIGNER will provide the *exact same* updates to a user view as the exhaustive algorithm. PREFERENTIALALIGNER does not have this guarantee, and instead focuses on the alignments specified in the prior, but gives even lower costs.

The differences in costs results from the fact that the number of comparisons in EXHAUSTIVE depends on the number of source relations in the schema graph, whereas the number of comparisons in the other cases is only dependent on the number of nodes in the

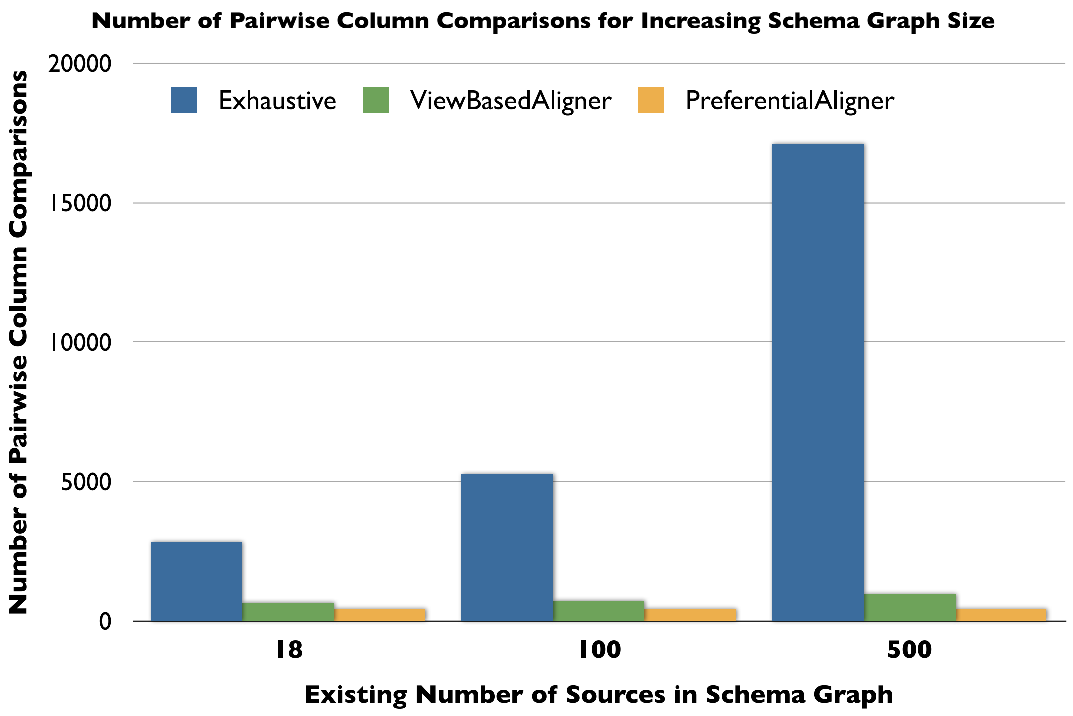


Figure 6.8: Number of pairwise attribute comparisons as the size of the search graph is increased (averaged over introduction of 40 sources). VIEWBASEDALIGNER and PREFERENTIALALIGNER are hardly affected by graph size.

local neighborhood of the query.

6.5.1.2 Scaling to Large Number of Sources

We next study how the cost of operations scales with respect to the search graph size. Since it is difficult to find large numbers of interlinked tables "in the wild," for this experiment we generated additional synthetic relations and associations for our graph. We started with the real search graph, and built upon it as follows. We initialized the original schema graph of 18 sources with default costs on all edges. Then we took our set of keyword queries and executed each in sequence, providing feedback on the output such that the base query was the top-scoring one. At this point, the costs on the edges were calibrated to provide meaningful results. Now we randomly generated new sources with two attributes, and then connected them to two random nodes in the search graph. We set the edge costs to the average cost in the calibrated original graph.

Once the schema graph of desired size was created, the three alignment methods were used to align the new sources in the expanded graph. Since our mostly-synthetic expanded search graph does not contain realistic node labels and attributes, we do not directly run COMA++ on the results, but instead focus on the number of column comparisons that must be performed. The results appear in Figure 6.8. Recall that Figure 6.6 shows that COMA++'s running times grow at a rate approximately proportional to the number of column comparisons. From Figure 6.8, we observe that the number of pairwise column comparisons needed by VIEWBASEDALIGNER and PREFERENTIALALIGNER remained virtually unchanged as the number of sources increased from 18 to 500, whereas EXHAUSTIVE grew quite quickly.

We conclude from the experiments in this subsection that localizing the search to the neighborhood around a query yields much better scalability. VIEWBASEDALIGNER gives the same results as the exhaustive strategy, and hence is probably the preferred choice.

6.5.2 Correcting Matchings

The previous section focused on the cost of running alignment algorithms, without looking at their quality. We now look at how well Q takes the suggested alignments from the individual alignment algorithms, as well as user feedback on query answers, to get the correct associations. These experiments were conducted over the InterPro-GO dataset described previously (shown visually in Figure 6.9), for which we were able to get a set of keyword queries based on common usage patterns suggested in the description of the GO and InterPro databases[3]. We know from the original schema specifications and documentation that there are 8 semantically meaningful join or alignment edges among these relations, but we remove this information from the metadata.

[3]`http://www.ebi.ac.uk/interpro/User-FAQ-InterPro.html`

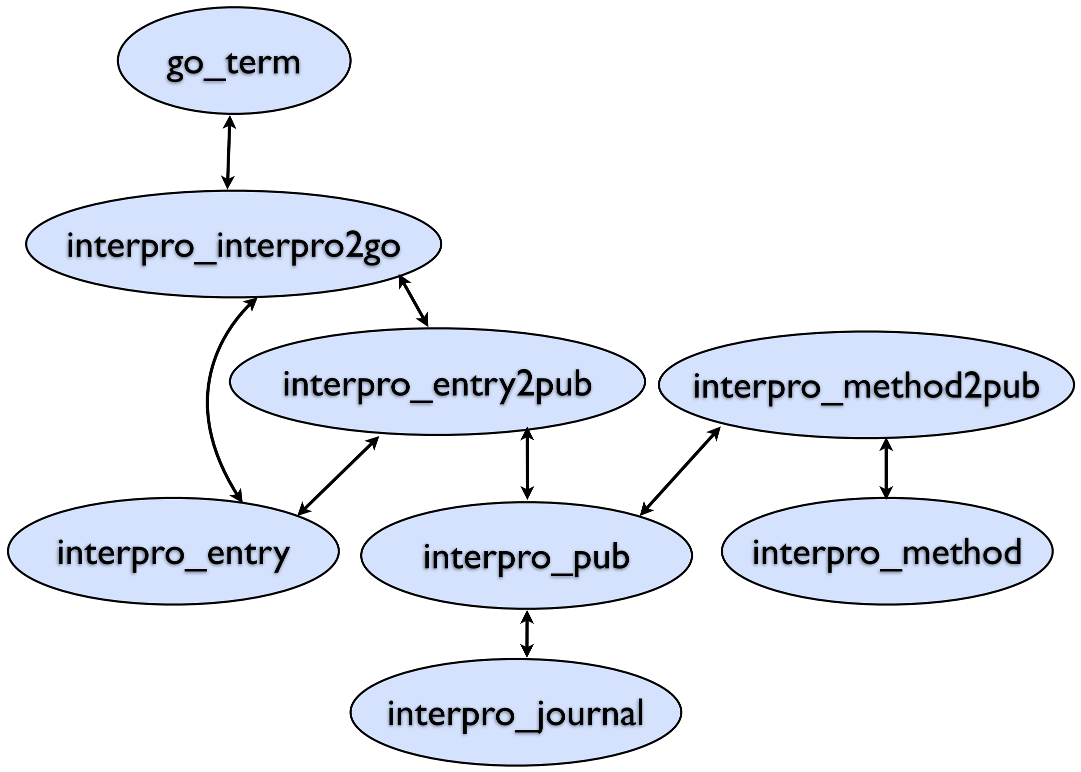


Figure 6.9: Schema graph used in the experiments of Section 6.5.2 (attributes are not shown).

Our experimental setup is to start with a schema graph that simply contains the tables in Figure 6.9, and then to run the association generation step (using COMA++ and/or MAD) to generate a search graph in the Y most promising alignments (for different values of Y) are recorded for each attribute. Next we execute the set of keyword queries obtained from the databases' documentation. For each query, we generate one *feedback response*, marking one answer that only makes use of edges in the gold standard. Since the gold standard alignments are known during evaluation, this feedback response step can be simulated on behalf of a user. Our goal is to "recover" all of the links shown in Figure 6.9, which forms the *gold standard*.

We now present our results using precision, recall and F-measure as our evaluation metrics. We compute these metrics with respect to the search graph, as opposed to looking

Y	System	Precision	Recall	F-measure
1	COMA++	62.5	62.5	62.5
	MAD	70	87.5	77.78
2	COMA++	63.64	87.5	73.68
	MAD	66.67	100	80
5	COMA++	63.64	87.5	73.68
	MAD	66.67	100	80

Table 6.1: Evaluation of top-Y edges (per node) induced by COMA++ and MAD for various values of Y (see Section 6.5.2.1). The schema graph of Figure 6.9 was used as the gold reference. Precision-Recall plots for COMA++ and MAD for the Y = 2 case are are shown in Figure 6.10.

at query answers. For different values of Y, we compare the top Y alignment edges in the search graph (that also fall under a cost threshold) for each attribute, versus the edges in the gold standard. Clearly, if the alignment edges in the schema graph exactly match the gold standard, then they will result in correct answers.

6.5.2.1 Baseline Matcher Performance

Our first set of experiments compares the relative performance of the individual matchers over our sample databases, as we increase the number of alternate attribute alignments we request from the matcher in order to create the search graph. We briefly describe setup before discussing the results.

COMA++ setup. As described in Section 6.3.2.1, COMA++ [Do and Rahm, 2007] was applied as a pairwise aligner among the relations in Figure 6.9. This involved computing alignments and scores in COMA++ for attributes in each pair of relations. Using this scheme we were able to induce up to 34 alignment edges.

MAD setup. We took the relations in Figure 6.9 and the values contained in the tables, and constructed a MAD graph resembling Figure 6.4. All nodes with degree one were pruned out from the MAD graph before the matching algorithm was run, as they are unlikely to contribute to the label propagation. Also, all nodes with numeric values were removed, as they are likely to induce spurious associations between attributes. The resulting graph had 87K nodes. We used the heuristics from [Talukdar and Crammer, 2009] to set the random walk probabilities.

MAD was run for 3 iterations (taking approximately 4 seconds total), with $\mu_1 = \mu_2 = 1$, and $\mu_3 = 1e{-}2$. Each unique column name (attribute) was used as a label, and so 28 labels were propagated.

Results. For each of the algorithms, we added to the search graph (up to) the top-Y-scoring alignments per attribute, for Y values ranging from 1 to 5, as shown in Table 6.1. Our general goal is to have the matchers produce 100% recall, even at the cost of precision: the Q learner must be able to find the correct alignment in the search graph if it is to be able to allow for mapping correction.

We conclude that our novel MAD scheme, which is purely based on data values, does very well in this bioinformatics setting, with a recall of 7 out of 8 edges even with $Y = 1$, and 100% recall with $Y = 2$. COMA++ produced good output (7 out of 8 alignments) with $Y = 2$, but we were not able to get it to detect all of the alignments even with high Y values.

Note that we compute precision under a fairly strict definition, and one might compellingly argue that some of the "wrongly" induced alignments are in fact useful in answering queries, even if they relate attributes that are not synonymous. For instance, if we look at the "incorrect" edges induced by MAD, we see one between `interpro.method.name` and `interpro.entry.name`. The data shows an overlap of 780 distinct values (out of 53,007 entries in `interpro.method.name` and 14,977 in `interpro.entry.name`). Joining these two tables according to this alignment may in fact produce useful results for exploratory queries (even if these results should be given a lower rank in the output). We hope in the future to conduct user studies to evaluate how useful biologists find Q's answers.

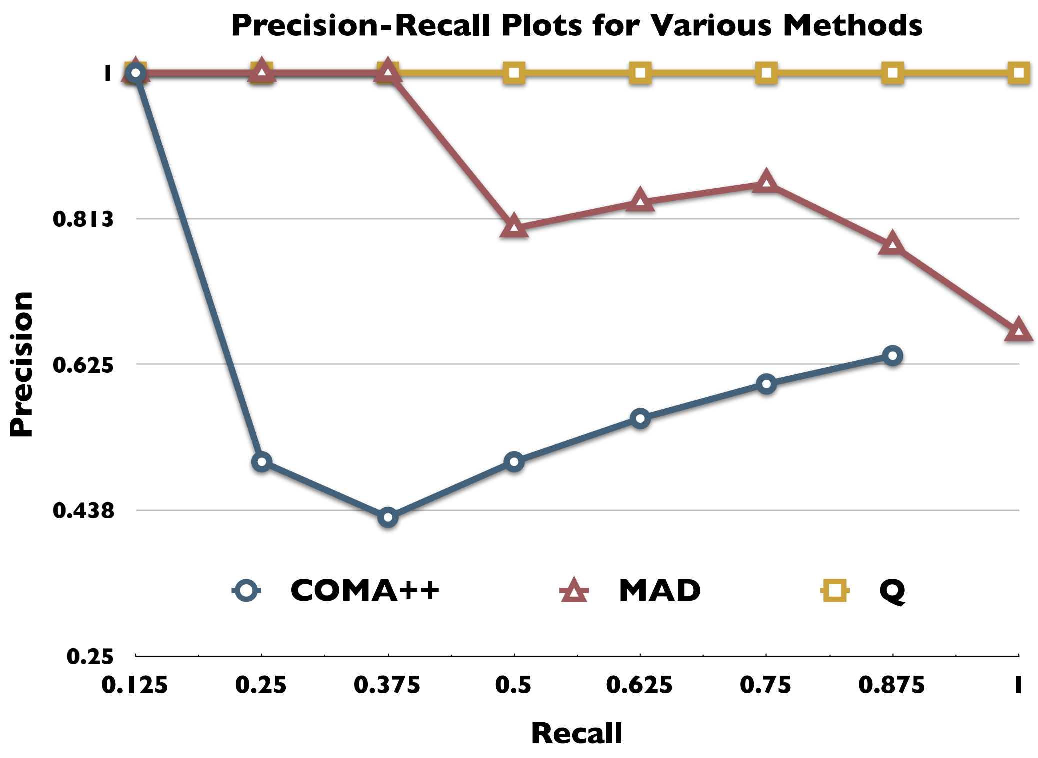


Figure 6.10: Precision vs. recall for COMA++, MAD and Q (which combines COMA++ and MAD). Q was trained from feedback on 10 keyword queries, replayed three times to reinforce the feedback. Precision and Recall were computed by comparing against the foreign key associations in Figure 6.9.

6.5.2.2 Correcting Associations

We next study Q's performance in combining the output of the two matchers, plus processing feedback to correct alignments. This performance (measured in precision and recall) is dependent on how high a similarity (how low a cost) we require between aligned attributes. Generally, the more strict our similarity threshold, the better our precision and the lower our recall will be.

Benefits of learning. In Figure 6.10, we take the schema alignments from both matchers (COMA++ and MAD) when $Y = 2$ (the lowest setting where we get 100% recall, see Table 6.1) and combine them, then provide feedback on 10 different two-keyword queries

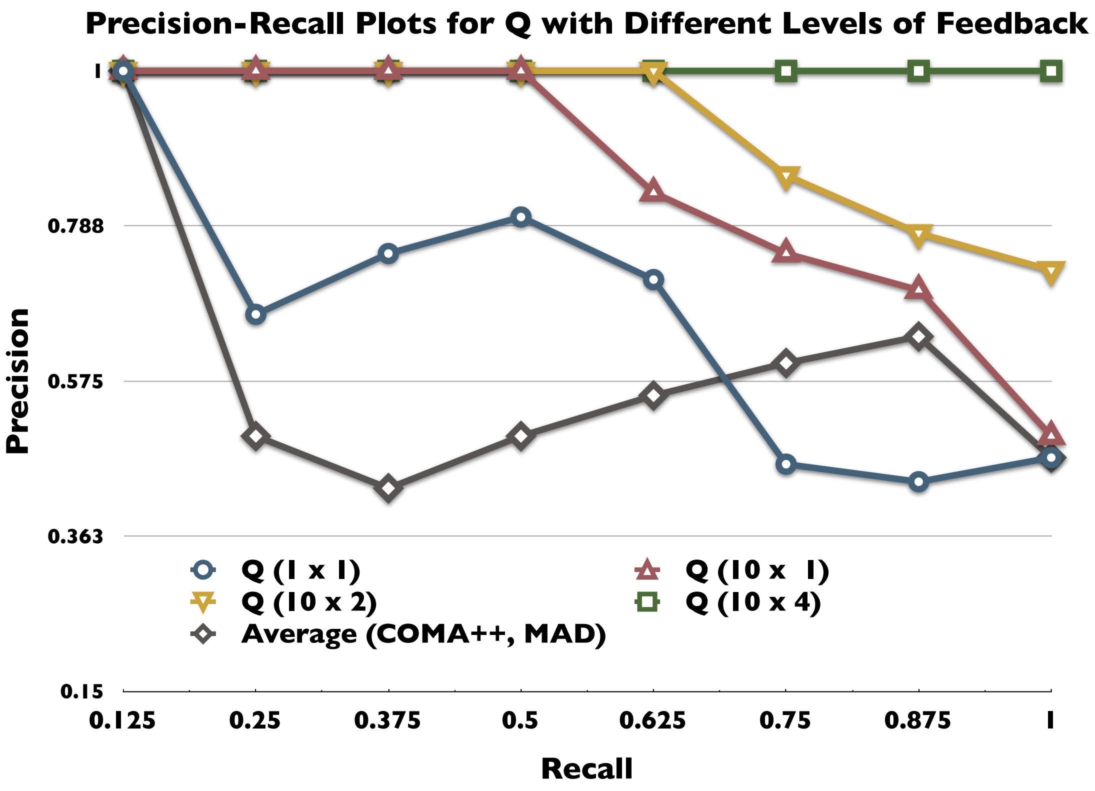


Figure 6.11: Precision versus recall in Q, given default weighting, then successively greater amounts of feedback.

(created as previously discussed), with $k = 5$ (see Algorithm 9). In order to ensure that weight updates are made in a way that consistently preserves all of the "good" answers, we actually apply the feedback *repeatedly* (we replay a log of the most recent feedback steps, recorded as a sliding window with a size bound). Here we input the 10 feedback items to the learner four times in succession (i.e., replay them three times) to reinforce them. In order to remove the edge cost variations resulting from intermediate feedbacks, we consider the average edge cost over all feedback steps.

To see the relationship between recall and precision levels, we vary a *pruning threshold* over the schema graph: any alignment edges with cost above this threshold will be ignored in query result generation, and any below will be included. Compared to both schema matchers in isolation, with the ten feedback steps, Q does a much better job of providing

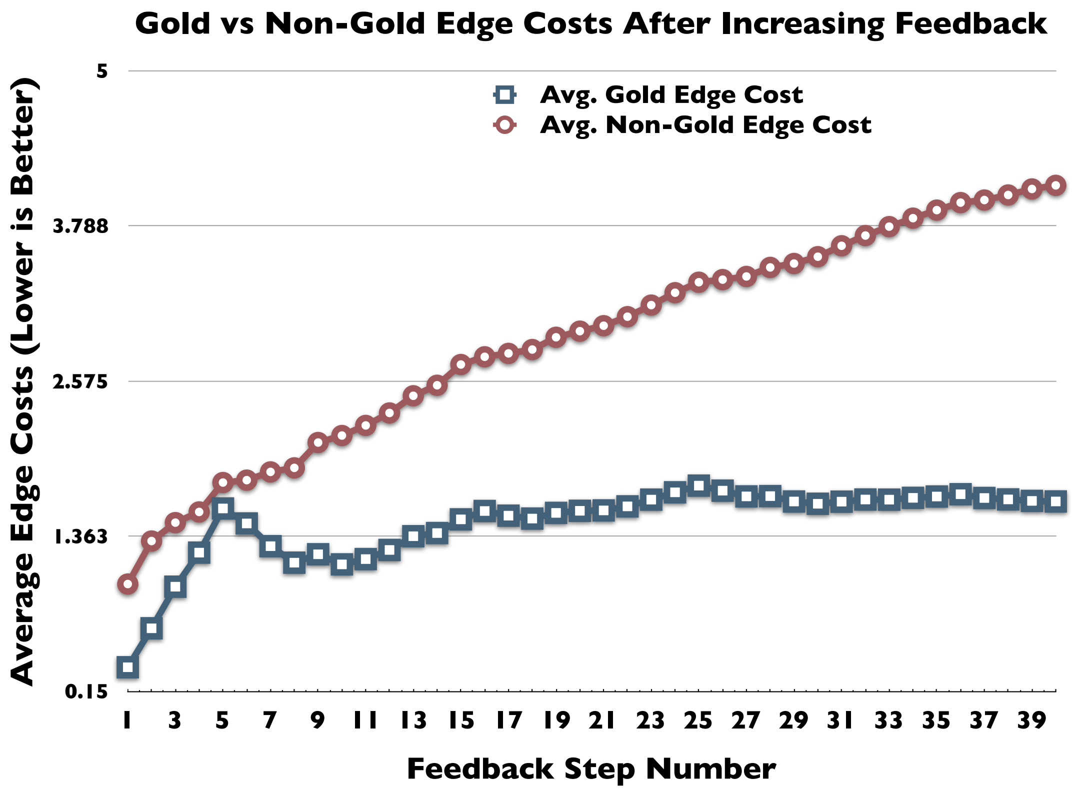


Figure 6.12: Average costs of gold edges (i.e., those in Figure 6.9) vs. non-gold edges in the search graph, as more feedback is applied. To obtain Steps 11–40 we repeat the feedback steps from 1–10 up to 3 times. Q continues to increase the gap between gold and non-gold edges' average scores.

both good precision and recall: we can get 100% precision with 100% recall.

Relative benefits of feedback. Next we study how performance improves with successive feedback steps. Figure 6.11 repeats the above experiment with increasing amounts of feedback. As a baseline, we start with the setting where the matchers' scores are simply averaged for every edge — in the absence of any feedback, we give equal weight to each matcher. Next we consider a single feedback step, designated Q (1x1), then ten feedback steps. Previously we had applied the feedback four successive times: we show here what happens if we do not repeat the feedback (10x1), if we repeat it once (10x2), and if we repeat it three times (10x4).

Recall Level	12.5	25	37.5	50	62.5	87.5	100
Feedback Steps	1	2	2	2	2	2	2

Table 6.2: Number of feedback steps required to initially get precision 1 with a certain recall level in the schema graph.

Looking at relative performance in Figure 6.11, we see that the baseline — the average of the two matchers' output — approximately follows the output of COMA++. It turns out that COMA++ gives higher confidence scores on average than MAD, and hence this simple average favors its alignments. Of course, we could adjust the default weighting accordingly — but it is far better to have the system automatically make this adjustment. We see from the graph that this happens quite effectively: after a single feedback step, we immediately see a noticeable boost in precision for most of the recall levels below 60%. Ten items of feedback with no repetitions makes a substantial difference, yielding precision of 100% for recall values all the way to 50%. However, repeating the feedback up to four times shows significant benefit.

Figure 6.12 shows the average costs of edges in the gold-standard (i.e., edges in Figure 6.9) versus non-gold edges, as we provide more feedback. Q assigns lower (better) costs on average to gold edges than to non-gold edges, and the gap increases with more feedback.

Feedback vs. precision for different recall levels. Finally, we consider the question of how much feedback is necessary to get perfect precision (hence, ultimately exact query answers) if we are willing to compromise on recall: Table 6.2 summarizes the results. Note that perfect precision is actually obtained with only 2 feedback steps even with 100% recall. At first glance this may seem incongruous with the results of the previous figures, but it is important to remember that each feedback step is given on a different query, and each time the online learner makes local adjustments that may counter the effects of the previous feedback steps. Hence we can see drops in precision with additional feedback steps, and it takes several more steps (plus, as we saw previously, multiple repetitions) before the overall effects begin to converge in a way that preserves all of the correct edges.

We conclude from these experiments that (1) the simple act of combining scores from different matchers is not enough to boost scores, (2) with a small number of feedback steps Q learns to favor the correct alignments, (3) particularly if a sequence of feedback steps is replayed several times, we can achieve very high precision and recall rates. Ultimately this means that we can learn to generate very high-quality answers directly using the output of existing schema matching components, plus feedback on the results.

6.6 Related Work

In this chapter, we addressed one of the shortcomings of the version of Q presented in [Talukdar et al., 2008b], namely, that all alignments were specified in advance. Many systems supporting keyword search over databases [Bhalotia et al., 2002, Botev and Shanmugasundaram, 2005, He et al., 2007, Hristidis and Papakonstantinou, 2002, Hristidis et al., 2003, Kacholia et al., 2005] use scores based on a combination of similarity between keywords and data values, length of join paths, and node authority [Balmin et al., 2004]. Existing "top-k query answering" [Cohen, 1998, Gravano et al., 2003, Li et al., 2005, Marian et al., 2004] provides the highest-scoring answers for ranked queries.

Schema alignment or matching is well-studied across the database, machine learning, and Semantic Web communities [Rahm and Bernstein, 2001]. General consensus is that methods that incorporate both data- and metadata-based features, and potentially custom learners and constraints, are most effective. Thus, most modern matchers combine output from multiple sub-matchers [Do and Rahm, 2007, Doan et al., 2001, Melnik et al., 2002]. Our focus is not on a new method for schema matching, but rather an *architecture* for incorporating the output of a matcher in a complete iterative, end-to-end pipeline where the matches or alignments are incorporated into existing user views, and *feedback on answers* is used to correct schema matching output. Our approach requires no special support within the matcher, simply leveraging it as a "black box."

The notion of propagating "influences" across node connectivity for schema alignment is used in similarity flooding [Melnik et al., 2002] and the Cupid system [Madhavan et al., 2001], among other schema matching studies. However, in the machine learning and Web communities, a great deal of work has been done to develop a principled family of *label propagation* algorithms [Baluja et al., 2008, Zhu et al., 2003]. In Section 6.3.2.2, we incorporate this kind of matching method not only to align compatible attributes in the output, but to discover synonymous tables and transitively related items. This builds upon recent observations (see Chapter 4) showing that one could find potential labelings of tables extracted from the Web using the *Modified Adsorption (MAD)* label propagation algorithm [Talukdar and Crammer, 2009]. Another benefit of the MAD-based method presented in Section 6.3.2.2 is that it doesn't require pariwise attribute comparisons, a scalability bottleneck, which is otherwise necessary in similarity flooding [Melnik et al., 2002].

Our ranked data model propagates uncertainty from uncertain mappings to output results. Intuitively, this resembles the model of *probabilistic schema mappings* [Dong et al., 2007], although we do not use a probabilistic model. Our goal is to *learn* rankings based on answer feedback, and hence we need a ranking model amenable to this.

Our work is complementary to efforts on learning to construct mashups [Tuchinda and Knoblock, 2008], in suggesting potential joins with new sources. Recent work on "pay as you go" integration has used decision theory to determine which feedback is most useful to a learner [Jeffery et al., 2008].

As opposed to feedback-driven query expansion and rewriting in [Pan et al., 2008], our goal here is to exploit user feedback to learn to correct schema matching errors. As mentioned in Chapter 5, a method that learns to rank pairs of nodes based on their graph-walk similarity is presented in [Minkov et al., 2006, Minkov and Cohen, 2007]. A similar method that learns the random walk probabilities in a graph satisfying pairwise node ordering constraints is presented in [Agarwal et al., 2006]. In contrast, the learning method used in this chapter learns to rank *trees* derived from the query graph, and not just node pairs. The method for incorporating user feedback as presented in [Chai et al., 2009] requires developers to implement declarative user feedback rules. We do not require any such intermediate rule implementation, and instead *learn* directly from user feedback over answers.

6.7 Summary of Chapter

In this chapter, we have developed an automatic, information need-driven strategy for automatically incorporating new sources and their information in a data integration setting. Schema matches or alignments, whether good or bad, only become apparent when they are used to produce query answers seen by a user; we exploit this to make the process of finding alignments with a new source more efficient, and also to allow the user with an information need to actually *correct* bad mappings through explicit feedback (from which the system *learns* new association weights). Through experiments on real-world datasets from the bioinformatics domain, we have demonstrated that our alignment scheme scales well. We have also demonstrated that our learning strategy is highly effective in combining the outputs of "black box" schema matchers and in re-weighting bad alignments. In doing this, we have also developed a new instance-based schema matcher using the MAD algorithm.

We believe that Q represents a step towards the ultimate goal of automated data integration, at least for particular kinds of datasets. In ongoing work we are expanding our experimental study to consider a wider array of domains, including Web sources with information extraction components.

Chapter 7

Conclusion

In this thesis, we argued in support of the statement: *Graph-based representation of data and learning over such graphs result in effective and scalable methods for large-scale information extraction and integration*. We made the following contributions:

- In Chapter 2, we proposed a novel context pattern induction method for entity extraction. We demonstrated effectiveness of the proposed method by extending seed entity lists of various types at fairly high precision. We also showed how performance of a state-of-the-art discriminative tagger can be improved by adding features derived from such extended entity lists.
- In Chapter 3, we used a graph-based semi-supervised label propagation algorithm, Adsorption, for acquiring open-domain labeled classes and their instances from a combination of unstructured and structured text sources. This allowed extractions from diverse sources and different methods to be put together in a single framework and perform joint learning and inference. This acquisition method significantly improved coverage compared to a previous set of labeled classes and instances derived from free text, while achieving comparable precision.
- Building on Adsorption, in Chapter 4, we presented a new label propagation algorithm, Modified Adsorption (MAD). We compared many label propagation methods

on a variety of real-world learning tasks, including class-instance acquisition, and found MAD to be the most effective. We also showed how class-instance acquisition performance in the graph-based SSL setting can be improved by including additional semantic constraints available in independently constructed knowledge bases.

- In Chapter 5, we focused on Information Integration and presented a novel system, Q, which drew ideas from machine learning and databases to help a non-expert user construct data-integrating queries based on keywords (across databases) and interactive feedback on answers. We evaluated the effectiveness of Q against gold standard costs from domain experts and demonstrated the method's scalability.

- In Chapter 6, we presented an information need-driven strategy for automatically incorporating new sources and their information in Q. This is particularly important in today's environment where new data sources are constantly showing up and there is a pressing need to make new source's data available to the user at the earliest. We also demonstrated that our learning strategy is highly effective in combining the outputs of "black box" schema matchers and in re-weighting bad alignments. This removes the need to develop an expensive mediated schema which has been necessary for most previous systems.

7.1 Future Work

In this section, we outline some of the promising avenues for future work:

- As one of the contributions of this thesis, we have demonstrated effectiveness of graph-based semi-supervised learning (SSL) algorithms in large-scale acquisition of class-instance pairs, which can be considered as extractions of a single relation: the IS-A relation. It will be interesting to explore whether similar methods could also be used to extract instantiations of other types of relations.

- In Chapter 4, we found that incorporation of additional semantic constraints in the form of (instance, attribute) pairs in the graph-based SSL setting can be quite helpful. Inclusion of other types of constraints (e.g., instance similarity, attribute similarity, per-node class sparsity, etc.) and measuring their impact is an interesting direction of future work. We note that importance of constraints during SSL in general have also been reported recently [Carlson et al., 2010]. Introduction of such additional constraints may also allow one to move beyond bipartite graphs, the dominant graph construction scheme used in Chapters 3, 4.

- For the experiments in Chapters 3 and 4, we have assumed that the entities are pre-segmented. It will be interesting to explore whether large class-instance repositories constructed by the methods proposed in this thesis can be used in conjunction with recently proposed methods [Bellare and McCallum, 2009] to quickly bootstrap extractors for large number of classes, with the extractors performing entity segmentation and classification at the same time. Additionally, exploring utility of such class-instance resources in non-IE tasks (e.g., machine translation, Web search) is a promising line of future work.

- So far, the language independent nature of the graph-based SSL methods proposed in this thesis has not been exploited. It will be very interesting to evaluate effectiveness of the proposed methods on non-English data sources.

- In Chapters 5 and 6, we have demonstrated Q's effectiveness on a variety of datasets obtained mostly from the life sciences domain. As part of future work, it will be interesting to apply Q on datasets from other domains, including Web sources. Moreover, a user study involving Q will also be very helpful in improving the system further. Initial step in this direction is currently underway.

- In the current setting, whenever a new model needs to be trained for a new user, the model parameters in Q are initialized to default values. Such *cold starts* may require larger amounts of feedback from the user, resulting in increased start-up time and inconvenience for the user. Exploring how to use an existing model, already trained for a current user, to *warm start* (or initialize) a model for a new user is an exciting avenue for future work. Social network information, e.g., appropriate inter-user similarity, and ideas from transfer learning [Raina et al., 2006] may be exploited for this task.

- Finally, it will be worthwhile to integrate the IE components developed in this thesis into the Q system, which will then allow a non-expert user to pose queries and integrate data from structured as well *unstructured* sources, in a *feedback-* and *need-driven* basis.

Bibliography

[Agarwal et al., 2006] Agarwal, A., Chakrabarti, S., and Aggarwal, S. (2006). Learning to rank networked entities. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 14–23. ACM New York, NY, USA. 86, 151

[Agichtein and Gravano, 2000] Agichtein, E. and Gravano, L. (2000). Snowball: Extracting relations from large plain-text collections. In *Proceedings of the Fifth ACM International Conference on Digital Libraries*. 6, 14, 21, 26, 30

[Ando and Zhang, 2005] Ando, R. and Zhang, T. (2005). A high-performance semi-supervised learning method for text chunking. In *Proceedings of ACL-2005*. Ann Arbor, USA. 24

[Azran, 2007] Azran, A. (2007). The rendezvous algorithm: multiclass semi-supervised learning with markov random walks. *Proceedings of the 24th international conference on Machine learning*, pages 49–56. 37

[Baeza-Yates and Ribeiro-Neto, 1999] Baeza-Yates, R. A. and Ribeiro-Neto, B. A. (1999). *Modern Information Retrieval*. ACM Press / Addison-Wesley. 84

[Balmin et al., 2004] Balmin, A., Hristidis, V., and Papakonstantinou, Y. (2004). Objectrank: Authority-based keyword search in databases. In *VLDB*. 85, 92, 133, 150

[Baluja et al., 2008] Baluja, S., Seth, R., Sivakumar, D., Jing, Y., Yagnik, J., Kumar, S., Ravichandran, D., and Aly, M. (2008). Video suggestion and discovery for youtube: taking random walks through the view graph. *Proceedings of WWW-2008*. 7, 31, 36, 37, 44, 45, 46, 49, 79, 126, 151

[Banko et al., 2007] Banko, M., Cafarella, M., Soderland, S., Broadhead, M., and Etzioni, O. (2007). Open information extraction from the web. *Procs. of IJCAI*. xiii, 65, 66

[Barabasi and Albert, 1999] Barabasi, A. and Albert, R. (1999). Emergence of scaling in random networks. *Science*, 286(5439):509. 132

[Bellare and McCallum, 2009] Bellare, K. and McCallum, A. (2009). Generalized Expectation Criteria for Bootstrapping Extractors using Record-Text Alignment. *EMNLP*. 155

[Bellare et al., 2007] Bellare, K., Talukdar, P., Kumaran, G., Pereira, F., Liberman, M., McCallum, A., and Dredze, M. (2007). Lightly-Supervised Attribute Extraction. *NIPS 2007 Workshop on Machine Learning for Web Search*. 31, 44, 68

[Bengio et al., 2007] Bengio, Y., Delalleau, O., and Roux, N. (2007). Semi-Supervised Learning, chapter Label Propogation and Quadratic Criterion. 54, 55

[Bernal et al., 2007] Bernal, A., Crammer, K., Hatzigeorgiou, A., and Pereira, F. (2007). Global discriminative training for higher-accuracy computational gene prediction. *PLoS Computational Biology*, 3. 104

[Bhalotia et al., 2002] Bhalotia, G., Hulgeri, A., Nakhe, C., Chakrabarti, S., and Sudarshan, S. (2002). Keyword searching and browsing in databases using BANKS. In *ICDE*. 9, 10, 84, 113, 118, 150

[Blitzer et al., 2007] Blitzer, J., Dredze, M., and Pereira, F. (2007). Biographies, bollywood, boom-boxes and blenders: Domain adaptation for sentiment classification. In *ACL*. 76

[Botev and Shanmugasundaram, 2005] Botev, C. and Shanmugasundaram, J. (2005). Context-sensitive keyword search and ranking for XML. In *WebDB*. 10, 84, 113, 150

[Boulakia et al., 2007] Boulakia, S. C., Biton, O., Davidson, S. B., and Froidevaux, C. (2007). BioGuideSRS: querying multiple sources with a user-centric perspective. *Bioinformatics*, 23(10). 3, 9, 10, 85, 104, 106, 107

[Brin, 1999] Brin, S. (1999). Extracting patterns and relations from the world wide web. *Lecture Notes in Computer Science*, pages 172–183. 6, 30

[Brin and Page, 1998] Brin, S. and Page, L. (1998). The anatomy of a large-scale hypertextual web search engine. *Computer Networks*, 30(1-7). 126

[Buneman et al., 2001] Buneman, P., Khanna, S., and Tan, W. C. (2001). Why and where: A characterization of data provenance. In *ICDT*. 89

[Bunescu and Mooney, 2005] Bunescu, R. and Mooney, R. (2005). A shortest path dependency kernel for relation extraction. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 724–731. Association for Computational Linguistics Morristown, NJ, USA. 6

[Cafarella et al., 2008] Cafarella, M., Halevy, A., Wang, D., Wu, E., and Zhang, Y. (2008). Webtables: Exploring the power of tables on the web. *VLDB*. 7, 31, 32, 33, 123

[Carey et al., 2000] Carey, M. J., Florescu, D., Ives, Z. G., Lu, Y., Shanmugasundaram, J., Shekita, E., and Subramanian, S. (2000). XPERANTO: Publishing object-relational data as XML. In *WebDB '00*. 99

[Carlson et al., 2010] Carlson, A., Betteridge, J., Wang, R., Hruschka Jr, E., and Mitchell, T. (2010). Coupled Semi-Supervised Learning for Information Extraction. In *Proceedings of the Third ACM International Conference on Web Search and Data Mining (WSDM)*, volume 2, page 110. 71, 155

[Chai et al., 2009] Chai, X., Vuong, B.-Q., Doan, A., and Naughton, J. F. (2009). Efficiently incorporating user feedback into information extraction and integration programs. New York, NY, USA. 152

[Chandrasekaran et al., 2003] Chandrasekaran, S., Cooper, O., Deshpande, A., Franklin, M. J., Hellerstein, J. M., Hong, W., Krishnamurthy, S., Madden, S., Raman, V., Reiss, F., and Shah, M. A. (2003). TelegraphCQ: Continuous dataflow processing for an uncertain world. In *CIDR*. 106

[Chapelle et al., 2006] Chapelle, O., Schölkopf, B., and Zien, A. (2006). *Semi-supervised learning*. MIT press. 7

[Chidlovskii, 2000] Chidlovskii, B. (2000). Wrapper generation by k-reversible grammar induction. *ECAI Workshop on Machine Learning for Information Extraction*. 18

[Cohen, 1998] Cohen, W. W. (1998). Integration of heterogeneous databases without common domains using queries based on textual similarity. In *SIGMOD*. 9, 85, 150

[Crammer et al., 2006a] Crammer, K., Dekel, O., Keshet, J., Shalev-Shwartz, S., and Singer, Y. (2006a). Online passive-aggressive algorithms. *Journal of Machine Learning Research*, 7:551–585. 91, 104

[Crammer et al., 2006b] Crammer, K., Dekel, O., Keshet, J., Shalev-Shwartz, S., and Singer, Y. (2006b). Online passive-aggressive algorithms. *Journal of Machine Learning Research*, 7:551–585. 115, 135

[Cui, 2001] Cui, Y. (2001). *Lineage Tracing in Data Warehouses*. PhD thesis, Stanford University. 89

[Culotta and Sorensen, 2004] Culotta, A. and Sorensen, J. (2004). Dependency tree kernels for relation extraction. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*. Association for Computational Linguistics Morristown, NJ, USA. 6

[Dalvi and Suciu, 2004] Dalvi, N. and Suciu, D. (2004). Efficient query evaluation on probabilistic databases. In *VLDB*. 92

[Dean and Ghemawat, 2008] Dean, J. and Ghemawat, S. (2008). Map Reduce: Simplified data processing on large clusters. *Communications of the ACM-Association for Computing Machinery-CACM*, 51(1):107–114. 37

[Do and Rahm, 2007] Do, H. H. and Rahm, E. (2007). Matching large schemas: Approaches and evaluation. *Inf. Syst.*, 32(6). 115, 125, 126, 144, 150

[Doan et al., 2001] Doan, A., Domingos, P., and Halevy, A. Y. (2001). Reconciling schemas of disparate data sources: A machine-learning approach. In *SIGMOD*. 150

[Dong et al., 2007] Dong, X. L., Halevy, A. Y., and Yu, C. (2007). Data integration with uncertainty. In *VLDB*. 151

[Duin and Volgenant, 1989] Duin, C. and Volgenant, A. (1989). Reduction tests for the steiner problem in graphs. *Netw.*, 19. 97

[Etzioni et al., 2005] Etzioni, O., Cafarella, M., Downey, D., Popescu, A.-M., Shaked, T., Soderland, S., Weld, D. S., and Yates, A. (2005). Unsupervised named-entity extraction from the web - an experimental study. *Artificial Intelligence Journal*. 6, 14, 21, 26, 30

[Fellbaum, 1998] Fellbaum, C., editor (1998). *WordNet: An Electronic Lexical Database and Some of its Applications*. MIT Press. 31

[Florian et al., 2003] Florian, R., Ittycheriah, A., Jing, H., and Zhang, T. (2003). Named entity recognition through classifier combination. In *Proceedings of CoNLL-2003*. 6, 25

[Franklin et al., 2005] Franklin, M., Halevy, A., and Maier, D. (2005). From databases to dataspaces: a new abstraction for information management. *SIGMOD Rec.*, 34(4). 114

[Freitag, 1997] Freitag, D. (1997). Using grammatical inference to improve precision in information extraction. In *ICML-97 Workshop on Automata Induction, Grammatical Inference, and Language Acquisition, Nashville*. 18

[Garey and Johnson, 1979] Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman. 94

[Ghahramani and Heller, 2006] Ghahramani, Z. and Heller, K. (2006). Bayesian sets. *Advances in Neural Information Processing Systems*, 18:435. 26, 34

[Gravano et al., 2003] Gravano, L., Ipeirotis, P. G., Koudas, N., and Srivastava, D. (2003). Text joins in an RDBMS for web data integration. In *WWW*. 85, 150

[Green et al., 2007a] Green, T. J., Karvounarakis, G., Ives, Z. G., and Tannen, V. (2007a). Update exchange with mappings and provenance. In *VLDB*. Amended version available as Univ. of Pennsylvania report MS-CIS-07-26. 101, 106

[Green et al., 2007b] Green, T. J., Karvounarakis, G., and Tannen, V. (2007b). Provenance semirings. In *PODS*. 89

[Guo et al., 2003] Guo, L., Shao, F., Botev, C., and Shanmugasundaram, J. (2003). XRANK: Ranked keyword search over XML documents. In *SIGMOD*. 85

[Halevy et al., 2003] Halevy, A. Y., Ives, Z. G., Suciu, D., and Tatarinov, I. (2003). Schema mediation in peer data management systems. In *ICDE*. 87

[He et al., 2007] He, H., Wang, H., Yang, J., and Yu, P. S. (2007). Blinks: ranked keyword searches on graphs. 150

[Hearst, 1992] Hearst, M. (1992). Automatic acquisition of hyponyms from large text corpora. In *Proceedings of the 14th International Conference on Computational Linguistics (COLING-92)*, pages 539–545, Nantes, France. 6, 30, 32

[Hristidis and Papakonstantinou, 2002] Hristidis, V. and Papakonstantinou, Y. (2002). Discover: Keyword search in relational databases. In *VLDB*. 9, 10, 84, 113, 118, 150

[Hristidis et al., 2003] Hristidis, V., Papakonstantinou, Y., and Balmin, A. (2003). Keyword proximity search on XML graphs. In *ICDE*. 150

[Indyk and Matousek, 2004] Indyk, P. and Matousek, J. (2004). Low-distortion embeddings of finite metric spaces. *Handbook of Discrete and Computational Geometry*. 37, 46

[Ives et al., 1999] Ives, Z. G., Florescu, D., Friedman, M. T., Levy, A. Y., and Weld, D. S. (1999). An adaptive query execution system for data integration. In *SIGMOD*. 106

[Jansen et al., 2000] Jansen, B., Spink, A., and Saracevic, T. (2000). Real life, real users, and real needs: a study and analysis of user queries on the Web. *Information Processing and Management*, 36(2):207–227. 29

[Jeffery et al., 2008] Jeffery, S. R., Franklin, M. J., and Halevy, A. Y. (2008). Pay-as-you-go user feedback for dataspace systems. In *SIGMOD*. 151

[Jiang and Conrath, 1997] Jiang, J. and Conrath, D. (1997). Semantic similarity based on corpus statistics and lexical taxonomy. *Arxiv preprint cmp-lg/9709008*. 71, 73

[Joachims, 1999] Joachims, T. (1999). Transductive inference for text classification using support vector machines. In *Sixteenth International Conference on Machine Learning*. 75

[Joachims, 2003] Joachims, T. (2003). Transductive learning via spectral graph partitioning. In *ICML*. 75

[Kacholia et al., 2005] Kacholia, V., Pandit, S., Chakrabarti, S., Sudarshan, S., Desai, R., and Karambelkar, H. (2005). Bidirectional expansion for keyword search on graph databases. In *VLDB*. 10, 84, 92, 113, 118, 150

[Kasneci et al., 2009] Kasneci, G., Ramanath, M., Sozio, M., Suchanek, F. M., and Weikum, G. (2009). Star: Steiner-tree approximation in relationship graphs. In *ICDE*. 121

[Kasneci et al., 2008] Kasneci, G., Suchanek, F. M., Ifrim, G., Ramanath, M., and Weikum, G. (2008). Naga: Searching and ranking knowledge. In *ICDE*. 85

[Katz, 1979] Katz, V. (1979). The history of Stokes' theorem. *Mathematics Magazine*, 52(3):146–156. 52

[Kimelfeld and Sagiv, 2006] Kimelfeld, B. and Sagiv, Y. (2006). Finding and approximating top-k answers in keyword proximity search. In *PODS*. 96

[Kissinger et al., 2002] Kissinger, J. C., Brunk, B. P., Crabtree, J., Fraunholz, M. J., Gajria, B., Milgram, A. J., Pearson, D. S., Schug, J., Bahl, A., Diskin, S. J., Ginsburg, H., Grant, G. R., Gupta, D., Labo, P., Li, L., Mailman, M. D., McWeeney, S. K., Whetzel, P., Stoeckert, Jr., C. J., and Roos, D. S. (2002). The Plasmodium genome database: Designing and mining a eukaryotic genomics resource. *Nature*, 419. 3

[Lafferty et al., 2001] Lafferty, J., McCallum, A., and Pereira, F. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. ICML 2001*. 24

[Lawler, 1972] Lawler, E. L. (1972). A procedure for computing the k best solutions to discrete optimization problems and its application to the shortest path problem. *Management Science*, 18. 96

[Li et al., 2005] Li, C., Chang, K. C.-C., Ilyas, I. F., and Song, S. (2005). RankSQL: Query algebra and optimization for relational top-k queries. In *SIGMOD*. 85, 150

[Liang, 2005] Liang, P. (2005). Semi-supervised learning for natural language. *MEng. Thesis, MIT*. 24

[Lin and Pantel, 2002] Lin, D. and Pantel, P. (2002). Concept discovery from text. In *Proceedings of the 19th International Conference on Computational linguistics (COLING-02)*, pages 1–7. 32

[Lin et al., 2003] Lin, W., Yangarber, R., and Grishman, R. (2003). Bootstrapped learning of semantic classes from positive and negative examples. In *Proceedings of ICML-2003 Workshop on The Continuum from Labeled to Unlabeled Data*. 14, 21, 26

[Madhavan et al., 2001] Madhavan, J., Bernstein, P. A., and Rahm, E. (2001). Generic schema matching with Cupid. In *VLDB*. 126, 151

[Madhavan et al., 2008] Madhavan, J., Ko, D., Kot, L., Ganapathy, V., Rasmussen, A., and Halevy, A. Y. (2008). Google's deep web crawl. *PVLDB*, 1(2). 123

[Marian et al., 2004] Marian, A., Bruno, N., and Gravano, L. (2004). Evaluating top-k queries over web-accessible databases. *ACM Trans. Database Syst.*, 29(2). 85, 150

[McCarthy and Lehnert, 1995] McCarthy, K. and Lehnert, W. (1995). Using decision trees for coreference resolution. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI-95)*, pages 1050–1055, Montreal, Quebec. 30

[McDonald and Pereira, 2006] McDonald, R. and Pereira, F. (2006). Online learning of approximate dependency parsing algorithms. In *European Association for Computational Linguistics*. 104

[Melnik et al., 2002] Melnik, S., Garcia-Molina, H., and Rahm, E. (2002). Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In *ICDE*. 126, 150, 151

[Metaweb Technologies, 2009] Metaweb Technologies (2009). Freebase data dumps. *http://download.freebase.com/datadumps/*. 60

[Miller et al., 2004] Miller, S., Guinness, J., and Zamanian, A. (2004). Name tagging with word clusters and discriminative training. In *Proceedings of HLT-NAACL 2004*. 24

[Minkov and Cohen, 2007] Minkov, E. and Cohen, W. (2007). Learning to rank typed graph walks: Local and global approaches. In *Proceedings of the 9th WebKDD and 1st SNA-KDD 2007 workshop on Web mining and social network analysis*, pages 1–8. ACM. 85, 151

[Minkov et al., 2006] Minkov, E., Cohen, W. W., and Ng, A. Y. (2006). Contextual search and name disambiguation in email using graphs. In *SIGIR*. 85, 151

[Mork et al., 2002] Mork, P., Shaker, R., Halevy, A., and Tarczy-Hornoch, P. (2002). PQL: A declarative query language over dynamic biological schemata. In *AMIA Symposium*. 3, 9, 10, 85, 106

[Pan et al., 2008] Pan, H., Schenkel, R., and Weikum, G. (2008). Fine-grained relevance feedback for XML retrieval. In *SIGIR*. 151

[Pantel et al., 2009] Pantel, P., Crestan, E., Borkovsky, A., Popescu, A., and Vyas, V. (2009). Web-scale distributional similarity and entity set expansion. *Proceedings of EMNLP-09, Singapore*. 64

[Pantel and Pennacchiotti, 2006] Pantel, P. and Pennacchiotti, M. (2006). Espresso: Leveraging generic patterns for automatically harvesting semantic relations. In *COLING/ACL-06, Sydney, Australia*. 30

[Pasca and Durme, 2007] Pasca, M. and Durme, B. V. (2007). What you seek is what you get: Extraction of class attributes from query logs. In *IJCAI-07. Ferbruary, 2007*. 7, 68

[Pennacchiotti and Pantel, 2009] Pennacchiotti, M. and Pantel, P. (2009). Entity Extraction via Ensemble Semantics. *Proceedings of EMNLP-09, Singapore*. 70

[Popescu et al., 2003] Popescu, A.-M., Etzioni, O., and Kautz, H. (2003). Towards a theory of natural language interfaces to databases. In *IUI '03*. 84

[Probst et al., 2007] Probst, K., Ghani, R., Krema, M., Fano, A., and Liu, Y. (2007). Semi-supervised learning of attribute-value pairs from product descriptions. In *IJCAI-07, Ferbruary, 2007*. 7, 68

[Rabiner, 1989] Rabiner, L. R. (1989). A tutorial on hidden markov models and selected applications in speech recognition. In *Proc. of IEEE, 77, 257–286*. 19

[Raghavan et al., 1989] Raghavan, V., Bollmann, P., and Jung, G. (1989). A critical investigation of recall and precision as measures of retrieval system performance. *ACM Transactions on Information Systems (TOIS)*, 7(3):205–229. 74, 75

[Rahm and Bernstein, 2001] Rahm, E. and Bernstein, P. A. (2001). A survey of approaches to automatic schema matching. *VLDB J.*, 10(4). 8, 85, 125, 150

[Raina et al., 2006] Raina, R., Ng, A., and Koller, D. (2006). Constructing informative priors using transfer learning. In *Proceedings of the 23rd international conference on Machine learning*, page 720. ACM. 156

[Reisinger and Pasca, 2009] Reisinger, J. and Pasca, M. (2009). Bootstrapped extraction of class attributes. In *Proceedings of the 18th international conference on World wide web*, pages 1235–1236. ACM. 71

[Riloff and Jones, 1999] Riloff, E. and Jones, R. (1999). Learning Dictionaries for Information Extraction by Multi-level Boot-strapping. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence*. 6, 14, 26, 30

[Saad, 2003] Saad, Y. (2003). *Iterative Methods for Sparse Linear Systems*. Society for Industrial Mathematics. 55, 57

[Sang and De Meulder, 1837] Sang, E. and De Meulder, F. (1837). Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. *Development*, 922:1341. 6

[Sarawagi, 2007] Sarawagi, S. (2007). Information Extraction. *Foundations and Trends® in Databases*, 1(3):261–377. 6

[Shen et al., 2008] Shen, W., DeRose, P., McCann, R., Doan, A., and Ramakrishnan, R. (2008). Toward best-effort information extraction. In *SIGMOD*, New York, NY, USA. 114

[Solan et al., 2005] Solan, Z., Horn, D., Ruppin, E., and Edelman, S. (2005). Unsupervised learning of natural languages. In *Proceedings of National Academy of Sciiences. 102:11629-11634*. 18

[Subramanya and Bilmes, 2008] Subramanya, A. and Bilmes, J. (2008). Soft-Supervised Learning for Text Classification. In *Proceedings of EMNLP. Honolulu, Hawaii*. xviii, 53, 73, 74, 75, 76

[Suchanek et al., 2007] Suchanek, F., Kasneci, G., and Weikum, G. (2007). Yago: a core of semantic knowledge. In *Proceedings of the 16th international conference on World Wide Web*, page 706. ACM. 64, 66, 69

[Szummer and Jaakkola, 2002] Szummer, M. and Jaakkola, T. (2002). Partially labeled classification with markov random walks. *Advances in Neural Information Processing Systems 14: Proceedings of the 2002 NIPS Conference*. 37, 46

[Talukdar et al., 2006] Talukdar, P. P., Brants, T., Pereira, F., and Liberman, M. (2006). A context pattern induction method for named entity extraction. In *Tenth Conference on Computational Natural Language Learning*, page 141. 13

[Talukdar and Crammer, 2009] Talukdar, P. P. and Crammer, K. (2009). New regularized algorithms for transductive learning. In *ECML-PKDD*. 45, 67, 115, 126, 127, 128, 145, 151

[Talukdar et al., 2010] Talukdar, P. P., Ives, Z., and Pereira, F. (2010). Automatically Incorporating New Sources in Keyword Search-Based Data Integration . *SIGMOD*. 113

[Talukdar et al., 2008a] Talukdar, P. P., Jacob, M., Mehmood, M., Crammer, K., Ives, Z., Pereira, F., and Guha, S. (2008a). Learning to create data-integrating queries. *Proceedings of the VLDB Endowment archive*, 1(1):785–796. 81

[Talukdar et al., 2008b] Talukdar, P. P., Jacob, M., Mehmood, M. S., Crammer, K., Ives, Z. G., Pereira, F., and Guha, S. (2008b). Learning to create data-integrating queries. In *VLDB*. 113, 114, 118, 121, 122, 135, 150

[Talukdar and Pereira, 2010] Talukdar, P. P. and Pereira, F. (2010). Experiments in graph-based semi-supervised learning methods for class-instance acquisition. In *48th Annual Meeting of the Association for Computational Linguistics (ACL)*. 45

[Talukdar et al., 2008c] Talukdar, P. P., Reisinger, J., Pasca, M., Ravichandran, D., Bhagat, R., and Pereira, F. (2008c). Weakly-Supervised Acquisition of Labeled Class Instances using Graph Random Walks. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 581–589. 7, 29, 59, 70

[Talukdar et al., 2008d] Talukdar, P. P., Reisinger, J., Pasca, M., Ravichandran, D., Bhagat, R., and Pereira, F. (2008d). Weakly supervised acquisition of labeled class instances using graph random walks. In *EMNLP*. 115

[Thelen and Riloff, 2002] Thelen, M. and Riloff, E. (2002). A bootstrapping method for learning semantic lexicons using extraction pattern contexts. In *Proceedings of EMNLP 2002*. 6, 14, 30

[Tuchinda and Knoblock, 2008] Tuchinda, R. and Knoblock, C. A. (2008). Building mashups by example. In *IUI '08*. 85, 151

[Van Durme and Paşca, 2008] Van Durme, B. and Paşca, M. (2008). Finding cars, goddesses and enzymes: Parametrizable acquisition of labeled instances for open-domain information extraction. *Twenty-Third AAAI Conference on Artificial Intelligence*. xvii, 7, 31, 32, 38

[Varadarajan et al., 2008] Varadarajan, R., Hristidis, V., and Raschid, L. (2008). Explaining and reformulating authority flow queries. In *ICDE*. 92

[Wang and Cohen, 2007] Wang, R. and Cohen, W. (2007). Language-Independent Set Expansion of Named Entities Using the Web. *Data Mining, 2007. ICDM 2007. Seventh IEEE International Conference on*, pages 342–350. 27, 43

[Winter, 1987] Winter, P. (1987). Steiner problem in networks: a survey. *Netw.*, 17(2). 98

[Winter and Smith, 1992] Winter, P. and Smith, J. M. (1992). Path-distance heuristics for the steiner problem in undirected networks. *Algorithmica*, 7(2&3):309–327. 97, 98

[Wolsey, 1998] Wolsey, L. (1998). *Integer Programming*. Wiley-Interscience. 94

[Wong, 1981] Wong, R. T. (1981). A dual ascent approach for steiner tree problems on a directed graph. *Mathematical Programming*, 28(3):271–287. 94

[Yen, 1971] Yen, J. Y. (1971). Finding the k shortest loopless paths in a network. *Management Science*, 18(17). 98

[Zhang and Johnson, 2003] Zhang, T. and Johnson, D. (2003). A robust risk minimization based named entity recognition system. In *Proceedings of CoNLL-2003*. 25

[Zhang et al., 2004] Zhang, Z., He, B., and Chang, K. C.-C. (2004). Understanding web query interfaces: Best-effort parsing with hidden syntax. In *SIGMOD*. 123

[Zhu, 2005] Zhu, X. (2005). Semi-supervised learning literature survey. 7

[Zhu and Ghahramani, 2002] Zhu, X. and Ghahramani, Z. (2002). Learning from labeled and unlabeled data with label propagation. Technical report, CMU CALD tech report CMU-CALD-02. 75, 76

[Zhu et al., 2003] Zhu, X., Ghahramani, Z., and Lafferty, J. (2003). Semi-supervised learning using gaussian fields and harmonic functions. *ICML-03, 20th International Conference on Machine Learning*. 37, 46, 53, 59, 126, 151